

Performance Optimization and Tuning Techniques for IBM Power Systems Processors Including IBM POWER8

Peter Bergner

Bernard King Smith

Brian Hall

Julian Wang

Alon Shalev Housfater

Suresh Warriar

Madhusudanan Kandasamy

David Wendt

Tulio Magno

Alex Mericas

Steve Munroe

Mauricio Oliveira

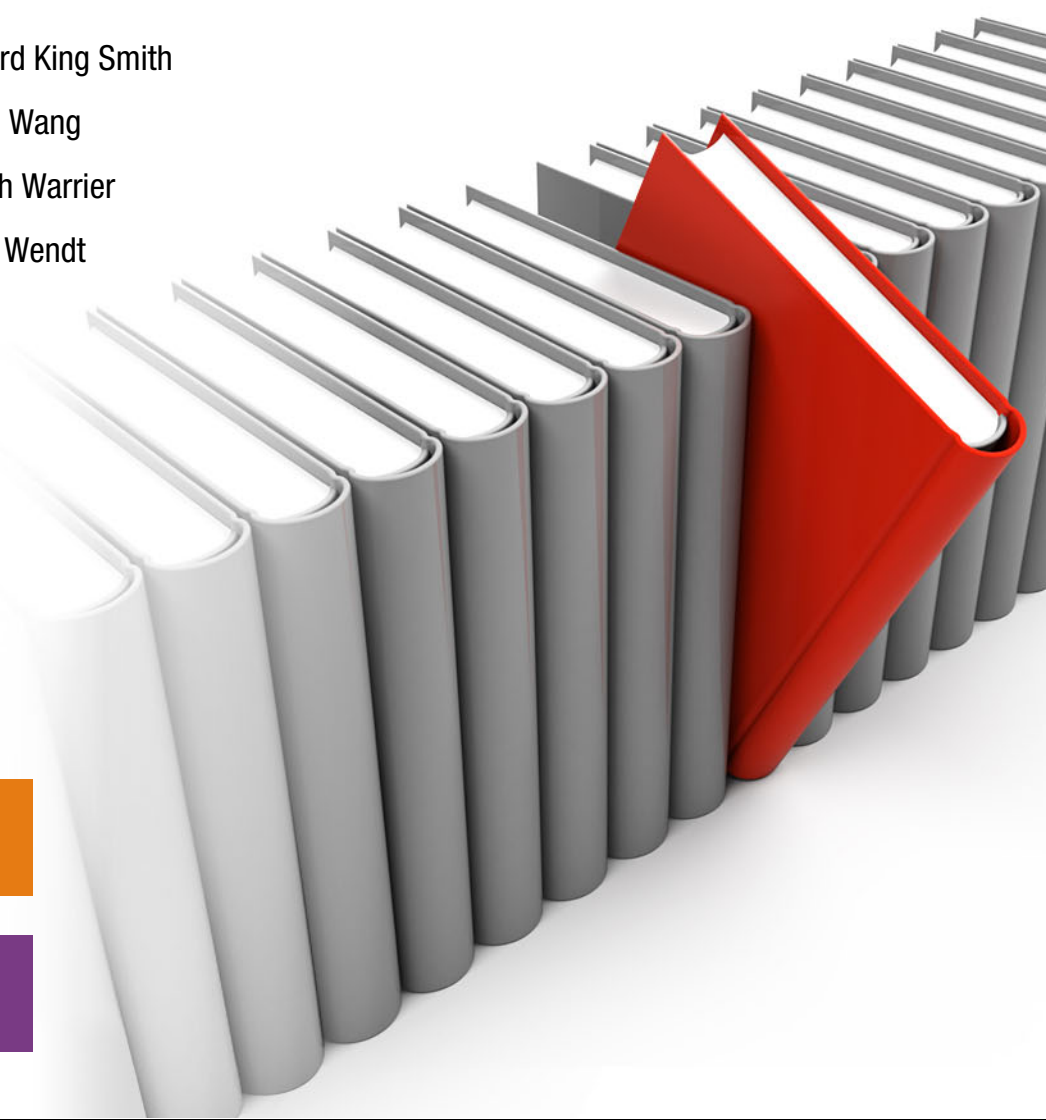
Bill Schmidt

Will Schmidt



Analytics

Power Systems





International Technical Support Organization

**Performance Optimization and Tuning Techniques for
IBM Power Systems Processors Including IBM
POWER8**

August 2015

Note: Before using this information and the product it supports, read the information in “Notices” on page ix.

Second Edition (August 2015)

This edition pertains to IBM Power Systems servers based on IBM Power Systems processor-based technology, including but not limited to IBM POWER8 processor-based systems. Specific software levels and firmware levels that are used are noted throughout the text.

© Copyright International Business Machines Corporation 2014, 2015. All rights reserved.

Note to U.S. Government Users Restricted Rights -- Use, duplication or disclosure restricted by GSA ADP Schedule Contract with IBM Corp.

Contents

Notices	ix
Trademarks	x
IBM Redbooks promotions	xi
Preface	xiii
Authors	xiii
Now you can become a published author, too!	xvii
Comments welcome	xvii
Stay connected to IBM Redbooks	xvii
Summary of changes	xix
August 2015, Second Edition	xix
Chapter 1. Optimization and tuning on IBM POWER8 processor-based systems	1
1.1 Introduction	2
1.2 Outline of this guide	2
1.3 Conventions that are used in this guide	5
1.4 Background	5
1.5 Optimizing performance on POWER8 processor-based systems	6
1.5.1 Lightweight tuning and optimization guidelines	7
1.5.2 Deployment guidelines	15
1.5.3 Deep performance optimization guidelines	21
Chapter 2. The IBM POWER8 processor	25
2.1 Introduction to the POWER8 processor	26
2.2 Using POWER8 features	28
2.2.1 Multi-core and multi-thread	28
2.2.2 Multipage size support (page sizes (4 KB, 64 KB, 16 MB, and 16 GB))	32
2.2.3 Efficient use of cache and memory	33
2.2.4 Transactional memory	42
2.2.5 Vector Scalar eXtension	45
2.2.6 Decimal floating point	47
2.2.7 In-core cryptography and integrity enhancements	47
2.2.8 On-chip accelerators	48
2.2.9 Storage synchronization (sync, lwsync, lwarx, stwcx., and eieio)	49
2.2.10 Fixed-point load and store quadword instructions	51
2.2.11 Instruction fusion	51
2.2.12 Event-based branches (or user-level fast interrupts)	52
2.2.13 Power management and system performance	52
2.2.14 Coherent Accelerator Processor Interface	53
2.3 I/O adapter affinity	55
2.4 Related publications	55
Chapter 3. The IBM POWER Hypervisor	57
3.1 Introduction to PowerVM	58
3.2 Power Systems virtualization with PowerVM	59
3.2.1 Virtual processors	59
3.2.2 Page table sizes for LPARs	63

3.2.3	Placing LPAR resources to attain higher memory affinity	63
3.2.4	Active memory expansion	66
3.2.5	Optimizing resource placement: Dynamic Platform Optimizer	67
3.2.6	Partition compatibility mode	67
3.3	Introduction to KVM Virtualization	67
3.4	Related publications	68
Chapter 4. IBM AIX		71
4.1	Introduction	72
4.2	Using Power Architecture features with AIX	72
4.2.1	Multi-core and multi-thread	72
4.2.2	Multipage size support on AIX	83
4.2.3	Efficient use of cache	86
4.2.4	Transactional memory	89
4.2.5	Vector Scalar eXtension	91
4.2.6	Decimal floating point	92
4.2.7	On-chip encryption accelerator	94
4.3	AIX operating system-specific optimizations	95
4.3.1	Malloc	95
4.3.2	Pthread tunables	97
4.3.3	pollset	98
4.3.4	File system performance benefits	98
4.3.5	Direct I/O	98
4.3.6	Concurrent I/O	99
4.3.7	Asynchronous I/O	99
4.3.8	I/O completion ports	100
4.3.9	shmat versus mmap	100
4.3.10	Large segment tunable aliasing (LSA)	101
4.3.11	64-bit versus 32-bit ABIs	101
4.3.12	Sleep and wake-up primitives (thread_wait and thread_post)	102
4.3.13	Shared versus private loads	103
4.3.14	Workload partition shared licensed program installations	104
4.4	AIX preferred practices	105
4.4.1	AIX preferred practices that are applicable to all Power Systems generations.	105
4.4.2	AIX preferred practices that are applicable to POWER7 and POWER8 processor-based systems	106
4.5	Related publications	107
Chapter 5. IBM i		111
5.1	Introduction	112
5.2	Using Power features with IBM i	112
5.2.1	Multi-core and multi-thread	112
5.2.2	Multipage size support on IBM i	113
5.2.3	Vector Scalar eXtension	113
5.2.4	Decimal floating point	113
5.3	IBM i operating system-specific optimizations	114
5.3.1	IBM i advanced optimization techniques	114
5.3.2	Performance management on IBM i	115
5.4	Related publications	116
Chapter 6. Linux		117
6.1	Introduction	118
6.2	Using Power features with Linux	118
6.2.1	Multi-core and multi-thread	119

6.2.2	Multipage size support on Linux	123
6.2.3	Efficient use of cache	123
6.2.4	Transactional memory.	124
6.2.5	Vector Scalar eXtension	125
6.2.6	Decimal floating point	126
6.2.7	Event-based branches	128
6.3	Linux operating system-specific optimizations	129
6.3.1	GCC, toolchain, and IBM Advance Toolchain.	129
6.3.2	Tuning and optimizing malloc	133
6.3.3	Large TOC -mmodel=medium optimization	137
6.3.4	POWER7 based distro considerations	137
6.3.5	Microthreading considerations	137
6.4	Little Endian	138
6.4.1	Application binary interface.	139
6.5	Related publications	139
Chapter 7. Compilers and optimization tools for C, C++, and Fortran.		141
7.1	Compiler versions and optimization levels	142
7.2	Advanced compiler optimization techniques	143
7.2.1	Common prerequisites	143
7.2.2	XL compiler family.	144
7.2.3	GCC compiler family.	146
7.3	Capitalizing on POWER8 features with the XL and GCC compilers.	148
7.3.1	In-core cryptography.	148
7.3.2	Compiler support for Vector Scalar eXtension	151
7.3.3	Built-in functions for storage synchronization	154
7.3.4	Data Streams Control Register controls	154
7.3.5	Transactional memory.	156
7.4	IBM Feedback Directed Program Restructuring	160
7.4.1	Introduction	160
7.4.2	Feedback Directed Program Restructuring supported environments.	162
7.4.3	Acceptable input formats	162
7.4.4	General operation	162
7.4.5	Instrumentation and profiling.	164
7.4.6	Optimization	165
7.5	Using the Advance Toolchain with IBM XLC and XLF	169
7.6	Using GPU accelerators with C/C++.	169
7.7	Related publications	171
Chapter 8. Java		173
8.1	Java levels	174
8.2	32-bit versus 64-bit Java.	174
8.2.1	Little Endian support.	175
8.3	Memory and page size considerations	175
8.3.1	Medium and large pages for Java heap and code cache	175
8.3.2	Configuring large pages for Java heap and code cache.	176
8.3.3	Prefetching	176
8.3.4	Compressed references	177
8.3.5	JIT code cache	180
8.3.6	Shared classes	181
8.4	Capitalizing on POWER8 features with IBM Java.	181
8.4.1	In-core Advanced Encryption Standard and Secure Hash Algorithm acceleration and instructions	181

8.4.2	Transactional memory	182
8.4.3	Runtime instrumentation	183
8.5	Java garbage collection tuning	183
8.5.1	GC strategy: Optthruput	183
8.5.2	GC strategy: Optavgpause	184
8.5.3	GC strategy: Gencon	184
8.5.4	GC strategy: Balanced	184
8.5.5	Optimal heap size	185
8.6	Application scaling	186
8.6.1	Choosing the correct simultaneous multithreading mode	186
8.6.2	Using resource sets	187
8.6.3	Java lock reservation	189
8.6.4	Java GC threads	189
8.6.5	Java concurrent marking	189
8.7	Using GPU accelerators with IBM Java	190
8.7.1	Automatic GPU compilation	190
8.7.2	Accessing the GPU through the CUDA4J application programming interface	191
8.7.3	The com.ibm.gpu application programming interface	191
8.7.4	NVIDIA Compute Unified Device Architecture: Java Native interface	191
8.8	Related publications	192
Chapter 9. IBM DB2		193
9.1	DB2 and the POWER processor	194
9.2	Taking advantage of the POWER processor	194
9.2.1	Affinitization	194
9.2.2	Page sizes	195
9.2.3	Decimal arithmetic	196
9.2.4	Using simultaneous multithreading priorities for internal lock implementation	196
9.2.5	Single Instruction Multiple Data	196
9.3	Capitalizing on the compilers and optimization tools for POWER	197
9.3.1	Whole-program analysis and profile-based optimizations	198
9.3.2	IBM Feedback Directed Program Restructuring	198
9.4	Capitalizing on POWER virtualization	198
9.4.1	DB2 virtualization	198
9.4.2	DB2 in an AIX workload partition	199
9.5	Capitalizing on the AIX system libraries	199
9.5.1	Using the thread_post_many API	199
9.5.2	File systems	200
9.6	Capitalizing on performance tools	201
9.6.1	High-level investigation	201
9.6.2	Low-level investigation	201
9.7	Conclusion	202
9.8	Related publications	202
Chapter 10. IBM WebSphere Application Server		205
10.1	IBM WebSphere	206
10.1.1	Installation	206
10.1.2	Deployment	206
10.1.3	Performance	207
10.1.4	Performance analysis, problem determination, and diagnostic tests	209
Appendix A. Analyzing malloc usage under IBM AIX		211
	Introduction	212
	How to collect malloc usage information	212

Appendix B. Performance tools and empirical performance analysis	215
Introduction	216
Performance advisors	216
Expert system advisors	216
IBM Rational Performance Advisor	221
IBM Power Virtualization Performance	223
AIX	223
CPU profiling	224
AIX trace-based analysis tools	226
Finding emulation issues	232
hpmstat, hpmcount, and tprof -E	232
Linux	233
Empirical performance analysis by using the IBM Software Development Kit for Linux on Power	233
Using the IBM SDK for Linux on Power Trace Analyzer	235
High library usage	235
Deeper empirical analysis	236
Java (either AIX or Linux)	239
32-bit or 64-bit JDK	240
Java heap size, and garbage collection policies and parameters	240
Hot method or routine analysis	241
Locking analysis	246
Thread state analysis	246

Notices

This information was developed for products and services offered in the U.S.A.

IBM may not offer the products, services, or features discussed in this document in other countries. Consult your local IBM representative for information on the products and services currently available in your area. Any reference to an IBM product, program, or service is not intended to state or imply that only that IBM product, program, or service may be used. Any functionally equivalent product, program, or service that does not infringe any IBM intellectual property right may be used instead. However, it is the user's responsibility to evaluate and verify the operation of any non-IBM product, program, or service.

IBM may have patents or pending patent applications covering subject matter described in this document. The furnishing of this document does not grant you any license to these patents. You can send license inquiries, in writing, to:

IBM Director of Licensing, IBM Corporation, North Castle Drive, Armonk, NY 10504-1785 U.S.A.

The following paragraph does not apply to the United Kingdom or any other country where such provisions are inconsistent with local law: INTERNATIONAL BUSINESS MACHINES CORPORATION PROVIDES THIS PUBLICATION "AS IS" WITHOUT WARRANTY OF ANY KIND, EITHER EXPRESS OR IMPLIED, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF NON-INFRINGEMENT, MERCHANTABILITY OR FITNESS FOR A PARTICULAR PURPOSE. Some states do not allow disclaimer of express or implied warranties in certain transactions, therefore, this statement may not apply to you.

This information could include technical inaccuracies or typographical errors. Changes are periodically made to the information herein; these changes will be incorporated in new editions of the publication. IBM may make improvements and/or changes in the product(s) and/or the program(s) described in this publication at any time without notice.

Any references in this information to non-IBM websites are provided for convenience only and do not in any manner serve as an endorsement of those websites. The materials at those websites are not part of the materials for this IBM product and use of those websites is at your own risk.

IBM may use or distribute any of the information you supply in any way it believes appropriate without incurring any obligation to you.

Any performance data contained herein was determined in a controlled environment. Therefore, the results obtained in other operating environments may vary significantly. Some measurements may have been made on development-level systems and there is no guarantee that these measurements will be the same on generally available systems. Furthermore, some measurements may have been estimated through extrapolation. Actual results may vary. Users of this document should verify the applicable data for their specific environment.

Information concerning non-IBM products was obtained from the suppliers of those products, their published announcements or other publicly available sources. IBM has not tested those products and cannot confirm the accuracy of performance, compatibility or any other claims related to non-IBM products. Questions on the capabilities of non-IBM products should be addressed to the suppliers of those products.

This information contains examples of data and reports used in daily business operations. To illustrate them as completely as possible, the examples include the names of individuals, companies, brands, and products. All of these names are fictitious and any similarity to the names and addresses used by an actual business enterprise is entirely coincidental.


COPYRIGHT LICENSE:

This information contains sample application programs in source language, which illustrate programming techniques on various operating platforms. You may copy, modify, and distribute these sample programs in any form without payment to IBM, for the purposes of developing, using, marketing or distributing application programs conforming to the application programming interface for the operating platform for which the sample programs are written. These examples have not been thoroughly tested under all conditions. IBM, therefore, cannot guarantee or imply reliability, serviceability, or function of these programs.

Trademarks

IBM, the IBM logo, and ibm.com are trademarks or registered trademarks of International Business Machines Corporation in the United States, other countries, or both. These and other IBM trademarked terms are marked on their first occurrence in this information with the appropriate symbol (® or ™), indicating US registered or common law trademarks owned by IBM at the time this information was published. Such trademarks may also be registered or common law trademarks in other countries. A current list of IBM trademarks is available on the Web at <http://www.ibm.com/legal/copytrade.shtml>

The following terms are trademarks of the International Business Machines Corporation in the United States, other countries, or both:

Active Memory™	Power Architecture®	PowerPC®
AIX®	POWER Hypervisor™	PowerVM®
AIX 5L™	Power Systems™	PowerVP™
Blue Gene/L®	Power Systems Software™	Rational®
DB2®	POWER6®	Redbooks®
FDPR®	POWER6+™	Redbooks (logo)  ®
IBM®	POWER7®	System z®
IBM Watson™	POWER7+™	Tivoli®
Micro-Partitioning®	POWER8®	WebSphere®
POWER®	PowerLinux™	

The following terms are trademarks of other companies:

Intel, Intel logo, Intel Inside logo, and Intel Centrino logo are trademarks or registered trademarks of Intel Corporation or its subsidiaries in the United States and other countries.

Linux is a trademark of Linus Torvalds in the United States, other countries, or both.

LTO, the LTO Logo and the Ultrium logo are trademarks of HP, IBM Corp. and Quantum in the U.S. and other countries.

Windows, and the Windows logo are trademarks of Microsoft Corporation in the United States, other countries, or both.

Java, and all Java-based trademarks and logos are trademarks or registered trademarks of Oracle and/or its affiliates.

UNIX is a registered trademark of The Open Group in the United States and other countries.

Other company, product, or service names may be trademarks or service marks of others.

Find and read thousands of IBM Redbooks publications

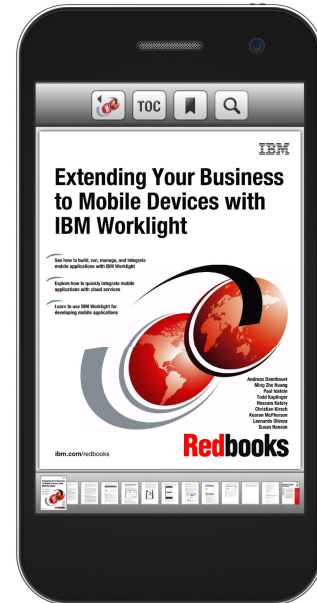
- ▶ Search, bookmark, save and organize favorites
- ▶ Get up-to-the-minute Redbooks news and announcements
- ▶ Link to the latest Redbooks blogs and videos

Get the latest version of the Redbooks Mobile App



Download
Now

iOS



Promote your business in an IBM Redbooks publication

Place a Sponsorship Promotion in an IBM® Redbooks® publication, featuring your business or solution with a link to your web site.

Qualified IBM Business Partners may place a full page promotion in the most popular Redbooks publications. Imagine the power of being seen by users who download millions of Redbooks publications each year!



ibm.com/Redbooks

About Redbooks → Business Partner Programs

THIS PAGE INTENTIONALLY LEFT BLANK

Preface

This IBM® Redbooks® publication focuses on gathering the correct technical information, and laying out simple guidance for optimizing code performance on IBM POWER8® processor-based systems that run the IBM AIX®, IBM i, or Linux operating systems. There is straightforward performance optimization that can be performed with a minimum of effort and without extensive previous experience or in-depth knowledge.

The POWER8 processor contains many new and important performance features, such as support for eight hardware threads in each core and support for transactional memory. The POWER8 processor is a strict superset of the IBM POWER7+™ processor, and so all of the performance features of the POWER7+ processor, such as multiple page sizes, also appear in the POWER8 processor. Much of the technical information and guidance for optimizing performance on POWER8 processors that is presented in this guide also applies to POWER7+ and earlier processors, except where the guide explicitly indicates that a feature is new in the POWER8 processor.

This guide strives to focus on optimizations that tend to be positive across a broad set of IBM POWER® processor chips and systems. Specific guidance is given for the POWER8 processor; however, the general guidance is applicable to the IBM POWER7+, IBM POWER7®, IBM POWER6®, IBM POWER5, and even to earlier processors.

This guide is directed at personnel who are responsible for performing migration and implementation activities on POWER8 processor-based systems. This includes system administrators, system architects, network administrators, information architects, and database administrators (DBAs).

Authors

This book was produced by a team of specialists from around the world working at the International Technical Support Organization, Poughkeepsie Center.



Peter Bergner is the GCC Compiler Team Lead within the Linux on Power Toolchain department. Since joining IBM in 1996, Peter has worked in various areas, including compiler optimizer development for the IBM i platform, as a core member of the teams that ported Linux and GLIBC to 64-bit POWER, and as a team lead for the IBM Blue Gene/L® compiler and runtime library development team. He obtained a PhD in Electrical Engineering from the University of Minnesota.

Brian Hall is the lead analyst for performance improvement efforts with the IBM Cloud Innovation Laboratory team. He works with many IBM software products to capitalize on the IBM Power Architecture® and develop performance preferred practices for software development and deployment. After joining IBM in 1987, Brian originally worked on the IBM XL C/C++/Fortran compilers and on the just-in-time compiler for IBM Java on Power. He has a Bachelor's degree in Computer Science from Queen's University at Kingston and a Master's degree in Computer Science from the University of Toronto.

Alon Shalev Housfater is software engineering professional at the IBM Runtime Technology Center at the IBM Toronto Lab. Alon's role involves deep system performance analysis and the exploitation of computational accelerators by Java's Just-in-Time compiler. He holds a PhD degree in Electrical Engineering from the University of Toronto.



Madhusudanan Kandasamy is an IBM Master Inventor and Technical Chief Engineering Manager (TCEM) for AIX Performance, Scalability, and DSO. He has more than a decade of experience in AIX development. He holds a Master's degree in Software Systems from Birla Institute of Technology, Pilani-India.



Tulio Magno is a Staff Software Engineer at the Linux Technology Center. He holds a Bachelor's degree in Electrical Engineering from Federal University of Minas Gerais and has been working on the Linux Toolchain for the last four years developing core libraries and the Advanced Toolchain.

Alex Mericas is a member of the IBM Systems and Technology Group in Austin, Texas. He is a Senior Technical Staff Member and is the Performance Architect for the POWER8 processor. He designed the performance monitoring unit on POWER4, POWER5, POWER6, POWER7, and IBM PowerPC® 970 processor-based systems. Alex is an IBM Master Inventor with 47 US patent applications and 22 issued patents covering microprocessor design and hardware performance monitors.



Steve Munroe is a Senior Technical Staff Member at the Rochester, Minnesota Lab in IBM US. He has 38 years of experience in the software development field. He holds a Bachelor's degree in Computer Science from Washington State University (1974). His areas of expertise include PowerISA, compilers, POSIX run times, and performance analysis. He has written extensively about IBM POWER performance and Java performance.

Mauricio Oliveira is a Staff Software Engineer at the Linux Technology Center at IBM Brazil. His areas of expertise include Linux performance and Debian and Ubuntu distributions on IBM Power Systems™. He also worked with official benchmark publications for Linux on IBM Power Systems and early development (bootstrap) of Debian on Little Endian 64-bit PowerPC. Mauricio holds a Master of Computer Science and Technology degree and a Bachelor of Engineering degree in Computer Engineering from Federal University of Itajubá, Brazil.



Bill Schmidt is a Senior Software Engineer with IBM in Rochester, Minnesota. He has 22 years of experience with design and implementation of compilers for the Power architecture, specializing in optimization technology. He contributes to the GCC and LLVM open source compilers. Bill is an IBM Master Inventor with over 50 issued US patents, and holds a PhD from Iowa State University.



Will Schmidt is an Advisory Software Engineer in Rochester, Minnesota. Since joining IBM in 1997, he has worked in various areas, most recently including Linux Toolchain and performance tools development. He obtained a BS in Mathematics and a BS in Computer Science from Bemidji State University.



Bernard King Smith is a Senior Software Engineer in the Power Systems Performance Department. He joined IBM in 1989 and has spent over 28 years in network performance of High Performance Computing (HPC) and clustered commercial systems. His primary work has been in TCP/IP and RDMA performance of high-speed networks. He was involved in the tuning and design of internal networks for both the IBM Deep Blue Chess Machine and the Jeopardy! Watson System. He is also the leading performance expert of InfiniBand network performance on Power Systems. He is an author or co-author of two patents and one Internet Engineering Task Force (IETF) standard. He is the team lead for networking performance for IBM Power Systems.



Julian Wang is the technical lead of JIT compiler and Java performance on Power Systems, and has been developing compiler and runtime products for the past 20 years. He has a passion for making the POWER architecture perform Java better and acute interests in parallel computing, operating system, performance analysis, and bit-twiddling.



Suresh Warriar is a Senior Technical Staff Member in IBM Power Systems Software™, specializing in KVM and Linux on Power architecture. Suresh has over 25 years of experience in systems software, including over 15 years leading AIX exploitation of POWER hardware technology. He has a Bachelor's degree in Electrical Engineering from the Indian Institute of Technology, Madras, India, and a Master's Degree in Computer Science from the University of Texas, Austin.



David Wendt is a Senior Staff Member of the IBM Watson™ Performance team in Research Triangle Park, NC. He has a Master's degree in Electrical Engineering from Johns Hopkins University. He is also an IBM Master Inventor with 12 granted US patents in software development.

Thanks to the following people for their contributions to this project:

- For technical reviews:
 - Clark Anderson, IBM Power Systems Storage I/O Subsystem Performance, Rochester, Minnesota
 - Yaoqing Gao, Senior Technical Staff Member, XL C/C++ and Fortran compilers, Ontario, Canada
 - Jenifer Hopper, Software Engineer - Linux Performance Analyst, Austin, Texas
 - Yan Luo, JIT Compiler POWER Optimization, Ontario, Canada
 - Younes Manton, JIT Compiler POWER Optimization, Ontario, Canada
 - Bruce Mealy, AIX Kernel Development, Austin, Texas
 - Greg Mewhinney, Power Systems Performance, Austin, Texas
 - Steve Munroe, Linux Toolchain Architect and TCEM, Rochester, Minnesota
 - David Tam, Ph.D., Staff Software Developer, Ontario, Canada
- For overall contributions to this project:
 - International Technical Support Organization, Poughkeepsie Center
 - Deana Coble, IBM Redbooks Technical Writer, RTP, North Carolina

Thanks to the authors of the previous versions of this book:

Ryan Arnold, Peter Bergner, Wainer dos Santos Moschetta, Robert Enenkel, Pat Haugen, Michael R. Meissner, Alex Mericas, Bernie Schiefer, Suresh Warriar, Daniel Zabawa, Adhemerval Zanella

Brian Hall, Mala Anand, Bill Buros, Miso Cilimdžić, Hong Hua, Judy Liu, John MacMillan, Sudhir Maddali, K Madhusudanan, Bruce Mealey, Steve Munroe, Francis P O'Connell, Sergio Reyes, Raul Silvera, Randy Swanberg, Brian Twichell, Brian F Veale, Julian Wang, Yaakov Yaari

Now you can become a published author, too!

Here's an opportunity to spotlight your skills, grow your career, and become a published author—all at the same time! Join an ITSO residency project and help write a book in your area of expertise, while honing your experience using leading-edge technologies. Your efforts will help to increase product acceptance and customer satisfaction, as you expand your network of technical contacts and relationships. Residencies run from two to six weeks in length, and you can participate either in person or as a remote resident working from your home base.

Find out more about the residency program, browse the residency index, and apply online at:

ibm.com/redbooks/residencies.html

Comments welcome

Your comments are important to us!

We want our books to be as helpful as possible. Send us your comments about this book or other IBM Redbooks publications in one of the following ways:

- Use the online **Contact us** review Redbooks form found at:

ibm.com/redbooks

- Send your comments in an email to:

redbooks@us.ibm.com

- Mail your comments to:

IBM Corporation, International Technical Support Organization
Dept. HYTD Mail Station P099
2455 South Road
Poughkeepsie, NY 12601-5400

Stay connected to IBM Redbooks

- Find us on Facebook:

<http://www.facebook.com/IBMRedbooks>

- Follow us on Twitter:

<http://twitter.com/ibmredbooks>

- ▶ Look for us on LinkedIn:
<http://www.linkedin.com/groups?home=&gid=2130806>
- ▶ Explore new Redbooks publications, residencies, and workshops with the IBM Redbooks weekly newsletter:
<https://www.redbooks.ibm.com/Redbooks.nsf/subscribe?OpenForm>
- ▶ Stay current on recent Redbooks publications with RSS Feeds:
<http://www.redbooks.ibm.com/rss.html>

Summary of changes

This section describes the technical changes that are made in this edition of the book and in previous editions. This edition might also include minor corrections and editorial changes that are not identified.

Summary of Changes for SG24-8171-01 for Performance Optimization and Tuning Techniques for IBM Power Systems Processors Including IBM POWER8 as created or updated on March 31, 2017.

August 2015, Second Edition

The second edition of this guide contains numerous minor updates and extensions across many topics, plus coverage of some new topics. Many of the updates are concentrated in Chapter 6, “Linux” on page 117, and are related to new developments in Linux for the POWER8 processor, such as Ubuntu for Power. Chapter 8, “Java” on page 173 contains updates covering the release of Java 8.

Coverage has been added for the following new topics:

- ▶ SMT modes as a deployment option (see “SMT mode” on page 19)
- ▶ Power management modes as a deployment option (see “Power management mode” on page 21)
- ▶ Coherent Accelerator Processor Interface (CAPI) (see 2.2.14, “Coherent Accelerator Processor Interface” on page 53)
- ▶ I/O adapter affinity performance considerations (see 2.3, “I/O adapter affinity” on page 55)
- ▶ PowerKVM (see 3.3, “Introduction to KVM Virtualization” on page 67)
- ▶ Little Endian (see 6.4, “Little Endian” on page 138)
- ▶ GPU acceleration for C/C++ (7.6, “Using GPU accelerators with C/C++” on page 169)
- ▶ GPU acceleration for Java (8.7, “Using GPU accelerators with IBM Java” on page 190)



Optimization and tuning on IBM POWER8 processor-based systems

This chapter describes the optimization and tuning of IBM POWER8 processor-based systems. It covers the following topics:

- ▶ 1.1, “Introduction” on page 2
- ▶ 1.2, “Outline of this guide” on page 2
- ▶ 1.3, “Conventions that are used in this guide” on page 5
- ▶ 1.4, “Background” on page 5
- ▶ 1.5, “Optimizing performance on POWER8 processor-based systems” on page 6

1.1 Introduction

This guide gathers the correct technical information and lays out simple guidance for optimizing code performance on IBM Power Systems that run the AIX, IBM i, or Linux operating systems.

This guide focuses on optimizations that tend to be positive across a broad set of IBM POWER processor chips and systems. Much of the technical information and guidance for optimizing performance on the POWER8 processor that is presented in this guide also applies to POWER7+ and earlier processors, except where the guide explicitly indicates that a feature is new in the POWER8 processor.

Straightforward performance optimization can be performed with a minimum of effort and without extensive previous experience or in-depth knowledge. This optimization work can accomplish the following goals:

- ▶ Substantially improve the performance of the application that is being optimized for the POWER8 processor (the focus of this guide).
- ▶ Typically, carry over improvements to systems that are based on related processor chips, such as the IBM POWER7+, IBM POWER7, and IBM POWER6 processor chips.
- ▶ Improve performance on other platforms.

The POWER8 processor contains many new and important performance features, such as support for eight hardware threads in each core and support for transactional memory. The POWER8 processor is a strict superset of the POWER7+ processor, and so all of the performance features of the POWER7+ processor, such as multiple page sizes, also appear in the POWER8 processor.

This guide is directed at personnel who are responsible for performing migration and implementation activities on POWER8 processor-based systems, including systems administrators, system architects, network administrators, information architects, program product developers, software architects, database administrators (DBAs), and compiler writers.

1.2 Outline of this guide

This chapter lays out simple strategies for optimizing performance and covers the opportunities that have been found to be the most universally applicable and valuable in past performance efforts (see 1.5, “Optimizing performance on POWER8 processor-based systems” on page 6). This chapter is not an exhaustive guide to Power Systems performance, but it presents a concise overview of typical methodology and areas to focus on in a performance improvement effort. There are references to later chapters in the guide that present a complete technical description of these areas. Later chapters also contain a complete list of opportunities and techniques to optimize performance that might be valuable in particular cases.

Section 1.5.1, “Lightweight tuning and optimization guidelines” on page 7 describes a set of straightforward steps to set up the environment for performance tuning and optimization, followed by an explanation about how to perform a set of straightforward and easy investigative steps. These steps are the most valuable to focus on for a short performance effort. These steps do not require a deep level of knowledge of the application being optimized, and (with one minor exception) do not involve changing application source code.

Section 1.5.2, “Deployment guidelines” on page 15 describes deployment choices, that is, system setup and configuration choices, so you can tune these designed-for-performance IBM Power Systems for your environment. Together with 1.5.1, “Lightweight tuning and optimization guidelines” on page 7, these simple optimization strategies and deployment guidance satisfy the requirements for most environments and can deliver substantial improvements.

Finally, 1.5.3, “Deep performance optimization guidelines” on page 21 describes some of the more advanced investigative techniques that can be used to identify performance bottlenecks in an application. It is here that optimization efforts move into the application code, and improvements are typically made by modifying source code. Coverage in this last area is fairly rudimentary, focusing on general areas of investigation and the tools that you can use.

Most of the remaining material in this guide is technical information that was developed by domain experts at IBM:

- ▶ This guide provides hardware information about the POWER8 processor (see Chapter 2, “The IBM POWER8 processor” on page 25), highlighting the important features from a performance perspective and laying out the basic information that is drawn upon by the material that follows.
- ▶ This guide describes the system software stack, examining the IBM POWER Hypervisor™ (see Chapter 3, “The IBM POWER Hypervisor” on page 57), the AIX, IBM i, and Linux operating systems and system libraries (see Chapter 4, “IBM AIX” on page 71, Chapter 5, “IBM i” on page 111, and Chapter 6, “Linux” on page 117), and the compilers (see Chapter 7, “Compilers and optimization tools for C, C++, and Fortran” on page 141). Java (see Chapter 8, “Java” on page 173) also receives extensive coverage.
- ▶ Chapter 4, “IBM AIX” on page 71 highlights some of the areas in which AIX exposes some new features of the POWER8 processor. Then, this chapter examines a set of operating system-specific optimization opportunities. The chapter concludes with a short description of AIX preferred practices regarding system setup and maintenance.
- ▶ Chapter 5, “IBM i” on page 111 describes IBM i support for a number of features in POWER8 processors (including features that are available in previous generations of POWER processors). The chapter describes how this operating system can be effective in automatically capitalizing on many new POWER architecture features without changes to existing programs. The chapter also provides information about IBM Portable Application Solutions Environment for i (PASE for i), a part of IBM i that allows some AIX application binary files to run on IBM i with little or no changes.
- ▶ Chapter 6, “Linux” on page 117 describes the primary Linux operating systems that are used on POWER8 processor-based systems. The chapter covers using features of the POWER architecture, and operating system-specific optimization opportunities.

Linux is based on community efforts that are focused not only on the Linux kernel, but also all of the complementary packages, tools, toolchains, and GNU Compiler Collection (GCC) compilers that are needed to use effectively POWER8 processor-based systems. IBM provides the expertise for Power Systems by developing, optimizing, and pushing open source changes to the Linux communities.

- ▶ Chapter 7, “Compilers and optimization tools for C, C++, and Fortran” on page 141 describes current compiler versions and optimization levels and how, for projects with increased focus on runtime performance, you can take advantage of the more advanced compiler optimization techniques. It describes XL compiler static analysis and runtime checking to validate the correctness of the program.
- ▶ Chapter 8, “Java” on page 173 describes the optimization and tuning of Java based applications that are running in a POWER environment.

- Finally, this book covers important information about IBM middleware, DB2® (see Chapter 9, “IBM DB2” on page 193) and IBM WebSphere® Application Server (see Chapter 10, “IBM WebSphere Application Server” on page 205). Various applications use middleware, and it is critical that the middleware is tuned correctly and performs well. The middleware chapters cover how these products are optimized for POWER8 processor-based systems, including select preferred practices for tuning and deploying these products.

The following appendixes are included:

- Appendix A, “Analyzing malloc usage under IBM AIX” on page 211 explains some simple techniques for analyzing how an application is using the system memory allocation routines (*malloc* and related functions in the C library). *malloc* is often a bottleneck for application performance, especially under AIX. AIX has an extensive set of optimized *malloc* implementations, and it is easy to switch between them without rebuilding or changing an application. Knowing how an application uses *malloc* is key to choosing the best memory allocation alternatives that AIX offers. Even Java applications often make extensive use of *malloc*, either in Java Native Interface (JNI) code that is part of the application itself or in the Java class libraries, or in binary code that is part of the software development kit (SDK).
- Appendix B, “Performance tools and empirical performance analysis” on page 215 describes some of the important performance tools that are available on the IBM Power Architecture under AIX or Linux, and strategies for using them in empirical performance analysis efforts.

These performance tools are most often used as part of the advanced investigative techniques that are described in 1.5.3, “Deep performance optimization guidelines” on page 21, except for the performance advisors, which are intended as investigative tools that are appropriate for a broader audience of users.

Throughout the book, there are links to related sections among the chapters. For example, Vector Scalar eXtension (VSX) is described in the processor chapter (Chapter 2, “The IBM POWER8 processor” on page 25), all of the OS chapters (Chapter 4, “IBM AIX” on page 71, Chapter 5, “IBM i” on page 111, and Chapter 6, “Linux” on page 117), and in the compiler chapter (Chapter 7, “Compilers and optimization tools for C, C++, and Fortran” on page 141). Therefore, after the description of VSX in the processor chapter, there are links to that same section in the OS chapters and in the compiler chapter.

After you review the advice in this guide, for more information, visit the IBM Power Systems website at:

<http://www.ibm.com/systems/power/index.html>

1.3 Conventions that are used in this guide

In this guide, the conventions for indicating sections of code or command examples are shown in Table 1-1.

Table 1-1 Conventions that are used in this guide

Type of example	Format that is used in this guide	Example of the convention
Commands and command options within text	Monofont, bolded	ldedit
Command lines or code examples outside of text	Monofont	<code>ldedit -btextpsize=64k -bdatapsize=64k -bstacksize=64k</code>
Variables in command lines	Monofont, italicized	<code>ldedit -btextpsize=64k -bdatapsize=64k -bstacksize=64k <executable></code>
Variables that are limited to specific choices	Monofont, italicized	<code>-mcmodel={medium large}</code>

1.4 Background

Continuing trends in processor design are making it more important than ever to consider analyzing and working to improve application performance. In the past, two of the ways in which newer processor chips delivered higher performance were by:

- ▶ Increasing the clock rate
- ▶ Making microarchitectural improvements that increase the performance of a single thread

Often, upgrading to a new processor chip gave existing applications a 50% or possibly 100% performance improvement, leaving little incentive to spend much effort to get an uncertain amount of additional performance. However, the approach in the industry has shifted, so that the newer processor chips do not substantially increase clock rates, as compared to the previous generation. In some cases, clock rates declined in newer designs. Recent designs also generally offer more modest improvements in the performance of a single execution thread.

Instead, the focus has shifted to delivering multiple cores per processor chip, and to delivering more hardware threads in each core (known as simultaneous multi-threading (SMT) in IBM Power Architecture terminology). This situation means that some of the best opportunities for improving the performance of an application are in delivering scalable code by having an application make effective use of multiple concurrent threads of execution.

Coupled with the trend toward aggressive multi-core and multi-threaded designs, there are sometimes changes in the amount of cache and memory bandwidth available to each hardware thread. Cache sizes and chip-level bandwidth are, in some cases, increasing at a slower rate than the growth of hardware threads, meaning that the amount of cache per thread is not growing as rapidly. In particular instances, it decreases from one generation to the next. Again, this situation shows where deeper analysis and performance optimization efforts can provide some benefits.

There is also a recent trend toward adding transactional memory support to processors and toward support for special purpose accelerators. Transactional memory is a feature that simplifies multi-threaded programming by providing safe access mechanisms to shared data. Special purpose accelerators may be based on adding new instructions to the core, on chip-level accelerators, or on fast and efficient access mechanisms to new off-chip accelerators, such as graphics processing units (GPUs) or field-programmable gate arrays (FPGAs).

1.5 Optimizing performance on POWER8 processor-based systems

This section provides guidance for optimizing performance on POWER8 processor-based systems. It covers the more prominent performance opportunities that have been found in past optimization efforts. The guidance is organized in to three broad categories:

1. Lightweight tuning and optimization guidelines

Lightweight tuning covers simple prescriptive steps for tuning application performance on POWER8 processor-based systems. These simple steps can be carried out without detailed knowledge of the internals of the application that is being optimized and usually without modifying the application source code. Simple system utilization and performance tools are used for understanding and improving your application performance. The steps and tools are general guidelines that apply to all types of applications. Although they are simple and straightforward, they often lead to significant performance improvements. It is possible to accomplish these steps in as little as two days or so for a small application. Two weeks might be required to perform these steps for a large and complex application.

Performance improvement: Consider lightweight tuning to be the starting point for any performance improvement effort.

2. Deployment guidelines

Deployment guidelines cover tuning considerations that are related to the:

- Configuration of a POWER8 processor-based system to deliver the best performance
- Associated runtime configuration of the application itself

There are many choices in a deployment, some of which are unrelated to the performance of a particular application. This section presents some guidelines and preferred practices. Understanding logical partitions (LPARs), energy management, I/O configurations, and using multi-threaded cores are examples of typical system considerations that can impact application performance.

Performance improvement: Consider deployment guidelines to be the second required activity for any reasonably extensive performance effort.

3. Deep performance optimization guidelines

Deep performance analysis covers performance tools and general strategies for identifying and fixing application bottlenecks. This type of analysis requires more familiarity with performance tools and analysis techniques, sometimes requiring a deeper understanding of the application internals, and often requiring a more dedicated and lengthy effort. Often, a simpler analysis is all that is required to identify serious bottlenecks in an application; however, a more detailed investigation is required to perform an exhaustive search for all of the opportunities for increasing performance.

Performance improvement: Consider this the last activity that is undertaken, with simpler analysis steps, for a moderately serious performance effort. The more complex iterative analysis is reserved for only the most performance critical applications.

This chapter provides only minimal background on the guidance provided. Detailed material about these topics is incorporated in the chapters that follow and in the appendixes. The following chapters and appendixes also cover many other performance topics that are not addressed here.

Guidance for POWER8 processor-based systems: The guidance that is provided in this book specifically applies to POWER8 processor chips and systems. The guidance that is provided also generally applies to previous generations of POWER processor chips and systems, including POWER7, POWER6, and POWER5 processor-based systems. When the guidance is not applicable to all generations of Power Systems, it is noted.

1.5.1 Lightweight tuning and optimization guidelines

This section covers building and performance testing applications on POWER8 processor-based systems, and gives a brief introduction to the most important simple performance tuning opportunities that are identified for POWER8 processor-based systems. More details about these and other opportunities are presented in the later chapters of this guide.

Performance test beds and workloads

In performance work, when you are tuning and optimizing an application for a particular processor, you must run and measure performance levels on that processor. Although there are some characteristics that are shared among processor chips in the same family, each generation of processor chip has unique performance features and characteristics. Optimizing code for POWER8 processor-based systems requires that you set up a test bed on a POWER8 processor-based system.

Some organizations want to see good performance across a range of newer systems, with a special emphasis on optimizing for the latest design. For Power Systems, the previous POWER7 generation is still commonly used, and it might be necessary to support even older POWER6 and earlier processor-based systems. For this reason, it is best to have multiple test bed environments: a POWER8 processor-based system for most optimization work, and POWER7 and possibly POWER6 processor-based systems for limited testing to ensure that all tuning is beneficial on the previous generations of hardware.

POWER8, POWER7, and POWER6 processors are dissimilar in some respects, and some simple steps can be taken to ensure good performance of a single binary running on any of these systems. In particular, see the information in “C, C++, and Fortran compiler options” on page 10.

Performance test beds must be sized and configured for performance and scalability testing. Choose your scalability goals based on the requirements that are placed on an application, and the test bed must accommodate at least the minimum requirements. For example, when you target a multi-threaded application to scale up to four cores on POWER8 processor-based systems, it is important that the test bed be at least a 4-core system and that tests are configured to run in various configurations (1-core, 2-core, and 4-core). You want to be able to measure performance across the different configurations such that the scalability can be computed. Ideally, a 4-core system delivers four times the performance of a 1-core system, but in practice, the scalability is less than ideal. Scalability bottlenecks might not be clearly visible if the only testing done for this example was in a 4-core configuration.

With the multi-threaded POWER8 cores (see 2.2, “Using POWER8 features” on page 28), each processor core can be instantiated with one, two, four, or eight logical CPUs within the operating system. A 4-core server, with SMT8 mode (eight hardware threads per core), means that the operating system is running 32 logical CPUs. Also, larger-core servers are becoming more pervasive, with scaling considerations well beyond 4-core servers.

The performance test bed should be a dedicated LPAR. You must ensure that there is no other activity on the system (including on other LPARs, if any, configured on the system) when performance tests are run. The initial performance testing should be done in a dedicated resource environment to minimize the factors that affect performance. Ensure that the LPAR is running an up-to-date version of the operating system, at the level that is expected for the typical usage of the application. Keep the test bed in place after any performance effort so that performance can occasionally be monitored, which ensures that later maintenance of an application does not introduce a performance regression.

Choosing the appropriate workloads for performance work is also important. Ideally, a workload has the following characteristics:

- ▶ Be representative of the expected actual usage of the application.
- ▶ Have simple measures of performance that are easily collected and compared, such as run time or transactions/second.
- ▶ Be easy to set up and run in an automated environment, with a fairly short run time for a fast turnaround in performance experiments.
- ▶ Have a low run-to-run variability across duplicated runs, such that extensive tests are not required to obtain a statistically significant measure of performance.
- ▶ Produce a result that is easily tested for correctness.

When an application is being optimized for multiple operating systems, much of the performance work can be undertaken on just one of the operating systems. However, some performance characteristics are operating system-dependent, so some analysis must be performed on each operating system. In particular, perform profiling and lock analysis separately for each operating system to account for differences in system libraries and kernels. Each operating system also has unique scalability considerations.

Build environment and build tools

The build environment, if separate from the performance test bed, must be running an up-to-date operating system. Only recent operating system levels include Application Binary Interface (ABI) extensions to use or control newer hardware features.

Critically, all compilers that are used to build an application must use up-to-date versions that offer full support for the target processor chip. Older levels of a compiler might tolerate newer processor chips, but they do not capitalize on the unique features of the latest processor chips. For the IBM XL compilers on AIX or Linux, XLC13 and XLF15 are the first compiler versions that have processor-specific tuning for POWER8 processor-based systems. For the GCC compiler on Linux, IBM Advance Toolchain Version 7.0 (and later versions) contain an updated GCC compiler that is preferred for POWER7 and POWER8 processor-based systems, and Version 8.0 and later support POWER Little Endian and Big Endian. The IBM XL Fortran Compiler is recommended over gfortran for the most optimized high floating point performance characteristics.

For more information about the Advance Toolchain features and supported environments, see the Introduction and Supported Linux Distributions sections of the Advance Toolchain wiki page at the following website:

https://www.ibm.com/developerworks/community/wikis/home?lang=en#!/wiki/W51a7ffcf4dfd_4b40_9d82_446ebc23c550/page/IBM%20Advance%20Toolchain%20for%20PowerLinux%20Documentation

For the GCC compiler on Linux, the GCC compilers, which come with the distributions, both recognize and take advantage of the POWER architecture and optimizations. For improved optimizations and newer GCC technology, the IBM Advance Toolchain package provides an updated GCC compiler and optimized toolchain libraries for use with POWER8 processor-based systems.

The Advance Toolchain is a key performance technology that is available for Power Systems running Linux. It includes newer, POWER-optimized versions of compilers (GCC, G++, GFortran, and GCCGo (since Version 8.0)), utilities, and libraries, along with various performance tools. The full Advance Toolchain must be installed in the build environment, and the Advance Toolchain runtime package must be installed in the performance test bed. The Toolchain is designed to coexist with the GCC compilers and toolchain that are provided in the standard Linux distributions. More information is available in 6.3.1, “GCC, toolchain, and IBM Advance Toolchain” on page 129.

Along with the compilers for C/C++ and Fortran, there is the separate IBM Feedback Directed Program Restructuring (FDPR®) tool to optimize performance. FDPR takes a post-link executable image (such as one produced by static compilers) and applies additional optimizations. FDPR is another tool that can be considered for optimizing applications that are based on an executable image. More details can be found in 7.4, “IBM Feedback Directed Program Restructuring” on page 160.

Java also contains a dynamic Just-In-Time (JIT) compiler, and only newer versions are tuned for POWER8 processor-based systems. However, Java compilations to binary code take place at application execution time, so a newer Java release must be installed on the performance test bed system.

C, C++, and Fortran compiler options

For the static compilers, the important compilation options to consider are as follows:

- ▶ *Basic optimization options:* The minimum suggested optimization level for the XL compilers and GCC is **-O2**. Higher levels of optimization are better for some types of code, and you might want to experiment with them. The XL compiler option **-O3** or **-qhot -O3** is recommended for numerical floating point compute-intensive applications, and **-O3** or **-O3 -qipa** is recommended for integer applications for better performance. More options are detailed in 7.1, “Compiler versions and optimization levels” on page 142. The more aggressive optimization options might not work for all programs and might need to be coupled with the strict options described in this list (see page 11).
- ▶ *Target processor chip options:* It is possible to build a single executable file that runs on various POWER processors. However, that executable file does not take advantage of some of the features added to later processor chips, such as new instructions. If only a restricted range of newer processor chips must be supported, consider using the compilation options that enable the usage of newer features. With the XL compilers, for example, if the executable file must run only on POWER7 or later processors (including the POWER8 processor), the **-qarch=pwr7** option can be specified. The equivalent GCC option is **-mcpu=power7**. Similarly, if the executable file must run only on POWER8 processors, the XL **-qarch=pwr8** option can be specified. The equivalent GCC option is **-mcpu=power8**.
- ▶ *Target processor chip tuning options:* The XL compiler **-qtune** option specifies that the code produced must be tuned to run optimally on particular processor chips in a specified SMT mode. The executable file that is produced still runs on other processor chips, but might not be tuned for them.

GCC uses the **-mcpu** and **-mtune=** options, which take a chip or platform name string (for example, Power platform names include **power7** or **power8**). For **-mcpu=**, the platform name implies a specific PowerISA version and specific PowerISA categories that are implemented for that chip. The same platform names, when applied to **-mtune=**, imply a specific chip micro-architecture. If the **-mtune=** is not specified, it is implied by the **-mcpu=** option. The **-mtune=** option can specify a different platform name than specified by **-mcpu=**. For example, **-mcpu=power7 -mtune=power8** generates code that runs on POWER7 and later POWER processors (including the POWER8 processor), but is tuned for the POWER8 micro-architecture. For the new Little Endian systems, use **-mcpu=power8 -mtune=power8** unless you know that is the compiler default.

Some other possible chip tuning options to consider are noted in the following list:

- The SMT suboptions for **-qtune** allow specification of a target SMT mode to direct optimizations for best performance in that mode:
 - **-qtune=ST** for optimizations that are tuned for single-threaded execution
 - **-qtune=SMT2** for SMT2 execution mode
 - **-qtune=SMT4** for SMT4 execution mode
 - **-qtune=SMT8** for SMT8 execution mode
- **-qarch=ppc64 -qtune=pwr8** for an executable file that is optimized to run on POWER8 processor-based systems, but that can run on all 64-bit implementations of the Power Architecture (POWER8, POWER7, POWER6, and other processor-based systems)
- **-qarch=pwr7 -qtune=pwr8** for an executable file that can run on POWER7 or POWER8 processor-based systems (with access to Vector Scalar eXtension (VSX) features), but is optimized to run on POWER8 processor-based systems

- **-qarch=pwr5 -qtune=balanced** for an executable file that can run on POWER5 and higher processor-based systems, and is tuned for good performance for all recent Power Systems (including POWER6, POWER7, and POWER8 processor-based systems)
- **-mtune=power7** to tune for the POWER7 processor on GCC and **-mtune=power8** for the POWER8 processor on GCC
- *Strict options:* Sometimes the compilers can produce faster code by subtly altering the semantics of the original source code. An example of this scenario is expression reorganization. Especially for floating point code, the effect of expression reorganization can produce different results. For some applications, these optimizations must be prevented to achieve valid results. For the XL compilers, certain semantic-altering transformations are allowed by default at higher optimization levels, such as **-O3**, but those transformations can be disabled by using the **-qstrict** option (for example, **-O3 -qstrict**). For GCC, the default is strict mode, but you can use **-ffast-math** to enable optimizations that are not concerned with Not a Number (NaN), signed zeros, infinities, floating point expression reorganization, or setting the errno variable. The new **-Ofast** GCC option includes **-O3** and **-ffast-math**, and might include other options in the future.
- *Source code compatibility options:* The XL compilers assume that the C and C++ source code conforms to language rules for aliasing. On occasion, older source code fails when compiled with optimization because the code violates the language rules. A workaround for this situation is to use the **-qalias=noansi** option. The GCC workaround is the **-fno-strict-aliasing** option.
- *Profile Directed Feedback (PDF):* PDF is an advanced optimization feature of the compilers to consider for performance-critical applications.
- *Interprocedural Analysis (IPA):* IPA is an advanced optimization feature of the compilers to consider for performance-critical applications.

A simple way to experiment with the C, C++, and Fortran compilation options is to repeatedly build an application with different option combinations, and then to run it and measure performance to see the effect. If higher optimization levels produce invalid results, try adding one or both of the **-qstrict** and **-qalias** options with the XL compilers, or **-fno-strict-aliasing** with GCC.

Not all source files must be compiled with the same set of options, but *all* files must be compiled at the minimum optimization level. There are cases where optimization was not used on just one or two important source files and that caused an application to suffer from substantially reduced performance.

Java options

Many Java applications are performance-sensitive to the configuration of the Java heap and garbage collection (GC). Experimentation with different heap sizes and GC policies is an important first optimization step. For generational GC, consider using the options that specify the split between nursery space (also known as the *new* or *young* space) and tenured space (also known as the *old* space). Most Java applications have modest requirements for long-lived objects in the tenured space, but frequently allocate new objects with a short life span in the nursery space. For more information, see 8.5, “Java garbage collection tuning” on page 183

If 64-bit Java is used, use the **-Xcompressedrefs** option. In newer Java releases, the compressed references option is the default for a 64-bit Java. For more information, see 8.3.4, “Compressed references” on page 177.

By default, newer releases of Java use 64 KB medium pages for the Java heap, which is the equivalent of explicitly specifying the **-X1p64k** option. Linux defaults to 64 KB pages, but AIX defaults to 4 KB pages. If older releases of Java are used on AIX, use the **-X1p64k** option; otherwise, those releases default to using 4 KB pages. Often, there is some additional performance improvement that is seen in using larger 16 MB large pages by using the **-X1p** option. However, using 16 MB pages normally requires explicit configuration by the administrator of the AIX or Linux operating system to reserve a portion of the memory to be used exclusively for large pages. (For more information, see 8.3.2, “Configuring large pages for Java heap and code cache” on page 176.) As such, the medium pages are a better choice for general use, and the large pages can be considered for performance critical applications.

Many Java applications benefit from turning off the default hardware prefetching on the POWER7 processor, and some applications might benefit from doing so on the POWER8 processor. Some recent Java releases turn off hardware prefetching by default. If you turned off hardware prefetching on the POWER7 processor, revisit that tuning because hardware changes on the POWER8 processor have made prefetching more beneficial across different types of code and applications. For more information, see in “Tuning to capitalize on hardware performance features” on page 14. The new and improved hardware prefetcher on the the POWER8 processor has proven to be beneficial to numerous Java applications.

On Power Systems, the **-Xcodecache** option often delivers a small improvement in performance, especially in a large Java application. This option specifies the size of each code cache that is allocated by the JIT compiler for the binary code that is generated for Java methods. Ideally, all of the compiled Java method binary code fits into a single code cache, eliminating the small penalty that might occur when one Java method calls another method when the binary code for the two methods is in different code caches. To use this option, determine how much code space is being used, and then set the size of the option correctly. The maximum size of each code cache that is allocated is 32 MB, so the largest value that can be used for this option is **-Xcodecache32m**. For more information, see 8.3.5, “JIT code cache” on page 180.

The JIT compiler automatically uses an appropriate optimization level when it compiles Java methods. Recent Java releases automatically fully use all of the new features of the target POWER8 processor of the system on which an application is running.

For more information about Java performance, see Chapter 8, “Java” on page 173.

Optimized libraries

Optimized libraries are important for application performance. This section covers some considerations that are related to standard libraries for AIX or Linux, libraries for Java, or specialized mathematical subroutine libraries that are available for the Power Architecture.

AIX malloc

The AIX operating system offers various memory allocation packages (the standard **malloc()** and related routines in the C library). The default package offers good space efficiency and performance for single-threaded applications, but it is not a good choice for the scalability of multi-threaded applications. Choosing the correct malloc package on AIX is important for performance. Even Java applications can make extensive use of malloc through JNI code or internally in the Java Runtime Environment (JRE).

Fortunately, AIX offers a number of different memory allocation packages that are appropriate for different scenarios. These different packages are chosen by setting environment variables and do not require any code modification or rebuilding of an application.

Choosing the best malloc package requires some understanding of how an application uses the memory allocation routines. Appendix A, “Analyzing malloc usage under IBM AIX” on page 211 shows how to collect easily the required information. Following the data collection, experiment with various alternatives, alone or in combination. Some alternatives that deliver high performance include:

- ▶ **Pool malloc:** The pool front end to the malloc subsystem optimizes the allocation of memory blocks of 512 bytes or less. It is common for applications to allocate many small blocks, and pools are particularly space- and time-efficient for that allocation pattern. Thread-specific pools are used for multi-threaded applications. The pool malloc is a good choice for both single-threaded and multi-threaded applications.
- ▶ **Multiheap malloc:** The multiheap malloc package uses up to 32 separate heaps, reducing contention when multiple threads attempt to allocate memory. It is a good choice for multi-threaded applications.

Using the pool front end and multiheap malloc in combination is a good alternative for multi-threaded applications. Small memory block allocations, typically the most common, are handled with high efficiency by the pool front end. Larger allocations are handled with good scalability by the multiheap malloc. A simple example of specifying the pool and multiheap combination is by using the following environment variable setting:

```
MALLOCOPTIONS=pool,multiheap
```

For more information about malloc alternatives, see 4.3.1, “Malloc” on page 95.

IBM Advance Toolchain libraries for Linux

The IBM Advance Toolchain contains replacements for various standard system libraries. These replacement libraries are optimized for specific processor chips, including POWER7 and POWER8 processors. After you install the IBM Advance Toolchain and relink your applications with it, the dynamic linker automatically has programs use the library that is optimized for the processor chip type in the system.

The libraries in IBM Advance Toolchain V7.0 and later are optimized to use the multi-core facilities in POWER7 and POWER8 processors.

Mathematical Acceleration Subsystem Library and Engineering and Scientific Subroutine Library

The Mathematical Acceleration Subsystem (MASS) libraries contain accelerated scalar, Single Instruction Multiple Data (SIMD), and vector versions of a collection of elementary mathematical functions (such as exp, log, and sin) that run on AIX and Linux. The MASS libraries are included with the XL compilers and are automatically used by the compilers when the **-O3 -qhot** compilation options are used. The MASS routines can be used automatically with the Advance Toolchain GCC by using the **-mvecLibabi=mass** option, but MASS is not included with GCC and must be separately installed. Explore the use of MASS for applications that use elementary mathematical functions. Substantial performance improvements can occur when you use the vector versions of the functions. The MASS routines do not necessarily provide the same accuracy of results or the same edge-case behavior as standard libraries do.

The Engineering and Scientific Subroutine Library (ESSL) contains an extensive set of advanced mathematical functions and runs on AIX and Linux. Avoid having applications write their own versions of functions, such as the Basic Linear Algebra Subprograms (BLAS). Instead, use the Power optimized versions in ESSL.

java/util/concurrent

For Java, all of the standard class libraries are included with the JRE. One package of interest for scalability optimization is `java/util/concurrent`. Some classes in `java/util/concurrent` are more scalable replacements for older classes, such as `java/util/concurrent/ConcurrentHashMap`, which can be used as a replacement for `java/util/Hashtable`. `ConcurrentHashMap` might be slightly less efficient than `Hashtable` when run in smaller system configurations where scalability is not an issue, so there can be trade-offs. Also, switching packages requires a source code change, albeit a simple one.

Tuning to capitalize on hardware performance features

For almost all applications, using 64 KB pages is beneficial for performance. Newer Linux releases (RHEL5, SLES11, and RHEL6) default to 64 KB pages, and AIX defaults to 4 KB pages. Applications on AIX have 64 KB pages that are enabled through one or a combination of the following methods:

- Using an environment variable setting:

```
LDR_CNTRL=TEXTPSIZE=64K@DATAPSIZE=64K@STACKPSIZE=64K@SHMPSIZE=64K
```

- Modifying the executable file with the following command:

```
ldedit -btextpsize=64k -bdatapsize=64k -bstacksize=64k <executable>
```

- Using linker options at build time:

```
cc -btextpsize:64k -bdatapsize:64k -bstacksize:64k ...  
ld -btextpsize:64k -bdatapsize:64k -bstacksize:64k ...
```

All of these mechanisms for enabling 64 KB pages can be safely used when the application must run on older hardware or operating system levels that do not support 64 KB pages. When the needed support is not in place, the system simply defaults to using 4 KB pages.

As mentioned in “Java options” on page 11, the newer Java releases default to using 64 KB pages. For Java, it is important that the Java heap space uses 64 KB pages, which are enabled by the **-X1p64k** option in older releases of Java.

Larger 16 MB pages are also supported on the Power Architecture and might provide an additional performance boost when compared to 64 KB pages. However, the usage of 16 MB pages requires explicit configuration by the administrator of the AIX or Linux operating system.

For certain types of non-numerical applications, turning off the default hardware prefetching improves performance. In specific cases, disabling hardware prefetching is beneficial for Java programs, WebSphere Application Server, and DB2. One way to control hardware prefetching is at the partition level, where prefetching is turned off by running the following commands:

- AIX: **dscrctl -n -s 1**
- Linux: **ppc64_cpu --dscr=1**

Controlling prefetching in this way might not be appropriate if different applications are running in a partition because some applications might run best with prefetching enabled. There are also mechanisms to control prefetching at the process level.

Since Java 7 SR3 (and Java 6 26 SR4), the JVM defaults to disable hardware prefetch on AIX. Option **-XXsetHWPrefetch:os-default** can be used to revert to the AIX default hardware prefetch setting.

Recent POWER processors allow not only prefetching to be enabled or disabled, but they also allow the fine-tuning of the prefetch engine. Such fine-tuning is especially beneficial for scientific and engineering and memory-intensive applications.¹ Because the effect of hardware prefetching is heavily dependent on the way that an application accesses memory, and also dependent on the cache sizes of a particular POWER chip, it is always best to test explicitly the effects of different prefetch settings on each chip on which the application is expected to run.

For more information about hardware prefetching and hardware and operating system tuning and usage for optimum performance, see Chapter 2, “The IBM POWER8 processor” on page 25, Chapter 4, “IBM AIX” on page 71, Chapter 6, “Linux” on page 117, and Chapter 5, “IBM i” on page 111.

1.5.2 Deployment guidelines

This section describes deployment guidelines, which relate to how you configure a Power Systems system or an application to achieve optimal performance.

Virtualized versus non-virtualized environments

Virtualization is a powerful technique that is applicable to situations where many applications are consolidated onto a single physical server. This consolidation leads to better usage of hardware and simplified system administration. Virtualization is efficient on the Power Architecture, but it does come with some costs. For example, the Virtual I/O Server (VIOS) partition in the IBM PowerVM® Hypervisor that is allocated for a virtualized deployment consumes a portion of the hardware resources to support the virtualization. For situations where few business-critical applications must be supported on a server, it might be more appropriate to deploy with non-virtualized resources. This situation is particularly true in cases where the applications have considerable network requirements.

Virtualized environments that are provided by the PowerVM Hypervisor offer many choices for deployment, such as dedicated or non-dedicated processor cores and memory, IBM Micro-Partitioning® that uses fractions of a physical processor core, and memory compression. These alternatives are explored in Chapter 3, “The IBM POWER Hypervisor” on page 57. When you set up a virtualized deployment, it is important that system administrators have a complete understanding of the trade-offs inherent in the different choices and the performance implications of those choices. Some deployment choices, such as enabling memory compression features, can disable other performance features, such as support for 64 KB memory pages.

POWER8 processor-based systems allow a second form of virtualization through the PowerKVM hypervisor, which is based on Linux Kernel-based Virtual Machine (KVM) technology. The traditional virtualization environment that is provided by PowerKVM is built upon the quick emulator (QEMU).

POWER8 processor-based systems also support bare metal Linux, which is a non-virtualized environment on the entire system, and Docker, which is a form of light-weight virtualization (see <https://www.docker.com>). Chapter 3, “The IBM POWER Hypervisor” on page 57 provides more information about these different forms of virtualized and non-virtualized environments.

¹ *Making data prefetch smarter: adaptive prefetching on POWER7*, found at: <http://dl.acm.org/citation.cfm?id=2370837> (available for purchase or with access to the ACM Digital Library)

The POWER8 processor and affinity performance effects

The POWER8 processor chip is available in configurations with up to 12 cores per chip, as compared to the POWER7 processor, which has up to eight cores per chip. Along with the increased number of cores, the POWER8 processor chip implements SMT8 mode, supporting eight hardware threads per core, as compared to the POWER7 processor, which supported only four hardware threads per core. Each POWER8 processor core supports running in single-threaded mode with one hardware thread, an SMT2 mode with two hardware threads, an SMT4 mode with four hardware threads, or an SMT8 mode with eight hardware threads.

Each SMT hardware thread is represented as a logical processor in AIX, IBM i, or Linux. When the hardware runs in SMT8 mode, the operating system has eight logical processors for each dedicated POWER8 processor core that is assigned to the partition. To gain the full benefit from the throughput improvement of SMT, applications must use all of the SMT threads of the processor cores.

Each POWER8 chip has memory controllers that allow direct access to a portion of the dual inline memory modules (DIMMs) in the system. Any processor core on any chip in the system can access the memory of the entire system, but it takes longer for an application thread to access the memory that is attached to a remote chip than to access data in the local memory DIMMs.

For more information about the POWER8 hardware, see Chapter 2, “The IBM POWER8 processor” on page 25. This short description provides some background to help understand two important performance issues that are known as *affinity effects*.

Cache affinity

The hardware threads for each core of a POWER8 processor share a core-specific cache space. For multi-threaded applications where different threads are accessing the same data, it can be advantageous to arrange for those threads to run on the same core. By doing so, the shared data remains resident in the core-specific cache space, as opposed to moving between different private cache spaces in the system. This enhanced *cache affinity* can provide more efficient utilization of the cache space in the system and reduce the latency of data references.

Similarly, the multiple cores on a POWER8 processor share a chip-specific cache space. Again, arranging the software threads that are sharing the data to run on the same POWER8 processor (when the partition spans multiple chips) often allows more efficient utilization of cache space and reduced data reference latencies.

Memory affinity

By default, the POWER Hypervisor attempts to satisfy the memory requirements of a partition by using the local memory DIMMs for the processor cores that are allocated to the partition. For larger partitions, however, the partition might contain a mixture of local and remote memory. For an application that is running on a particular core or chip, the application runs best when using only local memory. This enhanced *memory affinity* reduces the latency of memory accesses.

Partition sizes and affinity

In terms of partition sizes and affinity, this section describes POWER dedicated LPARs, shared resource environments, and memory requirements.

Power dedicated LPARs

Dedicated LPAR deployments generally use larger partitions, ranging from just one POWER8 core up to a partition that includes all of the cores and memory in a large symmetric multi-processor (SMP) system. A smaller partition might run a single application and a larger partition typically runs multiple applications, or multiple instances of a single application. A common example of multiple instances of a single application is in deployments of WebSphere Application Server.

With larger partitions, one of the most important performance considerations is often which cores and memory are allocated to a partition. For partitions of up to the number of cores on the chips that are used in the system, the POWER Hypervisor attempts to allocate all cores for the partition from a single POWER8 chip and attempts to allocate only memory local to the chip that is used. Those partitions generally and automatically have good cache and memory affinity. However, it might not be possible to obtain resources for each of the LPARs from a single chip.

For example, assume that you have a 32-core system with four chips, each with eight cores. If five partitions are configured, each with six cores, the fifth LPAR spreads across three chips. Start the most important partition first to obtain resources from a single chip. (The order of starting partitions is one consideration in obtaining the best performance for high priority workloads). This topic is described further in 3.2.3, “Placing LPAR resources to attain higher memory affinity” on page 63.

Another example is when the partition sizes are mixed. Here, starting smaller partitions might consume resources that are spread across many chips, resulting in larger partitions that are spread across multiple chips, which might be contained on a chip if the larger partitions are started first. It is a preferred practice to start higher priority partitions first, so that there is a better opportunity for them to obtain good affinity characteristics in their core and memory allocations. The affinity of the cores and memory that is allocated to a partition can be determined by running the AIX `lsrad -va` command or the Linux `numactl --hardware` command. For more information about partition resource allocation and the `lsrad` command, see Chapter 3, “The IBM POWER Hypervisor” on page 57.

For partitions larger than the number of cores on a chip, the partition always spans more than one chip and has a mixture of local and remote memory. For these larger partitions, it is often useful to force manually good affinity for an application. Manual affinity can be forced by binding applications so that they can run only on particular cores, and by specifying to the operating system that only local memory should be used by the application.

Consider an example where you run four instances of WebSphere Application Server on a partition of 16 cores on a POWER8 processor-based system that is running in SMT8 mode. Each instance of WebSphere Application Server is bound to run on four of the cores of the system. Because each of the cores has eight SMT threads, each instance of WebSphere Application Server is bound to 32 logical processors. Good memory and cache affinity on AIX can therefore be ensured by completing the following steps:

1. Set the AIX `MEMORY_AFFINITY` environment variable, typically to the value `MCM`. This setting tells the AIX operating system to use local memory when an application thread requires physical memory to be allocated.
2. Start the four instances of WebSphere Application Server by running the following `execrset` commands, which bind the execution to the specified set of logical processors:
 - `execrset -c 0-31 -m 0 -e <command to start first WebSphere Application Server instance>`
 - `execrset -c 32-63 -m 0 -e <command to start second WebSphere Application Server instance>`

- **execrset -c 64-95 -m 0 -e** *<command to start third WebSphere Application Server instance>*
- **execrset -c 96-127 -m 0 -e** *<command to start fourth WebSphere Application Server instance>*

Here are some important items to understand in this example:

- ▶ For a particular number of instances and available cores, the most important consideration is that each instance of an application runs only on the cores of one processor chip.
- ▶ Memory and logical processor binding is not done independently because doing so can negatively affect performance.
- ▶ The workload must be evenly distributed over WebSphere Application Server processes for the binding to be effective.
- ▶ There is an assumed mapping of AIX logical CPUs to cores and chips that is implicitly being used in this example, and that is always established at boot time. This mapping can be altered if the SMT mode of the system is changed by running **smtctl -w now**. If dynamic changes have been made to a partition, ensure that you understand the resulting mapping of logical CPUs to cores and chips before using any binding commands.

For more information about the **MEMORY_AFFINITY** environment variable, the **execrset** command, and related environment variables and commands, see Chapter 4, “IBM AIX” on page 71.

The same forced affinity can be established on Linux by running **taskset** or **numactl**. For example:

- ▶ **numactl -C 0-31 -l** *<command to start first WebSphere Application Server instance>*
- ▶ **numactl -C 32-63 -l** *<command to start second WebSphere Application Server instance>*
- ▶ **numactl -C 64-95 -l** *<command to start third WebSphere Application Server instance>*
- ▶ **numactl -C 96-127 -l** *<command to start fourth WebSphere Application Server instance>*

The **-l** option on these **numactl** commands is the equivalent of the AIX **MEMORY_AFFINITY=MCM** environment variable setting.

Even for partitions that are contained on a single chip, better cache affinity can be established with multiple application instances by using logical processor binding commands. With partitions contained on a single chip, the performance effects typically range up to about 10% improvement with binding. For partitions that span more than one POWER8 processor chip, using manual affinity results in a substantially bigger performance effect. For more information about this topic, see Chapter 3, “The IBM POWER Hypervisor” on page 57.

Shared resource environments

Virtualized deployments that share cores among a set of partitions also can use logical processor binding to ensure good affinity within the guest operating system. However, the real dispatching of physical cores is handled by the underlying host operating system (POWER Hypervisor).

The PowerVM Hypervisor uses a three-level affinity mechanism in its scheduler to enforce affinity as much as possible. The reason why absolute affinity is not always possible is that partitions can expand and use unused cycles of other LPARs. This process is done by using uncapped mode in Power, where the uncapped cycles might not always have affinity. Therefore, binding logical processors that are seen at the operating system level to physical threads seen at the hypervisor level works only in some cases in shared partitions. Achieving a high level of affinity is difficult when multiple partitions share resources from a single pool, especially at high utilization, and when partitions are expanding to use other partition cycles. Therefore, creating large shared processor core pools that span across chips tends to create remote memory accesses. For this reason, it might be less desirable to use larger partitions and large processor core pools where high-level affinity performance is expected.

Virtualized deployments can use Micro-Partitioning, where a partition is allocated a fraction of a core. Micro-Partitioning allow a core allocation as small as 0.1 cores in older firmware levels, and as small as 0.05 cores in more recent firmware levels, when coupled with supporting operating system levels. This powerful mechanism provides great flexibility in deployments. However, small core allocations can be more appropriate for situations in which many virtual machines are often idle. Therefore, active 0.05 core LPARs can use those idle cycles.

Also, there is one negative performance effect in deployments with considerably small partitions, in particular with 0.1 or fewer cores at high system utilization: Java warm-up times can be greatly increased. In a Java execution, the JIT compiler is producing binary code for Java methods dynamically. Steady-state optimal performance is reached after a portion of the Java methods are compiled to binary code. With considerably small partitions, there might be a long warm-up period before reaching steady-state performance, where a 0.05 core LPAR cannot get additional cycles from other LPARs because the other LPARs are consuming their cycles. Also, if the workload that is running on this small-size LPAR does not need more than 5% of a processor core capacity, then the performance impact is mitigated.

For more information about this topic, see Chapter 3, “The IBM POWER Hypervisor” on page 57.

Memory requirements

For good performance, there needs to be enough physical memory available so that application data does not need to be frequently paged in and out between memory and disk. The physical memory that is allocated to a partition must be enough to satisfy the requirements of the operating system and the applications that are running on the partition.

Java is sensitive to having enough physical memory available to contain the Java heap because Java applications often have frequent GC cycles where large portions of the Java heap are accessed. If portions of the Java heap are paged out to disk by the operating system because of a lack of physical memory, then GC cycles can cause a large amount of disk activity, which is known as *thrashing*.

SMT mode

The POWER8 processor is designed to support as many as eight SMT threads in each core. This is up to eight separate concurrent threads of instruction execution that share the hardware resources of the core. At the operating system level, this is seen as up to eight logical CPUs per core in the partition. The operating system therefore can schedule up to eight software threads to run concurrently on the core.

Different operating systems can choose to run by default at different SMT modes. AIX defaults to SMT4 when running on a POWER8 processor, and Linux defaults to SMT8 (assuming newer operating system levels that are POWER8 aware). As a deployment choice, configuring the system to run in a particular SMT mode is easily done by the system administrator by using the `smtctl` command on AIX or the `ppc64_cpu --smt` command on Linux.

Note: The SMT mode that the operating system is running in specifies a maximum SMT level, and not a fixed level. The AIX and Linux operating systems dynamically alter the SMT level up to the maximum permitted. During periods where there are few software threads available to run, the operating system can dynamically reduce the SMT mode. During periods where there are many software threads available to run, the operating system dynamically switches to the maximum SMT mode that the system administrator has configured.

SMT is sometimes a trade-off between the best performance a single thread of execution can achieve versus the best total throughput the partition can achieve. To understand this better, consider the following cases:

- ▶ A particular software thread is consuming only a modest fraction of the hardware resources of the core. This often occurs for threads in a Java application, for example. In Java, there is typically a higher frequency of loads, stores and branches, and a lower level of *instruction-level parallelism*, in the binary code. In a case such as this, SMT effectively supports many software threads running simultaneously on the same core and achieves a high level of total throughput without sacrificing the performance of the individual threads.
- ▶ A particular software thread can consume a large fraction of the hardware resources of the core. This sometimes occurs in numerically intensive C code, for example. At high SMT modes with many active threads, this particular software thread is competing with other threads for core resources, and can be starved for resources by the other threads. In a case such as this, the highest single thread performance of this particular software thread is achieved at lower SMT modes. Conversely, the highest throughput is still achieved with a high SMT mode, at the expense of reducing the performance of some individual threads.

With the larger number of cores per chip in POWER8 processor-based systems as compared to previous generations and the higher number of SMT threads per core on the POWER8 processor, one natural tendency is to create partitions with more logical CPUs than in the past. One effect that has been repeatedly seen with more logical CPUs is for an application to start suffering from scalability bottlenecks. All applications typically have a limit to their scalability, and more logical CPUs can cause or exacerbate a scalability issue.

Counterintuitively, application performance goes down when a scalability bottleneck appears. When this happens, users sometime experiment with different SMT modes and come to the conclusion that the application naturally prefers a lower SMT mode, and adjust the operating system configuration. This is often the incorrect conclusion. As explained previously, a small minority of applications do perform better at SMT2, but most applications run well at SMT4 or SMT8 if there are no scalability bottlenecks present.

For cases where an application is seeing reduced performance with higher SMT modes because of scalability effects, possible remediations include:

- ▶ Bind the application to run on a subset of the available logical CPUs. See the example under “Power dedicated LPARs” on page 17.
- ▶ Reduce the number of cores in the partition, which frees hardware resources that can be used to create other partitions.

- ▶ Perform the analysis steps that are outlined in 1.5.3, “Deep performance optimization guidelines” on page 21 for identifying scalability bottlenecks and fixing the application so that it scales better.
- ▶ As a temporary measure, lower the SMT mode of the partition. This should be considered a temporary measure only because by artificially lowering the SMT mode to address a scalability issue, you are effectively wasting some of the available hardware resources of the system.

For the cases where an application intrinsically runs better in a lower SMT mode, and where there is a desire to achieve the best possible single thread performance at the expense of overall throughput, possible remediations include:

- ▶ Segregate the system into partitions running in lower SMT modes and other partitions running in higher SMT modes. Run the applications that prefer lower SMT modes in the partitions with lower SMT levels.
- ▶ Use the hybrid thread and core feature to have some cores in a partition that is run in lower SMT modes, while other cores run in high SMT modes. Bind the applications that prefer lower SMT modes to the cores running in lower SMT modes and bind other applications to the other cores.

For more information see 2.2.1, “Multi-core and multi-thread” on page 28, 4.2.1, “Multi-core and multi-thread” on page 72, 5.2.1, “Multi-core and multi-thread” on page 112, and 6.2.1, “Multi-core and multi-thread” on page 119, all of which address multi-core and multi-thread from the processor and operating system standpoints.

Power management mode

As described in 2.2.13, “Power management and system performance” on page 52, there are different power management modes available that can dramatically affect system performance. Some modes are designed to reduce electrical power consumption at the expense of system performance. Other modes, such as Dynamic Power Saver - Favor Performance, run the system at the highest clock rate and deliver the highest performance. As a deployment choice, a system administrator should configure an appropriate power management mode.

1.5.3 Deep performance optimization guidelines

Performance tools for AIX and Linux are described in Appendix B, “Performance tools and empirical performance analysis” on page 215. A deep performance optimization effort typically uses those tools and follows this general strategy:

- ▶ Gather general information about the running of an application when it is running on a dedicated POWER8 performance system. Important statistics to consider are:
 - The user and system CPU usage of the application: Ideally, a multi-threaded application generates a high overall CPU usage with most of the CPU time in user code. Too high a system CPU usage is generally a sign of a locking bottleneck in the application. Too low an overall usage usually indicates some type of resource bottleneck, such as network or disk. For low CPU usage, look at the number of runnable threads reported by the operating system, and try to ensure that there are as many runnable threads as there are logical processors in the partition.
 - The network utilization of the application: Networks can be a bottleneck in execution either because of bandwidth or latency issues. Link aggregation techniques are often used to solve networking issues.
 - The disk utilization of the application: High disk I/O issues are increasingly being solved by using solid-state devices (SSDs).

Common operating system tools for gathering this general information include **topas** and **perfmpr** (AIX), **top** and **LPCPU** (Linux), **vmstat**, **iostat**, and **netstat**. Detailed CPU usage information is available by running **sar**. This command diagnoses cases where some logical processors are saturated and others are underutilized, an issue that is seen with network interrupt processing on Linux.

- ▶ Collect a time-based profile of the application to see where run time is concentrated. Some possible areas of concern are:
 - Particular user routines or Java methods with a high concentration of execution time. This situation is an indication of a poor coding practice or an inefficient algorithm that is being used in the application itself.
 - Particular library routines or Java class library methods with a high concentration of execution time. First, determine whether the hot routine or method is legitimately used to that extent. Look for alternatives or more efficient versions, such as using the optimized libraries in the IBM Advance Toolchain or the vector routines in the MASS library (for more information, see “Mathematical Acceleration Subsystem Library and Engineering and Scientific Subroutine Library” on page 13).
 - A concentration of run time in the pthreads library (see “Java profiling example” on page 242) or in kernel locking routines. This situation is associated with a locking issue. This locking might ultimately arise at the system level (as seen with malloc locking issues on AIX), or at the application level in Java code (associated with synchronized blocks or methods in Java code). The source of locking issues is not always immediately apparent from a profile. For example, with AIX malloc locking issues, the time that is spent in the malloc and free routines might be low, with almost all of the impact appearing in kernel locking routines.

The tools for gathering profiles are **tprof** (AIX), **OProfile** (Linux), and **perf** (Linux) (these tools are described in “IBM Rational Performance Advisor” on page 221). The **curt** tool (see “AIX trace-based analysis tools” on page 226) also provides a breakdown, describing where CPU time is consumed and includes more useful information, such as a system call summary.

- ▶ Where there are indications of a locking issue, collect locking information.

With locking problems, the primary concern is to determine where the locking originates in the application source code. Cases such as AIX malloc locking can be easily solved just by switching to a more scalable memory allocation package through the **MALLOCTYPE** and **MALLOCOPTIONS** environment variables. In this case, examine how malloc is used and consider making changes at the source code level. For example, rather than repeatedly allocating many small blocks of memory by calling malloc for each block, the application can allocate an array of blocks and then internally manage the space.

As mentioned in “java/util/concurrent” on page 14, Java locking issues that are associated with some older classes, such as java/util/Hashtable, can be easily solved by using java/util/concurrent/ConcurrentHashMap.

For Java programs, use *Java Lock Monitor* (see “Java Health Center” on page 241). For non-Java programs, use the **splat** tool on AIX (see “AIX trace-based analysis tools” on page 226).

- For Java, the WAIT tool is a powerful, easy-to-use analysis tool that is based on collecting thread state information.

Using the WAIT tool requires installing and running only a data collection shell. The shell collects various information about the Java program execution, the most important of which is a set of javacore files. The javacore files show the state of all of the threads at the time the file was dumped. The collected data is submitted to an online tool by using a web browser, and the tool analyzes the data and displays the results with a GUI. The GUI presents information about thread states and has powerful features to drill down to see call chains.

The WAIT tool results combine many of the features of a time-based profile, a lock monitor, and other tools. For Java programs, the WAIT tool might be one of the first analysis tools to consider because of its versatility and ease of use.

For more information about IBM Whole-system Analysis of Idle Time, which is the browser-based (that is, no-install) WAIT tool, go to:

<http://wait.researchlabs.ibm.com>



The IBM POWER8 processor

This chapter introduces the POWER8 processor and describes some of the technical details and features of this product. It covers the following topics:

- ▶ 2.1, “Introduction to the POWER8 processor” on page 26
- ▶ 2.2, “Using POWER8 features” on page 28
- ▶ 2.3, “I/O adapter affinity” on page 55
- ▶ 2.4, “Related publications” on page 55

2.1 Introduction to the POWER8 processor

The POWER8 processor is manufactured by using the IBM 22 nm Silicon-On-Insulator (SOI) technology. Each chip is 567 mm² and contains 1.2 billion transistors. As shown in Figure 2-1, the chip contains the following items:

- ▶ Twelve cores, each with its own 512 KB L2 and 8 MB L3 (embedded DRAM) cache
- ▶ Two memory controllers, PCIe Gen3 I/O controllers
- ▶ An interconnection system that connects all components within the chip

The interconnect also extends through module and board technology to other POWER8 processors in addition to DDR3 memory and various I/O devices.

POWER8 processor-based systems use memory buffer chips to interface between the POWER8 processor and DDR3 or DDR4 memory. Each buffer chip also includes an L4 cache to reduce the latency of local memory accesses. The number of memory controllers, memory buffer chips, PCIe lanes, and cores that are available for use depend upon the particular POWER8 processor-based system.

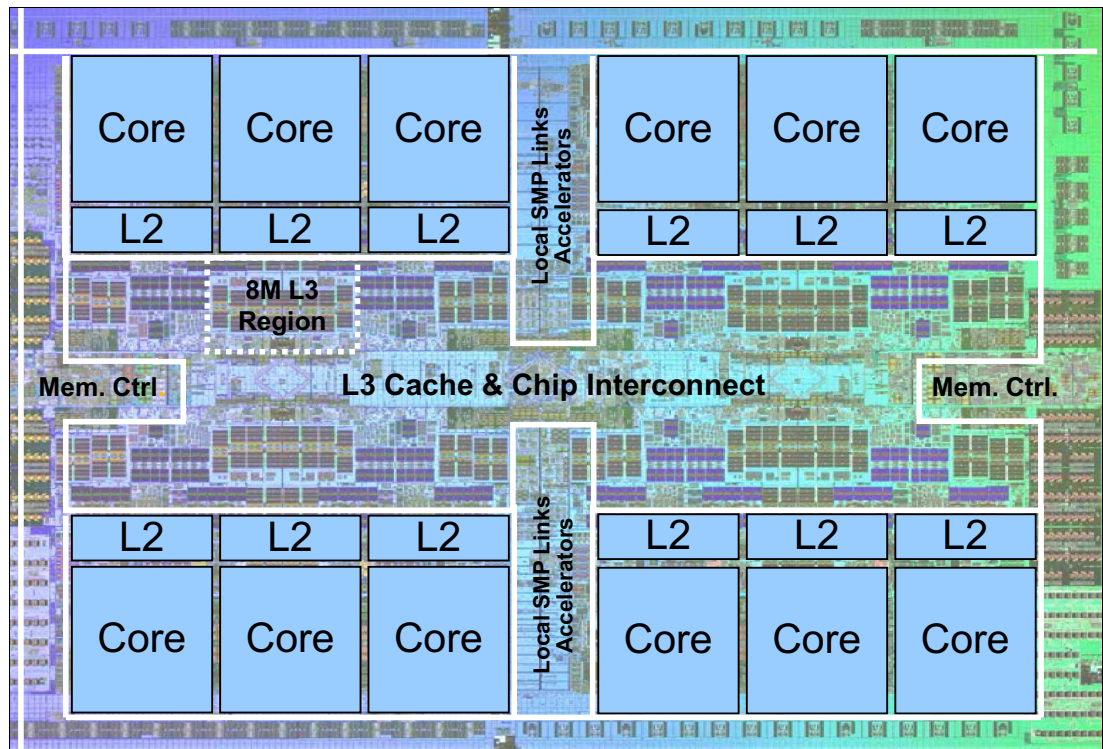


Figure 2-1 The POWER8 processor chip

Each core is a 64-bit implementation of the IBM Power Instruction Set Architecture (ISA) Version 2.07¹ and has the following features:

- ▶ Multi-threaded design, capable of up to eight-way simultaneous multithreading (SMT)
- ▶ 32 KB, eight-way set-associative L1 i-cache
- ▶ 64 KB, eight-way set-associative L1 d-cache

¹ Power ISA Version 2.07, found at <https://www.power.org/documentation/power-isa-v-2-07b/>

- ▶ 72-entry Effective to Real Address Translation (ERAT) for effective to real address translation for instructions (fully associative)
- ▶ 48-entry primary ERAT (fully associative) and 144-entry secondary ERAT for effective to real address translation for data
- ▶ Aggressive branch prediction, using both local and global prediction tables with a selector table to choose the best predictor
- ▶ 16-entry link stack
- ▶ 256-entry count cache
- ▶ Aggressive out-of-order execution
- ▶ Two symmetric fixed-point execution units
- ▶ Two symmetric load/store units and two load units, all four of which can also run simple fixed-point instructions
- ▶ An integrated, multi-pipeline vector-scalar floating point unit for running both scalar and SIMD-type instructions, including the Vector Multimedia eXtension (VMX) instruction set and the new Vector Scalar eXtension (VSX) instruction set, and capable of up to sixteen floating point operations (flops) per cycle (eight double precision or sixteen single precision)
- ▶ In-core Advanced Encryption Standard (AES) encryption capability
- ▶ Hardware data prefetching with 16 independent data streams and software control
- ▶ Hardware decimal floating point (DFP) capability

The POWER8 processor is designed for system offerings from single-socket blades to multi-socket Enterprise servers. It incorporates a triple-scope broadcast coherence protocol over local and global SMP links to provide superior scaling attributes. Multiple-scope coherence protocols reduce the amount of SMP link bandwidth that is required by attempting operations on a limited scope (single chip or multi-chip group) when possible. If the operation cannot complete coherently, the operation is reissued by using a larger scope to complete the operation.

Here are additional features that can augment performance of the POWER8 processor:

- ▶ Adaptive power management.
- ▶ Support for DDR3 and DDR4 memory through memory buffer chips that offload the memory support from the POWER8 memory controller.
- ▶ 16 MB L4 cache within the memory buffer chip that reduces the memory latency for local access to memory behind the buffer chip. The operation of the L4 cache is transparent to applications running on the POWER8 processor.
- ▶ On-chip accelerators, including on-chip encryption, compression, and random number generation accelerators.

For more information about this topic, see 2.3, “I/O adapter affinity” on page 55.

2.2 Using POWER8 features

This section describes several features of the POWER8 processor that can affect performance, including page sizes, cache sharing, SMT priorities, and others.

2.2.1 Multi-core and multi-thread

This section describes the advanced multi-core and multi-thread capabilities of the POWER8 processor. The effective use of the cores and threads is a critically important element of capitalizing on the performance potential of the processor.

Multi-core and multi-thread scalability

POWER8 processor-based system advancements in multi-core and multi-thread scaling are significant. A significant POWER8 processor performance opportunity comes from parallelizing workloads to enable the full potential of the Power platform. Application scaling is influenced by both multi-core and multi-thread technology in POWER8 processors. A single POWER8 chip can contain up to twelve cores. With SMT, each POWER8 core can present eight hardware threads. SMT is the ability of a single physical processor core to dispatch simultaneously instructions from more than one hardware thread context. Because there are multiple hardware threads per physical processor core, additional instructions can run at the same time. SMT is primarily beneficial in commercial environments where the speed of an individual transaction is not as important as the total number of transactions performed. SMT is expected to increase the throughput of workloads with large or frequently changing working sets, such as database servers and web servers.

Additional details about the SMT feature are described in Table 2-1.

Table 2-1 Multi-thread per core features by POWER generation

Technology	Cores/system	Maximum SMT mode	Maximum hardware threads per LPAR
IBM POWER4 processor	32	ST	32
IBM POWER5 processor	64	SMT2	128
IBM POWER6 processor	64	SMT2	128
IBM POWER7 processor	256	SMT4	1024
IBM POWER8 processor	192	SMT8	1536

Information about the multi-thread per core features by single LPAR scaling is available in the following tables:

- ▶ Table 4-1 on page 73 (*AIX*)
- ▶ Table 5-1 on page 112 (*IBM i*)
- ▶ Table 6-1 on page 119 (*Linux*)

Operating system enablement of multi-core and multi-thread technology varies by operating system and release:

- ▶ Power operating systems present an SMP view of the resources of a partition.
- ▶ Hardware threads are presented as logical CPUs to the application stack.
- ▶ Many applications can use the operating system scheduler to place workloads onto logical processors and maintain the SMP programming model.
- ▶ In some cases, the differentiation between hardware threads per core can be used to improve performance.
- ▶ Placement of a workload on hardware book, drawer and node, socket, core, and thread boundaries can improve application scaling.

Using multi-core and multi-thread features is a challenging prospect.

For more information about this topic, from the OS perspective, see:

- ▶ 4.2.1, “Multi-core and multi-thread” on page 72 (*AIX*)
- ▶ 5.2.1, “Multi-core and multi-thread” on page 112 (*IBM i*)
- ▶ 6.2.1, “Multi-core and multi-thread” on page 119 (*Linux*)

For more information about this topic, see 2.3, “I/O adapter affinity” on page 55.

Simultaneous multithreading

The Power Architecture uses simultaneous multithreading (SMT) to provide multiple streams of hardware execution. The POWER8 processor provides eight SMT hardware threads per core and can be configured to run in SMT8, SMT4, SMT2, or single-threaded mode (SMT1 mode or, as referred to in this publication, ST mode). The POWER7 and POWER7+ processors provide four SMT hardware threads per core and can be configured to run in SMT4, SMT2, or ST mode. POWER6 and POWER5 processors provide two SMT threads per core, and can be run in SMT2 mode or ST mode.

By using multiple SMT threads, a workload can take advantage of more of the hardware features that are provided in the POWER processor than if a single SMT thread is used per core. By configuring the processor core to run in multi-threaded mode, the operating system can maximize the use of the hardware capabilities that are provided in the system and the overall workload throughput by correctly balancing software threads across all of the cores and SMT hardware threads in the partition.

SMT does include some performance tradeoffs:

- ▶ SMT can provide a significant throughput and capacity improvement on POWER processors. When you are in SMT mode, there is a trade-off between overall CPU throughput and the performance of each hardware thread. SMT allows multiple instruction streams to be run simultaneously, but this concurrency can cause some resource conflict between the instruction streams. This conflict can result in a decrease in performance for an individual thread, but an increase in overall throughput.
- ▶ Some workloads do not run well with the SMT feature. This situation is not typical for commercial workloads, but it has been observed with scientific (floating point-intensive) workloads.

Information about the topic of SMT, from the OS perspective, is available in the following sections:

- ▶ “Simultaneous multithreading” on page 73 (*AIX*)
- ▶ “Simultaneous multithreading” on page 112 (*IBM i*)
- ▶ “Simultaneous multithreading” on page 119 (*Linux*)

Simultaneous multithreading priorities

The POWER5 processor introduced the capability for the SMT thread priority level for each hardware thread to be set, controlling the relative priority of the threads within a single core. This capability allows each SMT thread to be adjusted so that it can receive more or less favorable performance than the other threads in the same core. The relative difference between the priority of each hardware thread determines the number of decode cycles each thread receives during a period.² This mechanism can be used in various situations, for example, to boost the performance of other threads on the same processor core, while the thread with a lowered priority is waiting on a lock, or waiting on other cooperative threads to reach a synchronization point.

Table 2-2 lists various SMT thread priority levels that are supported in the Power Architecture. The level at which code can set the SMT priority level to is controlled by the privilege level that the code is running at (such as problem-state versus supervisor level). For example, code that is running in problem-state cannot set the SMT priority level to High.

Table 2-2 SMT thread priority levels for POWER5, POWER6, POWER7, POWER7+, and POWER8 processors

SMT thread priority level ^a	PPR (11:13)	Priority Nop	Minimum privilege required to set level in POWER5, POWER6, and POWER7 processors	Minimum privilege required to set level in a POWER7+ processor	Minimum privilege required to set level in a POWER8 processor
Thread shutoff (read only; set by disabling thread)	b'000'		Hypervisor	Hypervisor	Hypervisor
Very low	b'001'	or 31,31,31	Supervisor	Problem-state	Problem-state
Low	b'010'	or 1,1,1	Problem-state	Problem-state	Problem-state
Medium low	b'011'	or 6,6,6	Problem-state	Problem-state	Problem-state
Medium	b'100'	or 2,2,2	Problem-state	Problem-state	Problem-state
Medium high	b'101'	or 5,5,5	Supervisor	Supervisor	Problem-state
High	b'110'	or 3,3,3	Supervisor	Supervisor	Supervisor
Very high	b'111'	or 7,7,7	Hypervisor	Hypervisor	Hypervisor

a. The required privilege to set a particular SMT thread priority level is associated with the physical processor implementation that the LPAR is running on, and not the processor compatible mode. Therefore, setting Very Low SMT priority requires only user level privilege on POWER7+ processors, even when running in IBM POWER6-compatible, POWER6+™-compatible, or POWER7-compatible modes.

For more information about SMT priority levels, see *Power ISA Version 2.07*, found at:

<https://www.power.org/documentation/power-isa-v-2-07b/>

Changing the SMT priority level can generally be done in one of the following ways:

- ▶ Running a Priority Nop, a special form of the `or x,x,x nop`
- ▶ Writing a value to the Program Priority Register (PPR) by running `mtppr`
- ▶ Through a system call, which can be used by problem-state programs to set priorities in the range that is permitted for the supervisor state

² *thread_set_smt_priority* or *thread_read_smt_priority* System Call, found at:

http://www-01.ibm.com/support/knowledgecenter/api/redirect/aix/v7r1/index.jsp?topic=/com.ibm.aix.kerneltechref/doc/ktechrf1/thread_set_smt_priority.htm

On POWER5, POWER6, and POWER7 processor-based systems, problem-state programs can set thread priority values only in the range of low (2) to medium (4). On POWER7+ processor-based systems, a problem-state program can set the thread priority value to very low (1). POWER8 processor-based systems introduce the ability for a problem-state program to change temporarily the thread priority value to medium-high (5). However, access to medium-high priority is controlled by the operating system through the new Problem State Priority Boost Register that was introduced in POWER8 processor-based systems.³

For more information about the topic of SMT priorities, from the OS perspective, see:

- ▶ “Simultaneous multithreading priorities” on page 74 (*AIX*)
- ▶ “Simultaneous multithreading priorities” on page 120 (*Linux*)

Affinitization and binding to hardware threads

Functionally, it does not matter which core in the partition an application thread is running on, or what physical memory the data it is accessing is on. From a performance standpoint, however, software threads that all access the same data are best placed on the SMT threads of the same core, or on the cores of the same chip. Operating systems may provide facilities to bind applications or specific software threads to run on specific SMT threads or cores.

For more information about the topic of affinitization and binding, from the OS perspective, see:

- ▶ “Affinitization and binding” on page 74 (*AIX*)
- ▶ “Affinitization and binding” on page 121 (*Linux*)

Hybrid thread and core

The POWER8 processor allows the SMT mode of each core in a partition to be independently controlled by the operating system. Exactly how this facility is presented to the users is dependent on the specific operating system release and version. Some of the ways the operating systems can expose this feature include:

- ▶ The ability to set all of the cores in a partition to run in a specific SMT mode, such as to disable SMT and run all of the cores in ST mode.
- ▶ The ability to dynamically alter the SMT mode of specific cores based on load. When only a small number of software threads are ready to run, the operating system can lower the SMT mode of the cores to give each of the software threads the highest possible performance. When a large number of software threads are ready to run, the operating system can use higher SMT modes and maximize the overall throughput of the partition.
- ▶ The ability to specify a fixed asymmetric SMT configuration, where some cores are in high SMT mode and others have SMT mode disabled. This configuration allows critical software threads within a workload to receive an ST performance boost, and allows the remaining threads to benefit from SMT mode. Typical reasons to take advantage of this hybrid mode are:
 - For an asymmetric workload, where the performance of one thread serializes an entire workload. For example, one master thread dispatches work to many subordinate threads.
 - For software threads that are critical to a system administrator.

For more information about this topic, from the OS perspective, see:

- ▶ “Hybrid thread and core” on page 80 (*AIX*)
- ▶ “Hybrid thread and core” on page 122 (*Linux*)

³ Power ISA Version 2.07, found at <https://www.power.org/documentation/power-isa-v-2-07b/>

2.2.2 Multipage size support (page sizes (4 KB, 64 KB, 16 MB, and 16 GB))

The virtual address space of a program is divided into segments. The size of each segment can be either 256 MB or 1 TB on Power Systems. The virtual address space can also consist of a mix of these segment sizes. The segments are again divided into units, called *pages*. IBM Power Architecture supports multiple virtual memory page sizes, which provides performance benefits to an application because of hardware efficiencies that are associated with larger page sizes.⁴

The POWER5+ and later processors support four virtual memory page sizes: 4 KB, 64 KB, 16 MB, and 16 GB. The POWER6 and later processors also support using 64 KB pages inside segments along with a base page size of 4 KB.⁵ The 16 GB pages can be used only within 1 TB segments.

Large pages provide multiple technical advantages:

- ▶ **Reduced Page Faults and Translation Lookaside Buffer (TLB) Misses:** A single large page that is being constantly referenced remains in memory. This feature eliminates the possibility of several small pages often being swapped out.
- ▶ **Unhindered Data Prefetching:** A large page enables unhindered data prefetch (which is constrained by page boundaries).
- ▶ **Increased TLB Reach:** This feature saves space in the TLB by holding one translation entry instead of n entries, which increases the amount of memory that can be accessed by an application without incurring hardware translation delays.
- ▶ **Increased ERAT Reach:** The ERAT on Power Systems is a first level and fully associative translation cache that can go directly from effective to real address. Large pages also improve the efficiency and coverage of this translation cache as well.

Large segments (1 TB) also provide reduced Segment Lookaside Buffer (SLB) misses, and increases the reach of the SLB. The SLB is a cache of the most recently used Effective to Virtual Segment translations.

The 16 MB and 16 GB pages are intended only for high-performance environments; however, 64 KB pages are considered general-purpose, and most workloads benefit from using 64 KB pages rather than 4 KB pages.

For more information about this topic, from the OS perspective, see:

- ▶ 4.2.2, “Multipage size support on AIX” on page 83
- ▶ 5.2.2, “Multipage size support on IBM i” on page 113
- ▶ 6.2.2, “Multipage size support on Linux” on page 123

⁴ *Power ISA Version 2.07*, found at <https://www.power.org/documentation/power-isa-v-2-07b/>

⁵ *Multiple page size support*, found at:

http://www-01.ibm.com/support/knowledgecenter/api/redirect/aix/v7r1/index.jsp?topic=/com.ibm.aix.prftungd/doc/prftungd/multiple_page_size_support.htm

2.2.3 Efficient use of cache and memory

Hardware facilities for controlling the efficient use of cache and memory are described in this section.

Cache sharing

Power Systems consist of multiple processor cores and multiple processor chips that share caches and memory in the system. The architecture uses a processor and memory layout that you can use to scale the hardware to many nodes of processor chips and memory. One advantage is that systems can be used for multiple workloads and workloads that are large. However, these characteristics must be carefully weighed in the design, implementation, and evaluation of a workload. Aspects of a program, such as the allocation of data across cores and chips and the layout of data within a data structure, play a key role in maximizing performance, especially when scaling across many processor cores and chips.

Power Systems use a cache-coherent SMP design, in which all of the memory in the system is accessible to all of the processor cores in the system, and all of the cache is coherently maintained:

- ▶ Any processor core on any chip can access the memory of the entire system.
- ▶ Any processor core can access the contents of any core cache, even if it is on a different chip.

Processor core access: In both of these cases, the processor core can access only memory or cache that it has authorized access to using normal operating system and Hypervisor memory access permissions and controls.

In POWER8 processor-based systems, each chip consists of twelve processor cores, each with on-core L1 instruction and d-caches, an L2 cache, and an L3 cache, as shown in Figure 2-2.⁶

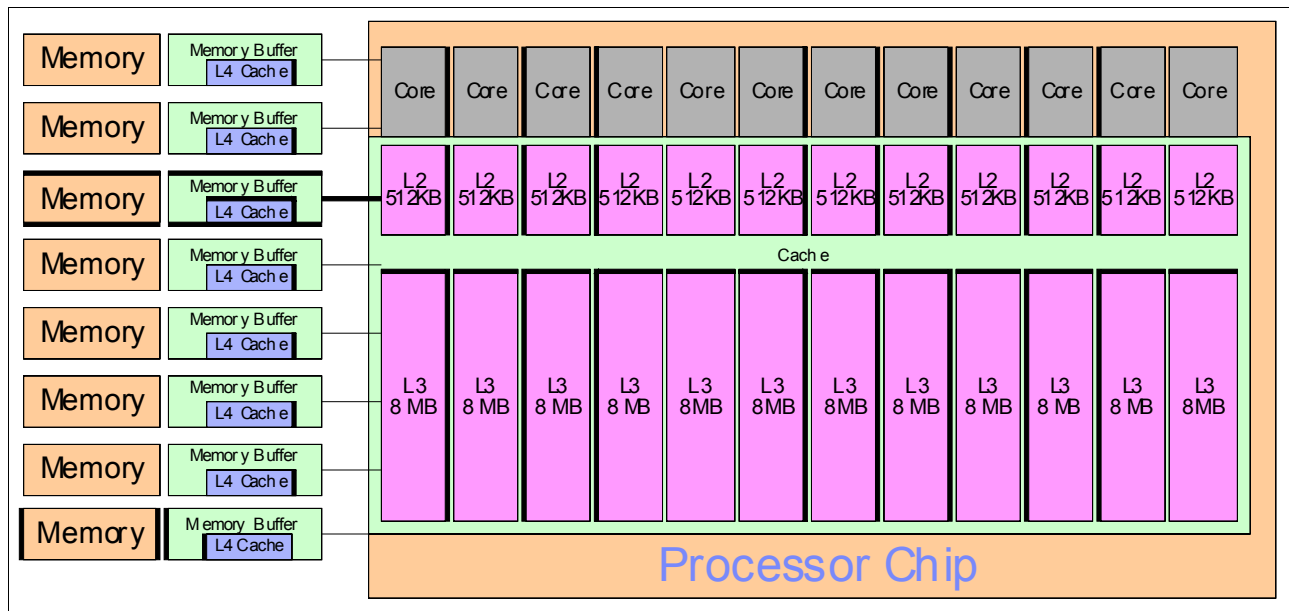


Figure 2-2 POWER8 chip and local memory

⁶ Ibid

All of these caches are effectively shared. The L2 cache has a longer access latency than L1, and L3 has a longer access latency than L2. Each chip also has memory controllers, allowing direct access to a portion of the memory DIMMs in the system.⁷ Thus, it takes longer for an application thread to access data in cache or memory that is attached to a remote chip than to access data in a local cache or memory. These types of characteristics are often referred to as *affinity performance effects* (for more information, see “The POWER8 processor and affinity performance effects” on page 16). In many cases, systems that are built around different processor models that have varying characteristics (for example, although L3 is supported, it might not be implemented on some models).

Functionally, it does not matter which core in the system an application thread is running on, or what memory the data it is accessing is on. However, this situation does affect the performance of applications because accessing a remote memory or cache takes more time than accessing a local memory or cache.⁸ This situation becomes even more imperative with the capability of modern systems to support massive scaling and the resulting possibility for remote accesses to occur across a large processor interconnection complex.

The effect of these system properties can be observed by application threads because they often move, sometimes rather frequently, between processor cores. This situation can happen for various reasons, such as a page fault or lock contention that results in the application thread being preempted while it waits for a condition to be satisfied, and then being resumed on a different core. Any application data that is in the cache local to the original core is no longer in the local cache because the application thread moved and a remote cache access is required.⁹ Although modern operating systems, such as AIX, attempt to ensure that cache and memory affinity is retained, this movement does occur, and can result in a loss in performance. For an introduction to the concepts of cache and memory affinity, see “The POWER8 processor and affinity performance effects” on page 16.

The POWER Hypervisor is responsible for:

- ▶ Virtualization of processor cores and memory that is presented to the operating system
- ▶ Ensuring that the affinity between the processor cores and memory an LPAR is using is maintained as much as possible

However, it is important for application designers to consider affinity issues in the design of applications, and to carefully assess the impact of application thread and data placement on the cores and the memory that is assigned to the LPAR the application is running in.

Various techniques that are employed at the system level can alleviate the effect of cache sharing. One example is to configure the LPAR so that the amount of memory that is requested for the LPAR is satisfied by the memories that are locally available to processor cores in the system (the memory DIMMs that are attached to the memory controllers for each processor core). It is more likely that the POWER Hypervisor can maintain affinity between the processor cores and memory that is assigned to the partition, improving performance.¹⁰

For more information about LPAR configuration and running the AIX `lsrad-va` command to query the affinity characteristics of a partition, see Chapter 3, “The IBM POWER Hypervisor” on page 57. The equivalent Linux command is `numactl --hardware`.

The rest of this section covers multiple topics that can affect application performance, including the effects of cache geometry, alignment of data, and sensitivity to the scaling of applications to more cores.

⁷ Of NUMA on POWER7 in IBM i, found at:

http://www.ibm.com/systems/resources/pwrsysperf_P7NUMA.pdf

⁸ Ibid

⁹ Ibid

¹⁰ Ibid

Cache geometry

Cache geometry refers to the specific layout of the caches in the system, including their location, interconnection, and sizes. These design details change for every processor chip, even within the Power Architecture. Figure 2-2 on page 33 shows the layout of a POWER8 chip, including the processor cores, caches, and local memory. Table 2-3 shows the cache sizes and related geometry information for POWER8 processor-based systems.¹¹

Table 2-3 POWER8 storage hierarchy

Cache	POWER7 processor-based system	POWER7+ processor-based system	POWER8 processor-based systems
L1 i-cache: Capacity/associativity	32 KB, 4-way	32 KB, 4-way	32 KB, 8-way
L1 d-cache: Capacity/associativity bandwidth	32 KB, 8-way 2 16 B reads or 1 16 B writes per cycle	32 KB, 8-way 2 16 B reads or 1 16 B writes per cycle	64 KB, 8-way 4 16 B reads or 1 16 B writes per cycle
L2 cache: Capacity/associativity bandwidth	256 KB, 8-way Private 32 B reads and 16 B writes per cycle	256 KB, 8-way Private 32 B reads and 16 B writes per cycle	512 KB, 8-way Private 64 B reads and 16 B writes per cycle
L3 cache: Capacity/associativity bandwidth	On-Chip 4 MB/core, 8-way 16 B reads and 16 B writes per cycle	On-Chip 10 MB/core, 8-way 16 B reads and 16 B writes per cycle	On-Chip 8 MB/core, 8-way 32 B reads and 32 B writes per cycle
L4 cache: Capacity/associativity bandwidth	N/A	N/A	On-Chip 16 MB/buffer chip, 16-way Up to 8 buffer chips per socket

Optimizing for cache geometry

There are several ways to optimize for cache geometry:

- Splitting structures into hot and cold elements

A technique for optimizing applications to take advantage of cache is to lay out data structures so that fields that have a high rate of reference (that is, hot) are grouped, and fields that have a relatively low rate of reference (that is, cold) are grouped.¹² The concept is to place the hot elements into the same *byte* region of memory, so that when they are pulled into the cache, they are co-located in to the same cache line or lines. Additionally, because hot elements are referenced often, they are likely to stay in the cache. Likewise, the cold elements are in the same area of memory and result in being in the same cache line, so that being written out to main storage and discarded causes less of a performance degradation. This situation occurs because they have a much lower rate of access.

Power Systems use 128-byte length cache lines. Compared to Intel processors (64-byte cache lines), these larger cache lines have the advantage of increasing the reach possible with the same size cache directory, and the efficiency of the cache by covering up to 128 bytes of hot data in a single line. However, it also has the implication of potentially bringing more data into the cache than needed for fine-grained accesses (that is, less than 64 bytes).

¹¹ Ibid

¹² *Splitting Data Objects to Increase Cache Utilization (Preliminary Version, 9th October 1998)*. found at: <http://citeseer.uark.edu:8080/citeseerx/viewdoc/summary?doi=10.1.1.84.3359>

As described in *Eliminate False Sharing, Stop your CPU power from invisibly going down the drain*,¹³ it is also important to assess carefully the impact of this strategy, especially when applied to systems where there are a high number of CPU cores and a phenomenon referred to as *false sharing* can occur. False sharing occurs when multiple data elements are in the same cache line that can otherwise be accessed independently. For example, if two different hardware threads wanted to update (store) two different words in the same cache line, only one of them at a time can gain exclusive access to the cache line to complete the store. This situation results in:

- Cache line transfers between the processors where those threads are
- Stalls in other threads that are waiting for the cache line
- Leaving all but the most recent thread to update the line without a copy in their cache

This effect is compounded as the number of application threads that share the cache line (that is, threads that are using different data in the cache line under contention) is scaled upwards.¹⁴ The discussion about cache sharing¹⁵ also presents techniques for analyzing false sharing and suggestions for addressing the phenomenon.

► Prefetching to avoid cache miss penalties

Prefetching to avoid cache miss penalties is another technique that is used to improve performance of applications. The concept is to prefetch blocks of data to be placed into the cache a number of cycles before the data is needed. This action hides the penalty of waiting for the data to be read from main storage. Prefetching can be speculative when, based on the conditional path that is taken through the code, the data might end up not being required. The benefit of prefetching depends on how often the prefetched data is used. Although prefetching is not strictly related to cache geometry, it is an important technique.

A caveat to prefetching is that, although it is common for the technique to improve performance for single-thread, single core, and low utilization environments, it can decrease performance in high thread-count per-socket and high-utilization environments. Most systems today virtualize processors and the memory that is used by the workload. Because of this situation, the application designer must consider that, although an LPAR might be assigned only a few cores, the overall system likely has a large number of cores. Further, if the LPARs are sharing processor cores, the problem becomes compounded.

The **dcbt** and **dcbtst** instructions are commonly used to prefetch data.^{16,17} *Power Architecture ISA 2.06 Stride N Prefetch Engines to boost Application's performance* provides an overview about how these instructions can be used to improve application performance. These instructions can be used directly in hand-tuned assembly language code, or they can be accessed through compiler built-ins or directives.

A preferred way to use pre-fetching is to have the compiler decide where in the application code to place prefetch instructions. The type of analysis that is required is highly suited for computers to perform.

Prefetching is also automatically done by the POWER8 hardware and is configurable, as described in “Data prefetching using d-cache instructions and the Data Streams Control Register (DSCR)” on page 39.

¹³ *Eliminate False Sharing, Stop your CPU power from invisibly going down the drain*, found at: <http://drdobbs.com/goparallel/article/showArticle.jhtml?articleID=217500206>

¹⁴ Ibid

¹⁵ Ibid

¹⁶ *dcbt (Data Cache Block Touch) instruction*, found at: http://www-01.ibm.com/support/knowledgecenter/api/redirect/aix/v7r1/topic/com.ibm.aix.aixassem/doc/clangref/clangref_dcbt_instrs.htm

¹⁷ *dcbtst (Data Cache Block Touch for Store) instruction*, found at: http://www-01.ibm.com/support/knowledgecenter/api/redirect/aix/v7r1/topic/com.ibm.aix.aixassem/doc/clangref/clangref_dcbstst_instrs.htm

Alignment of data

Processors are optimized for accessing data elements on their naturally aligned boundaries. Unaligned data accesses might require extra processing time by the processor for individual load or store instructions. They might require a trap and emulation by the host operating system. Ensuring natural data alignment also ensures that individual accesses do not span cache line boundaries.

Similar to the idea of splitting structures into hot and cold elements, the concept of data alignment seeks to optimize cache performance by ensuring that data does not span across multiple cache lines. The cache line size in Power Systems is 128 bytes.

The general technique for alignment is to keep operands (data) on *natural* boundaries, such as a word or doubleword boundary (that is, an int is aligned on a word boundary in memory). This technique might involve padding and reordering data structures to avoid cases such as the interleaving of chars and doubles: *char; double; char; double*. High-level language compilers can ensure optimal data alignment by inserting padding. However, data layout must be carefully analyzed to avoid an undue increase in size by such methods. For example, the previous case of a structure containing *char; double; char; double*; requires 14 bytes of padding. Such an increase in size might result in more cache misses or page misses (especially for rarely referenced groupings of data).

Additionally, to achieve optimal performance, floating point and VMX/VSX have different alignment requirements. For example, the preferred VSX alignment is 16 bytes instead of the element size of the data type being used. This situation means that VSX data that is smaller than 16 bytes must be padded out to 16 bytes. The compilers introduce padding as necessary to provide optimal alignment for vector data types.

Non-vector data that is intended to be accessed through VSX instructions should be aligned so that VSX loads and stores are performed on addresses that are aligned to 16-byte boundaries. However, the POWER8 processor improves the handling of misaligned accesses. Most loads, which cross cache lines and hit in the d-cache, are handled by the hardware with minimal impact on performance.

Byte ordering

The byte ordering (Big Endian or Little Endian) is specified by the operating system. In Little Endian mode, byte swapping is performed before data is written to storage and before data is fetched into the execution units. The Load and Store Multiple instructions and the Move Assist instructions are not supported in Little Endian mode. Attempting to run any of these instructions in Little Endian mode causes the system alignment error handler to be started.

The POWER8 processor can operate with the same byte ordering for both instruction and data, or with Split Endian, with instructions and data having different byte ordering.

Sensitivity of scaling to more cores

Different processor chip versions and system models provide less or more scaling of LPARs and workloads to cores. Different processor chips and systems might have different bus widths and latencies. All of these factors result in the sensitivity of the performance of an application/workload to the number of cores it is running on to change based on the processor chip version and system model.

In general terms, an application that tends to not access memory without CPU intervention (that are core-centric) scales perfectly across more cores. Performance loss when scaling across multiple cores tends to come from one or more of the following sources:

- ▶ Increased cache misses (often from invalidations of data by other processor cores, especially for locks)
- ▶ The increased cost of cache misses, which in turn drives overall memory and interconnect fabric traffic into the region of bandwidth limitations (saturating the memory busses and interconnect)
- ▶ The additional cores that are being added to the workload in other nodes, resulting in increased latency in reaching memory and caches in those nodes

Briefly, cache miss requests and returning data can end up being routed through busses that connect multiple chips and memory, which have particular bandwidth and latency characteristics. The goal for scaling across multiple cores, then, is to minimize the change in the potential penalties that are associated with cache misses and data requests as the workload size grows.

It is difficult to assess what strategies are effective for scaling to more cores without considering the complex aspects of a specific application. For example, if all of the cores that the application is running across eventually access all of the data, then it might be wise to interleave data across the processor sockets (which are typically a grouping of processor chips) to optimize them from a memory bus utilization point of view. However, if the access pattern to data is more localized so that, for most of the data, separate processor cores are accessing it most of the time, the application might obtain better performance if the data is close to the processor core that is accessing that data the most (maintaining affinity between the application thread and the data it is accessing). For the latter case, where the data ought to be close to the processor core that is accessing the data, the AIX `MEMORY_AFFINITY=MCM` environment variable can be set to achieve this behavior. For Linux, the equivalent is the `-l` option on a `numactl` command.

When multiple processor cores are accessing the same data and that data is being held by a lock, resulting in the data line in the cache that is invalidated, programs can suffer. This phenomenon is often referred to as *hot locks*, where a lock is holding data that has a high rate of contention. Hot locks result in cache-to-cache intervention and can easily limit the ability to scale a workload because all updates to the lock are serialized.

Tools such as `splat` (see “AIX trace-based analysis tools” on page 226) can be used to identify hot locks. Additionally, the transactional memory (TM) feature can speed up lock-based programs. Learn more about TM in 2.2.4, “Transactional memory” on page 42.

Hot locks can be caused by the programmer having lock control access to too large an area of data, which is known as *coarse-grained locking*.¹⁸ In that case, the strategy to deal effectively with a hot lock is to split the lock into a set of fine-grained locks, such that multiple locks, each managing a smaller portion of the data than the original lock, now manage the data for which access is being serialized. Hot locks can also be caused by trying to scale an application to more cores than the original design intended. In that case, using an even finer grain of locking might be possible, or changes can be made in data structures or algorithms, such that lock contention is reduced.

Additionally, the programmer must spend time considering the layout of locks in the cache to ensure that multiple locks, especially hot locks, are not in the same cache line because any updates to the lock itself results in the cache line being invalidated on other processor cores. When possible, pad the locks so that they are in their own distinct cache line.

¹⁸ *Synchronization & Deadlock Notes*, found at:
<http://www.read.seas.harvard.edu/~kohler/class/05s-osp/notes/notes8.html>

For more information about this topic, see 2.3, “I/O adapter affinity” on page 55.

Data prefetching using d-cache instructions and the Data Streams Control Register (DSCR)

The hardware data prefetch mechanism reduces the performance impact that is caused by the latency in retrieving cache lines from higher-level caches and from memory. The data prefetch engine of the processor can recognize sequential data access patterns in addition to certain non-sequential (stride-N) patterns and initiate prefetching of d-cache lines from L2 and L3 cache and memory into the L1 d-cache to improve the performance of these storage reference patterns.

The Power ISA architecture also provides cache instructions to supply a hint to prefetch engines for data prefetching to override the automatic stream detection capability of the data prefetcher. Cache instructions, such as **dcbt** and **dcbtst**, allow applications to specify stream direction, prefetch depth, and number of units. These instructions can avoid the starting cost of the automatic stream detection mechanism.

The d-cache instructions **dcbt** (d-cache block touch) and **dcbtst** (d-cache block touch for store) affect the behavior of the prefetched lines. The syntax for the assembly language instructions is:¹⁹

dcbt RA, RB, TH
dcbtst RA, RB, TH

- *RA* specifies a source general-purpose register for Effective Address (EA) computation.
- *RB* specifies a source general-purpose register for EA computation.
- *TH* indicates when a sequence of d-cache blocks might be needed.

The block that contains the byte addressed by the EA is fetched into the d-cache before the block is needed by the program. The program can later perform loads and stores from the block and might not experience the added delay that is caused by fetching the block into the cache.

The Touch Hint (TH) field is used to provide a hint that the program probably loads or stores to the storage locations specified by the EA and the TH field. The hint is ignored for locations that are caching-inhibited or guarded. The encodings of the TH field depend on the target architecture that is selected with the **-m** flag or the machine assembly language pseudo-op.

The **dcbt** and **dcbtst** instructions provide hints about a sequence of accesses to data elements, or indicate the expected use. Such a sequence is called a *data stream*. The range of values for the TH field describing data streams is 0b01000 - 0b01111. A **dcbt** or **dcbtst** instruction in which TH is set to one of these values is said to be a *data stream variant* of **dcbt** or **dcbtst**.

A data stream to which a program can perform *Load* accesses is said to be a *load data stream*, and is described by using the data stream variants of the **dcbt** instruction.

A data stream to which a program can perform *Store* accesses is said to be a *store data stream*, and is described by using the data stream variants of the **dcbtst** instruction.

¹⁹ Power ISA Version 2.07, found at <https://www.power.org/documentation/power-isa-v-2-07b/>

The **dcbt** and **dcbtst** instructions can also be used to provide hints about the transient nature of accesses to data elements. If TH=0b10000, the **dcbt** instruction provides a hint that the program will probably soon load from the block that contains the byte addressed by EA, and that the program's need for the block will be transient (this means the time interval during which the program accesses the block is likely to be short). If TH=0b10001, the **dcbt** instruction provides a hint that the program will probably not access the block that contains the byte addressed by EA for a relatively long period.

The contents of the DSCR, a special purpose register, affects how the data prefetcher responds to hardware-detected and software-defined data streams.

The layout of the DSCR register is shown in Table 2-4.

Table 2-4 DSCR register layout (field names are defined following the table)

	SWTE	HWTE	STE	LTE	SWUE	HWUE	UNT CNT	URG	LSD	SNSE	SSE	DPFD
0:38	39	40	41	42	43	44	45:54	55 57	58	59	60	61:63

Where:

- ▶ 39 Software Transient Enable (SWTE)
New field added in the POWER8 processor. Applies the transient attribute to software-defined streams.
- ▶ 40 Hardware Transient Enable (HWTE)
New field added in the POWER8 processor. Applies the transient attribute to hardware-detected streams.
- ▶ 41 Store Transient Enable (STE)
New field added in the POWER8 processor. Applies the transient attribute to store streams.
- ▶ 42 Load Transient Enable (LTE)
New field added in the POWER8 processor. Applies the transient attribute to load streams.
- ▶ 43 Software Unit count Enable (SWUE)
New field added in the POWER8 processor. Applies the unit count to software-defined streams.
- ▶ 44 Hardware Unit count Enable (HWUE)
New field added in the POWER8 processor. Applies the unit count to hardware-detected streams.
- ▶ 45:54 Unit Count (UNITCNT)
New field added in the POWER8 processor. Number of units in data stream. Streams that exceed this count are terminated.
- ▶ 55:57 Depth Attainment Urgency (URG)
New field added in the POWER7+ processor. This field indicates how quickly the prefetch depth can be reached for hardware-detected streams.
- ▶ Bits 58 Load Stream Disable (LDS)
New field added in the POWER7+ processor. Disables hardware detection and initiation of load streams.

- ▶ Bits 59 Stride-N Stream Enable (SNSE)
Enables hardware detection and initiation of load and store streams that have a stride greater than a single cache block. Such load streams are detected when LSD = 0, and such store streams are detected when SSE=1.
- ▶ Bits 60 Store Stream Enable (SSE)
Enables hardware detection and initiation of store streams.
- ▶ Bits 61:63 Default Prefetch Depth (DPFD)
Supplies a prefetch depth for hardware-detected streams and for software-defined streams for which a depth of zero is specified, or for which **dcbt** or **dcbtst** with TH=1010 is *not* used in their description.
- ▶ Bits 55:57 Depth Attainment Urgency (URG)
This field is a new one added in the POWER7+ processor. This field indicates how quickly the prefetch depth can be reached for hardware-detected streams. Values and their meanings are as follows:
 - 0: Default
 - 1: Not urgent
 - 2: Least urgent
 - 3: Less urgent
 - 4: Medium
 - 5: Urgent
 - 6: More urgent
 - 7: Most urgent

The ability to enable or disable the three types of streams that the hardware can detect (load streams, store streams, or stride-N streams), or to set the default prefetch depth, allows empirical testing of any application. There are no simple rules for determining which settings are optimum overall for an application: The performance of prefetching depends on many different characteristics of the application in addition to the characteristics of the specific system and its configuration. Data prefetches are purely speculative, meaning they can improve performance greatly when the data that is prefetched is, in fact, referenced by the application later, but can also degrade performance by expending bandwidth on cache lines that are not later referenced, or by displacing cache lines that are later referenced by the program.

Similarly, setting DPFD to a deeper depth tends to improve performance for data streams that are predominately sourced from memory because the longer the latency to overcome, the deeper the prefetching must be to maximize performance. But deeper prefetching also increases the possibility of stream overshoot, that is, prefetching lines beyond the end of the stream that are not later referenced. Prefetching in multi-core processor implementations has implications for other threads or processes that are sharing cache (in SMT mode) or the same system bandwidth.

For information about modifying the DSCR value by using the XL compiler family, see 7.3.4, “Data Streams Control Register controls” on page 154.

Instruction cache instructions

The **icbt** instruction provides a hint that the program will probably soon run code from a storage location and that the cache line containing that code will be loaded into the Level 2 cache. For example, see the instruction the follows:

icbt CT, RA, RB

Where:

- ▶ *RA* specifies a source general-purpose register for EA computation.
- ▶ *RB* specifies a source general-purpose register for EA computation.
- ▶ *CT* indicates the level of cache the block is to be loaded into. The only supported value for the POWER8 processor is 2.

Information about the efficient use of cache, from the OS perspective, is available in the following sections:

- ▶ 4.2.3, “Efficient use of cache” on page 86 (*AIX*)
- ▶ 6.2.3, “Efficient use of cache” on page 123 (*Linux*)
- ▶ 7.3.4, “Data Streams Control Register controls” on page 154 (*compilers*)

2.2.4 Transactional memory

Transactional memory is a shared-memory synchronization construct that allows process-threads to perform sequences of storage operations that appear to be atomic to other process-threads and applications. This allows for optimistic execution as a means to take advantage of the inherent parallelism that is found in the latest generation of Power Systems.

One of the main uses of TM is the speed-up of lock-based programs by using the speculative execution of lock-based critical sections (CSs), without first acquiring a lock. This allows applications that have not been carefully tuned for performance to take advantage of the benefits of fine-grain locking. The transactional programming model also provides productivity gains when developing lock-based shared memory programs.

Applications can also use TM to checkpoint and restore architectural state, independent of the atomic storage access guarantees that are provided by TM.

Using transactional memory

To use the TM facility in the most basic form, the process-thread marks the beginning and end of the sequence of storage accesses (namely, the transaction) by using the instructions **tbegin.** and **tend.**, respectively. The **tbegin.** instruction initiates transactional execution, during which the loads and stores appear to occur atomically. The **tend.** instruction terminates transactional execution.

A transaction may either succeed or fail. If a transaction succeeds, it is said to commit, and the transaction appears to have run as a single atomic unit when viewed by other processors and mechanisms. If a transaction fails, it is as though none of the instructions that were part of the transaction were ever run. The storage updates that were made since the **tbegin.** instruction was run are rolled back, and control is transferred to a software failure handler.

It is possible to nest transactions within one another, although the support is using a form of nesting called *flattened nesting*. New transactions that are begun during transactional execution are subsumed by the pre-existing transaction. The effects of a successful nested transaction do not become visible until the outermost (the first transaction that was started in the absence of any previous transactional execution) transaction commits. When a nested transaction fails, the entire set of transactions is rolled back, and control is transferred to the failure handler of the outermost transaction.

A transaction may be put into suspended state by the application by using the **tsuspend.** instruction. This allows a sequence of instructions within the transaction to have the same effect as though the sequence were run in the absence of a transaction. For example, such instructions are not run speculatively, and any storage updates are committed, regardless of transaction success or failure. The **tresume.** instruction is used to resume the transaction and to continue speculative execution of instructions.

Checkpoint state

When a transaction is initiated, and when it is restored following transaction failure, a set of registers is saved or restored, representing the checkpoint state of the processor (for example, the pre-transactional state). The checkpoint state includes all of the problem state, writable registers, with the exception of CR0, FXCC, EBBHR, EBBRR, BESCR, the performance monitor registers, and the TM special purpose registers (SPRs).

The checkpoint state is not directly accessible in either the supervisor or problem state. Instead, the checkpoint state is copied into the respective registers when the **treclaim.** instruction is run. This allows privileged code to save or modify values. The checkpoint state is copied back into the speculative registers (from the respective user-accessible registers) when the new **trechkpt.** instruction is run.

Transaction failure

A transaction might fail for various reasons, which can be either externally induced or self-induced. External causes include conflicts with the storage accesses of another process thread (for example, they both access the same storage area and one of the accesses is a store). There are many self-induced causes for a transaction to fail, for example:

- ▶ Explicitly aborted by using a set of conditional and unconditional abort instructions (for example, various forms of the **tabort.** instruction)
- ▶ Too many nested transactions
- ▶ Too many storage accesses performed in the transactional state, causing a state overflow
- ▶ Execution of certain instructions that are disallowed in transactional state (for example, **slbie**, **dcbi**, and so on)

When a transaction fails, a software failure handler may be started. This is accomplished by redirecting control to the instruction following the **tbegin.** instruction of the outermost transaction and setting CR0 to **0b1010**. Therefore, when writing a TM program, the **tbegin.** instruction must always be followed with a conditional branch (for example, **beq**), predicated on bit 2 of CR0. The target of the branch should be the software failure handler that is responsible for handling the transaction failure. For comparison, when **tbegin.** is successfully ran at the start of the transaction, CR0 is set to either **0b0000** or **0b0100**.

A transaction failure may be of a transient or a persistent type. Transient failures are typically considered temporary failures, and persistent failures indicate that it is unlikely that the transaction will succeed if restarted. The failure handler can retry the transaction or employ a different locking construct or logic path, depending on the nature of the failure. When handling transient type failures, applications might find it useful to keep a count of transient failures and to treat the failure as a persistent type failure on reaching a threshold. If the failure is of persistent type, the expectation is that the applications fall back to non-transactional logic.

When transaction failure occurs while in a suspended state, failure handling occurs after the transaction is resumed by using the **tresume.** instruction.

The software failure handler may identify the cause of the transaction failure by examining bits 0:31 of the Transaction EXception And Summary Register (TEXASR), a special purpose register that is associated with the TM architecture. In particular, bits 0:6 indicate the failure code, and bit 7 indicates whether the failure is persistent and whether the transaction will likely fail if attempted again. These bits are copied from the **trec1aim.** instruction (privileged code) or the **tabort.** instruction (problem state code) that are used by software to induce a transaction failure.

The Power Architecture Platform reserves a range of failure codes for use by client operating systems and a separate range for use by a hypervisor, leaving a range of codes free for use by software applications:

- ▶ 0x00 – 0x3F is reserved for use by the OS.
- ▶ 0x40 – 0xDF is free for use by problem state (application) code.
- ▶ 0xE0 – 0xFF is reserved for use by a hypervisor.

Problem state code is limited to using transaction failure codes to the range specified above to provide a failure reason when issuing a **tabort.** instruction.

Sample transaction

Example 2-1 is a sample of assembly language code, showing a simple transaction that writes the value in GPR 5 into the address in GPR 4, which is assumed to be shared among multiple threads of execution. If the transaction fails because of a persistent cause, the code falls back to an alternative code path at the label `lock_based_update` (the code for the alternative path is not shown) (based on sample code available from Power.org²⁰).

Example 2-1 A transaction that writes to an address that is shared among multiple execution threads

```
trans_entry:
    tbegin.                # Start transaction
    beq-   failure_hdlr    # Handle transaction failure

# Transaction Body
    stw r5, 0(r4)          # Write to memory pointed to by r4.
    tend.                 # End transaction
    b trans_exit

# Failure Handler
failure_hdlr:
    mfspr r4, TEXASRU      # Read high-order half of TEXASR
    andis. r5, r4, 0x0100  # Is the failure persistent?
    bne lock_based_update  # If persistent, acquire lock and
                          # then perform the write.
    b trans_entry          # If transient, try again.

# Alternate path for obtaining a lock and performing memory updates
# (non-transactional code path):

lock_based_update:

trans_exit:
```

²⁰ Power ISA Transactional Memory, found at:
<https://www.power.org/documentation/power-isa-transactional-memory/> (registration required).

Synchronization mechanisms

In multi-thread programs, synchronization mechanisms are used to ensure that threads have exclusive access to critical sections. Usually, compare-and-swap (CAS - x86_64) or load-link/store-conditional (LLSC - PowerPC) instructions are used to create locks, a synchronization mechanism. The semantics of locks is this: A running program acquires the lock, runs its CSs in a serialized way (only one thread of execution at a time), and releases the lock.

The serialization of threads because of CSs is a bottleneck to achieving high performance in multi-thread programs. There are some techniques for mitigating or removing such performance issues, for example, non-blocking algorithms, lock-free data, and fine-grained locking.

Lock Elision (LE) is another optimization technique that uses Hardware Transaction Memory (HTM) primitives to avoid lock acquiring. It relies on the behavior of some algorithms that do not have mutually exclusive executions of CS. For example, a hash table insertion where updates can be done in parallel, and locks are only needed when the same bucket is accessed at same time.

The LE uses an HTM to first try a transaction on a shared data resource. If it is successful, no locks are required. If the transaction cannot succeed, such as during concurrent access, it falls back to default locking mechanism.

For more information about the topic of transactional memory, from the OS and compiler perspectives, see:

- ▶ 4.2.4, “Transactional memory” on page 89 (*ALX*)
- ▶ 6.2.4, “Transactional memory” on page 124 (*Linux*)
- ▶ 7.3.5, “Transactional memory” on page 156 (*XL and GCC compiler families*)
- ▶ 8.4.2, “Transactional memory” on page 182 (*Java*)

2.2.5 Vector Scalar eXtension

Vector Scalar eXtension (VSX) in the Power ISA introduced more support for Vector and Scalar Binary flops conforming to the Institute of Electrical and Electronics Engineers-(IEEE)-754 Standard for Floating Point Arithmetic. The introduction of VSX into the Power Architecture increases the parallelism by providing SIMD execution functions for floating point double-precision to improve the performance of HPC applications.

The following VSX features are provided to increase opportunities for vectorization:

- ▶ A Unified Register File and a set of Vector-Scalar Registers (VSRs), supporting both scalar and vector operations, is provided, eliminating the impact of vector-scalar data transfer through storage.
- ▶ Support for word-aligned storage accesses for both scalar and vector operations is provided.
- ▶ Robust support for IEEE-754 for both vector and scalar flops is provided.
- ▶ Support for symmetric AES instructions that include polynomial multiply to support the Galois Counter Mode (GCM).

A 64-entry Unified Register File is shared across VSX, the Binary floating point unit (BFP), VMX, and the DFP unit. The thirty-two 64-bit Floating Point Registers (FPRs), which are used by the BFP and DFP units, are mapped to registers 0 - 31 of the Vector Scalar Registers. The 32 vector registers (VRs) that are used by the VMX are mapped to registers 32 - 63 of the VSRs, as shown in Table 2-5.

Table 2-5 The Unified Register File

FPR0		VSR0
FPR1		VSR1
....		
FPR30		
FPR31		
VR0		
VR1		
..		
..		
VR30		VSR62
VR31		VSR63

VSX supports Double Precision Scalar and Vector Operations and Single Precision Vector Operations. VSX instructions are broadly divided into two categories that can operate on 64 vector scalar registers:^{21, 22, 23, 24}

- ▶ Computational instructions: Addition, subtraction, multiplication, division, extracting the square root, rounding, conversion, comparison, and combinations of these operations
- ▶ Non-computational instructions: Loads/stores, moves, select values, and so on

In terms of compiler support for vectors, XLC supports vector processing technologies through language extensions on both AIX and Linux. GCC supports using the VSX engine on Linux. XL and GCC C implement and extend the AltiVec Programming Interface specification.

For more information about the topic of VSX, from the OS and compiler perspectives, see:

- ▶ 4.2.5, “Vector Scalar eXtension” on page 91 (*AIX*)
- ▶ 5.2.3, “Vector Scalar eXtension” on page 113 (*IBM i*)
- ▶ 6.2.5, “Vector Scalar eXtension” on page 125 (*Linux*)
- ▶ 7.3.2, “Compiler support for Vector Scalar eXtension” on page 151 (*XL and GCC compiler families*)

²¹ Support for POWER7 processors, found at:
<http://www-01.ibm.com/support/knowledgecenter/api/redirect/comphelp/v111v131/index.jsp?topic=/com.ibm.xlc111.aix.doc/getstart/architecture.html>

²² Vector built-in functions, found at:
http://www-01.ibm.com/support/knowledgecenter/api/redirect/comphelp/v111v131/index.jsp?topic=/com.ibm.xlc111.aix.doc/compiler_ref/vec_intrin_cpp.html

²³ Initialization of vectors (IBM extension), found at:
http://www-01.ibm.com/support/knowledgecenter/api/redirect/comphelp/v111v131/index.jsp?topic=/com.ibm.xlc111.aix.doc/language_ref/vector_init.html

²⁴ Engineering and Scientific Subroutine Library (ESSL), found at:
<http://www-01.ibm.com/support/knowledgecenter/api/redirect/clresctr/vxrx/index.jsp?topic=/com.ibm.cluster.essl.doc/esslbooks.html>

2.2.6 Decimal floating point

Decimal (base 10) data is widely used in commercial and financial applications. However, most computer systems have only binary (base two) arithmetic. There are two binary number systems in computers: integer (fixed-point) and floating point. Unfortunately, decimal calculations cannot be directly implemented with binary floating point. For example, the value 0.1 needs an infinitely recurring binary fraction, and a decimal number system can represent it exactly, as 1/10th. So, using binary floating point cannot ensure that results are the same as those results that use decimal arithmetic.

In general, DFP operations are emulated with binary fixed-point integers. Decimal numbers are traditionally held in a binary-coded decimal (BCD) format. Although BCD provides sufficient accuracy for decimal calculation, it imposes a heavy cost in performance because it is implemented in software.

POWER6, POWER7, and POWER8 processors provide hardware support for DFP arithmetic. The POWER6, POWER7, and POWER8 microprocessor cores include a DFP unit that provides acceleration for the DFP arithmetic. The IBM Power Systems instruction set is expanded: 54 new instructions were added to support the DFP unit architecture. DFP can provide a performance boost for applications that are using BCD calculations.²⁵

For more information about this topic, from the OS perspective, see:

- ▶ 4.2.6, “Decimal floating point” on page 92 (*AIX*)
- ▶ 5.2.4, “Decimal floating point” on page 113 (*IBM i*)
- ▶ 6.2.6, “Decimal floating point” on page 126 (*Linux*)

2.2.7 In-core cryptography and integrity enhancements

POWER8 in-core enhancements are targeting applications by the use of symmetric cryptography (Advanced Encryption Standard (AES)) and security (Secure Hash Algorithms (SHA-2)) and cyclic redundancy check (CRC) algorithms. In cryptography, the information is scrambled so that only an authorized receiver can read the message. Asymmetric-key algorithms require two separate keys, one private and one public, and symmetric-key algorithms use the same key for encryption and decryption (for example, AES). Many applications not only require information protection (for confidentiality) but they also need to ensure that data is not changed when sent to the receiver (for integrity). This is realized by cryptographic hash functions, which take an arbitrary block of data (often called a *message*) and return a fixed-size bit string (called a *message digest* or *digest*). Well-established algorithms are SHA and CRC.

AES

AES was established for the encryption of electronic data by the US National Institute of Standards and Technology (NIST) in 2001 (FIPS PUB 197). AES is a symmetric-key algorithm that processes data blocks of 128 bits (a block cipher algorithm), and, therefore, naturally fits into the 128-bit VSX data flow. The AES algorithm is covered in five new instructions, available in *Power ISA Version 2.07*.²⁶

²⁵ *How to Leverage Decimal Floating-Point unit on POWER6 for Linux*, found at:

<https://www.ibm.com/developerworks/community/wikis/home?lang=en#!/wiki/Welcome%20to%20High%20Performance%20Computing%20%28HPC%29%20Central/page/How%20to%20Leverage%20Decimal%20Floating-Point%20unit%20on%20POWER6%20for%20Linux>

²⁶ *Power ISA Version 2.07*, found at <https://www.power.org/documentation/power-isa-v-2-07b/>

AES special mode of operation: Galois Counter Mode

The AES Galois Counter Mode (GCM) mode of operation is designed to provide both confidentiality and integrity (for authentication). GCM is defined for block ciphers (block sizes of 128, 192, and 256 bits). The key feature is that Galois Field multiplication (used for authentication) can be computed in parallel, resulting in higher throughput than the authentication algorithms that use chaining modes.

SHA-2

SHA-2 was designed by the US National Security Agency (NSA) and published in 2001 by the NIST (FIPS PUB 180-2). It is a set of four hash functions (SHA-224, SHA-256, SHA-384, and SHA-512) with message digests that are 224, 256, 384, and 512 bits. The SHA-2 functions compute the digest based on 32-bit words (SHA-224 and SHA-256) or 64-bit words (SHA-384 and SHA-512). Different combinations of *rotate* and *xor* vector instructions have been identified to be merged into a new instruction to accelerate the SHA-2 family. The new instruction comes in two flavors:

- ▶ In word (32-bit), targeting SHA-224 and SHA-256
- ▶ In doubleword (64 bit), accelerating SHA-384 and SHA-512 (Power ISA v2.07)

CRC

CRC can be seen as an error-detecting code. It is used in storage devices and digital networks to protect data from accidental (or hacker-intended) changes to raw data. Data to be stored or information that is sent over the network (in a stream) gets a short **checksum** attached (based on the remainder of the polynomial division and modulo operations). CRC is a reversible function, which makes it unsuitable for use in digital signatures, but it is in use for error detection when data is transferred, for example, in an Ethernet network protocol.

CRC algorithms are defined by the different generator polynomial used. For example, an n-bit CRC is defined by an n-bit polynomial. Examples for applications using CRC-32 are Ethernet (Open Systems Interconnection (OSI) physical layer), Serial Advanced Technology Attachment (Serial ATA), Moving Picture Experts Group (MPEG-2), GNU Project file compression software (Gzip), and Portable Network Graphics (PNG, fixed 32-bit polynomial). In contrast, Internet Small Computer System Interface (iSCSI) and the Stream Control Transmission Protocol (SCTP transport layer protocol) are based on a different, 32-bit polynomial.²⁷ The POWER8 enhancements focus on a specific application that supports only one single generator polynomial, and they help to accelerate any kind of CRC size, ranging from 8-bit CRC, 16-bit CRC, and 32-bit CRC, to 64-bit CRC.

For more information about the topic of in-core cryptography, from the OS and compiler perspectives, see:

- ▶ 4.2.7, “On-chip encryption accelerator” on page 94 (*AIX*)
- ▶ 7.3.1, “In-core cryptography” on page 148 (*XL and GCC compiler families*)

2.2.8 On-chip accelerators

On-chip accelerators, initially available in the POWER7 processor, provide the following benefits:

- ▶ *On-chip encryption*: AIX transparently uses on-chip encryption accelerators. There are no application visible changes or awareness required.
- ▶ *On-chip compression*.

²⁷ Optimization of cyclic redundancy-check codes with 24 and 32 parity bits, found at <http://ieeexplore.ieee.org/xpl/login.jsp?tp=&arnumber=231911&url=http%3A%2F%2Fieeexplore.ieee.org%2Fiee1%2F26%2F5993%2F00231911>

- *On-chip random number generator*: AIX capitalizes on the on-chip random number generator, providing the advantages of stronger hardware-based random numbers. In some instances, there can also be a performance advantage.

For more information about this topic, from the AIX perspective, see:

- 4.2.7, “On-chip encryption accelerator” on page 94 (*AIX*)
- “AIX /dev/random (random number generation)” on page 94 (*AIX*)

2.2.9 Storage synchronization (**sync**, **lwsync**, **lwarx**, **stwcx.**, and **eieio**)

The Power Architecture storage model provides for out-of-order storage accesses, providing opportunities for performance enhancement when accesses do not need to be in order. However, when accessing storage that is shared by multiple processor cores or shared with I/O devices, it is important that accesses occur in the correct order that is required for the sharing mechanisms that are used.

The architecture provides mechanisms for synchronization of such storage accesses and defines an architectural model that ought to be adhered to by software. Several synchronization instructions are provided by the architecture, such as **sync**, **lwsync**, **lwarx**, **stwcx.**, and **eieio**. There are also operating system-specific locking services provided that enforce such synchronization. Software must be carefully designed when you use these mechanisms to ensure optimal performance while providing appropriate data consistency because of their inherent heavyweight nature.

Concepts and benefits

The Power Architecture defines a storage model that provides weak ordering of storage accesses. The order in which memory accesses are performed might differ from the program order and the order in which the instructions that cause the accesses are run.²⁸

The Power Architecture provides a set of instructions that enforce storage access synchronization, and the AIX kernel provides a set of kernel services that provide locking mechanisms and associated synchronization support.²⁹ However, such mechanisms come with an inherent cost because of the nature of synchronization. Thus, it is important to use intelligently the correct storage mechanisms for the various types of storage access scenarios to ensure that accesses are performed in program order while minimizing their impact.

Associated instructions

The following instructions provide various storage synchronization mechanisms:

- | | |
|---------------|---|
| sync | This instruction provides an ordering function, so that all instructions issued before the sync complete and no subsequent instructions are issued until after the sync completes. ³⁰ |
| lwsync | This instruction provides an ordering function similar to sync , but it is only applicable to load , store , and dcbb instructions that are run by the processor (hardware thread) running the lwsync instruction, and only for specific combinations of storage control attributes. ³¹ |

²⁸ *PowerPC storage model and AIX programming: What AIX programmers need to know about how their software accesses shared storage*, found at: <http://www.ibm.com/developerworks/systems/articles/powerpc.html>

²⁹ *Ibid*

³⁰ *sync (Synchronize) or dcs (Data Cache Synchronize) instruction*, found at: http://www-01.ibm.com/support/knowledgecenter/api/redirect/aix/v7r1/index.jsp?topic=%2Fcom.ibm.aix.aixassem%2Fdoc%2Fdalangref%2Fidalangref_sync_dcs_instrs.htm

³¹ *PowerPC storage model and AIX programming: What AIX programmers need to know about how their software accesses shared storage*, found at: <http://www.ibm.com/developerworks/systems/articles/powerpc.html>

lwarx	This instruction reserves a storage location for subsequent store by using a stwcx. instruction and notifies the memory coherence mechanism of the reservation. ³²
stwcx.	This instruction performs a store to the target location only if the location specified by a previous lwarx instruction is not used for storage by another processor (hardware thread) or mechanism, which invalidates the reservation. ³³
eieio	This instruction creates a memory barrier that provides an order for storage accesses caused by load , store , dcbz , eciwx , and ecowx instructions. ³⁴
makeitso	New in the POWER8 processor, this instruction allows data to push out to the coherence point as quickly as possible. An attempt to run the makeitso instruction provides a hint that preceding stores are made visible with higher priority.
lbarx/stbcx.	These instructions were added in the POWER8 processor and are similar to lwarx/stwcx. , except that they load and store a byte.
lharx/sthcx.	These instructions were added in the POWER8 processor and are similar to lwarx/stwcx. , except that they load and store a 16-bit halfword.
ldarx/stdcx.	These instructions are similar to lwarx/stwcx. , except that they load and store a 64-bit doubleword (requires 64-bit mode).
lqarx/stqcx.	These instructions were added in the POWER8 processor and are similar to lwarx/stwcx. , except that they load and store a 128-bit quad word (requires 64-bit mode).

Where to use

Care must be taken when you use synchronization mechanisms in any processor architecture because the associated load and store instructions have a heavier weight than normal loads and stores, and the barrier operations have a cost that is associated with them. Thus, it is imperative that the programmer carefully consider when and where to use such operations, so that data consistency is ensured without adversely affecting the performance of the software and the overall system.

*PowerPC storage model and AIX programming*³⁵ describes where synchronization mechanisms must be used to ensure that the code adheres to the Power Architecture. Although this documentation covers how to write compliant code, it does not cover the performance aspect of using the mechanisms.

Unless the code is hand-tuned assembly language code, take advantage of the locking services that are provided by the operating system because they are tuned and provide the necessary synchronization mechanisms. *Power Instruction Set Architecture Version 2.07*³⁶ provides assembly language programming examples for sharing storage. For more information, see Appendix B, “Performance tools and empirical performance analysis” on page 215.

³² *lwarx* (Load Word and Reserve Indexed) instruction, found at: http://www-01.ibm.com/support/knowledgecenter/api/redirect/aix/v7r1/index.jsp?topic=%2Fcom.ibm.aix.aixassem%2Fdoc%2Falangref%2Fidalangref_lwarx_lwri_instrs.htm

³³ *stwcx* (Store Word Conditional Indexed) instruction, found at: <http://www-01.ibm.com/support/knowledgecenter/api/redirect/pseries/v5r3/index.jsp?topic=/com.ibm.aix.aixassem/doc/alangref/stwcx.htm>

³⁴ *eieio* (Enforce In-Order Execution of I/O) instruction, found at: http://www-01.ibm.com/support/knowledgecenter/api/redirect/aix/v7r1/topic/com.ibm.aix.aixassem/doc/alangref/idalangref_eieio_instrs.htm

³⁵ *PowerPC storage model and AIX programming: What AIX programmers need to know about how their software accesses shared storage*, found at: <http://www.ibm.com/developerworks/systems/articles/powerpc.html>

³⁶ *Power ISA Version 2.07*, found at <https://www.power.org/documentation/power-isa-v-2-07b/>

For more information about this topic, from the perspective of compiler built-ins, see 7.3.3, “Built-in functions for storage synchronization” on page 154.

2.2.10 Fixed-point load and store quadword instructions

The Power Architecture provides load and store instructions that operate on quadwords (16 bytes) of storage. The Load Quadword (**lq**) instruction loads an even-odd pair of general-purpose registers from the storage that is addressed by the effective address that is specified by the instruction. The store quadword (**stq**) instruction stores the contents of an even-odd pair of general-purpose registers in to the storage that is addressed by the effective address that is specified by the instruction.

2.2.11 Instruction fusion

POWER8 instruction fusion combines information from two adjacent instructions into one instruction, such that it runs faster than the non-fused instruction. Two forms of fusion are supported for loads with immediate fields that are larger than the allotted 16 bits that are provided by the Power Architecture. This is typically accomplished by running an **addis** instruction to compute the address followed by a load from that address. The two forms of fusion are described in “Table of content fusion” on page 51 and “Vector load fusion” on page 51.

Capitalizing on instruction fusion

The instruction fusion capabilities of the POWER8 processor are a feature of the processor and do not require special options for the compilers to use them. However, for best performance, **-qtune=pwr8** (XL family) or **-mtune=power8** (GCC) are recommended for the best use of this feature.

For hand-tuned assembly language code, ensure that the appropriate pattern of code is used and that the two instructions to be fused are adjacent.

Table of content fusion

Here is an example of table of content fusion:

ADDIS RT, RA, SI

LD RT, RA, DS (eligible instructions are **LD**, **LBZ**, **LHZ**, and **LWZ**)

Where the **RT** of the **ADDIS** is the same as **RA** of the **LD** instruction. The POWER8 processor internally fuses them into a single instruction.

Vector load fusion

Here is an example of vector load fusion:

addi RT,0,SI

lvx VRT, RA, RB (eligible instructions are **lxvd2x**, **lxvw4x**, **lxvdsx**, **lvebx**, **lvehx**, **lvewx**, **lvx**, and **lxsdx**)

Where **RT** of **ADDI** is the same as **RB** of the **LVX** instruction and **RA** cannot be zero. The POWER8 processor internally fuses them into a single instruction.

2.2.12 Event-based branches (or user-level fast interrupts)

The event-based branch facility is a hardware facility that generates event-based exceptions when certain event criteria are met. For example, this facility allows application programs to enable hardware to change the EA of the next instruction to be run when certain events occur to an EA specified by the program.

For more information about this topic, from the OS perspective, see 6.2.7, “Event-based branches” on page 128 (*Linux*).

2.2.13 Power management and system performance

The POWER8 processor has power saving and performance enhancing features that can be used to lower overall energy usage, and yielding higher performance when needed. The following modes can be enabled and modified to use these features.

Dynamic Power Saver: Favor Performance

This mode is intended to provide the best performance. If the processor is being used even moderately, the frequency is raised to the maximum frequency possible to provide the best performance. If the processors are lightly used, the frequency is lowered to the minimum frequency, which is potentially far below the nominal shipped frequency, to save energy. The top frequency that is achieved is based on system type and is affected by environmental conditions. Also, when running at the maximum frequency, more energy is being consumed, which means this mode can potentially cause an increase in overall energy consumption.

Dynamic Power Saver: Favor Power

This mode is intended to provide the best performance per watt consumed. The processor frequency is adjusted based on the processor utilization to maintain the workload throughput without using more energy than required to do so. At high processor utilization levels, the frequency is raised above nominal, as in the Favor Performance mode. Likewise, at low processor utilization levels, the frequency is lowered to the minimum frequency. The frequency ranges are the same for the two Dynamic Power Saver modes, but the algorithm that determines which frequency to set is different.

Dynamic Power Saver: Tunable Parameters

The Favor Performance and Favor Power modes are tuned to provide both energy savings and performance increases. However, there might be situations where only top performance is of concern, or, conversely, where peak power consumption is an issue. The tunable parameters can be used to modify the setting of the processor frequency in these modes to meet these various objectives. Modifying these parameters should be done only by advanced users. If there are issues that must be addressed by the Tunable Parameters, IBM should be directly involved in the parameter value selection.

Idle Power Saver

This mode is intended to save the maximum amount of energy when the system is nearly idle. When the processors are found to be nearly idle, the frequency of all processors is lowered to the minimum. Additionally, workloads are dispatched onto a smaller number of processor cores so that the other processor cores can be put into a low energy usage state. When processor utilization increases, the process is reversed: The processor frequency is raised back up to nominal, and the workloads are spread out once again over all of the processor cores. There is no performance boosting aspect in this mode, but entering or exiting this mode might affect overall performance. The delay times and utilization levels for entering and exiting this mode can be adjusted to allow for more or less aggressive energy savings.

The controls for all modes that are listed above are available on the Advanced System Management Interface and are described in more detail in a white paper that is found at <http://www.ibm.com/common/ssi/cgi-bin/ssialias?infotype=SA&subtype=WH&htmlfid=POW03125USEN>. Additionally, the appendix of this white paper includes links to other papers that detail the performance benefits and impacts of using these controls.

2.2.14 Coherent Accelerator Processor Interface

The Coherent Accelerator Interface Architecture (CAIA) defines a coherent accelerator interface structure for attaching peripheral devices to Power Systems. This allows accelerators to work coherently with system memory, removing additional processing from the main system processor and reducing the overall memory requirements for the accelerator.

The Coherent Accelerator Processor Interface (CAPI) can attach accelerators that have coherent shared memory access to the processors in the server and share full virtual address translation with these processors by using a standard PCIe Gen3 bus.

Applications can access customized functions in Field Programmable Gate Arrays (FPGAs), allowing them to enqueue work requests directly in shared memory queues to the FPGA, and using the same effective addresses (pointers) it uses for any of its threads running on a host processor. From the practical perspective, CAPI allows a specialized hardware accelerator to be seen as an additional processor in the system, with access to the main system memory, and coherent communication with other processors in the system.

The benefits of using CAPI include the ability to access shared memory blocks directly from the accelerator, the ability to perform memory transfers directly between the accelerator and processor cache, and a reduction in the code path length between the adapter and the processors. The latter occurs because the adapter is not operating as a traditional I/O device, and there is no device driver layer to perform processing. It also presents a simpler programming model.

Figure 2-3 shows a high-level view of how an accelerator communicates with the POWER8 processor through CAPI. The POWER8 processor provides a Coherent Attached Processor Proxy (CAPP), which is responsible for extending the coherence in the processor communications to an external device. The coherency protocol is tunneled over standard PCIe Gen3 connections, effectively making the accelerator part of the coherency domain.

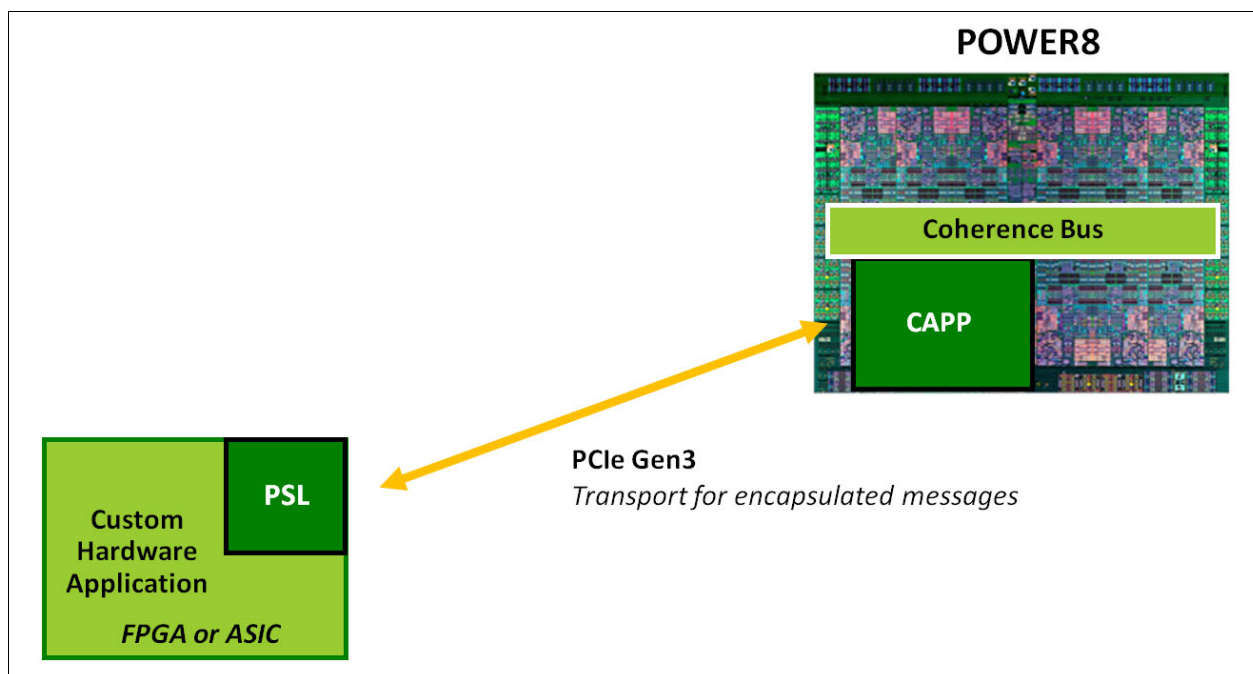


Figure 2-3 CAPI accelerator that is attached to the POWER8 processor

The accelerator adapter implements the Power Service Layer (PSL), which provides address translation and system memory cache for the accelerator functions. The custom processors on the board, which might consist of an FPGA or an Application Specific Integrated Circuit (ASIC) use this layer to access shared memory regions and cache areas as though they were a processor in the system. This ability greatly enhances the performance of the data access for the device and simplifies the programming effort to use the device. Instead of treating the hardware accelerator as an I/O device, it is treated as a processor. This eliminates the requirement of a device driver to perform communication, and the need for Direct Memory Access that requires system calls to the operating system kernel. By removing these layers, the data transfer operation requires fewer clock cycles in the processor, greatly improving the I/O performance.

The implementation of CAPI on the POWER8 processor allows hardware companies to develop solutions for specific application demands and use the performance of the POWER8 processor for general applications. The developers can also provide custom acceleration of specific functions by using a hardware accelerator, with a simplified programming model and efficient communication with the processor and memory resources.

2.3 I/O adapter affinity

The POWER8 processor benefits from the next generation PCIe Gen3 and shorter hardware paths to the adapters. The POWER8 chip design includes on-chip PCIe buses, which means that there are fewer hardware delays and lower latency to PCIe slots. This results in lower latency for networking and storage protocols, reducing latency by over 1 μ s from the previous I/O hub-based POWER7 systems.

The PCIe Gen3 x16 bus increases the PCIe bus peak bandwidth to 112 Gbps (for a single x16 bus), which is about four times the bandwidth in previous POWER PCIe Gen2 x8 slots in POWER7 and POWER7+ processor-based systems. Each processor module or socket provides two or four PCIe buses depending on the system model. Some buses are x16 and others are x8. Depending on the system model several buses can connect to PCIe I/O hub chips on the system board. These PCIe slots are for slower speed adapters so they can share a single higher speed bus. In addition, for systems that support I/O drawers, the PCIe buses in the I/O drawer all use a PCIe I/O hub. For applications that are sensitive to latency or those requiring high bandwidth or high message rates, the adapters that are used by the application should use, when possible, the PCIe slots in the chip where the application runs.

In all cases, bandwidth should be the same no matter where in a POWER8 processor-based system an adapter is plugged. However, for latency, there are small increases if the adapter is in a different socket, different node, or different central electronic complex (CEC) drawer.

As a general rule, high-speed and low-latency adapters should be placed in the direct PCIe slots on each socket. The PCIe slots behind I/O hubs and in I/O drawers should be used for lower bandwidth and non-latency sensitive adapters. There are a number of adapters that are restricted to certain slots. Consult a *Power Systems PCI Adapter Placement Guide* for a specific adapter.

2.4 Related publications

The publications that are listed in this section are considered suitable for a more detailed description of the topics that are covered in this chapter:

- ▶ *AIX dscr_ctl API sample code*, found at:
<https://www.power.org/documentation/performance-guide-for-hpc-applications-on-ibm-power-755-system/> (registration required)
- ▶ *AIX Version 7.1 Release Notes*, found at:
<http://www-01.ibm.com/support/knowledgecenter/api/redirect/aix/v7r1/index.jsp?topic=/com.ibm.aix.ntl/RELNOTES/GI11-9815-00.htm>
Refer to the “The **dscrctl** command” section.
- ▶ *Application configuration for large pages*, found at:
http://www-01.ibm.com/support/knowledgecenter/api/redirect/aix/v7r1/index.jsp?topic=/com.ibm.aix.prftungd/doc/prftungd/config_apps_large_pages.htm
- ▶ *False Sharing*, found at:
<http://msdn.microsoft.com/en-us/magazine/cc872851.aspx>
- ▶ *sync (Synchronize) or dcs (Data Cache Synchronize) instruction*, including information about **sync** and **lwsync** (lightweight sync), found at:
http://www-01.ibm.com/support/knowledgecenter/api/redirect/aix/v7r1/topic/com.ibm.aix.aixassem/doc/alongref/idalangref_sync_dcs_instrs.htm

- ▶ *The Performance of Runtime Data Cache Prefetching in a Dynamic Optimization System*, found at:
<http://www.microarch.org/micro36/html/pdf/lu-PerformanceRuntimeData.pdf>
- ▶ *POWER6 Decimal Floating Point (DFP)*, found at:
<https://www.ibm.com/developerworks/community/wikis/home?lang=en#!/wiki/Power+Systems/page/POWER6+Decimal+Floating+Point+%28DFP%29>
- ▶ *Power ISA Transactional Memory*, found at:
<https://www.power.org/documentation/power-isa-transactional-memory/>
- ▶ *Power ISA Version 2.07*, found at
<https://www.power.org/documentation/power-isa-v-2-07b/>

Refer to the following sections:

- Section 3.1: Program Priority Registers
 - Section 3.2: “or” Instruction
 - Section 4.3.4: Program Priority Register
 - Section 4.4.3: OR Instruction
 - Section 5.3.4: Program Priority Register
 - Section 5.4.2: OR Instruction
 - Book I – 4 Floating Point Facility
 - Book I – 5 Decimal Floating Point
 - Book I – 6 Vector Facility
 - Book I – 7 Vector-Scalar Floating Point Operations (VSX)
 - Book I – Chapter 5 Decimal Floating-Point.
 - Book II – 4.2 Data Stream Control Register
 - Book II – 4.3.2 Data Cache Instructions
 - Book II – 4.4 Synchronization Instructions
 - Book II – A.2 Load and Reserve Mnemonics
 - Book II – A.3 Synchronize Mnemonics
 - Book II – Appendix B. Programming Examples for Sharing Storage
 - Book III – 5.7 Storage Addressing
- ▶ *PowerPC storage model and AIX programming: What AIX programmers need to know about how their software accesses shared storage*, found at:
<http://www.ibm.com/developerworks/systems/articles/powerpc.html>
- Refer to the following sections:
- Power Instruction Set Architecture
 - Section 4.4.3 Memory Barrier Instructions – Synchronize
- ▶ *Product documentation for XL C/C++ for AIX, V12.1 (PDF format)*, found at:
<http://www.ibm.com/support/docview.wss?uid=swg27024811>
 - ▶ *Simple performance lock analysis tool (splat)*, found at:
http://www-01.ibm.com/support/knowledgecenter/api/redirect/aix/v7r1/topic/com.ibm.aix.prftools/doc/prftools/idprftools_splat.htm
 - ▶ *What makes Apple's PowerPC memcpy so fast?*, found at:
<http://stackoverflow.com/questions/1990343/what-makes-apples-powerpc-memcpy-so-fast>
 - ▶ *What programmers need to know about hardware prefetching?*, found at:
<http://www.futurechips.org/chip-design-for-all/prefetching.html>



The IBM POWER Hypervisor

This chapter introduces the POWER8 Hypervisor and describes some of the technical details for this product. It covers the following topics:

- ▶ 3.1, “Introduction to PowerVM” on page 58
- ▶ 3.2, “Power Systems virtualization with PowerVM” on page 59
- ▶ 3.3, “Introduction to KVM Virtualization” on page 67
- ▶ 3.4, “Related publications” on page 68

3.1 Introduction to PowerVM

Power Virtualization was introduced in POWER5 processor-based systems, so there are many reference materials that are available that cover all three resources (CPU, memory, and I/O), virtualization, capacity planning, and virtualization management. Some of these documents are shown in the reference section at the end of this chapter, which focuses on POWER8 virtualization. As for any workload deployments, capacity planning, selecting the correct set of technologies, and appropriate tuning are critical to deploying high performing workloads. However, in deploying workloads in virtualized environments, there are more aspects to consider, such as consolidation ratio, workload resource usage patterns, and the suitability of a workload to run in a shared resource environment (or latency requirements).

The first step in the virtualization deployment process is to understand whether the performance of a workload in a shared resource environment meets customer requirements. If the workload requires consistent performance with stringent latency requirements, then such workloads must be deployed on a dedicated partition rather than on a shared LPAR. The exceptions are where a shared processor pool is not heavily over committed and over-utilized; such workloads can meet stringent requirements in a shared LPAR configuration.

It is a preferred practice to understand the resource usage of all workloads that are planned for consolidation on a single system, especially when you plan to use a shared resource model, such as shared LPARs, IBM Active Memory™ Sharing, and VIOS technologies. The next step is to use a capacity planning tool that takes virtualization impacts into consideration, such as the IBM Workload Estimator, to estimate capacity for each partition. One of the goals of virtualization is maximizing usage. This improved usage can be achieved by consolidating workloads that peak at different times. This is done in a non-overlapping manner, so each workload (or partition) does not have to be sized for peak usage but rather for average usage. At the same time, each workload can grow to consume free resources from the shared pool that belong to other partitions on the system. This situation allows the packing of more partitions (workloads) on a single system, producing a higher consolidation ratio or higher density on the deployed system. A higher consolidation ratio is a key metric to achieve in the data center, as it helps to reduce the total cost of ownership (TCO).

Let us look at a list of key attributes that require consideration when deploying workloads on a shared resource model (virtualization):

- ▶ Levels of variation between average and peak usage of workloads:
 - A large difference between average and peak usage
 - A small difference between average and peak usage
- ▶ Workloads and their peak duration, frequency, and estimate when they potentially peak. Select workloads that peak at different times (non-overlapping).
- ▶ Workload Service Level Agreement SLA requirements (latency requirements and their tolerance levels).
- ▶ Ratio of active to inactive (mostly idle) partitions on a system.
- ▶ Provisioning and de-provisioning frequency.
- ▶ IBM PowerVM has a richer set of technology options than virtualization on other platforms. It supports dedicated, shared, and a mix of dedicated and shared resource models for each of the system resources, such as processor cores, memory, and I/O:
 - Shared LPAR: Capped versus uncapped.
 - Shared LPAR: Resources over-commit levels to meet the peak usage (the ratio of virtual processors to physical processor entitled capacity).

- Shared LPAR: Weight selection to assign a level of priority to get uncapped capacity (excess cycles to address the peak usage).
- Shared LPAR: Multiple shared pools to address software licensing costs, which prevents a set of partitions from exceeding its capacity consumption.
- Active Memory Sharing: The size of a shared pool is based on active workload memory consumption:
 - Inactive workload memory is used for active workloads, which reduces the memory capacity of the pool.
 - The Active Memory Deduplication option can reduce memory capacity further.
 - AIX file system cache memory is loaned to address memory demands that lead to memory savings.
 - Workload load variation changes active memory consumption, which leads to opportunity for sharing.
- Active Memory Sharing: A shared pool size determines the levels of memory over-commit. Starts without over-commit and is based on workload consumption that reduces the pool.
- Active Memory Expansion: AIX working set memory is compressed.
- Active Memory Sharing and Active Memory Expansion can be deployed on the same workload.
- Active Memory Sharing: VIOS sizing is critical for CPU and memory.
- Virtual Ethernet: An inter-partition communication VLANs option that is used for higher network performance.
- Shared Ethernet versus host Ethernet.
- Virtual disk I/O: Virtual small computer system interface (vSCSI), N_Port ID Virtualization (NPIV), file-backed storage, and storage pool.
- Dynamic resource movement (DLPAR) to adopt to growth.

3.2 Power Systems virtualization with PowerVM

PowerVM hypervisor and the AIX operating system (AIX V6.1 TL 7, and AIX V7.1 TL 1 and later versions) on POWER8 processor-based systems implement enhanced affinity in a number of areas to achieve optimized performance for workloads that are running in a virtualized shared processor logical partition (SPLPAR) environment. By using the preferred practices that are described in this guide, customers can attain optimum application performance in a shared resource environment. This guide covers preferred practices in the context of POWER8 processor-based systems, so this section can be used as an addendum to other PowerVM preferred practice documents.

3.2.1 Virtual processors

A virtual processor is a unit of a virtual processor resource that is allocated to a partition or virtual machine. PowerVM hypervisor can map a whole physical processor core, or it can create a time slice of a physical processor core.

The PowerVM hypervisor creates time slices of Micro-Partitioning on physical CPUs by dispatching and undischatching the various virtual processors for the partitions that are running in the shared pool.

If a partition has multiple virtual processors, they might be scheduled to run simultaneously on the physical processor cores.

Partition entitlement is the guaranteed resource that is available to a partition. A partition that is defined as capped can consume only the processors units that are explicitly assigned as its entitled capacity. An uncapped partition can consume more than its entitlement, but is limited by many factors:

- ▶ Uncapped partitions can exceed their entitlement if there is unused capacity in the shared pool, dedicated partitions that share their physical processor cores while active or inactive, unassigned physical processors, and Capacity on Demand (CoD) utility processors.
- ▶ If the partition is assigned to a virtual shared processor pool, the capacity for all of the partitions in the virtual shared processor pool might be limited.
- ▶ The number of virtual processors in an uncapped partition is throttled depending on how much CPU it can consume. For example:
 - An uncapped partition with one virtual CPU can consume only one physical processor core of CPU resources under any circumstances.
 - An uncapped partition with four virtual CPUs can consume only four physical processor cores of CPU.
- ▶ Virtual processors can be added or removed from a partition by using HMC actions.

Sizing and configuring virtual processors

The number of virtual processors in each LPAR in the system ought not to *exceed* the number of cores available in the system (central electronic complex (CEC)/framework). Or, if the partition is defined to run in a specific virtual shared processor pool, the number of virtual processors ought not to exceed the maximum that is defined for the specific virtual shared processor pool. Having more virtual processors that are configured than can be running at a single point in time does not provide any additional performance benefit and can cause more context switches of the virtual processors, which reduces performance.

If there are sustained periods during which there is sufficient demand for all the shared processing resources in the system or a virtual shared processor pool, it is prudent to configure the number of virtual processors to match the capacity of the system or virtual shared processor pool.

A single virtual processor can consume a whole physical core under two conditions:

1. SPLPAR has an entitlement of 1.0 or more processors.
2. The partition is uncapped and there is idle capacity in the system.

Therefore, there is no need to configure more than one virtual processor to get one physical core.

For example, a shared pool is configured with 16 physical cores. Four SPLPARs are configured, each with entitlement 4.0 cores. To configure virtual processors, consider the sustained peak demand capacity of the workload. If two of the four SPLPARs were to peak to use 16 cores (the maximum available in the pool), then those two SPLPARs need 16 virtual CPUs. If the other two SPLPARs peak only up to eight cores, those two SPLPARs are configured with eight virtual CPUs.

Entitlement versus virtual processors

Entitlement is the capacity that an SPLPAR is ensured to get its share from the shared pool. Uncapped mode allows a partition to receive excess cycles when there are free (unused) cycles in the system.

Entitlement also determines the number of SPLPARs that can be configured for a shared processor pool. The sum of the entitlement of all the SPLPARs cannot exceed the number of physical cores that are configured in a shared pool.

For example, a shared pool has eight cores and 16 SPLPARs are created, each with 0.1 core entitlement and one virtual CPU. In our example, we configured the partitions with 0.1 core entitlement because these partitions are not running that frequently. In this example, the sum of the entitlement of all the 16 SPLPARs comes to 1.6 cores. The rest of the 6.4 cores and any unused cycles from the 1.6 entitlement can be dispatched as uncapped cycles.

At the same time, keeping entitlement low when there is capacity in the shared pool is not always a preferred practice. Unless the partitions are frequently idle, or there is a plan to add more partitions, the preferred practice is that the sum of the entitlement of all the SPLPARs configured is close to the capacity in the shared pool. Entitlement cycles are guaranteed, so when a partition is using its entitlement cycles, the partition is not preempted; however, a partition can be preempted when it is dispatched to use excess cycles. Following this preferred practice allows the hypervisor to optimize the affinity of the partition's memory and processor cores and also reduces unnecessary preemptions of the virtual processors.

Entitlement also affects the choice of memory and processors that are assigned by the hypervisor for the partition. The hypervisor uses the entitlement value as a guide to the amount of CPU that a partition consumes. If the entitlement is undersized, performance can be adversely affected, for example, if there are four cores per processor chip and two partitions are consistently consuming about 3.5 processors of CPU capacity. If the partitions are undersized with four virtual processors and 2.0 entitlement (that is, entitlement is set below normal usage levels), the hypervisor may allocate both of the partitions on the same processor chip, as the entitlement of 2.0 allows two partitions to fit into a 4-core processor chip. If both partitions consistently consume 3.5 processors worth of capacity, the hypervisor is forced to dispatch some of the virtual processors on chips that do not contain memory that is associated with the partitions. If the partitions were configured with an entitled capacity of 3.5 instead of 2.0, the hypervisor places each partition on its own processor chip to ensure that there is sufficient processor capacity for each partition. This improves the locality, resulting in better performance.

Matching the entitlement of an LPAR close to its average usage for better performance

The aggregate entitlement (minimum or wanted processor) capacity of all LPARs in a system is a factor in the number of LPARs that can be allocated. The minimum entitlement is what is needed to boot the LPARs; however, the wanted entitlement is what an LPAR gets if there are enough resources available in the system. The preferred practice for LPAR entitlement is to match the entitlement capacity to average usage and let the peak be addressed by more uncapped capacity.

When to add more virtual processors

When there is sustained need for a shared LPAR to use more resources in the system in uncapped mode, increase the virtual processors.

How to estimate the number of virtual processors per uncapped shared LPAR

The first step is to monitor the usage of each partition and for any partition where the average utilization is about 100%, and then add one virtual processor, that is, use the capacity of the configured virtual processors before you add more. Additional virtual processors run concurrently if there are enough free processor cores available in the shared pool.

If the peak usage is below the 50% mark, then there is no need for more virtual processors. In this case, look at the ratio of virtual processors to configured entitlement and if the ratio is greater than 1, then consider reducing the ratio. If there are too many virtual processors that are configured, AIX can *fold* those virtual processors so that the workload can run on fewer virtual processors to optimize virtual processor performance.

For example, if an SPLPAR is given a CPU entitlement of 2.0 cores and four virtual processors in an uncapped mode, then the hypervisor can dispatch the virtual processors to four physical cores concurrently if there are free cores available in the system. The SPLPAR uses unused cores and the applications can scale up to four cores. However, if the system does not have free cores, then the hypervisor dispatches four virtual processors on two cores so that the concurrency is limited to two cores. In this situation, each virtual processor is dispatched for a reduced time slice as two cores are shared across four virtual processors. This situation can impact performance, so AIX operating system processor folding support might be able to reduce to number of virtual processors that are dispatched so that only two or three virtual processors are dispatched across the two physical cores.

Virtual processor management: Processor folding

The AIX operating system monitors the usage of each virtual processor and the aggregate usage of a shared processor partition to manage the use of virtual processors that are actively engaged by a partition. This management task is carried out by using a threshold value that is used to increase, decrease, or hold steady the number of engaged virtual processors for the partition. The threshold is observable as the `vpm_fold_threshold` output by the `schedo` command.

When the aggregate usage goes below the threshold, AIX starts folding down the virtual CPUs so that fewer virtual CPUs are dispatched. This action has the benefit of virtual CPUs running longer before being preempted, which helps improve performance. If a virtual CPU gets a shorter dispatch time slice, then more workloads are cut into time slices on the processor core, which can cause higher cache misses. If the aggregate usage of an SPLPAR goes above the threshold, AIX starts unfolding virtual CPUs so that additional processor capacity can be given to the SPLPAR. AIX cannot engage more virtual processors than are currently defined for the partition. Virtual processor management dynamically adopts the number of virtual processors to match the load on an SPLPAR. This threshold (`vpm_fold_threshold`) represents the SMT thread usage starting with AIX V6.1 TL6. In versions before AIX V6.1 TL6, `vpm_fold_threshold` represents the core utilization. The threshold is processor type specific.

When folding increases the number of virtual processors that are engaged, and there are free cores available in the shared processor pool, then unfolding another virtual processor results in the partition getting another core along with its associated caches. Now, the partition can run on two primary threads of two cores, instead of two threads (primary and secondary) on the same core. A workload that is running on two primary threads of two cores can achieve higher performance if there is less sharing of data than the workload that is running on primary and secondary threads of the same core. The AIX virtual processor management default policy aims at using the primary thread of each virtual processor first; therefore, it unfolds the next virtual processor without using the SMT threads of the first virtual processor. After it unfolds all the virtual processors and consumes the primary thread of all the virtual processors, it starts using the secondary and tertiary threads of the virtual processors.

For more information, see the following website:

<http://www-01.ibm.com/common/ssi/cgi-bin/ssialias?infotype=SA&subtype=WH&htmlfid=P0W03049USEN#loaded>

Processor bindings in a shared LPAR

In AIX V6.1 TL 7, and in AIX V7.1 TL 1 and later versions, binding virtual processors is available to an application that is running in a shared LPAR. An application process can be bound to a virtual processor in a shared LPAR. In a shared LPAR, a virtual processor is dispatched by the PowerVM hypervisor. The PowerVM hypervisor maintains three levels of affinity for dispatching, such as core, chip, and node level. By maintaining affinity at the hypervisor level and in AIX, applications can achieve higher-level affinity through virtual processor bindings.

3.2.2 Page table sizes for LPARs

The hardware page table of an LPAR is sized based on the maximum memory size of an LPAR and not what is assigned to (or wanted for) the LPAR. There are some performance considerations if the maximum size is set higher than the wanted memory:

- ▶ A larger page table tends to help performance of the workload, as the hardware page table can hold more pages. This larger table reduces translation page faults. Therefore, if there is enough memory in the system and you want to improve translation page faults, set your max memory to a higher value than the LPAR wanted memory.
- ▶ On the downside, more memory is used for the hardware page table, which wastes memory and makes the table become sparse, which results in the following situations:
 - A dense page table tends to help with better cache affinity because of reloads.
 - Less memory that is consumed by the hypervisor for the hardware page table means that more memory is made available to the applications.
 - There is less page walk time as page tables are small.

3.2.3 Placing LPAR resources to attain higher memory affinity

POWER8 PowerVM optimizes the allocation of resources for both dedicated and shared partitions as each LPAR is activated. Correct planning of the LPAR configuration enhances the possibility of getting both CPU and memory in the same domain in relation to the topology of a system.

PowerVM hypervisor selects the required processor cores and memory that is configured for an LPAR from the system free resource pool. During this selection process, the hypervisor takes the topology of the system into consideration and allocates processor cores and memory where both resources are close. This situation ensures that the workload on an LPAR has lower latency in accessing its memory.

When you install a system, power on the partitions of highest importance first. By doing so, the partitions have first access to available memory and processing resources. After a partition is powered on, the server undergoes IPL, or the Dynamic Platform Optimizer is run, the processors and memory assignment are predetermined by the hypervisor, so that the order of activation is not important. On the HMC, there is an option to activate the current configuration, and when you use this option, there is no change in the current placement of the partition. Activating with a partition profile might change the current placement of the partition.

Partition powering on: Even though a partition is dependent on a VIOS, it is safe to power on the partition before the VIOS; the partition does not fully power on because of its dependency on the VIOS, but claims its memory and processing resources.

How to determine whether an LPAR is contained within a domain

From an AIX LPAR, run **lssrad** to display the number of domains across which an LPAR is spread.

The **lssrad** syntax is:

```
lssrad -av
```

If all the cores and memory are in a single domain, you receive the following output with only one entry under REF1:

REF1	SRAD	MEM	CPU
0	0	31806.31	0-31
	1	31553.75	32-63

REF1 represents a domain, and domains vary by platform. SRAD always references a chip. However, **lssrad** does not report the actual physical domain or chip location of the partition: it is a relative value whose purpose is to inform you whether the resources of the partition are within the same domain or chip. The output of this **lssrad** example indicates that the LPAR is allocated with 16 cores from two chips within the same domain. The **lssrad** command output was taken from an SMT4 platform, so CPU 0-31 represents eight cores.

When all the resources are free (an initial machine state or restart of the CEC), the PowerVM allocates memory and cores as optimally as possible. At partition boot time, PowerVM is aware of all of the LPAR configurations, so placement of processors and memory are made regardless of the order of activation of the LPARs.

However, after the initial configuration, the setup might not stay static. Numerous operations take place, such as:

- ▶ Reconfiguration of existing LPARs with new profiles
- ▶ Reactivating existing LPARs and replacing them with new LPARs
- ▶ Adding and removing resources to LPARs dynamically (DLPAR operations)

Any of these changes might result in memory fragmentation, causing LPARs to be spread across multiple domains. There are ways to minimize or even eliminate the spread. For the first two operations, the spread can be minimized by releasing the resources that are assigned to the deactivated LPARs.

Resources of an LPAR can be released by running the following commands:

- ▶ **chhwres -r mem -m <system_name> -o r -q <num_of_Mbytes> --id <lp_id>**
- ▶ **chhwres -r proc -m <system_name> -o r --procunits <number> --id <lp_id>**

The first command frees the memory, and the second command frees cores.

Fragmentation because of frequent movement of memory or processor cores between partitions is avoidable with correct planning. DLPAR actions can be done in a controlled way so that the performance impact of resource addition or deletion is minimal. Planning for growth helps alleviate the fragmentation that is caused by DLPAR operations. Knowing the LPARs that must grow or shrink dynamically, and placing them with LPARs that can tolerate nodal crossing latency (less critical LPARs), is one approach to handling the changes of critical LPARs dynamically. In such a configuration, when growth is needed for the critical LPAR, the resources that are assigned to the non-critical LPAR can be reduced so that the critical LPAR can grow. Another method of managing fragmentation is to monitor the affinity score of the system or important partitions and use the Dynamic Platform Optimizer to reoptimize the memory and processor that is assigned to the partitions.

Affinity groups

PowerVM firmware has support for affinity groups that can be used to group multiple LPARs within the same processor chip, processor socket, or drawer. When using affinity groups, it is important to understand the physical configuration of the processor cores and memory that is contained within the processor chips, processor sockets, and drawers, such that the size of the affinity group does not exceed the capacity of the wanted domain. For example, if the system has 4 cores and 64 GB of memory per processor chip, and you want to contain the partitions to a single processor chip, ensure that the size of the affinity group does not exceed four cores and 64 GB of memory. When calculating the memory size of an affinity group and what is available on a chip, the computed value must account for the memory that is used by the hypervisor for I/O space and for objects that are associated with the partition, such as the hardware page table.

Note: As a general rule, the size of the affinity group wanted memory should allocate only 90 - 95% of the physical memory that is contained in a domain. If the affinity group is larger than the wanted domain, the hypervisor cannot contain the affinity group within a single domain.

This affinity group feature can be used in multiple situations:

- ▶ LPARs that are dependent or related, such as server and client, and application server and database server, can be grouped so they are in the same book.
- ▶ Affinity groups can be created that are large enough such that they force the assignment of LPARs to be in different books. For example, if you have a two-socket system and the total resources (memory and processor cores) assigned to the two groups exceeds the capacity of a single socket, these two groups are forced to be in separate sockets.

If a pair of LPARs is created with the intent of one being a failover to another partition, and one partition fails, the other partition (which is placed in the same node, if both are in the same affinity group) uses all of the resources that were freed up from the failed LPAR.

The following HMC CLI command adds or removes a partition from an affinity group:

```
chsyscfg -r prof -m <system_name> -i name=<profile_name>  
lpar_name=<partition_name>,affinity_group_id=<group_id>
```

group_id is a number 1 - 255 (255 groups can be defined), and **affinity_group_id=none** removes a partition from the group.

When the hypervisor places resources at frame restart, it first places all the LPARs in group 255, then the LPARs in group 254, and so on. Place the most important partitions regarding affinity in the highest configured group.

PowerVM resource consumption for capacity planning considerations

PowerVM hypervisor consumes a portion of the memory resources in the system. During your planning stage, consider the layout of LPARs. Factors that affect the amount of memory that is consumed are the size of the hardware page tables in the partitions, the number of I/O devices, hypervisor memory mirroring, and other factors. Use the *IBM System Planning Tool* to estimate the amount of memory that is reserved by the hypervisor. This tool is found at the following website:

<http://www.ibm.com/systems/support/tools/systemplanningtool/>

Licensing resources and Capacity Upgrade on Demand

Some Power Systems support Capacity Upgrade on Demand (CUoD) so that customers can license capacity on demand as business needs for compute capacity grows. Therefore, a Power Systems server might not have usage of all of the resources that are installed, which poses a challenge to allocate both cores and memory from a local domain. PowerVM correlates customer configurations and licensed resources to allocated cores and memory from the local domain to each of the LPARs. For systems with unlicensed memory, the licensing is governed on a quantity of memory basis and not on a physical DIMM basis. Therefore, any installed memory can be used to optimize the affinity of partitions. For systems with unlicensed processors, during a CEC restart, Dynamic Platform Optimizer (see 3.2.5, “Optimizing resource placement: Dynamic Platform Optimizer” on page 67), and some DLPAR requests, the hypervisor can readjust which processors are licensed to optimize the affinity of the partitions.

For more information about this topic, see 3.4, “Related publications” on page 68.

3.2.4 Active memory expansion

Active memory expansion (AME) is a capability that is supported on POWER8 processor-based systems that employs memory compression technology to expand the effective memory capacity of an LPAR. The operating system identifies the least frequently used memory pages and compresses them. The result is that more memory capacity within the LPAR is available to sustain more load, or the ability to remove memory from the LPAR to be used to deploy more LPARs. The POWER8 processor provides enhanced support of AME with the inclusion of on-chip accelerators onto which the work of compression and decompression is offloaded.

AME is deployed by first using the **amepat** tool to model the projected expansion factor and CPU usage of a workload. This modeling looks at the compressibility of the data, the memory reference patterns, and current CPU usage of the workload. AME can then be enabled for the LPAR by setting the expansion factor. The operating system then reports the physical memory that is available to applications as *actual memory* times the *expansion factor*. Then, transparently, the operating system locates and compresses cold pages to maintain the appearance of expanded memory.

Applications do not need to change, and they are not aware that AME is active. However, not all applications or workloads have suitable characteristics for AME. Here is a partial list of guidelines for the workload characteristics that can be a good fit for AME:

- ▶ The memory footprint is dominated by application working storage (such as heap, stack, and shared memory).
- ▶ Workload data is compressible.
- ▶ Memory access patterns are concentrated in a subset of the overall memory footprint.
- ▶ Workload performance is acceptable without the use of larger page sizes, such as 64 KB pages. AME disables the usage of large pages and uses only 4 KB pages.
- ▶ The average CPU usage of the workload is below 60%.
- ▶ Users of the application and workload are relatively insensitive to response time increases.

For more information about AME usage, see *Active Memory Expansion: Overview and Usage Guide*, found at:

<http://www-01.ibm.com/common/ssi/cgi-bin/ssialias?infotype=SA&subtype=WH&htmlfid=POW03049USEN#loaded>

3.2.5 Optimizing resource placement: Dynamic Platform Optimizer

The Dynamic Platform Optimizer feature automates the manual steps to improve resource placement. For more information, go to the following website and select the Doc-type **Word document** *P7 Virtualization Best Practice*. An update to this document from the POWER8 perspective is planned.

<https://www.ibm.com/developerworks/community/wikis/home?lang=en#!/wiki/61ad9cf2-c6a3-4d2c-b779-61ff0266d32a/page/64c8d6ed-6421-47b5-a1a7-d798e53e7d9a/attachments>

Note: This document is intended to address POWER8 processor technology-based PowerVM preferred practices to attain the best LPAR performance. Use this document with other PowerVM documents.

3.2.6 Partition compatibility mode

When partitions are created, the processor compatibility mode can be specified. On POWER8 processor-based systems, partitions can run in POWER6, POWER6+, POWER7, POWER8, or default compatibility modes. Different modes support different SMT levels and hardware instructions, based on the hardware model that is chosen.

For example, to migrate to a POWER6 processor-based system, the partition must be selected to run in POWER6 mode. In addition to allowing migration, the partition, even on a POWER8 processor-based system, runs at most in SMT2 mode, and only instructions that are available on a POWER6 processor-based system can be used by the partition. SMT2 mode is used when POWER6 or POWER6+ is selected, SMT4 mode is used for POWER7 processor-based systems, and SMT8 mode is used for POWER8 processor-based systems. AIX also supports the `smtctl` command, which can reduce the SMT level of the partition if that is wanted. A value of `default` means that the partition runs in whatever mode was available when the partition was activated. The selection of the default prevents the partition from migrating to earlier generations of POWER processors.

3.3 Introduction to KVM Virtualization

Starting with POWER8 processor-based systems, IBM introduced a new family of servers that are called *scale-out systems*. Scale-out systems are targeted at scale-out workloads and support a complete stack of open software, ranging from the hypervisor to cloud management. Examples of POWER8 scale-out systems are the single-socket IBM Power System S812L and the two-socket IBM Power System S822L systems. Although scale-out systems can be virtualized with the PowerVM hypervisor, a second virtualization mechanism, which is known as the kernel-based virtual machine (KVM), is supported on these systems.

KVM is a Linux based complete virtualization solution, which consists of at least the following components:

- ▶ A loadable kernel module that is called `kvm.ko` that provides the core virtualization infrastructure
- ▶ A loadable kernel module that is called `kvm-<arch>.ko`, which is a processor or architecture-specific module, such as `kvm-intel.ko`
- ▶ A **qemu** (quick emulator) command, which emulates CPUs and provides a set of device models

IBM PowerKVM is the port of KVM for hardware virtualization of Power Systems and provides full virtualization on POWER8 scale-out systems. The architecture-specific model for Power Systems is called `kvm-hv.ko`. PowerKVM also includes the virtualization packages such as **libvirt**, which provide the tools, runtime libraries, and a daemon for managing platform virtualization. There is both a CLI interface (the **virsh** command, which is part of the `libvirt-client` package) and a web-based interface, *Kimchi*, for managing virtualization, including starting and stopping virtual machines.

In KVM terminology, a virtual machine is more commonly referred to as the *guest*. The hypervisor is often referred to as running on the *host* machine. The hypervisor consists of the operating system (including the virtualization modules) and firmware that directly runs on the hardware and supports running guests.

Note: On the PowerVM hypervisor, a virtual machine or guest is called a logical partition (LPAR).

PowerKVM V2.1, released June 2014, is the first PowerKVM release. Only Linux distributions (such as RHEL, Ubuntu, SLES, or Fedora) are supported as guest OSes by PowerKVM.

Unlike PowerVM, there is no need for a Hardware Management Console (HMC) to manage PowerKVM. Instead, the industry-standard Intelligent Platform Management Interface (IPMI) interface is used to manage the host. On IBM Power Systems, the IPMI server runs on the service controller and not on the host. Therefore, commands directed to the IPMI server must use the service processor IP address in the command line.

For more information about the KVM technology on IBM systems and the various virtualization support tools (such as `qemu`, `libvirt`, *Kimchi*, IPMI, and so on), see *IBM PowerKVM Configuration and Use*, SG24-8231.

Note: *IBM PowerKVM Configuration and Use*, SG24-8231 can help in configuring PowerKVM and the guest OSes optimally.

3.4 Related publications

The publications that are listed in this section are considered suitable for a more detailed description of the topics that are covered in this chapter:

- ▶ *Active Memory Expansion: Overview and Usage Guide*, found at:
<ftp://ftp.software.ibm.com/common/ssi/sa/wh/n/pow03037usen/POW03037USEN.PDF>
- ▶ *IBM PowerVM Active Memory Sharing Performance*, found at:
http://public.dhe.ibm.com/common/ssi/rep_wh/n/POW03017USEN/POW03017USEN.PDF
- ▶ *IBM PowerVM Virtualization Introduction and Configuration*, SG24-7940
- ▶ *IBM PowerVM Virtualization Managing and Monitoring*, SG24-7590
- ▶ *POWER7 Virtualization Best Practice Guide*, found at:
https://www.ibm.com/developerworks/wikis/download/attachments/53871915/P7_virtualization_bestpractice.doc?version=1
- ▶ *PowerVM Migration from Physical to Virtual Storage*, SG24-7825

- ▶ *Virtual I/O (VIO) and virtualization*, found at:
<https://www.ibm.com/developerworks/community/wikis/home?lang=en#!/wiki/Power%20Systems/page/Virtual%20I%20and%20virtualization>
- ▶ *Virtualization Best Practice*, found at:
<https://www.ibm.com/developerworks/community/wikis/home?lang=en#!/wiki/Power%20Systems/page/Virtualization%20best%20practices>



IBM AIX

This chapter describes the optimization and tuning of POWER8 and other Power Systems processor-based servers running the AIX operating system. It covers the following topics:

- ▶ 4.1, “Introduction” on page 72
- ▶ 4.2, “Using Power Architecture features with AIX” on page 72
- ▶ 4.3, “AIX operating system-specific optimizations” on page 95
- ▶ 4.4, “AIX preferred practices” on page 105
- ▶ 4.5, “Related publications” on page 107

4.1 Introduction

AIX is regarded as a good choice for building an IT infrastructure on IBM systems that are designed with Power Architecture technology. With its proven scalability, advanced virtualization, security, manageability, and reliability features, it is an enterprise-class OS. In particular, AIX is the only operating system that uses decades of IBM technology innovation that is designed to provide the highest level of performance and reliability of any UNIX operating system. AIX has demonstrated leadership performance on various system benchmarks.

The performance benefits of AIX include:

- ▶ Deep integration with the Power Architecture (core design with the Power Architecture)
- ▶ Autonomic optimization
 - A single OS image configures itself to support any POWER processor.
 - Dynamic workload optimization.
- ▶ Performs on a wide variety of system configurations
 - Scales from 0.05 to 256 cores (up to 1024 logical processors).
 - Horizontal (native clustering) and vertical scaling.
- ▶ Strong virtualization support for PowerVM virtualization
 - Tight integration with PowerVM.
 - Enabler for virtual I/O (VIO).
- ▶ Full set of integrated performance tools.

AIX V6.1 and AIX V7.1 run on and maximize the capabilities of systems based on the POWER8 processor-based system, which is the latest generation of POWER processor-based systems, while supporting POWER4, POWER5, POWER6, and POWER7 (including POWER7+) processor-based systems.

For more information about this topic, see 4.5, “Related publications” on page 107.

4.2 Using Power Architecture features with AIX

Various significant features of the Power Architecture with POWER7 and POWER8 extensions in an AIX environment are described in this section.

4.2.1 Multi-core and multi-thread

Operating system enablement usage of multi-core and multi-thread technology varies by operating system and release. Table 4-1 on page 73 shows the maximum processor cores and threads for a (single) logical partition running AIX.

Table 4-1 Multi-thread per core features by single LPAR scaling

Single LPAR scaling	AIX release
32-core/32-thread	5.3/6.1/7.1
64-core/128-thread (SMT2)	5.3/6.1/7.1
64-core/256-thread (SMT4)	6.1(TL4)/7.1
256-core/1024-thread (SMT4) or 128-core/1024-thread (SMT8)	7.1

Information about multi-thread per core features by POWER generation is available in Table 2-1 on page 28.

Using multi-core and multi-thread features is a challenging prospect. In addition to the overview material in this section, the following specific scaling topics are described:

- ▶ Malloc tuning (see 4.3.1, “Malloc” on page 95)
- ▶ Pthread tuning (see 4.3.2, “Pthread tunables” on page 97)

For more information about this topic, from the processor and OS perspectives, see:

- ▶ 2.2.1, “Multi-core and multi-thread” on page 28 (*processor*)
- ▶ 5.2.1, “Multi-core and multi-thread” on page 112 (*IBM i*)
- ▶ 6.2.1, “Multi-core and multi-thread” on page 119 (*Linux*)

For more information about this topic, see 4.5, “Related publications” on page 107.

Simultaneous multithreading

Simultaneous Multithreading (SMT) is a feature of the Power Architecture and is described in “Simultaneous multithreading” on page 29. SMT is supported in AIX, as described in *Simultaneous multithreading*.¹

AIX provides options to allow SMT customization. The `smtctl` command allows the SMT feature to be enabled, disabled, or capped (SMT2 versus SMT4 mode on POWER7 processor-based systems and SMT2 or SMT4 modes versus SMT8 on POWER8 processor-based systems). The partition-wide tuning option, `smtctl`, changes the SMT mode of all processor cores in the partition. It is built on the AIX dynamic reconfiguration (AIX DR) framework to allow hardware threads (logical processors) to be added and removed in a running partition. Because of this option’s global nature, it is normally set by system administrators. Most AIX systems (commercial) use the default SMT settings enabled (that is, SMT2 mode on POWER5 and POWER6 processor-based systems, and SMT4 mode on POWER7 and POWER8 processor-based systems).

When SMT is enabled (SMT2, SMT4, or SMT8 mode), the AIX kernel takes advantage of the platform feature to change SMT modes dynamically. These mode switches are done based on partition load (the number of running or waiting to run software threads) to choose the optimal SMT mode for the CPUs in the partition. The mode switching policies optimize overall workload throughput, but do not attempt to optimize individual software threads.

For more information about the topic of SMT, from the processor and OS perspectives, see:

- ▶ “Simultaneous multithreading” on page 29 (*processor*)
- ▶ “Simultaneous multithreading” on page 112 (*IBM i*)
- ▶ “Simultaneous multithreading” on page 119 (*Linux*)

¹ *Simultaneous multithreading*, found at:

<http://www-01.ibm.com/support/knowledgecenter/api/redirect/aix/v7r1/topic/com.ibm.aix.genprogc/doc/genprogc/smt.htm>

Simultaneous multithreading priorities

Simultaneous multithreading (SMT) priorities in the Power hardware are introduced in “Simultaneous multithreading priorities” on page 30.

AIX kernel usage of simultaneous multithreading thread priority and effects

The AIX kernel is optimized to take advantage of SMT thread priority by lowering the SMT thread priority in select code paths, such as when spinning in the wait process. When the kernel modifies the SMT thread priority and execution is returned to a process-thread, the kernel sets the SMT thread priority back to Medium or the level that is specified by the process-thread by using an AIX system call that modified the SMT thread priority (see “Application programming interfaces” on page 74).

Where to use

SMT thread priority can be used to improve the performance of a workload by lowering the SMT thread priority that is being used on an SMT thread that is running a particular process-thread in the following situations:

- ▶ The thread is waiting on a lock.
- ▶ The thread is waiting on an event, such as the completion of an IO event.

Alternatively, process-threads that are performance-sensitive can maximize their performance by ensuring that the SMT thread priority level is set to an elevated level.

Application programming interfaces

There are three ways to set the SMT priority when it is running on POWER processors:^{2, 3}

1. Modify the SMT priority directly by using the PPR register.⁴
2. Modify the SMT priority by using a special no-ops.⁵
3. Use the AIX `thread_set_smt_priority` system call.⁶

For more information about this topic, see Table 2-2 on page 30.

For more information about the topic of SMT priorities, from the processor and OS perspectives, see:

- ▶ “Simultaneous multithreading priorities” on page 30 (*processor*)
- ▶ “Simultaneous multithreading priorities” on page 120 (*Linux*)

Affinitization and binding

Affinity performance effects are explained in “The POWER8 processor and affinity performance effects” on page 16. Establishing good affinity is accomplished by understanding the placement of a partition on the underlying cores and memory of a Power Systems server, and then by using operating system facilities to bind application threads to run on specific hardware threads or cores.

² Power ISA Version 2.07, found at <https://www.power.org/documentation/power-isa-v-2-07b/>

³ `thread_set_smt_priority` or `thread_read_smt_priority` System Call, found at: http://www-01.ibm.com/support/knowledgecenter/api/redirect/aix/v7r1/index.jsp?topic=/com.ibm.aix.kerneltechref/doc/ktechrf1/thread_set_smt_priority.htm

⁴ Power ISA Version 2.07, found at: <https://www.power.org/documentation/power-isa-v-2-07b/>

⁵ Ibid

⁶ `thread_set_smt_priority` or `thread_read_smt_priority` System Call, found at: http://www-01.ibm.com/support/knowledgecenter/api/redirect/aix/v7r1/index.jsp?topic=/com.ibm.aix.kerneltechref/doc/ktechrf1/thread_set_smt_priority.htm

Flexible simultaneous multithreading

On POWER7 and POWER7+ processors, there is a correlation between the hardware thread number (0 - 3) and the hardware resources within the processor. Matching the thread numbers to the number of active threads was required for optimum performance. For example, if only one thread was active, it was thread0; if two threads were active, they were thread0 and thread1. On the POWER8 processor, the same performance is obtained regardless of which thread is active. The processor balances resources according to the number of active threads. There is no need to match the thread numbers with the number of active tasks. Thus, when using the **bindprocessor** command or API, it is not necessary to bind the job to thread0 for optimal performance.

Affinity APIs

Most applications must be bound to logical processors to get a performance benefit from memory/cache affinity to prevent the AIX dispatcher from moving the application to processor cores in different SRADs while the application runs.

AIX provides bind-ids, Resource Sets (RSETs), and Scheduler Resource Allocation Domains (SRADs) for affinity tuning in the following ways:

- ▶ **Bindprocessor**: Provides affinity to a single hardware thread that is identified by bind-id. It does not provide topology.
- ▶ **RSET**: Provides affinity to a group of hardware threads and supports memory binding. It provides topology.
- ▶ **SRAD**: Provides affinity to a scheduler resource domain and supports memory binding. It provides topology.

The most likely way to obtain a benefit from memory affinity is to limit the application to running only on the processor cores that are contained in a single SRAD. You can accomplish this task with the help of RSET (commands/API) and SRAD (APIs). If the application just needs a single processor, then the **bindprocessor** command or the **bindprocessor()** function can be used. It can also be done with the resource set affinity commands (**rset**) and service applications. Often, affinity is provided as an administrator option that can be optionally enabled on large systems.

When the application requires more processor cores than are contained in a single SRAD, the performance benefit through memory affinity depends on the memory allocation and access patterns of the various threads in the application. Applications with threads that individually allocate and reference unique data areas can see improved performance.

Bind processor

Processor affinity is the probability of dispatching a thread to the logical processor that was previously running it. If a thread is interrupted and later redispached to the same logical processor, the processor's cache might still contain lines that belong to the thread. If the thread is dispatched to a different logical processor, it probably experiences a series of cache misses until its cache working set is retrieved from RAM or the other logical processor's cache. If a dispatchable thread must wait until the logical processor that it was previously running on is available, the thread might experience an even longer delay.

The highest possible degree of processor affinity is to bind a thread to a specific logical processor. Binding means that the thread is dispatched to that logical processor only, regardless of the availability of other logical processors.

The **bindprocessor** command and the **bindprocessor()** subroutine bind the thread (or threads) of a specified process to a particular logical processor. Explicit binding is inherited through **fork()** and **exec()** system calls. The **bindprocessor** command requires the process identifier of the process whose threads are to be bound or unbound, and the bind CPU identifier of the logical processor to be used. This bind-id is different from the logical processor number and does not have any topology that is associated with it. Bind-ids cannot be associated to a specific chip/core because they tend to change on every DR operation (such as an SMT mode change). For NUMA affinity, RSETs or SRADs should be used to restrict the application to a set of logical processors in the same core/SRAD and its local memory.

CPU binding is useful for CPU-intensive applications; however, it can sometimes be counter-productive for I/O-intensive applications.

RSETs

Every process and kernel thread can have an RSET attached to it. The CPUs on which a thread can be dispatched are controlled by a hierarchy of resource sets. RSETs are mandatory bindings and are accepted by the AIX kernel always. Also, RSETs can affect dynamic reconfiguration (DR) activities.

Resource sets

These resource sets are:

Thread effective RSET	Created by ra_attachrset() . Must be a subset (improper or proper) of “Other RSETs” on page 76.
Thread partition RSET	Used by WLM. Partition RSETs allow WLM to limit where a thread can run.
Process effective RSET	Created by ra_attachrset() , ra_exec() , and ra_fork() . Must be a subset (improper or proper) of the process partition RSET.
Process partition RSET	Used by WLM to limit where processes in a WLM class are allowed to run. Can also be created by root users that use the rs_setpartition() service.

Other RSETs

Another type of RSET is the exclusive RSET. Exclusive use processor resource sets (XRSETs) allow an installation to limit the usage of the processors in XRSETs; they are used only by work that is attached to those XRSETs. They can be created by running the **mkrset** command in the 'sysxrset' namespace.

RSET data types and operations

The public shipped header file **rset.h** contains declarations for the public RSET data types and function prototypes.

An RSET is an opaque data type. Applications allocate an RSET by calling **rs_alloc()**. Applications receive a *handle* to the RSET. The RSET handle (data type **rsethandle_t** in **sys/rset.h**) is then used in RSET APIs to manipulate or attach the RSET.

Summary of RSET commands

Here is a summary of the RSET commands:

- **lsrset**: Displays RSETs that are stored in the system registry or RSETS that are attached to a process. For example:

lsrset -av Displays all RSETS in the system registry.

lsrset -p 28026 Displays the effective RSET that is attached to PID 28026.

- **mkrset**: Makes a named RSET containing specific CPU and memory pools and places the RSET in the system registry. For example, **mkrset -c 6 10-12 test/lotsofcpus** creates an RSET named test/lotsofcpus that contains the specified CPUs.

- **rmrset**: Removes an RSET from the system registry. For example:

rmrset test/lotsofcpus

- **attachrset**: Attaches an RSET to a specified PID. The RSET can either be in the system registry, or CPUs or mem pools that are specified in the command. For example:

attachrset test/lotsofcpus 28026 Attaches an RSET in a register to a process.

attachrset -c 4-8 28026 Attaches an RSET with CPUs 4 - 8 to a process as an effective RSET.

attachrset -P -c 4-8 28026 Attaches an RSET with CPUs 4 - 8 to process as a partition rset.

- **detachrset**: Detaches an RSET from a specified PID. For example:

detachrset 28026 Detaches an effective RSET from a PID.

detachrset -P 20828 Detaches a partition RSET from a PID.

- **execrset**: Runs a specific program or command with a specified RSET. For example:

execrset sys/node.04.00000 -e test Runs a program test with an effective RSET from the system registry.

execrset -c 0-1 -e test2 Runs program test2 with an effective RSET that contains logical CPU IDs 0 and 1.

execrset -P -c 0-1 -e test3 Runs program test3 with a partition RSET that contains logical CPU IDs 0 and 1.

RSET manipulation and information services

This list contains only user space APIs. There are also similar kernel extension APIs. For example, **krs_alloc()** is the kernel extension equivalent to **rs_alloc()**.

rs_alloc() Allocates and initializes an RSET and returns an RSET handle to a caller.

rs_free() Frees a resource set. The input is an RSET handle.

rs_init() Initializes a previously allocated RSET. The initialization options are the same as for **rs_alloc()**.

rs_op() Performs one of a set of operations against one or two RSETS.

rs_getinfo() Get information about an RSET.

rs_getrad() Get resource allocation domain information from an input RSET.

rs_numrads() Returns the number of system resource allocation domains at the specified system detail level that have available or online resources.

rs_getpartition()	Gets a process's partition RSET.
rs_setpartition()	Sets a process's partition RSET.
rs_discardname()	
rs_getnameattr()	
rs_getnamedrset()	
rs_setnameattr()	
rs_registername()	These are services that are used to manage the RSET system registry. There are services to create, obtain, and delete RSETs in the registry.

Attachment services

Here are the RSET attachment services:

ra_attachrset()	A service to attach a work component to an RSET. The service uses the rstype_t and rsid_t parameters to identify the work component to attach to the input RSET (specified by an rsethandle_t).
ra_detachrset()	Detaches an RSET from the work unit that is specified by the rstype_t/rsid_t parameters.
ra_exec()	Runs a program that is attached to a specific work component. The service uses rstype_t and rsid_t to specify the work component. However, the only supported rstype_t is R_RSET . All of the various versions of exec() are supported.
ra_fork()	Forks a process that is attached to a specific work component. The service uses rstype_t and rsid_t to specify the work component. However, the only supported rstype_t is R_RSET .
ra_get_attachinfo()	The ra_attachrset() also allows RSETs to be attached to ranges of memory in a file or in a shared memory segment.
ra_free_attachinfo()	This service frees the memory that was allocated for the attachment information that was returned by ra_get_attachinfo() .
ra_getrset()	Retrieves the RSET attachment to a process or thread. The return code indicates where the returned RSET is attached.
ra_mmap() and ra_mmapv()	Maps a file or memory region into a process and attaches it to the resource set that is specified by the rstype_t and rsid_t parameters. A memory allocation policy similar to ra_attachrset() allows a caller to specify how memory is preferentially allocated when the area is accessed.
ra_shmget() and ra_shmgetv()	Gets a shared memory segment with an attachment to a resource set. The RSET is specified by the rstype_t and rsid_t parameters. A memory allocation policy similar to ra_attachrset() allows a caller to specify how memory is preferentially allocated when the area is accessed.

AIX Enhanced Affinity (SRADs)

AIX Enhanced Affinity is a collection of AIX internal system changes and API extensions to improve performance on POWER7 processor-based systems. Enhanced Affinity improves performance by increasing CPU and memory locality on POWER7 processor-based systems. Enhanced Affinity extends the AIX existing memory affinity support. AIX V6.1 technology level 6100-05 contains AIX Enhanced Affinity support.

Enhanced Affinity status is determined during system start and remains unchanged for the life of the system. A restart is required to change the Enhanced Affinity status. In AIX V6.1.0 technology level 6100-05, Enhanced Affinity is enabled by default on POWER7 processor-based systems. Enhanced Affinity is available only on POWER7 processor-based systems. Enhanced Affinity is disabled by default on POWER6 and earlier processor-based systems. A `vmo` command tunable (`enhanced_memory_affinity`) is available to disable Enhanced Affinity support on POWER7 processor-based systems.

Here are two concepts that are related to Enhanced Affinity:

- ▶ **SRAD:** SRAD is the collection of logical CPUs and physical memory resources that are close from a hardware affinity perspective. An AIX system (partition) can consist of one or more SRADs. An SRAD represents the same collection of system resources as an existing MCM. A specific SRAD in a partition is identified by a number. It is an `sradit_t` data type and is often referred to as an SRADID.
- ▶ **SRADID:** The numeric identifier of a specific SRAD. It is a short integer data type. An SRADID value is the index of the resource allocation domain at the R_SRADSDL system detail level in the system's resource set topology.

Power Systems before POWER7 processor-based systems provided only system topology information to dedicated CPU logical partitions. This setup limited the usefulness of RSET attachments for CPU and memory locality purposes to dedicated CPU partitions. POWER7 processor-based systems provide system topology information for shared CPU logical partitions (SPLPAR).

The `lssrad` command can be used to display the processor and memory resources at the SRAD and REF1 levels (where REF1 is the next higher-level affinity domain).

You can use the AIX Enhanced Affinity services to attach SRADs to threads and memory ranges so that the application preferentially identifies the logical CPUs or physical memory to use to run the application. AIX continues to support RSET attachments to identify resources for an application.

RSET versus SRADs

When you compare RSET with SRADIDs:

1. SRADIDs can be attached to threads, shared memory segments, memory map regions, and process memory subranges. SRADIDs may not be attached at the process level (R_PROCESS). SRADIDs may not be attached to files (R_FILDES).
2. SRADID attachments are considered advisory. There are no mandatory SRADID attachments. AIX may ignore advisory SRADID attachments.
3. Process and thread RSET attachments continue to be mandatory. The process and thread resource set hierarchy continues to be enforced. Memory RSET attachments (shared memory, file, and process subrange) continue to be advisory. This situation is unchanged from previous affinity support.

API support

SRADIDs can be attached to threads and memory by using the following functions:

- ▶ `ra_attach()` (new)
- ▶ `ra_fork()`
- ▶ `ra_exec()`
- ▶ `ra_mmap()` and `ra_mmapv()`
- ▶ `ra_shmget()` and `ra_shmgetv()`

SRADIDs can be detached from thread and memory by using the `sra_detach()` function (new).

For more information about the topic of affinitization and binding, from the processor and OS perspectives, see:

- ▶ “Affinitization and binding to hardware threads” on page 31 (*processor*)
- ▶ “Affinitization and binding” on page 121 (*Linux*)

Hybrid thread and core

AIX provides facilities to customize SMT characteristics of CPUs running within a partition. The features require some partition-wide CPU configuration options, so their use is limited to specific workloads.

Hybrid thread features

AIX provides some basic features that allow more control in SMT mode. With these features, specific software threads can be bound to hardware threads that are assigned to ST mode CPUs. This configuration allows for an asymmetric SMT configuration, where some CPUs are in high SMT mode, and others have SMT mode disabled. This configuration allows critical software threads within a workload to receive an ST performance boost, and allows the remaining threads to benefit from SMT mode. Typical reasons to take advantage of this hybrid mode are:

- ▶ Asymmetric workload, where the performance of one thread serializes an entire workload. For example, one master thread dispatches work to many subordinate threads.
- ▶ Software threads that are critical to a system administrator.

The ability to create hybrid SMT configurations is limited under current AIX releases and does require administrator or privileged configuration changes. CPUs that provide ST mode hardware threads must be placed into XRSETs. XRSETs contain logical CPUs that are segregated from the general kernel dispatching pool. Software threads must be explicitly bound to CPUs in an XRSET. The only way to create an XRSET is by running the `mkrset` command. All of the hardware threads for logical CPUs must be contained in the XRSET created RSET. To accomplish this task, run the following commands:

- | | |
|------------------------------------|--|
| lsrset -av | Displays the RSET topology. The system CPU topology is broken down into a hierarchy that has the form <code>sys/node.XX.YYYYY</code> . The largest XX value is the CPU (core) level. This command provides logical processor groups by core. |
| mkrset -c 4-7 sysxrset/set1 | Creates an XRSET <code>sysxrset/set1</code> containing logical CPUs 4 - 7. |

An XRSET alone can be used to ensure that only specific work uses a CPU set. There is also the ability to restrict work execution to primary threads in an XRSET. This ability is known as an *STRSET*. STRSETs allow software threads to use ST execution mode independently of the load on the other CPUs in the system. Work can be placed onto STRSETs by running the following commands:

execrset -S This command allows external programs to start and be bound to an exclusive RSET.

ra_attach(R_STRSET) This API allows a thread to be bound to an STRSET.

For more information about this topic, from the processor and OS perspectives, see:

- ▶ “Hybrid thread and core” on page 31 (*processor*)
- ▶ “Hybrid thread and core” on page 122 (*Linux*)

For more information about this topic, see 4.5, “Related publications” on page 107.

AIX folding

Folding is a key AIX feature on shared processor LPARs that can improve both system and partition performance. Folding is needed for supporting many partitions in a system. It is an integrated feature, requiring both hardware and PowerVM support. The AIX component that manages folding is the Virtual Processor Manager (VPM).

The basic concept of folding is to compress work to a smaller number of cores, based on CPU utilization, by folding the remaining cores. The unused cores are folded by VPM, and PowerVM does not schedule them for dispatch in the partition unless the operating system requests that the cores be unfolded (or woken up), for example, when the workload changes or when a timer interrupt needs to be fired on that core.

As an example, an LPAR might have 24 virtual cores (processors) assigned, but is consuming only a total of three physical processors across all of these virtual cores. Folding compresses (moves) all work to a smaller number of cores (three cores plus some extra cores to handle spikes in workload), allowing PowerVM to allocate the unused cores for use elsewhere on the system.

Folding generally improves LPAR and system performance by reducing context switching of cores between partitions across a system, thus reducing context switching of software threads across multiple cores in an LPAR. It improves overall affinity at both the LPAR and system levels.

VPM runs once per second and computes how many cores are kept unfolded based on the overall CPU utilization of the LPAR. On POWER8 processor-based systems, the folding algorithm has been enhanced to include the average load (or the average number of runnable software threads) as a factor in the computation.

Folding can be enabled and disabled by using the **schedo** command to adjust the value of the **vpm_fold_policy** tunable. To respond faster to spikes in workloads or on partitions with a high interrupt load, a second tunable, **vpm_xvcpus**, can also be used to increase the number of spare, unfolded CPUs. This can improve response time for workloads with steep utilization spikes, or on partitions with a high interrupt load, although this can result in higher core usage.

AIX V6.1 TL8 and AIX V7.1 TL2 introduced a new scaled throughput-based folding algorithm that can be enabled by using the **schedo** command to adjust the value of the **vpm_throughput_mode** tunable. The default folding algorithm favors single-threaded performance and overall LPAR throughput over core utilization. The new scaled throughput algorithm can favor reduced core utilization and higher core throughput, instead of overall LPAR throughput. The new algorithm applies both load and utilization data to make folding decisions. It can switch unfolded cores to SMT2, SMT4, or SMT8 modes when the workload increases, rather than unfolding more cores, as shown in Figure 4-1.

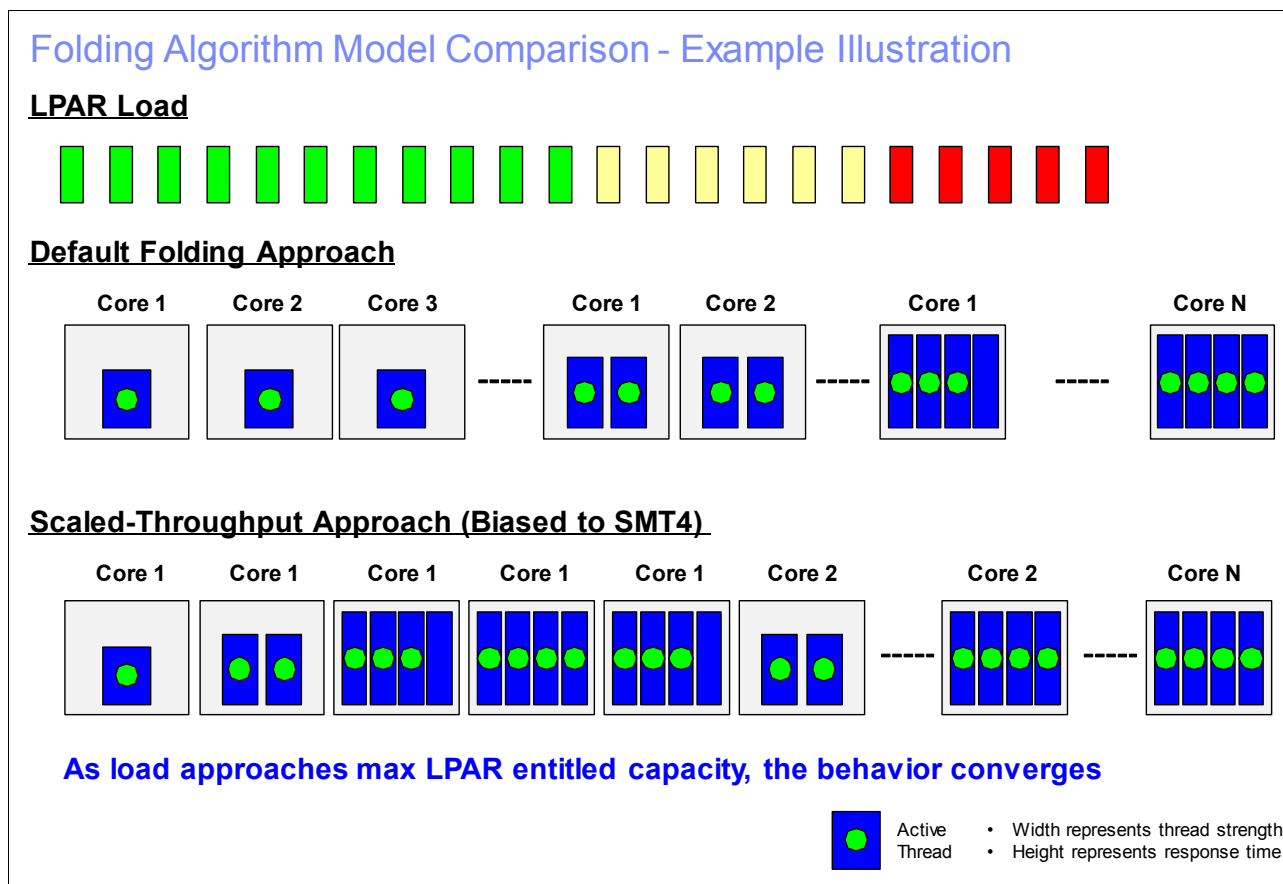


Figure 4-1 Folding algorithm model comparison

The degree of SMT mode (SMT2 or SMT4 (or SMT8 for POWER8 processor-based systems)) to favor reduced core utilization can be controlled by assigning the appropriate value to the **vpm_throughput_mode** tunable (2 for SMT2 mode, 4 for SMT4 mode, and 8 for SMT8 mode). When the **vpm_throughput_mode** tunable is set to a value of 1, the folding algorithm behaves like the legacy folding algorithm and favors single-threaded (ST mode) performance. However, unlike the legacy algorithm, which uses only utilization data, the new algorithm employs both load and utilization data to make folding decisions.

The default value of the **vpm_throughput_mode** tunable is 1 on POWER8 processor-based systems, and on POWER7 and earlier processor-based systems, the default value is zero (the legacy folding algorithm continues to be applicable).

If the `vpm_throughput_mode` is set to a value of 1 or greater, then the `vpm_throughput_core_threshold` tunable can also be set to specify the number of cores that must be unfolded before the `vpm_throughput_mode` parameter is accepted. One scheme that balances between performance and core utilization when enabling higher SMT modes is to set `vpm_throughput_core_threshold` to the integer value of the entitled capacity.

The scaled throughput algorithm can reduce overall core utilization at the frame level for certain workloads.

4.2.2 Multipage size support on AIX

AIX supports up to four different page sizes (see Table 4-2), but the actual page sizes that are supported by a particular system vary, based on processor chip type. The `pagesize -a` command on AIX determines all of the page sizes that are supported by AIX on a particular system.

Because the 64 KB page size is easy to use, and because it is expected that many applications perform better when they use the 64 KB page size rather than the 4 KB page size, AIX has rich support for the 64 KB page size. No system configuration changes are necessary to enable a system to use the 64 KB page size. On systems that support the 64 KB page size, the AIX kernel automatically configures the system to use it. Table 4-2 and Table 4-3 list the page size specifications for Power Systems.

Table 4-2 Page size support for Power HW and AIX configuration support⁷

Page size	Required hardware	Requires user configuration	Restricted
4 KB	ALL	No	No
64 KB	POWER5+ processor-based system or later	No	No
16 MB	POWER4 processor-based system or later	Yes	Yes
16 GB	POWER5+ processor-based system or later	Yes	Yes

Table 4-3 Supported segment page sizes on AIX⁸

Segment base page size	Supported page sizes	Minimum required hardware
4 KB	4 KB/64 KB	POWER6 processor-based system
64 KB	64 KB	POWER5+ processor-based system
16 MB	16 MB	POWER4 processor-based system
16 GB	16 GB	POWER5+ processor-based system

⁷ Multiple page size support, found at:
http://www-01.ibm.com/support/knowledgecenter/api/redirect/aix/v7r1/topic/com.ibm.aix.prftungd/doc/prftungd/multiple_page_size_support.htm

⁸ Ibid

Page sizes are an attribute of an individual segment. Earlier POWER processors supported only a single page size per segment. The system administrator or user had to choose the optimal page size for a specific application based on its memory footprint. The POWER5+ processor introduced the concept of mixed or multiple page sizes within a single segment: 4 KB and 64 KB. POWER7 and later processors support mixed page segment sizes of 4 KB, 64 KB, and 16 MB.

Starting with Version 6.1, AIX takes advantage of this new hardware capability on POWER6 and later processor-based systems to combine the conservative memory usage aspects of the 4 KB page size in sparsely referenced memory regions with the performance benefits of the 64 KB page size in densely referenced memory regions. AIX V6.1 takes advantage of this automatically, without user intervention, although it is disabled in segments that have an explicit page size that is selected by the user. This AIX feature is referred to as dynamic Variable Page Size Support (VPSS). Some applications might prefer to use a larger page size, even when a 64 KB region is not fully referenced. The page size promotion aggressiveness factor (PSPA) can be used to reduce the memory-referenced requirement, at which point a group of 4 KB pages is promoted to a 64 KB page size. The `vmo` command on AIX allows configuration of the VMM tunable parameters. The PSPA can be set for the whole system by using the `vmm_default_pspa vmo` tunable, or for a specific process by using the `vm_attr` system call.⁹

In addition to 4 KB and 64 KB page sizes, AIX supports 16 MB pages, also called *large pages*, and 16 GB pages, also called *huge pages*. These page sizes are intended for use only in high-performance environments, and AIX by default does not automatically configure a system to use these page sizes.

Use the `vmo` tunables `lgpg_regions` and `lgpg_size` to configure the number of 16 MB large pages on a system.

The following example allocates 1 GB of 16 MB large pages:

```
vmo -r -o lgpg_regions=64 -o lgpg_size=16777216
```

To use large pages, non-root users must have the `CAP_BYPASS_RAC_VMM` capability in AIX enabled. The system administrator can add this capability by running `chuser`:

```
chuser capabilities=CAP_BYPASS_RAC_VMM,CAP_PROPAGATE <user_id>
```

Huge pages must be configured by using the Hardware Management Console (HMC). To do so, complete the following steps:

1. On the managed system, click **Properties** → **Memory** → **Advanced Options** → **Show Details** to change the number of 16 GB pages.
2. Assign 16 GB huge pages to a partition by changing the partition profile.

The `vmo` tunable `vmm_mpsize_support` can be used to limit multiple page size support. The default value of 1 supports all four page sizes, but the tunable can be set to other values to configure which page sizes will to be supported.

Application support to use multisize pages on AIX¹⁰

As described in *Power Instruction Set Architecture Version 2.07*,¹¹ you can specify page sizes to use for four regions of a 32-bit or 64-bit process address space.

⁹ Ibid

¹⁰ Ibid

¹¹ *Power ISA Version 2.07*, found at <https://www.power.org/documentation/power-isa-v-2-07b/>

These page sizes can be configured with an environment variable or with settings in an application XCOFF binary with the **ldedit** or **ld** commands, as shown in Table 4-4.

Table 4-4 Page sizes for four regions of a 32-bit or 64-bit process address space

Region	ld or ldedit option	LDR_CNTRL environment variable	Description
Data	bdatapsize	DATAPSIZE	Initialized data, bss, and heap
Stack	bstacksize	STACKSIZE	Initial thread stack
Text	btextpsize	TEXTPSIZE	Main executable text
Shared memory	None	SHMPSIZE	Shared memory that is allocated by the process

You can specify a different page size to use for each of the four regions of a process address space. Only the 4 KB and 64 KB page sizes are supported for all four memory regions. The 16 MB page size is supported only for the process data, process text, and process shared memory regions. The 16 GB page size is supported only for a process shared memory region.

You can set the preferred page sizes for an application in the XCOFF/XCOFF64 binary file by running the **ldedit** or **ld** commands.

The **ld** or **cc** commands can be used to set these page size options when you are linking an executable command:

- ▶ **ld -o mpsize.out -btextpsize:4K -bstacksize:64K sub1.o sub2.o**
- ▶ **cc -o mpsize.out -btextpsize:4K -bstacksize:64K sub1.o sub2.o**

The **ldedit** command can be used to set these page size options in an existing executable command:

```
ldedit -btextpsize=4K -bdatapsize=64K -bstacksize=64K mpsize.out
```

You can set the preferred page sizes of a process with the **LDR_CNTRL** environment variable. As an example, the following command causes the **mpsize.out** process to use 4 KB pages for its data, 64 KB pages for its text, 64 KB pages for its stack, and 64 KB pages for its shared memory on supported hardware:

```
LDR_CNTRL=DATAPSIZE=4K@TEXTPSIZE=64K@SHMPSIZE=64K mpsize.out
```

Page size environment variables override any page size settings in an executable XCOFF header. Also, the **DATAPSIZE** environment variable overrides any **LARGE_PAGE_DATA** environment variable setting.

Rather than using the **LDR_CNTRL** environment variable, consider marking specific executable files to use large pages because this limits the large page usage to the specific application that benefits from large page usage.

Page size and shared memory

To back shared memory segments of an application with large pages, specify the **SHM_LGPAGE** and **SHM_PIN** flags in the **shmget()** function. In addition, set the **vmo v_pinshm** tunable to a value of 1 with, for example, **vmo -r -o v_pinshm=1**. If large pages are unavailable, the 4 KB pages back the shared memory segment.

Support for specifying the page size to use for the shared memory of a process with the **SHMPSIZE** environment variable is available starting in IBM AIX 5L™ Version 5.3 with the 5300-08 Technology Level, or later, and AIX Version 6.1 with the 6100-01 Technology Level, or later.

Monitoring the page size that is used by an application

Monitoring the page size is accomplished by running the following commands:¹²

- ▶ The **ps** command can be used to monitor the base page sizes that are used for process data, stack, and text.
- ▶ The **vmstat** command has two options available to display memory statistics for a specific page size:
 - The **vmstat -p** command displays global **vmstat** information, along with a breakdown of statistics per page size.
 - The **vmstat -P** command displays per page size statistics.

For more information about this topic, from the processor and OS perspectives, see:

- ▶ 2.2.2, “Multipage size support (page sizes (4 KB, 64 KB, 16 MB, and 16 GB))” on page 32 (*processor*)
- ▶ 4.2.2, “Multipage size support on AIX” on page 83
- ▶ 5.2.2, “Multipage size support on IBM i” on page 113
- ▶ 6.2.2, “Multipage size support on Linux” on page 123

For more information about this topic, see 4.5, “Related publications” on page 107.

4.2.3 Efficient use of cache

Generally, with Power Architecture, unlike some other architectures, users do not need to be concerned about cache management or optimizing cache usage. This section describes AIX facilities for controlling hardware prefetching through the Data Streams Control Register (DSCR) and is meant for advanced users who understand their workload characteristics and want to experiment with the register settings for improving performance. For a more detailed description of the DSCR register and its settings, see 2.2.3, “Efficient use of cache and memory” on page 33.

Controlling Data Streams Control Register under AIX

Under AIX, DSCR settings can be controlled both by programming API and from the command line by using the **dscr_ctl()** API and running the **dscrctl** command running the **dscr_ctl()** API and **dscrctl** commands.^{13,14}

dscr_ctl() API

```
#include <sys/machine.h>
int dscr_ctl(int op, void *buf_p, int size)
```

¹² *Multiple page size support*, found at:

http://www-01.ibm.com/support/knowledgecenter/api/redirect/aix/v7r1/topic/com.ibm.aix.prftungd/doc/prftungd/multiple_page_size_support.htm

¹³ *dscr_ctl Subroutine*, found at:

http://www-01.ibm.com/support/knowledgecenter/api/redirect/aix/v7r1/topic/com.ibm.aix.basetechref/doc/basetechref/dscr_ctl.htm

¹⁴ *dscrctl Command*, found at:

<http://www-01.ibm.com/support/knowledgecenter/api/redirect/aix/v7r1/topic/com.ibm.aix.cmds/doc/aixcmds2/dscrctl.htm>

Where:

op: Operation. Possible values are **DSCR_WRITE**, **DSCR_READ**, **DSCR_GET_PROPERTIES**, and **DSCR_SET_DEFAULT**.

Buf_p: Pointer to an area of memory where the values are copied from (**DSCR_WRITE**) or copied to (**DSCR_READ** and **DSCR_GET_PROPERTIES**). For **DSCR_WRITE**, **DSCR_READ**, and **DSCR_SET_DEFAULT** operations, **buf_p** must be a pointer to a 64-bit data area (long long *). For **DSCR_GET_PROPERTIES**, **buf_p** must be a pointer to a **struct dscr_properties** (defined in <sys/machine.h>).

Size: Size in bytes of the area pointed to by **buf_p**.

Function:

The action that is taken depends on the value of the operation parameter that is defined in <sys/machine.h>:

DSCR_WRITE	Stores a new value from the input buffer into the process context and in the DSCR.
DSCR_READ	Reads the current value of DSCR and returns it in the output buffer.
DSCR_GET_PROPERTIES	Reads the number of hardware streams that are supported by the platform, the platform (firmware) default Prefetch Depth, and the Operating System default Prefetch Depth from kernel memory, and returns the values in the output buffer (struct dscr_properties , which is defined in <sys/machine.h>).
DSCR_SET_DEFAULT	Sets a 64-bit DSCR value in a buffer pointed to by buf_p as the operating system default. Returns the old default in the buffer pointed to by buf_p . Requires root authority. The new default value is used by all the processes that do not explicitly set a DSCR value by using DSCR_WRITE . The new default is not permanent across restarts. For an operating system default prefetch depth that is permanent across restarts, use the dscrctl command, which adds an entry in the inittab to initialize the system-wide prefetch depth default value upon restart (for a description of this command, see “The dscrctl command” on page 88).

Here are the return values:

- ▶ 0 if successful.
- ▶ -1 if an error detected. In this case, **errno** is set to indicate the error. Here are the possible values:

EINVAL	Invalid value for DSCR (DSCR_WRITE , DSCR_SET_DEFAULT).
EFAULT	Invalid address that is passed to function.
EPERM	Operation not permitted (DSCR_SET_DEFAULT by non-root user).
ENOTSUP	Data streams that are not supported by platform hardware.

Symbolic values for the following SSE and DPFDF fields are defined in <sys/machine.h>:

DPFD_DEFAULT	0
DPFD_NONE	1
DPFD_SHALLOWEST	2
DPFD_SHALLOW	3
DPFD_MEDIUM	4
DPFD_DEEP	5
DPFD_DEEPER	6

DPFD_DEEPEST	7
DSCR_SSE	8

Here is a description of the **dscr_properties** structure in `<sys/machine.h>`:

```
struct dscr_properties {
    uintversion;
    uintnumber_of_streams; /* Number of HW streams */
    longlongplatform_default_pd; /* PFW default */
    longlongos_default_pd; /* AIX default */
    longlong dscr_res[5]; /* Reservd for future use */
};
```

Here is an example of this structure:

```
#include <sys/machine.h>
int rc;
long long dscr = DSCR_SSE | DPFD_DEEPER;
rc = dscr_ctl(DSCR_WRITE, &dscr);
...
```

A new process inherits the DSCR from its parent during a **fork**. This value is reset to the system default during **exec** (**fork** and **exec** are system calls).

When a thread is dispatched (starts running on a CPU), the value of the DSCR for the owning process is written in the DSCR. You do not need to save the value of the register in the process context when the thread is *undispatched* because the system call writes the new value both in the process context and in the DSCR.

When a thread runs **dscr_ctl** to change the prefetch depth for the process, the new value is written into the AIX process context and the DSCR register of the thread that is running the system call. If another thread in the process is concurrently running on another CPU, it starts using the new DSCR value only after the new value is reloaded from the process context area after either an interrupt or a redispatch. This action can take as much as 10 ms (a clock tick).

The dscrctl command

The system administrator can use this command to read the current settings for the hardware streams mechanism and set a system-wide value for the DSCR. The DSCR is privileged. It can be read or written only by the operating system.

To query the characteristics of the hardware streams on the system, run the following command:

```
dscrctl -q
```

Here is an example of this command:

```
# dscrctl -q
Current DSCR settings:
    number_of_streams = 16
    platform_default_pd = 0x5 (DPFD_DEEP)
    os_default_pd = 0xd (DSCR_SSE | DPFD_DEEP)
```

To set the operating system default prefetch depth on the system temporarily (that is, for the current session) or permanently (that is, after each restart), run the following command:

```
dscrctl [-n] [-b] -s <dscr_value>
```

The `dscr_value` is treated as a decimal number unless it starts with 0x, in which case it is treated as hexadecimal.

To cancel a permanent setting of the operating system default prefetch depth at start time, run the following command:

```
dscrctl -c
```

Applications that have predictable data access patterns, such as numerical applications that process arrays of data in a sequential manner, benefit from aggressive data prefetching. These applications must run with the default operating system prefetch depth, or whichever settings are empirically found to be the most beneficial.

Applications that have considerably unpredictable data access patterns, such as some transactional applications, can be negatively affected by aggressive data prefetching. The data that is prefetched is unlikely to be needed, and the prefetching uses system bandwidth and might displace useful data from the caches. Some WebSphere Application Server and DB2 workloads have this characteristic. Performance can be improved by disabling hardware prefetching in these cases by running the following command:

```
dscrctl -n -s 1
```

This system (partition) wide disabling is only appropriate if it is expected to benefit all of the applications that are running in the partition. However, the same effect can be achieved on an application-specific basis by using the programming API.

For more information about the efficient use of cache, from the processor and OS perspectives, see:

- ▶ 2.2.3, “Efficient use of cache and memory” on page 33 (*processor*)
- ▶ 6.2.3, “Efficient use of cache” on page 123 (*Linux*)

For more information about this topic, see 4.5, “Related publications” on page 107.

4.2.4 Transactional memory

Transactional memory (TM) is a POWER8 shared-memory synchronization construct that allows process-threads to perform storage operations that appear to be atomic to other process-threads and applications. One of the main uses of TM is that it speeds up the lock-based programs through the speculative execution of lock-based, critical sections, and it does so without first acquiring a lock. This allows applications that have not been carefully tuned for performance to take advantage of the benefits of fine-grain locking. The transactional programming model also provides productivity gains when developing lock-based, shared memory programs.

Although POWER8 processor-based systems support TM, you must explicitly check for support of TM before using the facility because the processor might be running in a compatibility mode, or the operating system or hypervisor might not support the use of TM. In AIX, the preferred API that determines whether TM is supported is the `getsystemcfg()` system call. A new `SC_TM_VER` system variable setting is provided that reports whether TM is supported. A new `__power_tm()` macro is provided that allows the caller to determine whether TM is supported. For more information, see the `/usr/include/sys/systemcfg.h` file.

Software failure handler

Upon transaction failure, the hardware redirects control to the failure handler that is associated with the outermost transaction. “Transaction failure” on page 43 explains this process and provides details about how control is passed to the software failure handler and the machine state of the status registers.

The Power Architecture Platform reserves a range of failure codes for the hypervisor, for client operating systems, and for user applications, to indicate a failure reason when issuing a **tabort.** instruction. These failure codes are noted in the following list:

- ▶ **0x00 – 0x3F** is reserved for use by AIX.
- ▶ **0x40 – 0xDF** is free for use by problem state (application) code.
- ▶ **0xE0 – 0xFF** is reserved for use by a hypervisor.

The failure codes that are reserved by AIX to indicate the cause of the failure are defined in `/usr/include/sys/machine.h`.

Debugger support

The dbx AIX debugger, found in `/usr/ccs/bin/dbx`, supports machine-level debugging of TM programs. This support includes the ability to disassemble the new TM instructions, and to display the TM SPRs.

Setting a breakpoint inside of a transaction causes the transaction to unconditionally fail whenever the breakpoint is encountered. To determine the cause and location of a failing transaction, the approach is to set a breakpoint on the transaction failure handler, and then to view the TEXASR and TFIAR registers when the breakpoint is encountered.

The TEXASR, TFIAR, and TFHAR registers can be displayed by using the **print** subcommand with the **\$texasr**, **\$tfiar**, or **\$tfhar** parameter. The line of code that is associated with the address that is found in TFIAR and TFHAR can be displayed by using the **list** subcommand, for example:

```
(dbx) list at $tfiar
```

A new **tm_status** subcommand is provided that displays and interprets the contents of the TEXASR register. This is useful in determining the nature of a transaction failure.

Tracing support

The AIX trace facility has been expanded to include a set of trace events for TM operations that are performed by AIX, including the processing of TM-type facility unavailable interrupts, preemptions that cause transaction failure, and other operations that can cause transaction failure. The trace event identifier **675** can be used as input to the **trace** and **trcrpt** commands to view TM-related trace events.

System call support

When a system call is made while a processor or thread is transactional (and the transaction has not been suspended), the system call is not started by the AIX kernel. The associated transaction persistently fails, and the system call handler returns control to the calling code with an error code of **ENOSYS**. When this occurs, the FC field of the TEXASR register contains the failure code **TM_ILL_SC**, which is defined in `/usr/sys/include/machine.h`.

It is assumed that any operations that are performed under a suspended transaction (when the application programmer has explicitly suspended the transaction) are intended to be persistent. Any operations that are performed by a system call that is made while in the suspended state are not rolled back if the transaction fails.

The reason that AIX cannot allow system calls to be made while in the transactional state is that any operations (writes or updates, including I/O) that are performed by AIX underneath a system call cannot be rolled back.

AIX threads library support

The use of TM is not supported for applications that use M:N threads. Undefined behavior might be encountered by transactional threads in an environment where more than one thread shares a single kernel thread. Usage of TM by an application that uses M:N threads can lead to a persistent transaction failure with the failure code **TM_PTH_PREEMPTED** being set in TEXASR.

Support of context management subroutines

The use of the context management subroutines, such as the **libc** subroutines **getcontext()**, **setcontext()**, **makecontext()**, **swapcontext()**, **setjmp()**, and **longjmp()** are not supported while in the transactional or suspended state. Such operations, where non-transactional context is restored while in the transactional or suspended state or context, is saved off while in the transactional or suspended state, and then restored while in the non-transactional state, leads to an inconsistent state and can result in undefined behavior. Under certain circumstances, AIX fails a transaction attempting to call such routines.

For more information about the topic of transactional memory, from the processor, OS, and compiler perspectives, see:

- ▶ 2.2.4, “Transactional memory” on page 42 (*processor*)
- ▶ 6.2.4, “Transactional memory” on page 124 (*Linux*)
- ▶ 7.3.5, “Transactional memory” on page 156 (*XL and GCC compiler families*)
- ▶ 8.4.2, “Transactional memory” on page 182 (*Java*)

4.2.5 Vector Scalar eXtension

A program can determine whether a system supports the vector extension by reading the **vmx_version** field of the **_system_configuration** structure. If this field is nonzero, then the system processor chips and operating system contain support for the vector extension. A value of 1 means that the processor chips on the system are Vector Multimedia eXtension (VMX) capable, and a value of 2 means that they are both VMX and Vector Scalar eXtension (VSX) capable. Alternatively, the **__power_vmx()** and **__power_vsx()** macros that are provided in **/usr/include/sys/systemcfg.h** can be used to perform these tests.

Vector capability support in AIX

The AIX Application Binary Interface (ABI) is extended to support the addition of vector register state and conventions. AIX supports the AltiVec programming interface specification.

A set of malloc subroutines (**vec_malloc**, **vec_free**, **vec_realloc**, and **vec_calloc**) is provided by AIX that give 16-byte aligned allocations. Vector-enabled compilation, with **_VEC_** implicitly defined by the compiler, result in any calls to older mallocs and callocs being redirected to their vector-safe counterparts, **vec_malloc** and **vec_calloc**. Non-vector code can also be explicitly compiled to pick up these same malloc and calloc redirections by explicitly defining **__AIXVEC**.

The alignment of the default **malloc()**, **realloc()**, and **calloc()** allocations can also be controlled at run time. This task can be done externally to any program by using the **MALLOCALIGN** environment variable, or internally to a program by using the **mallopt()** interface command option.¹⁵

For more information about the topic of VSX, from the processor, OS, and compiler perspectives, see:

- ▶ 2.2.5, “Vector Scalar eXtension” on page 45 (*processor*)
- ▶ 5.2.3, “Vector Scalar eXtension” on page 113 (*IBM i*)
- ▶ 6.2.5, “Vector Scalar eXtension” on page 125 (*Linux*)
- ▶ 7.3.2, “Compiler support for Vector Scalar eXtension” on page 151 (*XL and GCC compiler families*)

For more information about this topic, see 4.5, “Related publications” on page 107.

4.2.6 Decimal floating point

Decimal (base 10) data is widely used in commercial and financial applications. However, most computer systems have only binary (base two) arithmetic. There are two binary number systems in computers: integer (fixed-point) and floating point. Unfortunately, decimal calculations cannot be directly implemented with binary floating point. For example, the value 0.1 needs an infinitely recurring binary fraction, and a decimal number system can represent it exactly as 1/10th. So, using binary floating point cannot ensure that results are the same as those results that use decimal arithmetic.

In general, decimal floating point (DFP) operations are emulated with binary fixed-point integers. Decimal numbers are traditionally held in a binary-coded decimal (BCD) format. Although BCD provides sufficient accuracy for decimal calculation, it imposes a heavy cost in performance because it is implemented in software.

IBM Power Systems processor-based systems, starting with POWER6, provide hardware support for DFP arithmetic. Their microprocessor cores include a DFP unit that provides acceleration for the DFP arithmetic. The IBM Power Systems instruction set is expanded: 54 new instructions were added to support the DFP unit architecture. DFP can provide a performance boost for applications that are using BCD calculations.

How to take advantage of DFP unit on POWER

You can take advantage of the DFP unit on POWER by using the following features:¹⁶

- ▶ Native DFP language support with a compiler

The C draft standard includes the following new data types (these are native data types, as are int, long, float, double, and so on):

<code>_Decimal32</code>	Seven decimal digits of accuracy
<code>_Decimal64</code>	Sixteen decimal digits of accuracy
<code>_Decimal128</code>	Thirty-four decimal digits of accuracy

Note: The `printf()` function uses new options to print these new data types:

- ▶ `_Decimal32` uses `%Hf`
- ▶ `_Decimal64` uses `%Df`
- ▶ `_Decimal128` uses `%DDf`

¹⁵ *AIX vector programming*, found at:

http://www-01.ibm.com/support/knowledgecenter/api/redirect/aix/v7r1/index.jsp?topic=com.ibm.aix.genprog/doc/genprog/vector_prog.htm

¹⁶ *How to compile DFPAL?*, found at: <http://speleotrove.com/decimal/dfpal/compile.html>

- The IBM XL C/C++ Compiler, Release 9 or later, includes native DFP language support. Here is a list of compiler options for IBM XL compilers that are related to DFP:
 - **-qdfp**: Enables DFP support. This option makes the compiler recognize DFP literal suffixes, and the **_Decimal32**, **_Decimal64**, and **_Decimal128** keywords.
 - **-qfloat=dfpemulate**: Instructs the compiler to use calls to library functions to handle DFP computation, regardless of the architecture level. You might experience performance degradation when you use software emulation.
 - **-qfloat=nodfpemulate** (the default when the **-qarch** flag specifies POWER6, POWER7, or POWER8): Instructs the compiler to use DFP hardware instructions.
 - **-D__STDC_WANT_DEC_FP__**: Enables the referencing of DFP-defined symbols.

For hardware supported DFP, with **-qarch=pwr6**, **-qarch=pwr7**, or **-qarch=pwr8**, run the following command:

```
cc -qdfp
```

For software emulation of DFP (on earlier processor chips), run the following command:

```
cc -qdfp -qfloat=dfpemulate
```

- The GCC compilers for Power Systems also include native DFP language support. Here is a list of GCC compiler options that are related to DFP:
 - **-mhard-dfp** (the default when **-mcpu=power6** or **-mcpu=power7** is specified): Instructs the compiler to take direct advantage of DFP hardware instructions for decimal arithmetic.
 - **-mno-hard-dfp**: Instructs the compiler to use calls to library functions to handle DFP computation, regardless of the architecture level. If your application is dynamically linked to the **libdfp** variant and running on POWER6 or POWER7 processor-based systems, then the run time automatically binds to the **libdfp** variant implemented with hardware DFP instructions. Otherwise, the software DFP library is used. You might experience performance degradation when you use software emulation.
 - **-D__STDC_WANT_DEC_FP__**: Enables the reference of DFP defined symbols.

- Decimal Floating Point Abstraction Layer (DFPAL), which is a no additional cost, downloadable library from IBM.¹⁷

Many applications that are using BCD today use a library to perform math functions. Changing to a native data type can be hard work, after which you might have an issue with one code set for AIX on POWER6, POWER7, or POWER8 processor-based systems, and one for other platforms that do not support native DFP. The solution to this problem is DFPAL, which is an alternative to the native support. DFPAL contains a header file to include in your code and the DFPAL library.

The header file is downloadable from General Decimal Arithmetic at <http://speleotrove.com/decimal/> (search for “DFPAL”). Download the complete source code, and compile it on your system.

If you have hardware support for DFP, use the library to access the functions.

If you do not have hardware support (or want to compare the hardware and software emulation), you can force the use of software emulation by setting a shell variable before you run your application by running the following command:

```
export DFPAL_EXE_MODE=NSW
```

¹⁷ Ibid

Determining whether your applications are using DFP

There are two AIX commands that are used for monitoring:

- ▶ **hpmstat** (for monitoring the whole system)
- ▶ **hpmcount** (for monitoring a single program)

The **PM_DFU_FIN** (DFU instruction finish) field in the output of the **hpmstat** and **hpmcount** commands verifies that the DFP operations finished.

The **-E PM_MRK_DFU_FIN** option in the **tprof** command uses the AIX trace subsystem, which tells you which functions are using DFP and how often.

For more information about this topic, from the processor and OS perspectives, see:

- ▶ 2.2.6, “Decimal floating point” on page 47 (*processor*)
- ▶ 5.2.4, “Decimal floating point” on page 113 (*IBM i*)
- ▶ 6.2.6, “Decimal floating point” on page 126 (*Linux*)

For more information about this topic, see 4.5, “Related publications” on page 107.

4.2.7 On-chip encryption accelerator

When the AIX operating system runs on POWER7+ or POWER8 processors, it transparently uses on-chip encryption accelerators. For each of the uses that are described in this section, there are no application visible changes or awareness required.

AIX encrypted file system

Integrated with the AIX Journaled File System (JFS2) is the ability to create an encrypted file system (EFS) where all data at rest in the file system is encrypted. When AIX EFS runs on POWER7+ or POWER8 processor-based systems, it uses the encryption accelerators, which can show up to a 40% advantage in file system I/O-intensive operations. Applications do not need to be aware of this situation, but application and workload deployments might be able to take advantage of higher levels of security by using AIX EFS for sensitive data.

AIX Internet Protocol Security

When Internet Protocol Security (IPSec) is enabled on AIX running on POWER7+ or POWER8 processor-based systems, AIX transparently uses the on-chip encryption accelerators for all data in transit. The advantage that is provided by the accelerators is more pronounced when jumbo frames (a maximum transmission unit (MTU) of 9000 bytes) are used. Applications do not need to be aware of this situation, but application and workload deployments might be able to take advantage of higher levels of security by enabling IPSec.

AIX /dev/random (random number generation)

AIX capitalizes on the on-chip random number generator on POWER7+ and POWER8 processors. Applications that use the AIX special files **/dev/random** or **/dev/urandom** transparently get the advantages of stronger hardware-based random numbers. If an application is making high frequency usage of random number generation, there can also be a performance advantage.

AIX PKCS11 Library

On POWER7+ and POWER8 processor-based systems, the AIX operating system PKCS11 library transparently uses the on-chip encryption accelerators. For an application that uses the PKCS11 APIs, no change or awareness by the application is required. The AIX library interfaces dynamically decide, based on the algorithm and data size, when to use the accelerators. Because of the cost of setup and programming of the on-chip accelerators, the advantage is limited to operations on large blocks of data (tens to hundreds of kilobytes).

For more information about this topic, from the processor perspective, see 2.2.8, “On-chip accelerators” on page 48 (*processor*).

4.3 AIX operating system-specific optimizations

This section describes optimization methods that are specific to AIX.

4.3.1 Malloc

Every application needs a fast, scalable, and memory efficient allocator. However, each application’s memory request patterns are different. It is difficult to provide one common allocator or tunable that can satisfy the needs of all applications. AIX provides different memory allocators and suboptions within the allocator so that a system administrator or developer can choose more suitable settings for their application. This section explains the available choices and when to choose them.

Memory allocators

AIX provides three different allocators, and each of them uses a different memory management algorithm and data structures. These allocators work independently, so the application developer must choose one of them by exporting the **MALLOCTYPE** environment variable. The allocators are:

- Default allocator

The default allocator is selected when the **MALLOCTYPE** environment variable is unset. This setting maintains a consistent performance, even in a worst case scenario, but might not be as memory-efficient as a Watson allocator. This allocator is ideal for 32-bit applications, which do not make frequent calls to **malloc()**.

- Watson allocator

This allocator is selected when **MALLOCTYPE=watson** is set. This allocator is designed for 64-bit applications. It is memory efficient, scalable, and provides good performance. This allocator has a built-in bucket component for allocation requests up to 512 bytes. Table 4-5 provides the mapping for the allocation requests to bucket size.

Table 4-5 Mapping for allocation requests to bucket size

Request size	Bucket size	Request size	Bucket size	Request size	Bucket size	Request size	Bucket size
1 - 4		33-40	40	129-144	144	257-288	288
5 - 8		41 - 48	48	145 - 160	160	289 - 320	320
9 - 12	12	49 - 56	56	161 - 176	176	321 - 352	352
13 - 16	16	57 - 64	64	177 - 192	192	353 - 384	384
17 - 20	20	65 - 80	80	193 - 208	208	385 - 416	416

21 - 24	24	81 - 96	96	209 - 224	224	417 - 448	448
25 - 28	28	97 - 112	112	224 - 240	240	449 - 480	480
29 - 32	32	113 - 128	128	241 - 256	256	481 - 512	512

This allocator is ideal for 64-bit memory-intensive applications.

► **Malloc 3.1 allocator**

This allocator is selected when `MALLOCTYPE=3.1` is set. This is a bucket allocator that divides the heap into 28 hash buckets, each with a size of $2^{\text{pow}(x+4)}$, where x stands for bucket index. This allocator provides the best performance at the cost of memory. In most cases, this algorithm can use as much as twice the amount of memory that is requested by the application. In addition, an extra page is required for buckets larger than 4096 bytes because objects of a page in size or larger are page-aligned. Interestingly, some earlier customer applications still use this allocator, as it is more tolerant for application memory overwrite bugs.

Memory allocator suboptions

There are many suboptions available that can be selected by exporting the **MALLOCOPTIONS** environment variable. This section covers a few of the suboptions that are more relevant to performance tuning. For a complete list of options, see *System memory allocation using the malloc subsystem*, found at:

http://www-01.ibm.com/support/knowledgecenter/api/redirect/aix/v7r1/topic/com.ibm.aix.genprogc/doc/genprogc/sys_mem_alloc.htm

► **Multiheap**

By default, the malloc subsystem uses a single heap, which causes lock contention for internal locks that are used by malloc in case of multi-threaded applications. By enabling this option, you can configure the number of parallel heaps to be used by allocators. You can set the multiheap by exporting `MALLOCOPTIONS=multiheap[:n]`, where n can vary between 1- 32 and 32 is the default if n is not specified.

Use this option for multi-threaded applications, as it can improve performance.

► **Buckets**

This suboption is similar to the built-in bucket allocator of the Watson allocator. However, with this option, you can have fine-grained control over the number of buckets, number of blocks per bucket, and the size of each bucket. This option also provides a way to view the usage statistics of each bucket, which be used to refine the bucket settings.

If the application has many requests of the same size, then the bucket allocator can be configured to preallocate the required size by correctly specifying the bucket options. The block size can go beyond 512 bytes, compared to the Watson allocator or malloc pool options.

You can enable the buckets allocator by exporting `MALLOCOPTIONS=buckets`. Details about the buckets options for fine-grained control are available¹⁸. Enabling the buckets allocator turns off the built-in bucket component if the Watson allocator is used.

¹⁸ *System memory allocation using the malloc subsystem*, found at:

http://www-01.ibm.com/support/knowledgecenter/api/redirect/aix/v7r1/topic/com.ibm.aix.genprogc/doc/genprogc/sys_mem_alloc.htm

- ▶ **malloc pools**

This option enables a high performance front end to malloc subsystem for managing storage objects smaller than 513 bytes. This suboption is similar to the built-in bucket allocator of the Watson allocator. However, this suboption maintains the bucket for each thread, providing lock-free allocation and deallocation for blocks smaller than 513 bytes. This suboption improves the performance for multi-threaded applications, as the time spent on locking is avoided for blocks smaller than 513 bytes.

The pool option makes small memory block allocations fast (no locking) and memory efficient (no header on each allocation object). The pool malloc both speeds up single-threaded applications, and improves the scalability of multi-threaded applications.

- ▶ **malloc disclaim**

By enabling this option, **free()** automatically disclaims memory. This suboption is useful for reducing the paging space requirement. This option can be set by exporting `MALLOCOPTIONS=disclaim`.

Use cases

Here are some uses cases that you can use to set up your environment:

- ▶ For a 32-bit single-threaded application, use the default allocator.
- ▶ For a 64-bit application, use the Watson allocator.
- ▶ Multi-threaded applications use the **multiheap** option. Set the number of heaps proportional to the number of threads in the application.
- ▶ For single-threaded or multi-threaded applications that make frequent allocation and deallocation of memory blocks smaller than 513, use the **malloc pool** option.
- ▶ For a memory usage pattern of the application that shows high usage of memory blocks of the same size (or sizes that can fall to common block size in bucket option) and sizes greater than 512 bytes, use the **configure malloc bucket** option.
- ▶ For older applications that require high performance and do not have memory fragmentation issues, use **malloc 3.1**.
- ▶ Ideally, the Watson allocator, along with the **multiheap** and **malloc pool** options, is good for most multi-threaded applications. The pool front end is fast and scalable for small allocations, and with **multiheap**, ensures scalability for larger and less frequent allocations.
- ▶ If you notice high memory usage in the application process even after you run **free()**, the **disclaim** option can help.

For more information about this topic, see 4.5, “Related publications” on page 107.

4.3.2 Pthread tunables

The AIX pthread library can be customized with a set of environment variables. Specific variables that improve scaling and CPU usage are listed here. A full description is provided in the following settings:

- ▶ **AIXTHREAD_SCOPE={P|S}**

The **P** option signifies a process-wide contention scope (M:N), and the **S** option signifies a system-wide contention scope (1:1). Use system scope (1:1) for AIX. Although process scope (M:N) continues to be supported, it is no longer being enhanced in AIX.

► **SPINLOOPTIME=*n***

The **SPINLOOPTIME** variable controls the number of times the system tries to get a busy mutex or spin lock without taking a secondary action, such as calling the kernel to yield the process. This control is intended for MP systems, where it is hoped that the lock that is held by another actively running pthread is released. The parameter works only within libpthreads (user threads). If locks are available within a short period, you might want to increase the spin time by setting this environment variable. The number of times to try a busy lock before yielding to another pthread is *n*. The default is 40 and *n* must be a positive value.

► **YIELDLOOPTIME=*n***

The **YIELDLOOPTIME** variable controls the number of times that the system yields the logical processor when it tries to acquire a busy mutex or spin lock before it goes to sleep on the lock. The logical processor is yielded to another kernel thread, assuming that there is another executable thread with sufficient priority. This variable is effective in complex applications, where multiple locks are in use. The number of times to yield the logical processor before blocking on a busy lock is *n*. The default is 0 and *n* must be a positive value.

For more information about this topic, see 4.5, “Related publications” on page 107.

4.3.3 pollset

AIX 5L V5.3 introduced the pollset APIs. Pollsets are an AIX replacement for UNIX **select()** and **poll()**. **Pollset**, **select()**, and **poll()** all allow an application to query efficiently the status of file descriptors. This action is typically done to allow a single application to multiplex I/O across many file descriptors. Pollset APIs can be more efficient when the number of file descriptors that are queried becomes large.

Efficient I/O event polling through the pollset interface on AIX contains a pollset summary and outlines the most advantageous use of Java. To see this document, go to the following website:

<http://www.ibm.com/developerworks/aix/library/au-pollset/index.html>

For more information about this topic, see 4.5, “Related publications” on page 107.

4.3.4 File system performance benefits

AIX JFS2 is the default file system for 64-bit kernel environments. Applications can capitalize on the features of JFS2 for better performance.

4.3.5 Direct I/O

The AIX read-ahead and write-behind JFS2 feature might not be suitable for applications that perform large-sized I/O operations, as the cache hit ratio is low. In those cases, an application developer must evaluate Direct I/O for I/O-intensive applications.

Programs that are good candidates for direct I/O are typically CPU-limited and perform much disk I/O. Technical applications that have large sequential I/Os are good candidates. Applications that benefit from striping are also good candidates.

The direct I/O access method bypasses the file cache and transfers data directly from disk into the user space buffer, as opposed to using the normal cache policy of placing pages in kernel memory.

At the user level, file systems can be mounted by using the **dio** option with the **mount** command.

At the programming level, applications enable direct I/O access to a file by passing the **O_DIRECT** flag to the open subroutine. This flag is defined in the `fcntl.h` file. Applications must be compiled with **_ALL_SOURCE** enabled to see the definition of **O_DIRECT**.

For more information, see *Working with file I/O*, found at:

http://www-01.ibm.com/support/knowledgecenter/api/redirect/aix/v6r1/index.jsp?topic=%2Fcom.ibm.aix.genprogc%2Fdoc%2Fgenprogc%2Fworking_file_io.htm

4.3.6 Concurrent I/O

An AIX JFS2 inode lock imposes write serialization at the file level. Serializing write accesses prevents data inconsistency because of overlapping writes. Serializing reads regarding writes ensures that the application does not read stale data.

However, some applications can choose to implement their own data serialization, usually at a finer level of granularity than the file. Therefore, they do not need the file system to implement this serialization for them. The inode lock hinders performance in such cases by unnecessarily serializing non-competing data accesses. For such applications, AIX offers the concurrent I/O (CIO) option. Under CIO, multiple threads can simultaneously perform reads and writes on a shared file. For applications that do not enforce serialization for accesses to shared files, do not use CIO, as it can result in data corruption because of competing accesses.

Enhanced JFS supports concurrent file access to files. Similar to direct I/O, this access method bypasses the file cache and transfers data directly from disk into the user space buffer.

CIO can be specified for a file either by running **mount -o cio** or by using the **open()** system call (by using **O_CIO** as the **OFlag** parameter).

4.3.7 Asynchronous I/O

If an application does a synchronous I/O operation, it must wait for the I/O to complete. In contrast, asynchronous I/O operations run in the background and do not block user applications, which improves performance because I/O operations and applications processing can run simultaneously. Many applications, such as databases and file servers, take advantage of the ability to overlap processing and I/O.

Applications can use the **aio_read()**, **aio_write()**, or **lio_listio()** subroutines (or their 64-bit counterparts) to perform asynchronous disk I/O. Control returns to the application from the subroutine when the request is queued. The application can then continue processing while the disk operation is being performed.

4.3.8 I/O completion ports

A limitation of the AIO interface that is used in a threaded environment is that `aio_nwait()` collects completed I/O requests for *all* threads in the same process. One thread collects completed I/O requests that are submitted by another thread.

Another limitation is that multiple threads cannot start the collection routines (such as `aio_nwait()`) at the same time. If one thread issues `aio_nwait()` when another thread is calling it, the second `aio_nwait()` returns EBUSY. This limitation can affect I/O performance when many I/Os must run at the same time and a single thread cannot run fast enough to collect all the completed I/Os.

On AIX, using I/O completion ports with AIO requests provides the capability for an application to capture the results of various AIO operations on a per-thread basis in a multi-threaded environment. This function provides threads with a method of receiving a completion status for only the AIO requests that are initiated by the thread.

You can enable IOCP on AIX by running `smitty iocp`. Verify that IOCP is enabled by running the following command:

```
lsdev -Ccc iocp
```

The resulting output is shown in the following example:

```
iocp0 Available I/O Completion Ports
```

4.3.9 shmat versus mmap

Memory-mapped files provide a mechanism for a process to access files by directly incorporating file data into the process address space. The use of mapped files can reduce I/O data movement because the file data does not have to be copied into process data buffers, as is done by the read and write subroutines. When more than one process maps the same file, its contents are shared among them, providing a low-impact mechanism by which processes can synchronize and communicate.

AIX provides two methods for mapping files and anonymous memory regions. The first set of services, which are known collectively as the *shmat* services, are typically used to create and use shared memory segments from a program. The second set of services, which are known collectively as the *mmap* services, is typically used for mapping files, although it can be used for creating shared memory segments as well.

Both the *mmap* and *shmat* services provide the capability for multiple processes to map the same region of an object so that they share addressability to that object. However, the *mmap* subroutine extends this capability beyond that provided by the *shmat* subroutine by allowing a relatively unlimited number of such mappings to be established. Although this capability increases the number of mappings that are supported per file object or memory segment, it can prove inefficient for applications in which many processes map the same file data into their address space. The *mmap* subroutine provides a unique object address for each process that maps to an object. The software accomplishes this task by providing each process with a unique virtual address, which is known as an *alias*. The *shmat* subroutine allows processes to share the addresses of the mapped objects.

shmat can be used to share memory segments in a way that is similar to how it creates and uses files. An *extended shmat* capability is available for 32-bit applications with their limited address spaces. If you define the `EXTSHM=ON` environment variable, then processes running in that environment can create and attach more than 11 shared memory segments.

Use the **shmat** services under the following circumstances:

- ▶ When mapping files larger than 256 MB
- ▶ When mapping shared memory regions that must be shared among unrelated processes (no parent-child relationship)
- ▶ When mapping entire files

In general, **shmat** is more efficient but less flexible.

Use **mmap** under the following circumstances:

- ▶ Many files are mapped simultaneously.
- ▶ Only a portion of a file must be mapped.
- ▶ Page-level protection must be set on the mapping (allows a 4 K boundary).

For more information, see *General Programming Concepts: Writing and Debugging Programs*, found at:

http://publib16.boulder.ibm.com/doc_link/en_US/a_doc_lib/aixprgdd/genprogc/understanding_mem_mapping.htm

For more information about this topic, see 4.5, “Related publications” on page 107.

4.3.10 Large segment tunable aliasing (LSA)

AIX V6.1 TL5 and AIX V7.1 introduce the 1 TB Segment Aliasing. 1 TB segments can improve the performance of 64-bit large memory applications. The optimization is specific to large shared memory (**shmat()** and **mmap()**) regions.

1 TB segments are a feature present in POWER5+ and later processor-based systems. They can be used to reduce Segment Lookaside Buffer (SLB) misses, and increase the reach of the SLB, reducing the impact of effective-to-virtual address to real translation impact. Applications that are 64-bit and that have large shared memory regions can benefit from incorporating 1 TB segments. This feature is enabled by default on AIX V7.1, but can be enabled by using the **vmo** command to adjust the **esid_allocator** tunable.

An overview of 1 TB segment usage can be found in *IBM AIX Version 7.1 Differences Guide*, SG24-7910.

For more information about this topic, see 4.5, “Related publications” on page 107.

4.3.11 64-bit versus 32-bit ABIs

AIX provides complete support for both 32-bit and 64-bit ABIs. Applications can be developed by using either ABI with some performance trade-offs. The 64-bit ABI provides more scaling benefits. With both ABIs, there are performance trade-offs to be considered.

Overview of 64-bit/32-bit ABI

All current POWER processors support a 32-bit and 64-bit execution mode. The 32-bit execution mode is a subset of the 64-bit execution mode. The modes are similar, where the most significant difference is addresses in address generation (effective addresses are truncated to 32 bits) and computation of some fixed-point status registers (carry, overflow, and so on). Although hardware 32-bit/64-bit mode does not affect performance, the 32-bit/64-bit ABIs that are provided by AIX do have performance implications and tradeoffs.

The 32-bit ABI provides an ILP32 model (32-bit integers, longs, and pointers). The 64-bit ABI provides an LP64 model (32-bit integer and 64-bit longs/pointers). Although current POWER CPUs have 64-bit fixed-point registers, they are treated as 32-bit fixed-point registers by the ABI (the high 32 bits of all fixed-point registers are treated as volatile or undefined by the ABI). The 32-bit ABI preserves only 32-bit fixed-point context across subroutine linkage, non-local goto (**longjmp()**), or signal delivery. 32-bit programs cannot attempt to use 64-bit registers when they run in 32-bit mode (32-bit ABI). In general, other registers (floating point, vector, and status registers) are the same size in both 32-bit/64-bit ABIs.

Starting with AIX V6.1, all supervisor code (kernel, kernel extensions, and device drivers) uses the 64-bit ABI. In general, a unified system call interface is provided to applications that provides efficient system call linkage to both 32-bit and 64-bit applications. Because the AIX V6.1 kernel is 64-bit, it implies that all systems supported by AIX V6.1 support the 64-bit ABI. Some older IBM PowerPC CPUs supported on AIX 5L V5.3 cannot run the 64-bit ABI.

Operating system libraries provide both 32-bit and 64-bit objects, allowing full support for either ABI. Development tools (assembly language, linker, and debuggers) support both ABIs.

Trade-offs

The primary motivation to choose the 64-bit ABI is to go beyond the 4 GB directly memory addressability barrier. A second reason is to improve scalability by extending some 32-bit data type limits that are in the 32-bit ABI (`time_t`, `pid_t`, and `offset_t`). Lastly, 64-bit mode provides access to 64-bit fixed-point registers and instructions that can improve the performance of specific fixed-point operations (long long arithmetic and 64-bit memory copies).

The 64-bit ABI does have some performance drawbacks, such as the 64-bit fixed-point registers and the LP64 model grow stack usage and data structures. These items can cause a performance drawback for some applications. Also, 64-bit text is larger for most compiles, producing a larger i-cache footprint.

The most significant issue is typically the porting effort (for existing applications), as changing between ILP32 and LP64 normally requires a port. Large memory addressability and scalability are normally the deciding factor when you chose an application execution model.

For more information about this topic, see 4.5, “Related publications” on page 107.

4.3.12 Sleep and wake-up primitives (`thread_wait` and `thread_post`)

AIX provides proprietary `thread_wait()` and `thread_post()` APIs that can be used to optimize thread synchronization and communication in instructions per cycle (IPC). AIX also provides several standard APIs that can be used for thread synchronization and communication. These APIs include `pthread_cond_wait()`, `pthread_cond_signal()`, and `semop()`. Although many applications use these standard APIs, the low-level primitives are available to optimize these operations. `thread_wait()` and `thread_post()` can be used to optimize critical applications services, such as user-mode locking or message passing. They are more efficient than the portable/standard APIs.

The following list has more information about the associated subroutines:

► **thread_wait**

The **thread_wait** subroutine allows a thread to wait or block until another thread posts it with the **thread_post** or the **thread_post_many** subroutine or until the time limit that is specified by the timeout value expires.

If the event for which the thread is waiting and for which it is posted occurs only in the future, the **thread_wait** subroutine can be called with a timeout value of 0 to clear any pending posts. This action can be accomplished by running the following command:

```
thread_wait (timeout)
```

► **thread_post**

The **thread_post** subroutine posts the thread whose thread ID is indicated by the value of the **tid** parameter, of the occurrence of an event. If the posted thread is waiting in **thread_wait**, it is awakened immediately. If it is not waiting in **thread_wait**, the next call to **thread_wait** is not blocked, but returns with success immediately.

Multiple posts to the same thread without an intervening wait by the specified thread counts only as a single post. The posting remains in effect until the indicated thread calls the **thread_wait** subroutine, upon which the posting is cleared.

► **thread_post_many**

The **thread_post_many** subroutine posts one or more threads of the occurrence of the event. The number of threads to be posted is specified by the value of the **nthreads** parameter, and the **tidp** parameter points to an array of thread IDs of threads that must be posted. The subroutine works just like the **thread_post** subroutine, but can be used to post to multiple threads at the same time. A maximum of 512 threads can be posted in one call to the **thread_post_many** subroutine.

For more information about this topic, see 4.5, “Related publications” on page 107.

4.3.13 Shared versus private loads

You can use AIX to share text for libraries and dynamically loaded modules. File permissions can be used to enable and disable sharing of loaded text.

Documentation

AIX provides optimizations that enable sharing of loaded text (libraries and dynamically loaded modules). Sharing text among processes often improves performance because it reduces resource usage (memory and disk space). It also allows unrelated software-threads to share cache space when they run concurrently. Lastly, it can reduce load times when the code is already loaded by a previous program.

Applications can control whether private or shared loads are performed to shared text regions. Shared loads require that execute permissions be set for group/other on the text files. As a preferred practice, enable sharing.

For more information about this topic, see 4.5, “Related publications” on page 107.

4.3.14 Workload partition shared licensed program installations

Starting with AIX V6.1, the workload partition (WPAR) feature gives the system administrator the ability to create easily an isolated AIX operating system that can run services and applications. WPAR provides a secure and isolated environment for enterprise applications in terms of process, signal, and file system space. Any software that is running within the context of a workload partition appears to have its own separate instance of AIX.

The usage of multiple virtual operating systems within a single global operating environment can have multiple advantages. It increases administrative efficiency by reducing the number of AIX instances that must be maintained.

Applications can be installed in a shared environment or a non-shared environment. When an application is installed in a shared environment, it means that it is installed in the global environment and then the application is shared with one or more WPARs. When an application is installed in a non-shared environment, it means that it is installed in the WPAR only. Other WPARs do not have access to that application.

Note: WPARs can be considered the AIX version of containers, and were invented before Linux Docker containers. Thus, the use case of WPARs has some commonalities with Docker containers.

Shared workload partition installation

A shared installation is straightforward because installing software in the global environment is accomplished in the normal manner. What must be considered is whether the system WPARs that share a single installation will interfere with each other's operation.

For software to function correctly in a shared-installation environment, the software package must be split into shareable and non-shareable files:

- ▶ Shareable files (such as executable code and message catalogs) must be installed into the shared global file systems that are read-only to all system WPARs.
- ▶ Non-shareable files (such as configuration and runtime-modifiable files) must be installed into the file systems that are writable to individual WPARs. This configuration allows multiple WPARs to share a single installation, yet still have unique configuration and runtime data.

In addition to splitting the software package, the software installation process must include a synchronization step to install non-shareable files into system WPARs. To accomplish this task, the application must provide a means to encapsulate the non-shareable files within the shared global file systems so that the non-shared files can be extracted into the WPAR by some means. For example, if a vendor creates a custom-installation system that delivers files into `/usr` and `/`, then the files that are delivered into `/` must be archived within `/usr` and then extracted into `/` by using some vendor-provided mechanism. This action can occur automatically the first time that the application is started or configured.

Finally, the software update process must work so that the shareable and non-shareable files stay synchronized. If the shared files in the global AIX instance are updated to a certain fix level, then the non-shared files in individual WPARs also must be updated to the same level. Either the update process discovers all the system WPARs that must be updated or, at start time, the application detects the out-of-synchronization condition and applies the update. Some software products manage to never change their non-sharable files in their update process, so they do not need any special handling for updates.

This type of installation sometimes takes a little effort on the part of the application, but it allows you to get the most value from using WPARs. If there is a need to run the same version of the software in several WPARs, this type of installation provides the following benefits:

- ▶ It increases administrative efficiency by reducing the number of application instances that users must maintain. The administrator saves time in application-maintenance tasks, such as applying fixes and performing backups and migrations.
- ▶ It allows users to deploy quickly multiple instances of the same application, each in its own secure and isolated environment. It can take only a matter of minutes to create and start a WPAR to run a shared installation of the application.
- ▶ By sharing one AIX or application image among multiple WPARs, the memory resource usage is reduced because only one copy of the application image is in real memory.

For more information about WPAR, see *WPAR concepts*, found at:

<http://www-01.ibm.com/support/knowledgecenter/api/redirect/aix/v7r1/topic/com.ibm.aix.wpar/wpar-overview.htm>

For more information about the topic of operating system-specific optimizations, from the IBM i and Linux perspectives, see:

- ▶ 5.3, “IBM i operating system-specific optimizations” on page 114 (*IBM i*)
- ▶ 6.3, “Linux operating system-specific optimizations” on page 129 (*Linux*)

4.4 AIX preferred practices

This section describes AIX preferred practices, and includes these subsections:

- ▶ 4.4.1, “AIX preferred practices that are applicable to all Power Systems generations” on page 105
- ▶ 4.4.2, “AIX preferred practices that are applicable to POWER7 and POWER8 processor-based systems” on page 106 systems

4.4.1 AIX preferred practices that are applicable to all Power Systems generations

Preferred practices for the installation and configuration of all Power Systems generations are noted in the following list:

- ▶ If this server is a VIOS, then run the VIO Performance Advisor on the VIOS. Instructions are available for Virtual I/O Server Advisor at the following website:

<http://www.ibm.com/developerworks/community/wikis/home?lang=en#!/wiki/Power%20Systems/page/VIOS%20Advisor>

For more information, see “VIOS Performance Advisor” on page 217.

- ▶ For logical partitions (LPARs) with Java applications, run and evaluate the output from the Java Performance Advisor, which can be run on POWER5 and POWER6 processor-based systems, to determine whether there is an existing issue before you migrate to a POWER7 processor-based systems. Instructions are available for Java Performance Advisor (JPA) at the following website:

[https://www.ibm.com/developerworks/community/wikis/home/wiki/Power%20Systems/page/Java%20Performance%20Advisor%20\(JPA\)](https://www.ibm.com/developerworks/community/wikis/home/wiki/Power%20Systems/page/Java%20Performance%20Advisor%20(JPA))

For more information, see “Java Performance Advisor” on page 219.

- ▶ For virtualized environments, you can also use the IBM PowerVM Virtualization Performance Advisor. Instructions for the IBM PowerVM Virtualization Performance Advisor are found at the following website:
<https://www.ibm.com/developerworks/community/wikis/home?lang=en#!/wiki/Power%20Systems/page/PowerVM%20Virtualization%20Performance%20Advisor>
For more information, see “Virtualization Performance Advisor” on page 218.
- ▶ The number of online virtual CPUs of a single LPAR cannot exceed the number of active CPUs in a pool. See the output of `lparstat -i` from the LPAR to see the values for online virtual CPUs and active CPUs in pool.
- ▶ IBM maintains a strong focus on the quality and reliability of Power Systems servers. To maintain this reliability, the currency of Licensed Internal Code levels on your systems is critical. Therefore, apply the latest Power Systems Firmware and management console levels as soon as possible. These service pack updates contain a collective number of High Impact or PERvasive (HIPER) fixes that continue to provide you with the system availability you expect from Power Systems.
- ▶ When you install firmware from the HMC, avoid the **do not auto accept** option. Selecting this advanced option can cause firmware installation problems.
- ▶ Subscribe to My Notifications to provide you with customizable communications that contain important news, new or updated support content, such as publications, hints, and tips, technical notes, product flashes (alerts), downloads, and drivers.

4.4.2 AIX preferred practices that are applicable to POWER7 and POWER8 processor-based systems

This section covers the AIX preferred practices that are applicable to POWER7 and POWER8 processor-based systems.

Preferred practices for installation and configuration

Preferred practices for installation and configuration are noted in the following list:

- ▶ To ensure that your system conforms to the minimum requirements, see Chapter 3, “The IBM POWER Hypervisor” on page 57 and the references that are provided for that chapter (see 4.5, “Related publications” on page 107).
- ▶ Review the *POWER7 Virtualization Best Practice Guide*, found at:
https://www.ibm.com/developerworks/wikis/download/attachments/53871915/P7_virtualization_bestpractice.doc?version=1
- ▶ For POWER7 and POWER7+ processor-based systems, review the “Active System Optimizer/Dynamic System Optimizer” section in *POWER7 and POWER7+ Optimization and Tuning Guide*, SG24-8079 to identify whether those optimizations are useful.

For more information about this topic, see 4.5, “Related publications” on page 107.

4.5 Related publications

The publications that are listed in this section are considered suitable for a more detailed description of the topics that are covered in this chapter:

- ▶ *1 TB Segment Aliasing*, found at:

http://www-01.ibm.com/support/knowledgecenter/api/redirect/aix/v7r1/index.jsp?topic=/com.ibm.aix.prftungd/doc/prftungd/1TB_segment_aliasing.htm

- ▶ *AIX 64-bit Performance in Focus*, SG24-5103

- ▶ *AIX dscr_ctl API sample code*, found at:

<https://www.power.org/documentation/performance-guide-for-hpc-applications-on-ibm-power-755-system/> (registration required)

- ▶ *AIX Version 7.1 Release Notes*, found at:

<http://www-01.ibm.com/support/knowledgecenter/api/redirect/aix/v7r1/index.jsp?topic=/com.ibm.aix.ntl/RELNOTES/GI11-9815-00.htm>

Refer to the “The dscrctl command” section.

- ▶ *Application configuration for large pages*, found at:

http://www-01.ibm.com/support/knowledgecenter/api/redirect/aix/v7r1/index.jsp?topic=/com.ibm.aix.prftungd/doc/prftungd/config_apps_large_pages.htm

- ▶ *AIX Linking and Loading Mechanisms*, found at:

http://download.boulder.ibm.com/ibmdl/pub/software/dw/aix/es-aix_ll.pdf

- ▶ *Efficient I/O event polling through the pollset interface on AIX*, found at:

<http://www.ibm.com/developerworks/aix/library/au-pollset/index.html>

- ▶ *Exclusive use processor resource sets*, found at:

<http://www-01.ibm.com/support/knowledgecenter/api/redirect/aix/v7r1/topic/com.ibm.aix.baseadm/doc/baseadmdita/excluseprocreset.htm>

- ▶ *execrset command*, found at:

<http://www-01.ibm.com/support/knowledgecenter/api/redirect/aix/v7r1/topic/com.ibm.aix.cmds/doc/aixcmds2/execrset.htm>

- ▶ *General Programming Concepts: Writing and Debugging Programs*, found at:

http://publib16.boulder.ibm.com/doc_link/en_US/a_doc_lib/aixprggd/genprogc/understanding_mem_mapping.htm

- ▶ *IBM AIX Version 7.1 Differences Guide*, SG24-7910

See 1.2, “Improved performance using 1 TB segments”

- ▶ *load and loadAndInit Subroutines*, found at:

<http://www-01.ibm.com/support/knowledgecenter/api/redirect/aix/v7r1/index.jsp?topic=/com.ibm.aix.basetechref/doc/basetrf1/load.htm>

- ▶ *sync (Synchronize) or dcs (Data Cache Synchronize) instruction*, including information about **sync** and **lwsync** (lightweight sync), found at:

http://www-01.ibm.com/support/knowledgecenter/api/redirect/aix/v7r1/topic/com.ibm.aix.aixassem/doc/alangref/idalangref_sync_dcs_instrs.htm

- ▶ *mkrset Command*, found at:

<http://www-01.ibm.com/support/knowledgecenter/api/redirect/aix/v7r1/topic/com.ibm.aix.cmds/doc/aixcmds3/mkrset.htm>

- ▶ *Multiprocessing*, found at:
http://www-01.ibm.com/support/knowledgecenter/api/redirect/aix/v7r1/topic/com.ibm.aix.prftungd/doc/prftungd/intro_muiltproc.htm
- ▶ *Oracle Database and 1 TB Segment Aliasing*, found at:
<http://www.ibm.com/support/techdocs/atsmastr.nsf/WebIndex/TD105761>
- ▶ *pollset_create, pollset_ctl, pollset_destroy, pollset_poll, and pollset_query Subroutines*, found at:
<http://www-01.ibm.com/support/knowledgecenter/api/redirect/aix/v7r1/topic/com.ibm.aix.basetechref/doc/basetrf1/pollset.htm>
- ▶ *The Performance of Runtime Data Cache Prefetching in a Dynamic Optimization System*, found at:
<http://www.microarch.org/micro36/html/pdf/lu-PerformanceRuntimeData.pdf>
- ▶ *POWER6 Decimal Floating Point (DFP)*, found at:
<http://www.ibm.com/developerworks/wikis/display/WikiPtype/Decimal+Floating+Point>
- ▶ *POWER7 Virtualization Best Practice Guide*, found at:
https://www.ibm.com/developerworks/wikis/download/attachments/53871915/P7_virtualization_bestpractice.doc?version=1
- ▶ *ra_attach Subroutine*, found at:
http://www-01.ibm.com/support/knowledgecenter/api/redirect/aix/v7r1/index.jsp?topic=/com.ibm.aix.basetechref%2Fdoc%2Fbasetrf2%2Fra_attach.htm
- ▶ *Shared library memory footprints on AIX 5L*, found at:
http://www.ibm.com/developerworks/aix/library/au-slib_memory/index.html
- ▶ *Simultaneous multithreading*, found at:
<http://www-01.ibm.com/support/knowledgecenter/api/redirect/aix/v7r1/topic/com.ibm.aix.genprogc/doc/genprogc/smt.htm>
- ▶ *splat Command*, found at:
<http://www-01.ibm.com/support/knowledgecenter/api/redirect/aix/v7r1/index.jsp?topic=/com.ibm.aix.cmds/doc/aixcmds5/splat.htm>
- ▶ *trace Daemon*, found at:
<http://www-01.ibm.com/support/knowledgecenter/api/redirect/aix/v7r1/index.jsp?topic=/com.ibm.aix.cmds/doc/aixcmds5/trace.htm>
- ▶ *thread_post Subroutine*, found at:
http://www-01.ibm.com/support/knowledgecenter/api/redirect/aix/v7r1/index.jsp?topic=/com.ibm.aix.basetechref/doc/basetrf2/thread_post.htm
- ▶ *thread_post_many Subroutine*, found at:
http://www-01.ibm.com/support/knowledgecenter/api/redirect/aix/v7r1/index.jsp?topic=/com.ibm.aix.basetechref/doc/basetrf2/thread_post_many.htm
- ▶ *thread_wait Subroutine*, found at:
http://www-01.ibm.com/support/knowledgecenter/api/redirect/aix/v7r1/index.jsp?topic=/com.ibm.aix.basetechref/doc/basetrf2/thread_wait.htm

- *Thread environment variables*, found at:

http://www-01.ibm.com/support/knowledgecenter/api/redirect/aix/v7r1/index.jsp?topic=/com.ibm.aix.prftungd/doc/prftungd/thread_env_vars.htm

- *Power ISA Version 2.07*, found at:

<https://www.power.org/documentation/power-isa-v-2-07b/>

See the following sections:

- Section 3.1: Program Priority Registers
- Section 3.2: “or” Instruction
- Section 4.3.4: Program Priority Register
- Section 4.4.3: OR Instruction
- Section 5.3.4: Program Priority Register
- Section 5.4.2: OR Instruction
- Book I – 4 Floating Point Facility
- Book I – 5 Decimal Floating Point
- Book I – 6 Vector Facility
- Book I – 7 Vector-Scalar Floating Point Operations (VSX)
- Book I – Chapter 5 Decimal Floating-Point.
- Book II – 4.2 Data Stream Control Register
- Book II – 4.3.2 Data Cache Instructions
- Book II – 4.4 Synchronization Instructions
- Book II – A.2 Load and Reserve Mnemonics
- Book II – A.3 Synchronize Mnemonics
- Book II – Appendix B. Programming Examples for Sharing Storage
- Book III – 5.7 Storage Addressing



IBM i

This chapter describes the optimization and tuning of the POWER8 processor and other Power Systems processor-based servers running the IBM i operating system. It covers the following topics:

- ▶ 5.1, “Introduction” on page 112
- ▶ 5.2, “Using Power features with IBM i” on page 112
- ▶ 5.3, “IBM i operating system-specific optimizations” on page 114
- ▶ 5.4, “Related publications” on page 116

5.1 Introduction

IBM i provides an operating environment that emphasizes integration, security, and ease of use.

5.2 Using Power features with IBM i

The operating system and most applications for IBM i are built on a Technology Independent Machine Interface (TIMI) that isolates programs from differences in processor architectures, and allows the system to *automatically capitalize* on many new Power Architecture features without changes to existing programs. For example, TIMI allows a program to use decimal floating point (DFP) on POWER5 processors (without special hardware support), and that same program automatically uses hardware support for DFP on POWER6, POWER7, and POWER8 processors.

IBM Portable Application Solutions Environment for i (PASE for i) is a part of IBM i that allows some AIX application binary files to run on IBM i with little or no changes, so many optimizations that are described for AIX are applicable to PASE for i.

5.2.1 Multi-core and multi-thread

Operating system enablement usage of multi-core and multi-thread technology varies by operating system and release. Table 5-1 shows the maximum processor cores, threads, and simultaneous multithreading (SMT) level for a (single) logical partition running IBM i. Customers who need more capacity can contact IBM to request more processor cores than are shown in Table 5-1. IBM Systems Lab Services works with clients to determine whether IBM i can support the customer workload in a partition with a larger number of cores.

Table 5-1 Maximum processor cores, threads, and SMT level for a (single) logical partition running IBM i

Release	POWER6 processor-based systems	POWER7 processor-based systems	POWER8 processor-based systems
IBM i 6.1	32 Cores / 64 Threads / SMT2	Not supported	Not supported
IBM i 6.1.1	32 Cores / 64 Threads / SMT2	32 Cores / 128 Threads / SMT4	Not supported
IBM i 7.1 TR8	32 Cores / 64 Threads / SMT2	32 Cores / 128 Threads / SMT4	32 Cores / 256 Threads / SMT8
IBM i 7.2	32 Cores / 64 Threads / SMT2	32 Cores / 128 Threads / SMT4	48 Cores / 384 Threads / SMT8

For more information about this topic, from the processor and OS perspectives, see:

- ▶ 2.2.1, “Multi-core and multi-thread” on page 28 (*processor*)
- ▶ 4.2.1, “Multi-core and multi-thread” on page 72 (*AIX*)
- ▶ 6.2.1, “Multi-core and multi-thread” on page 119 (*Linux*)

Simultaneous multithreading

Simultaneous multithreading (SMT) is a feature of the Power Architecture and is described in “Simultaneous multithreading” on page 29.

Simultaneous multithreading dispatch control

IBM i 7.2 adds a job attribute named *Processor Resources Priority* (PRCRSCPTY) to influence how threads for the job are dispatched. The PRCRSCPTY attribute can request that the system isolate threads for the job on processors that are running fewer threads concurrently, or that the system run threads for the job on processors that are running as many concurrent threads as possible.

For more information about the topic of SMT, from the processor and OS perspectives, see:

- ▶ “Simultaneous multithreading” on page 29 (*processor*)
- ▶ “Simultaneous multithreading” on page 73 (*AIX*)
- ▶ “Simultaneous multithreading” on page 119 (*Linux*)

5.2.2 Multipage size support on IBM i

Most of IBM i uses 4 KB pages, but select system functions automatically use 64 KB pages. Applications running on IBM i 6.1 or later can create shared memory objects that use 64 KB pages (typically by using `shmctl` with `SHM_PAGESIZE`). IBM technology for Java programs running on IBM i 6.1 or later can use 64 KB pages for Java heap. PASE for i programs running on IBM i 7.1 or later automatically use 64 KB pages for shared library text and data, and can request 64 KB pages for program text, stack, and data.

IBM Power Systems Firmware does not support 64 KB pages for all configurations. For example, 64 KB pages are not available in a logical partition that is configured for Active Memory Sharing (AMS).

For more information about this topic, from the processor and OS perspectives, see:

- ▶ 2.2.2, “Multipage size support (page sizes (4 KB, 64 KB, 16 MB, and 16 GB))” on page 32 (*processor*)
- ▶ 4.2.2, “Multipage size support on AIX” on page 83
- ▶ 6.2.2, “Multipage size support on Linux” on page 123

5.2.3 Vector Scalar eXtension

IBM i 7.2 automatically uses POWER8 vector instructions to improve the performance of some cryptographic operations. PASE for i applications running on IBM i 7.2 on POWER7 or newer processors can use Vector Scalar eXtension (VSX).

For more information about the topic of VSX, from the processor, OS, and compiler perspectives, see:

- ▶ 2.2.5, “Vector Scalar eXtension” on page 45 (*processor*)
- ▶ 4.2.5, “Vector Scalar eXtension” on page 91 (*AIX*)
- ▶ 6.2.5, “Vector Scalar eXtension” on page 125 (*Linux*)
- ▶ 7.3.2, “Compiler support for Vector Scalar eXtension” on page 151 (*XL and GCC compiler families*)

5.2.4 Decimal floating point

IBM i 6.1 and later supports DFP in select programming languages and in DB2 for i. DFP operations (outside of PASE for i) automatically use DFP instructions when running on POWER6 or newer processors, and use software support on older architectures. IBM i 7.2 improves the performance of many DFP operations (compared to prior releases) by the increased use of DFP instructions.

For more information about this topic, from the processor and OS perspectives, see:

- ▶ 2.2.6, “Decimal floating point” on page 47 (*processor*)
- ▶ 4.2.6, “Decimal floating point” on page 92 (*AIX*)
- ▶ 6.2.6, “Decimal floating point” on page 126 (*Linux*)

5.3 IBM i operating system-specific optimizations

This section describes optimization methods that are specific to IBM i.

5.3.1 IBM i advanced optimization techniques

Optimization methods specific to the creation of IBM i programs and service programs include the following methods:¹

- ▶ *8-byte pointers in C and C++ code*: The performance of C and C++ code that uses pointers can be improved when the code is compiled to use 8-byte pointers, rather than 16-byte pointers (default). To take full advantage of 8-byte pointers, specify `STGMDL(*TERASPACE)` and `DTAMD(*LLP64)` when you compile code.
- ▶ *Program profiling*: Program profiling is an advanced optimization technique to reorder procedures, or code within procedures, and to direct code generation decisions in ILE programs and service programs based on statistical data that is gathered while running the program. The reordering can improve instruction cache utilization and reduce the paging that is required by the program, improving performance.
- ▶ *Argument optimization*: The Argument optimization parameter, with `ARGOPT(*YES)`, is available with the `CRTPGM` and `CRTSRVPGM` commands to support advanced argument optimization, where an analysis across modules that are bound to the program is performed. In general, this improves the performance of most procedure calls within the program. Argument optimization is a technique for passing arguments (parameters) to ILE procedures to improve performance of call-intensive applications.
- ▶ *Interprocedural analysis*: Interprocedural analysis that is performed by the `IPA(*YES)` option on `CRTPGM` or `CRTSRVPGM` performs optimizations across function bodies in the entire program during program creation. In particular, this occurs across the modules that are bound into the program and that were compiled with the `MODCRTOPT(*KEEPILDTA)` option. In contrast, intraprocedural is a mechanism for performing optimization for each function within a compilation unit, by using only the information that is available for that function and compilation unit.
- ▶ *Licensed Internal Code Options (LICOPTs)*: LICOPTs are compiler options that are passed to the Licensed Internal Code to affect how code is generated or packaged. You can use some of the options to fine-tune the optimization of your code.

The `TargetProcessorModel` LICOPT instructs the translator to perform optimizations that are tuned for the specified processor model. Programs that are created with this option run on all supported hardware models, but run faster on the specified processor model. For IBM i 7.2, a `TargetProcessorModel` value can be specified so that the code is tuned to run optimally on the POWER8 processor.

The `CodeGenTarget` LICOPT specifies the creation target model for a program or module object. The creation target model indicates the hardware features that the code that is generated for that object can use. For IBM i 7.2, a `CodeGenTarget` model of the POWER8 processor can be specified.

¹ ILE Concepts, SC41-5606

CodeGenTarget features and their associated Power Systems hardware include:

- CodeGenTarget features that are associated with POWER6 processor-based systems:
 - A hardware decimal floating point unit
 - Efficient hardware support for ILE pointer handling
- CodeGenTarget features associated with POWER7 processor-based systems: A number of new instructions that might speed up certain computations, such as conversions between integer and floating-point values.
- CodeGenTarget features associated with POWER8 hardware: New move instructions between floating point and general-purpose registers.

The TargetProcessorModel and CodeGenTarget LICOPTs are two of several factors, including Adaptive Code Generation, which determine the processor model to which code should be tuned and targeted when creating a module, changing a module or program, or re-creating a module or program. The default behavior is to use all features available on the current machine. For more information, see *ILE Concepts*, SC41-5606.

- *Adaptive Code Generation (ACG)*: ACG allows you to take advantage of all of the processor features on your systems, regardless of whether those features are present on other system models that are supported by the same release. Furthermore, programs can be moved from one system model to another and continue to run correctly, even if the new machine does not have all of the processor features that were available on the original machine. The technology for achieving this task is ACG. ACG can work without user intervention in most scenarios. However, if you build and distribute software to run on various system models, you might want to exercise some control over which processor features are used by ACG.

The first time a program object is activated on a system to which it is moved, the system performs a compatibility check to ensure that your program does not use any features that are unavailable on your system. If the program requires any processor feature that is not supported by the system to which it was moved, then the system automatically calls the optimizing translator to convert the program to be compatible. Options that are associated with restoring objects exist to cause incompatible module and program objects that are restored to be immediately converted, rather than on the first activation.

For more information about these optimizations in an IBM i environment, see *ILE Concepts*, SC41-5606. In particular, see Chapter 13, "Advanced Optimization Techniques".

5.3.2 Performance management on IBM i

For links to general performance resources, performance education resources, performance papers, and articles for IBM i, see the *Performance management on IBM i*, found at:

<http://www.ibm.com/systems/power/software/i/management/performance/resources.html>

For a basic understanding of IBM i on Power Systems performance concepts, workloads and benchmarks on Power Systems, capacity planning, performance monitoring and analysis, frequently asked questions, and guidelines addressing common performance issues, see *IBM i on Power - Performance FAQ*, found at:

http://www-01.ibm.com/common/ssi/cgi-bin/ssialias?subtype=WH&infotype=SA&appname=S TGE_PO_PO_USEN&htmlfid=POW03102USEN&attachment=POW03102USEN.PDF

For more information about the topic of operating system-specific optimizations, from the AIX and Linux perspectives, see:

- ▶ 4.3, “AIX operating system-specific optimizations” on page 95 (*AIX*)
- ▶ 6.3, “Linux operating system-specific optimizations” on page 129 (*Linux*)

5.4 Related publications

- ▶ *Advanced Optimization Techniques*, found at:
<http://www-01.ibm.com/support/knowledgecenter/api/redirect/series/v7r1m0/topic/ilec/sc415606206.htm>
- ▶ *IBM i on Power - Performance FAQ*, available found at:
http://www-01.ibm.com/common/ssi/cgi-bin/ssialias?subtype=WH&infotype=SA&appname=STGE_PO_PO_USEN&htmlfid=POW03102USEN&attachment=POW03102USEN.PDF
- ▶ *ILE Concepts*, SC41-5606
- ▶ *Performance management on IBM i*, found at:
<http://www.ibm.com/systems/power/software/i/management/performance/resources.html>



Linux

This chapter describes the optimization and tuning of the POWER8 and other Power Systems processor-based servers running the Linux operating system. It covers the following topics:

- ▶ 6.1, “Introduction” on page 118
- ▶ 6.2, “Using Power features with Linux” on page 118
- ▶ 6.3, “Linux operating system-specific optimizations” on page 129
- ▶ 6.4, “Little Endian” on page 138
- ▶ 6.5, “Related publications” on page 139

6.1 Introduction

When you work with POWER7, POWER7+, or POWER8 processor-based servers and solutions, a solid choice for running enterprise-level workloads is Linux. Red Hat Enterprise Linux (RHEL), SUSE Linux Enterprise Server (SLES), and Ubuntu provide operating systems that are optimized and targeted for the Power Architecture. These operating systems run natively on the Power Architecture and are designed to take full advantage of the specialized features of Power Systems.

RHEL and SLES support both POWER7 and POWER8 processor-based systems. Ubuntu is supported on POWER8 processor-based systems only (starting with Ubuntu Version 14.04). Unless otherwise stated, the references to the POWER8 processor or POWER8 processor-based systems in this chapter applies to all three Linux distributions, and references to POWER7 or POWER7 processor-based systems applies only to RHEL or SLES.

All of these Linux distributions provide the tools, kernel support, optimized compilers, and tuned libraries for Power Systems to achieve excellent performance. For advanced users, more application and customer-specific tuning approaches are also available.

Additionally, IBM provides a number of added value packages, tools, and extensions that provide for more tunings, optimizations, and products for the best possible performance on POWER8 processor-based systems. The typical Linux open source performance tools that Linux users are comfortable with are available on Linux on Power systems.

The IBM Linux on Power Tools repository enables the use of standard Linux package management tools (such as yum and zypper) to provide easy access to IBM recommended tools:

- ▶ IBM Linux on Power hardware diagnostic aids and productivity tools
- ▶ IBM Software Development Toolkit for Linux on Power servers
- ▶ IBM Advance Toolchain for Linux on Power Systems servers

The IBM Linux on Power Tools repository is found at:

<http://www.ibm.com/support/customer/sas/f/lopdiags/yum.html>

Under a PowerVM hypervisor, Linux on Power supports small virtualized Micro-Partitioning partitions up through large dedicated partitions containing all of the resources of a high-end server. Under a PowerKVM hypervisor, the Linux on Power supports running as a KVM guest on POWER8 processor-based systems.

IBM premier products, such as IBM XL compilers, IBM Java products, IBM WebSphere, and IBM DB2 database products, all provide Power Systems optimized support with the RHEL, SLES, and Ubuntu operating systems.

For more information about this topic, see 6.5, “Related publications” on page 139.

6.2 Using Power features with Linux

Some of the significant features of POWER with POWER7 and POWER8 extensions in a Linux environment are described in this section.

6.2.1 Multi-core and multi-thread

Operating system enablement usage of multi-core and multi-thread technology varies by operating system and release. Linux defines a logical CPU as any schedulable entity. So, every core/thread in a multi-core/thread processor is a logical CPU. Table 6-1 shows the maximum number of logical cores for a (single) logical partition/guest running Linux on Power that is supported by each distribution. However, the maximum number of hardware threads per LPAR (see Table 2-1 on page 28) for a POWER generation, limits the maximum number of logical CPUs on a POWER server of that generation.

Table 6-1 Maximum logical CPUs by single LPAR scaling

Single LPAR scaling	Linux release
128	SLES 10
256	RHEL 5
1024	RHEL 6 SLES 11
2048	RHEL 7 SLES 12 Ubuntu 14.04

Information about multi-thread per core features by POWER generation is available in Table 2-1 on page 28.

For more information about this topic, from the processor and OS perspectives, see:

- ▶ 2.2.1, “Multi-core and multi-thread” on page 28 (*processor*)
- ▶ 4.2.1, “Multi-core and multi-thread” on page 72 (*ALX*)
- ▶ 5.2.1, “Multi-core and multi-thread” on page 112 (*IBM i*)

Simultaneous multithreading

Simultaneous multithreading (SMT) is a feature of the Power Architecture and is described in “Simultaneous multithreading” on page 29.

On a POWER8 processor-based system, with a properly enabled Linux distribution, or distro, the Linux operating system supports up to eight hardware threads per core (SMT=8).

With the POWER8 processor cores, the SMT hardware threads are more equal in the execution implementation, which allows the system to support flexible SMT scheduling and management.

Application throughput and SMT scaling from SMT=1 to SMT=2, to SMT=4, and to SMT=8 is highly application-dependent. With additional hardware threads that are available for scheduling, the ability of the processor cores to switch from a waiting (stalled) hardware thread to another thread that is ready for processing can improve overall system effectiveness and throughput.

High SMT modes are best for maximizing total system throughput, and lower SMT modes might be appropriate for high performance threads and low latency applications. For code with low levels of instruction-level parallelism (often seen in Java code, for example), high SMT modes are preferred.

For more information about the topic of SMT, from the processor and OS perspectives, see:

- ▶ “Simultaneous multithreading” on page 29 (*processor*)
- ▶ “Simultaneous multithreading” on page 73 (*ALX*)
- ▶ “Simultaneous multithreading” on page 112 (*IBM i*)

Boot-time enablement of simultaneous multithreading

When starting a Linux distro, **SMT=8** is the default boot mode. To disable SMT at start time, simply add the **ppc64_cpu --smt=off** command to the `systemd` start script.

Dynamically selecting different simultaneous multithreading modes

Linux enables Power SMT capabilities. By default, the system runs at the highest SMT level.

Changing SMT settings remains a dynamic (runtime) option in the operating system. The **ppc64_cpu** command is provided in the `powerpc_utils` package. Running this command requires root access. The **ppc64_cpu** command can be used to force the system kernel to use lower SMT levels (ST, SMT2, or SMT4 mode). For example:

- ▶ **ppc64_cpu --smt=1** sets the SMT mode to ST.
- ▶ **ppc64_cpu --smt** shows the current SMT mode.

POWER8 processor-based systems support up to 8 SMT hardware threads per core. The **ppc64_cpu** command can specify hardware threads from a single thread per core, two threads, four threads, or eight threads.

When using the **ppc64_cpu** command to control SMT settings, the normal Linux approach of *holes in the CPU numbering* continues as it was in previous POWER generations, such as POWER7 processor-based systems.

In different POWER8 SMT modes, CPUs are numbered as follows:

SMT=8:	0,1,2,3,4,5,6,7,	8,9,10,11,12,13,14,15,	16,17,18,19,20,21,22,23, ...
SMT=4:	0,1,2,3,	8,9,10,11,	16,17,18,19, ...
SMT=2:	0,1,	8,9,	16,17, ...
SMT=1:	0,	8,	16, ...

The setaffinity application programming interface (API) allows processes and threads to have affinity to specific logical processors, as described in “Affinitization and binding” on page 121. Because the POWER8 processor supports running up to eight threads per core, the CPU numbering is different than in POWER7 processor-based systems, which supported only up to four threads per core. Therefore, an application that specifically binds processes to threads must be aware of the new CPU numbering to ensure that the binding is correct because there are now more threads available for each core.

For more information about this topic, see 6.5, “Related publications” on page 139.

Querying the simultaneous multithreading setting

The command for querying the SMT setting is **ppc64_cpu --smt**. A programmable API is not available.

Simultaneous multithreading priorities

Simultaneous multithreading (SMT) priorities in the POWER hardware are introduced in “Simultaneous multithreading priorities” on page 30. Linux supports selecting SMT priorities by using the Priority Nop mechanism or by writing to the PPR, as described in that section.

The current GLIBC (from Version 2.16) provides the system header `sys/platform/ppc.h`, which contains a wrapper for setting the PPR by using the Priority Nop mechanism, as shown in Example 6-1.

Example 6-1 GLIBC PPR set functions

```
void __ppc_set_ppr_med (void)
void __ppc_set_ppr_med_low (void)
void __ppc_set_ppr_low (void)
```

Earlier versions of RHEL and SLES do not provide this header; however, it is supported on IBM Advance Toolchain for Linux on Power Version 6.0 and later.

Where to use

SMT thread priority can be used to improve the performance of a workload by lowering the SMT thread priority that is being used on an SMT thread that is running a particular process-thread when:

- ▶ The thread is waiting on a lock
- ▶ The thread is waiting on an event, such as the completion of an I/O event

Alternatively, process-threads that are performance-sensitive can maximize their performance by ensuring that the SMT thread priority level is set to an elevated level.

For more information about the topic of SMT priorities, from the processor and OS perspectives, see:

- ▶ “Simultaneous multithreading priorities” on page 30 (*processor*)
- ▶ “Simultaneous multithreading priorities” on page 74 (*AIX*)

Affinitization and binding

Affinity performance effects are explained in “The POWER8 processor and affinity performance effects” on page 16. Establishing good affinity is accomplished by understanding the placement of a partition on the underlying cores and memory of a Power Systems server, and then by using operating system facilities to bind application threads to run on specific hardware threads or cores.

The `numactl --hardware` command shows the relative positioning of the underlying cores and memory, if that information is available from the hypervisor or firmware. In the case of PowerVM shared LPARS or PowerKVM guests, this information cannot be directly mapped to the underlying cores and memory.

Flexible simultaneous multithreading support

On POWER7 and POWER7+ processors, there is a correlation between the hardware thread number (0 - 3) and the hardware resources within the processor. Matching the thread numbers to the number of active threads is recommended for optimum performance. For example, if only one thread is active, it should be hardware thread 0. If two threads are active, they should be hardware threads 0 and 1. The Linux operating system automatically shifts the threads to those modes.

On the POWER8 processor, any process or thread can run in any SMT mode. The processor balances the processor core resources according to the number of active hardware threads. There is no need to match the application thread numbers with the number of active hardware threads. Hardware threads on the POWER8 processor have equal weight, unlike the hardware threads under POWER7 processor. Therefore, as an example, a single process running on thread 7 runs as fast as running on thread 0, presuming nothing else is on the other hardware threads for that processor core.

Linux scheduler

The Linux Completely Fair Scheduler (CFS) handles load balancing across CPUs and uses scheduler modules to make policy decisions. CFS works with multi-core and multi-thread processors and balances tasks across real processors. CFS also groups and tunes related tasks together.

The Linux topology considers physical packages, threads, siblings, and cores. The CFS scheduler domains help to determine load balancing. The base domain contains all sibling threads of the physical CPU, the next parent domain contains all physical CPUs, and the next parent domain takes NUMA nodes into consideration.

Because of the specific asymmetrical thread ordering of POWER7 processors, special Linux scheduler modifications were added for the POWER7 CPU type. With the POWER8 processor, this logic is no longer needed because any of the SMT8 threads can act as the primary thread by design. This means that the number of threads that are active in the core at one time determines the dynamic SMT mode (for example, from a performance perspective, thread 0 can be the same as thread 7). Idle threads should be napping or in a deeper sleep if they are idle for a period.

CPU sets, cgroups, and scheduler domains

It is possible to target (and limit) processes for a specific set of CPUs or cores. This can provide Linux applications with more fine-grained control of the cores and characteristics of application process and thread requirements.

taskset

Use the **taskset** command to retrieve, set, and verify the CPU affinity information of a process that running.

numactl

Similar to the **taskset** command, use the **numactl** command to retrieve, set, and verify the CPU affinity information of a process that running. The **numactl** command, however, provides additional performance information about local memory allocation.

Using setaffinity to bind to specific logical processors

The setaffinity API allows processes and threads to have affinity to specific logical processors. The number and numbering of logical processors is a product of the number of processor cores (in the partition) and the SMT capability of the machine (eight-way SMT for the POWER8 processor).

For more information about the topic of affinitization and binding, from the processor and OS perspectives, see:

- ▶ “Affinitization and binding to hardware threads” on page 31 (*processor*)
- ▶ “Affinitization and binding” on page 74 (*ALX*)

Hybrid thread and core

Linux provides facilities to customize SMT characteristics of CPUs running within a partition.

SMT can be enabled or disabled at boot time, as described in “Boot-time enablement of simultaneous multithreading” on page 120. SMT modes can be dynamically controlled for a whole partition, as described in “Dynamically selecting different simultaneous multithreading modes” on page 120.

In Linux, each CPU is associated with a processor core hardware thread.

When there is no work to be done on a CPU, the scheduler goes into the idle loop, and Linux calls into the hypervisor to report that the CPU is truly idle. The kernel-to-hypervisor interface is defined in the Power Architecture Platform Reference (PAPR) found at <http://power.org>. In this case, it is the H_CEDDE hypervisor call.

For more information about this topic, from the processor and OS perspectives, see:

- ▶ “Hybrid thread and core” on page 31 (*processor*)
- ▶ “Hybrid thread and core” on page 80 (*AIX*)

6.2.2 Multipage size support on Linux

On Power Systems running Linux, the default page size is 64 KB, so most, but not all, applications are expected to see a performance benefit from this default. There are cases in which an application uses many small files, which can mean that each file is loaded into a 64 KB page, resulting in poor memory utilization.

Support for 16 MB pages (huge pages in Linux terminology) is available through various mechanisms and is typically used for databases, Java engines, and high-performance computing (HPC) applications. The `libhugetlbfs` package is available in Linux distributions, and using this package gives you the most benefit from 16 MB pages.

Transparent huge pages (THP) is an alternative means of using huge pages for backing virtual memory. It does this through the automatic promotion and demotion of pages between 64 K (normal) and 16 MB (huge) page sizes. Unlike `libhugetlbfs`, huge pages do not need to be set aside or reserved at boot time to use this feature, and applications do not need to map them explicitly either. The memory for the SPLPAR partition (under PowerVM) or the guest (under PowerKVM) must be configured to be explicitly backed by huge pages. THP can be enabled by running the following command (root privileges needed):

```
echo always > /sys/kernel/mm/transparent_hugepage/enabled
```

Although THP is expected to improve the performance of most workloads with large memory footprints, it is not recommended for database workloads.

For more information about this topic, from the processor and OS perspectives, see:

- ▶ 2.2.2, “Multipage size support (page sizes (4 KB, 64 KB, 16 MB, and 16 GB))” on page 32 (*processor*)
- ▶ 4.2.2, “Multipage size support on AIX” on page 83
- ▶ 5.2.2, “Multipage size support on IBM i” on page 113

6.2.3 Efficient use of cache

Operating system facilities for controlling hardware prefetching are described in this section.

Controlling DSCR under Linux

DSCR settings on Linux are controlled with the `ppc64_cpu` command. Controlling DSCR settings for an application is considered advanced and specific tuning.

Currently, setting the DSCR value is a cross-LPAR setting.

For more information about the efficient use of cache, from the processor and OS perspectives, see:

- ▶ 2.2.3, “Efficient use of cache and memory” on page 33 (*processor*)
- ▶ 4.2.3, “Efficient use of cache” on page 86 (*AIIX*)

For more information about this topic, see 6.5, “Related publications” on page 139.

6.2.4 Transactional memory

Transactional memory (TM) is a POWER8 shared-memory synchronization construct that allows a thread to perform a sequence of storage operations that appear to occur atomically with respect to other threads. One of the main advantages of TM is that it can speed up lock-based programs through the speculative execution of lock-based, critical sections because it does this without first acquiring a lock. This allows applications that have not been carefully tuned for performance to take advantage of the benefits of fine-grain locking. It also helps simplify programming threaded applications, especially code sections that deal with synchronizing access to shared data. Transactions are a well-known database concept, and in the context of TM it is the sequence of storage operations that must be performed atomically that constitute a transaction. As with databases, the transaction must complete in its entirety (the speculative execution must complete without another thread attempting to access the same storage) or the transaction must be failed. Transactions, in short, provide synchronization guarantees that are similar to what is guaranteed when using a lock to protect the storage accesses.

Software using TM in general uses one of the following two approaches:

- ▶ Lock elision: First, try lock elision by converting a normally locked critical section to run as a transaction when there is less contention. Some parts of the software can use transactions, and other parts can still use normal lock acquisition semantics.
- ▶ TM aware software: Convert critical sections in software to use transactions as compared to locks. Add code to fall back to a lock-based method if contention is excessive.

GCC defines several new syntax extensions to support TM-based programming and the GLIBC library provides runtime support. For more information about GCC support, see the following website:

<http://gcc.gnu.org/wiki/TransactionalMemory>

Conflict detection (the fact that two transactions are affecting each other and hence they need to be failed) is based on cache line granularity on Power Systems. For more information about the TM implementation on Power Systems, see the following website:

<https://www.power.org/documentation/power-isa-transactional-memory/>

Although POWER8 processor-based systems support TM, you must explicitly check for support of TM before using the facility because the processor might be running in a compatibility mode, or the operating system or hypervisor might not support the use of TM. In Linux, the preferred API that determines whether TM is supported is to query the `AT_HWCAP2` field in the Auxiliary Vector (AUXV). The `libauxv` library provides easy functions for querying the AUXV. For more information, see the following website:

<https://github.com/Libauxv/libauxv>

Software failure handler

Upon transaction failure, the hardware redirects control to the failure handler that is associated with the outermost transaction. “Transaction failure” on page 43 explains this situation and provides details about how control is passed to the software failure handler and the machine state of the status registers. Section 7.3.5, “Transactional memory” on page 156 describes several built-in functions that the user can use, in addition to various macros in `htmintrin.h`, to determine the failure code for an failed transaction.

Transaction failure causes

There are several reasons why a particular transaction might fail. Some failures may be persistent, meaning retrying the transaction is pointless, and others might be more transitory in nature and retrying the transaction is fine. Some common reasons why transactions fail are noted in the following list:

- ▶ A **tabort.** instruction was run by the user program
- ▶ Cache line conflicts that are used by other processors
- ▶ Context switches
- ▶ Signals

Refer to the ISA documentation for a more extensive list of failure causes.

Making syscalls within an active transaction is problematic because some syscall side effects cannot be rolled back. Syscalls made within a suspended transaction are theoretically possible, but again, some of the syscall side effects might lead to transaction failure. For an in-depth description of transaction failures that are caused by the Linux kernel, see the following website:

https://git.kernel.org/cgit/linux/kernel/git/torvalds/linux.git/tree/Documentation/powerpc/transactional_memory.txt

Debugger support

The GDB debugger currently supports only machine-level debugging of TM programs. This support includes the ability to disassemble the new TM instructions. This support does not allow the setting of breakpoints within a transaction. Setting a breakpoint inside of a transaction causes the transaction to unconditionally fail whenever the breakpoint is encountered. To determine the cause and location of a failing transaction, set a breakpoint on the transaction failure handler, and then view the TEXASR and TFIAR registers when the breakpoint is encountered.

For more information about the topic of transactional memory, from the processor, OS, and compiler perspectives, see:

- ▶ 2.2.4, “Transactional memory” on page 42 (*processor*)
- ▶ 4.2.4, “Transactional memory” on page 89 (*AIX*)
- ▶ 7.3.5, “Transactional memory” on page 156 (*XL and GCC compiler families*)
- ▶ 8.4.2, “Transactional memory” on page 182 (*Java*)

6.2.5 Vector Scalar eXtension

GCC makes an interface available for PowerPC processors to access built-in functions. For more information about the various revisions of the GCC compiler, see the following website:

<http://gcc.gnu.org/onlinedocs>

For more information about the topic of Vector Scalar eXtension (VSX), from the processor, AIX, IBM i, and compiler perspectives, see:

- ▶ 2.2.5, “Vector Scalar eXtension” on page 45 (*processor*)
- ▶ 4.2.5, “Vector Scalar eXtension” on page 91 (*AIX*)
- ▶ 5.2.3, “Vector Scalar eXtension” on page 113 (*IBM i*)
- ▶ 7.3.2, “Compiler support for Vector Scalar eXtension” on page 151 (*XL and GCC compiler families*)

6.2.6 Decimal floating point

Decimal (base 10) data is widely used in commercial and financial applications. However, most computer systems have only binary (base two) arithmetic. There are two binary number systems in computers: integer (fixed-point) and floating point. Unfortunately, decimal calculations cannot be directly implemented with binary floating point. For example, the value 0.1 needs an infinitely recurring binary fraction, and a decimal number system can represent it exactly as 1/10th. So, using binary floating point cannot ensure that results are the same as those results that use decimal arithmetic.

In general, decimal floating point (DFP) operations are emulated with binary fixed-point integers. Decimal numbers are traditionally held in a binary-coded decimal (BCD) format. Although BCD provides sufficient accuracy for decimal calculation, it imposes a heavy cost in performance because it is implemented in software.

POWER6, POWER7, and POWER8 processor-based systems provide hardware support for DFP arithmetic. These microprocessor cores include a DFP unit that provides acceleration for the DFP arithmetic. The IBM POWER instruction set is expanded to include 54 new instructions that were added to support the DFP unit architecture. DFP can provide a performance boost for applications that are using BCD calculations.

How to take advantage of a DFP unit on POWER

You can take advantage of the DFP unit on POWER with the following features:¹

- ▶ Native DFP language support with a compiler

The C draft standard includes the following new data types (these are native data types, as are int, long, float, double, and so on):

<code>_Decimal32</code>	Seven decimal digits of accuracy
<code>_Decimal64</code>	Sixteen decimal digits of accuracy
<code>_Decimal128</code>	Thirty-four decimal digits of accuracy

Note: The `printf()` function uses new options to print these new data types:

- ▶ `_Decimal32` uses `%Hf`.
 - ▶ `_Decimal64` uses `%Df`.
 - ▶ `_Decimal128` uses `%DDf`.
- The IBM XL C/C++ Compiler, release 9 or later for AIX and Linux, includes native DFP language support. Here is a list of the compiler options for IBM XL compilers that are related to DFP:
 - `-qdfp`: Enables DFP support. This option makes the compiler recognize DFP literal suffixes, and the `_Decimal32`, `_Decimal64`, and `_Decimal128` keywords.

¹ How to compile DFPAL?, found at: <http://speleotrove.com/decimal/dfpal/compile.html>

- **-qfloat=dfpemulate**: Instructs the compiler to use calls to library functions to handle DFP computation, regardless of the architecture level. You might experience performance degradation when you use software emulation.
- **-qfloat=nodfpemulate** (the default when the **-qarch** flag specifies POWER6, POWER7, or POWER8): Instructs the compiler to use DFP hardware instructions.
- **-D__STDC_WANT_DEC_FP__**: Enables the referencing of DFP-defined symbols.
- **-ldfp**: Enables the DFP function that is provided by the Advance Toolchain on Linux.

For hardware supported DFP, with **-qarch=pwr6**, **-qarch=pwr7**, or **-qarch=pwr8**, use the following command:

```
cc -qdfp
```

For software emulation of DFP (on earlier processor chips), use the following command:

```
cc -qdfp -qfloat=dfpemulate
```

- The GCC compilers for Power Systems also include native DFP language support.

As of SLES 11 SP1 and RHEL 6, and in accord with the Institute of Electrical and Electronics Engineers (IEEE) 754R, DFP is fully integrated with compiler and runtime (printf and DFP math) support. For older Linux distribution releases (RHEL 5/SLES 10 and earlier), you can use the freely available Advance Toolchain compiler and runtime libraries. The Advance Toolchain runtime libraries can also be integrated with recent XL (V9+) compilers for DFP exploitation.

The latest Advance Toolchain compiler and run times can be downloaded from the following website:

<ftp://ftp.unicamp.br/pub/linuxpatch/toolchain/at/>

Advance Toolchain is a self-contained toolchain that does not rely on the base system toolchain for operability. In fact, it is designed to coexist with the toolchain that is shipped with the operating system. You do not have to uninstall the regular GCC compilers that come with your Linux distribution to use the Advance Toolchain.

The latest Enterprise distributions and Advance Toolchain run time use the Linux CPU tune library capability to select automatically hardware DFP or software implementation library variants, which are based on the hardware platform.

Here is a list of GCC compiler options for Advance Toolchain that are related to DFP:

- **-mhard-dfp** (the default when **-mcpu=power6**, **-mcpu=power7** or **-mcpu=power8** is specified): Instructs the compiler to take direct advantage of DFP hardware instructions for decimal arithmetic.
- **-mno-hard-dfp**: Instructs the compiler to use calls to library functions to handle DFP computation, regardless of the architecture level. If your application is dynamically linked to the `libdfp` variant and running on POWER6, POWER7, or POWER8 processors, then the run time automatically binds to the `libdfp` variant that is implemented with hardware DFP instructions. Otherwise, the software DFP library is used. You might experience performance degradation when you use software emulation.
- **-D__STDC_WANT_DEC_FP__**: Enables the reference of DFP defined symbols.
- **-ldfp**: Enables the DFP function that is provided by recent Linux Enterprise Distributions or the Advance Toolchain run time.

- Decimal Floating Point Library (`libdfp`) is an implementation of the joint efforts of the International Organization for Standardization and the International Electrotechnical Commission (ISO/IEC). ISO/IEC technical report ISO/IEC TR 24732² describes the C-Language library routines that are necessary to provide the C library runtime support for decimal floating point data types, as introduced in IEEE 754-2008, namely `_Decima132`, `_Decima164`, and `_Decima1128`.

The library provides functions, such as `sin` and `cos`, for the decimal types that are supported by GCC. Current development and documentation can be found at <https://github.com/libdfp/libdfp>, and RHEL6 and SLES11 provide this library as a supplementary extension. Advance Toolchain also ships with the library.

Determining whether your applications are using DFP

The Linux `perf` tool is used for application performance profiling. The `PM_MRK_DFU_FIN` performance counter event indicates that the Decimal Floating Point Unit finished a marked instruction. To profile an application for `PM_MRK_DFU_FIN` samples, use `perf` to set the event name and sample count and run the application:

```
perf -e PM_MRK_DFU_FIN:1000 application
```

To view the results and see what symbols the event samples are associated with, run the following command:

```
opreport --symbols
```

If you see this message, there were no samples found for the event that is specified when running the application:

```
opreport error: No sample file found
```

For more information about this topic, from the processor and OS perspectives, see:

- 2.2.6, “Decimal floating point” on page 47 (*processor*)
- 4.2.6, “Decimal floating point” on page 92 (*AIX*)
- 5.2.4, “Decimal floating point” on page 113 (*IBM i*)

For more information, see 6.5, “Related publications” on page 139.

6.2.7 Event-based branches

The event-based branching (EBB) facility is a new Power Architecture ISA 2.07 hardware facility, under [Category:Server], that generates event-based exceptions when a certain event criteria is met. Currently, ISA 2.07 (on POWER8 hardware) defines only one type of EBB: the performance monitoring unit (PMU) EBB. Following an EBB exception, the branch event status and control register (BESCR) tells which kind of event triggered the exception.

The EBB facility is a per-hardware-thread problem-state facility with access to the PMU and initialization under privileged-state. A problem-state application with direct access to the facility registers a callback function as an EBB handler by setting the handler address into the EBBHR register.

When a specified or requested PMU overflows, an exception is generated and as a result, the problem-state application EBB handler is started. Execution continues in event-based exception context until the handler returns control to the address in the event-based branch return register (EBBRR) by using the `rfebb` instruction.

² Information technology -- Programming languages, their environments, and system software interfaces -- Extension for the programming language C to support decimal floating-point arithmetic, found at: http://www.iso.org/iso/catalogue_detail.htm?csnumber=38842

There are interoperability considerations with the Power Architecture Executable File and Linkable format (ELF) application binary interface (ABI), and these can complicate the usage of this facility to remain ABI compliant. As a result, user applications should use an API that is provided by `libpaf-ebb` that handles the ABI implications consistently and correctly and provides a handler by proxy.

For more information about this topic, from the processor perspective, see 2.2.12, “Event-based branches (or user-level fast interrupts)” on page 52 (*processor*).

For more information about EBB, see the following website:

<https://github.com/paflib/paflib/wiki/Event-Based-Branching----Overview,-ABI,-and-API>

6.3 Linux operating system-specific optimizations

This section describes optimization methods that are specific to Linux.

6.3.1 GCC, toolchain, and IBM Advance Toolchain

This section describes 32-bit and 64-bit modes and CPU-tuned libraries.

Linux support for 32-bit and 64-bit modes

The compiler and run time can support either 32-bit or 64-bit mode applications simultaneously. The compilers can select the target mode through the `-m32` or `-m64` compiler options.

For the SLES, Ubuntu, and RHEL distributions, the shared libraries have both 32-bit and 64-bit versions. The toolchain (compiler, assembly language, linker, and dynamic linker) selects the correct libraries based on the `-m32` or `-m64` option or the mode of the application program.

The Advance Toolchain defaults to 64-bit, as do SLES 11, RHEL 6, and RHEL7. POWER Little Endian distributions support only 64-bit mode, such as SLES 12, RHEL 7, and Ubuntu 14.04.

Applications can use 32-bit and 64-bit execution modes, depending on their specific requirements, if their dependent libraries are available for the wanted mode.

The 32-bit mode is lighter with a simpler function call sequence and smaller footprint for stack and C++ objects, which can be important for some dynamic language interpreters and applications with many small functions.

The 64-bit mode has a larger footprint because of the larger pointer and general register size, which can be an asset when you handle large data structures or text data, where larger (64-bit) general registers are used for high bandwidth in the memory and string functions.

Linux on Power also supports 64-bit direct memory access (DMA), which can have a significant impact on I/O performance. For more information, see *Taking Advantage of 64-bit DMA capability on PowerLinux*, found at:

https://www.ibm.com/developerworks/community/wikis/home?lang=en#!/wiki/W51a7ffcf4df4b40_9d82_446ebc23c550/page/Taking%20Advantage%20of%2064-bit%20DMA%20capability%20on%20PowerLinux

The handling of floating point and vector data is the same (registers size and format and instructions) for 32-bit and 64-bit modes. Therefore, for these applications, the key decision depends on the address space requirements. For 32-bit POWER applications (32-bit mode applications that are running on 64-bit POWER hardware with a 64-bit kernel), the address space is limited to 4 GB, which is the limit of a 32-bit address. 64-bit applications are limited to 16 TB of application program or data per process. This limitation is not a hardware one, but is a restriction of the shared Linux virtual memory manager implementation. For applications with low latency response requirements, using the larger, 64-bit addressing to avoid I/O latencies that use memory mapped files or large local caches is a good trade-off.

CPU-tuned libraries

If an application must support only one POWER hardware platform (such as POWER7 and later processor-based systems), then compiling the entire application with the appropriate **-mcpu=** and **-mtune=** compiler flags might be the best option.

For example, **-mcpu=power7** allows the compiler to use all the POWER7 instructions, such as the VSR category. The **-mcpu=power7** option also implies **-mtune=power7** if it is not explicitly set.

The GCC compiler does not have any specific POWER7+ optimizations, so use **-mcpu=power7** or **-mtune=power7**.

The **-mcpu=power8** option allows the compiler to use instructions that were added for the POWER8 processor, such as cryptography built-in functions, direct move instructions that allow data movement between the general-purpose registers and the floating point or floating vector registers, and additional vector scalar instructions that were introduced.

-mcpu generates code for a specific machine. If you specify **-mcpu=power7**, the code also runs on a POWER8 processor-based system, but not on a POWER6 processor-based system. **-mcpu=power6x** generates instructions that are not implemented on POWER7 or POWER8 processor-based systems, and **-mcpu=power6** generates code that runs on POWER7 and POWER8 processor-based systems. The **-mtune** option focuses on optimizing the order of the instructions.

Most applications do need to run on more than one platform, for example, in POWER7 mode and POWER8 mode. For applications composed of a main program and a set of shared libraries or applications that spend significant execution time in other (from the Linux run time or extra package) shared libraries, you can create packages that automatically select the best optimization for each platform.

Linux also supports automatic CPU tuned library selection. There are a number of implementation options for CPU tuned library implementers as described here. For more information, see *Optimized Libraries*, found at:

https://www.ibm.com/developerworks/community/wikis/home?lang=en#/wiki/W51a7ffcf4dfd_4b40_9d82_446ebc23c550/page/Optimized%20Libraries

The Linux Technology Center works with the SUSE, Canonical, and Red Hat Linux Distribution Partners to provide some automatic CPU-tuned libraries for the C/POSIX runtime libraries. However, these libraries might not be supported for all platforms or have the latest optimization.

One advantage of the Advance Toolchain is that the runtime RPMs for the current release do include CPU-tuned libraries for all the supported POWER processors and the latest processor-specific optimization and capabilities, which are constantly updated. Additional libraries are added as they are identified. The Advance Toolchain run time can be used with either Advance Toolchain GCC or XL compilers and includes configuration files to simplify linking XL compiled programs with the Advance Toolchain runtime libraries.

These techniques are not restricted to systems libraries, and can be easily applied to application shared library components. The dynamic code path and processor tuned libraries are good starting points. With this method, the compiler and dynamic linker do most of the work. You need only some additional build time and extra media for the multiple library images.

In this example, the following conditions apply:

- ▶ Your product is implemented in your own shared library, such as `libmyapp.so`.
- ▶ You want to support Linux running on POWER5, POWER6, POWER7, and POWER8 processor-based systems.
- ▶ DFP and Vector considerations:
 - Your oldest supported platform is a POWER5 processor-based system, which does not have a DFP or the Vector unit.
 - The POWER6 processor has DFP and a Vector Unit implementing the older Vector Multimedia eXtension (VMX) (vector float but no vector double) instructions.
 - POWER7 and POWER8 processors have DFP and the new VSX (the original VMX instructions plus Vector Double and more).
 - Your application benefits greatly from both Hardware Decimal and high performance vector, but if you compile your application with `-mcpu=power7 -03`, it does not run on POWER5 (no hardware DFP instructions) or POWER6 (no vector double instructions) processor-based systems.

You can optimize all three Power platforms if you build and install your application and libraries correctly by completing the following steps:

1. Build the main application binary file and the default version of `libmyapp.so` for the oldest supported platform (in this case, use `-mcpu=power5 -03`). You can still use decimal data because the Advance Toolchain and the newest SLES 11 and RHEL 6 include a DFP emulation library and run time.
2. Install the application (`myapp`) into the appropriate `./bin` directory and `libmyapp.so` into the appropriate `./lib64` directory. The following paths provide the application main and default run time for your product:
 - `/opt/ibm/myapp1.0/bin/myapp`
 - `/opt/ibm/myapp1.0/lib64/libmyapp.so`
3. Compile and link `libmyapp.so` with `-mcpu=power6 -03`, which enables the compiler to generate DFP and VMX instructions for POWER6 processor-based systems.
4. Install this version of `libmyapp.so` in to the appropriate `./lib64/power6` directory. For example:
`/opt/ibm/myapp1.0/lib64/power6/libmyapp.so`

5. Compile and link the fully optimized version of `libmyapp.so` for POWER7 processors with `-mcpu=power7 -O3`, which enables the compiler to generate DFP and all the VSX instructions. Install this version of `libmyapp.so` into the appropriate `./lib64/power7` directory. For example:

```
/opt/ibm/myapp1.0/lib64/power7/libmyapp.so
```

6. Compile and link the fully optimized version of `libmyapp.so` for the POWER8 processor with `-mcpu=power8 -O3`, which enables the compiler to generate DFP and all the VSX instructions. Install this version of `libmyapp.so` into the appropriate `./lib64/power8` directory. For example:

```
/opt/ibm/myapp1.0/lib64/power8/libmyapp.so
```

By simply running some extra builds, your `myapp1.0` is fully optimized for the current and N-1/N-2 POWER hardware releases. When you start your application with the appropriate `LD_LIBRARY_PATH` (including `/opt/ibm/myapp1.0/lib64`), the dynamic linker automatically searches the subdirectories under the library path for names that match the current platform (POWER5, POWER6, POWER7, or POWER8 processor-based systems). If the dynamic linker finds the shared library in the subdirectory with the matching platform name, it loads that version; otherwise, the dynamic linker looks in the base `lib64` directory and uses the default implementation. This process continues for all directories in the library path and recursively for any dependent libraries.

Using the Advance Toolchain

The latest Advance Toolchain compilers and run time can be downloaded from the following link:

<ftp://ftp.unicamp.br/pub/linuxpatch/toolchain/at>

The Advance Toolchain V7.0 was the first release with full Power ISA-2.07 POWER8 support, in addition to improved POWER7 optimization. Advance Toolchain V7.0 supports RHEL6, SLES11, and the Big Endian RHEL7 releases.

The Advance Toolchain V7.1 was the first release with full POWER8 Little Endian support, including the new ELF V2 64-bit ABI. Advance Toolchain V7.1 supports the Little Endian RHEL7.1 and Ubuntu-14.04 (Trusty) releases.

The Advance Toolchain V8.0 supports both Big and Little Endian POWER8 processor-based systems with a number of enhancements, including a GCC-4.9 compiler. Advance Toolchain V8.0 supports Big Endian RHEL7 and Little Endian RHEL7.1, SLES12, and Ubuntu 14.04 releases.

In addition, all the latest Advance Toolchain releases (starting with Advance Toolchain V5.0) add multi-core runtime libraries to enable you to take advantage of application level multi-cores. The toolchain currently includes a POWER port of the open source version of Intel Threading Building Blocks, the Concurrent Building Blocks software transactional memory library, the Userspace RCU library (the application level version of the Linux kernel's Read-Copy-Update concurrent programming technique), and the Shared Persistent Heap Data Environment library, which provides mechanisms to create persistent storage in a shared address space. Additional libraries are added to the Advance Toolchain run time as needed and if resources allow it.

Linux on Power Enterprise Distributions default to 64 KB pages, so most applications automatically benefit from large pages. Larger (16 MB) segments can be best used with the `libhugetlbfs` API, which is provided with Advance Toolchain. Large segments can be used to back shared memory, malloc storage, and (main) program text and data segments (incorporating large pages for shared library text or data is not supported).

6.3.2 Tuning and optimizing malloc

Methods for tuning and optimizing malloc are described in this section.

Linux malloc

Generally, tuning malloc invocations on Linux systems is an application-specific focus.

Improving malloc performance

Linux is flexible regarding the system and application tuning of malloc usage.

By default, Linux manages malloc memory to balance the ability to reuse the memory pool against the range of default sizes of memory allocation requests. Small chunks of memory are managed on the **sbrk** heap. This **sbrk** heap is labeled as [heap] in /proc/self/maps.

When you work with Linux memory allocation, there are a number of tunables available to users. These tunables are coded and used in the Linux malloc.c program. Our examples (“Malloc environment variables” on page 133 and “Linux malloc considerations” on page 133) show two of the key tunables, which force the large sized memory allocations away from using mmap, to using the memory on the program stack by using the **sbrk** system directive.

When you control memory for applications, the Linux operating system automatically makes a choice between using the stack for mallocs with the **sbrk** command, or mmap regions. mmap regions are typically used for larger memory chunks. When you use mmap for large mallocs, the kernel must zero the newly mmapmed chunk of memory.

Malloc environment variables

Users can define environment variables to control the tunables for a program. The environment variables that are shown in the following examples caused a significant performance improvement across several real-life workloads.

To disable the usage of mmap for mallocs (which includes Fortran allocates), set the max value to zero:

```
MALLOC_MMAP_MAX=0
```

To disable the trim threshold, set the value to negative one:

```
MALLOC_TRIM_THRESHOLD=-1
```

Trimming and using mmap are two different ways of releasing unused memory back to the system. When used together, they change the normal behavior of malloc across C and Fortran programs, which in some cases can change the performance characteristics of the program. You can run one of the following commands to use both actions:

- ▶ # ./my_program
- ▶ # MALLOC_MMAP_MAX=0 MALLOC_TRIM_THRESHOLD=-1 ./my_program

Depending on your application's behavior regarding memory and data locality, this change might do nothing or might result in performance improvement.

Linux malloc considerations

The Linux GNU C run time includes a default malloc implementation that is optimized for multi-threading and medium-sized allocations. For smaller allocations (less than the MMAP_THRESHOLD), the default malloc implementation allocates blocks of storage with **sbrk()** called arenas, which are then suballocated for smaller malloc requests. Larger allocations (greater than MMAP_THRESHOLD) are allocated by an anonymous mmap, one per request.

The default values are listed here:

DEFAULT_MXFAST	64 (for 32-bit) or 128 (for 64-bit)
DEFAULT_TRIM_THRESHOLD	128 * 1024
DEFAULT_TOP_PAD	0
DEFAULT_MMAP_THRESHOLD	128 * 1024
DEFAULT_MMAP_MAX	65536

Storage within arenas can be reused without kernel intervention. The default malloc implementation uses trylock techniques to detect contentions between POSIX threads, and then tries to assign each thread its own arena. This action works well when the same thread frees storage that it allocates, but it does result in more contention when malloc storage is passed between producer and consumer threads. The default malloc implementation also tries to use atomic operations and more granular and critical sections (lock and unlock) to enhance parallel thread execution, which is a trade-off for better multi-thread execution at the expense of a longer malloc path length with multiple atomic operations per call.

Large allocations (greater than MMAP_THRESHOLD) require a kernel syscall for each **malloc()** and **free()**. The Linux Virtual Memory Management (VMM) policy does not allocate any real memory pages to an anonymous **mmap()** until the application touches those pages. The benefit of this policy is that real memory is not allocated until it is needed. The downside is that, as the application populates the new allocation with data, the application experiences multiple page faults, on first touch to allocate and zero fill the page. This situation means that on the initial touching of memory, there is more processing then, as opposed to the earlier timing when the original mmap is done. In addition, this first touch timing can impact the NUMA placement of each memory page.

Such storage is unmapped by **free()**, so each new large malloc allocation starts with a flurry of page faults. This situation is partially mitigated by the larger (64 KB) default page size of the RHEL and SLES on Power Systems; there are fewer page faults than with 4 KB pages.

Malloc tuning parameters

The default malloc implementation provides a **mallopt()** API to allow applications to adjust some tuning parameters. For some applications, it might be useful to adjust the MMAP_THRESHOLD, TOP_PAD, and MMAP_MAX limits. Increasing MMAP_THRESHOLD so that most (application) allocations fall below that threshold reduces syscall and page fault impact, and improves application start time. However, this situation can increase fragmentation within the arenas and **sbrk()** storage. Fragmentation can be mitigated to some extent by also increasing TOP_PAD, which is the extra memory that is allocated for each **sbrk()**.

Reducing MMAP_MAX, which is the maximum number of chunks to allocate with **mmap()**, can also limit the use of **mmap()** when MMAP_MAX is set to 0. Reducing MMAP_MAX does not always solve the problem. The run time reverts to **mmap()** allocations if **sbrk()** storage, which is the gap between the end of program static data (bss) and the first shared library, is exhausted.

Linux malloc and memory tools

There are several readily available tools in the Linux open source community:

- ▶ A website that describes the heap profiler that is used at Google to explore how C++ programs manage memory, found at the following website:
<http://gperftools.googlecode.com/svn/trunk/doc/heapprofile.html>
- ▶ *Massif: a heap profiler*, found at:
<http://valgrind.org/docs/manual/ms-manual.html>

For more information about memory management tools, see “Empirical performance analysis by using the IBM Software Development Kit for Linux on Power” on page 233.

For more information about tuning malloc parameters, see *Malloc Tunable Parameters*, found at:

<http://www.gnu.org/software/libtool/manual/libc/Malloc-Tunable-Parameters.html>

Thread-caching malloc

Under some circumstances, an alternative malloc implementation can prove beneficial for improving application performance. Packaged as part of Google's Perftools package (<http://code.google.com/p/gperftools/?redir=1>), and in the IBM Advance Toolchain, this specialized malloc implementation can improve performance across a number of C and C++ applications.

Thread-caching malloc (TCMalloc) uses a thread-local cache for each thread and moves objects from the memory heap into the local cache as needed. Small objects with less than 32 KB are mapped into allocatable size-classes. A thread cache contains a singly linked list of free objects per size-class. Large objects are rounded up to a page size (4 KB) and handled by a central page heap, which is an array of linked lists.

For more information about how TCMalloc works, see *TCMalloc: Thread-Caching Malloc*, found at:

<http://gperftools.googlecode.com/svn/trunk/doc/tcmalloc.html>

The TCMalloc implementation is part of the gperftools project. For more information about this topic, go to the following website:

<http://code.google.com/p/gperftools/>

Usage

To use TCMalloc, link TCMalloc in to your application by using the `-ltcmalloc` linker flag by running the following command:

```
$ gcc [...] -ltcmalloc
```

You can also use TCMalloc in applications that you did not compile yourself by using `LD_PRELOAD` as follows:

```
$ LD_PRELOAD="/usr/lib/libtcmalloc.so"
```

These examples assume that the TCMalloc library is in `/usr/lib`. With the Advance Toolchain V5.0.4, the 32-bit and 64-bit libraries are in `/opt/at5.0/lib` and `/opt/at5.0/lib64`.

Using TCMalloc with huge pages

To use large pages with TCMalloc, complete the following steps:

1. Set the environment variables for `libhugetlbfs`.
2. Allocate the number of large pages from the system.
3. Set up the `libhugetlbfs` mount point.
4. Monitor large pages usage.

TCMalloc backs up the heap allocation on the large pages only.

Here is a more detailed version of these steps:

1. Set the environment variables for `libhugetlbfs` by running the following commands:
 - `# export TCMALLOC_MEMFS_MALLOC_PATH=/libhugetlbfs/`
 - `# export HUGETLB_ELFMAP=RW`
 - `# export HUGETLB_MORECORE=yes`

Where:

- `TCMALLOC_MEMFS_MALLOCC_PATH=/libhugetlbfs/` defines the `libhugetlbfs` mount point.
- `HUGETLB_ELFMAP=RW` allocates both RSS and BSS (text/code and data) segments on the large pages, which is useful for codes that have large static arrays, such as Fortran programs.
- `HUGETLB_MORECORE=yes` allows heap usage on the large pages.

2. Allocate the number of large pages from the system by running one of the following commands:

- **# echo *N* > /proc/sys/vm/nr_hugepages**
- **# echo *N* > /proc/sys/vm/nr_overcommit_hugepages**

Where:

- *N* is the number of large pages to be reserved. A peak usage of 4 GB by your program requires 256 large pages (4096/16).
- `nr_hugepages` is the static pool. The kernel reserves $N * 16$ MB of memory from the static pool to be used exclusively by the large pages allocation.
- `nr_overcommit_hugepages` is the dynamic pool. The kernel sets a maximum usage of *N* large pages and dynamically allocates or deallocates these large pages.

3. Set up the `libhugetlbfs` mount point by running the following commands:

- **# mkdir -p /libhugetlbfs**
- **# mount -t hugetlbfs hugetlbfs /libhugetlbfs**

4. Monitor large pages usage by running the following command:

```
# cat /proc/meminfo | grep Huge
```

This command produces the following output:

```
HugePages_Total:
HugePages_Free:
HugePages_Rsvd:
HugePages_Surp:
Hugepagesize:
```

Where:

- `HugePages_Total` is the total pages that are allocated on the system for LP usage.
- `HugePages_Free` is the total free memory available.
- `HugePages_Rsvd` is the total of large pages that are reserved but not used.
- `Hugepagesize` is the size of a single LP.

You can monitor large pages by NUMA nodes by running the following command:

```
# watch -d grep Huge /sys/devices/system/node/node*/meminfo
```

MicroQuill SmartHeap

MicroQuill SmartHeap is an optimized malloc that is used for SPECcpu2006 publishes for optimizing performance on selected benchmark components. For more information, see *SmartHeap for SMP: Does your app not scale because of heap contention?*, found at:

<http://www.microquill.com/smartheapsmp/index.html>

6.3.3 Large TOC -mmodel=medium optimization

The Linux ABI on the Power Architecture is enhanced to optimize larger programs. This ABI both simplifies an application build and improves overall performance.

Previously, the TOC (`-mfull-toc`) defaulted to a single instruction access form that restricts the total size of the TOC to 64 KB. This configuration can cause large programs to fail at compile or link time. Previously, the only effective workaround was to compile with the `-minimal-toc` option (which provides a private TOC for each source file). The minimal TOC strategy adds a level of indirection that can adversely impact performance.

The `-mmodel=medium` option extends the range of the TOC addressing to +/-2 GB. This setup eliminates most TOC-related build issues. Also, as the Linux ABI TOC includes Global Offset Table (GOT) and local data, you can enable a number of compiler- and linker-based optimizations, including TOC pointer relative addressing for local static and constant data. This setup eliminates a level of indirection and improves the performance of large programs.

Currently, this optimization is available on SLES 11 and RHEL 6 when you are using the system compilers if you use the `-mmodel=medium` option (it is not on by default with those compilers). This optimization is on by default when using Advance Toolchain V4.0 and later.

The medium and large code models are 64-bit only. If you use medium or large code models, you must not use the `-minimal-toc` option.

6.3.4 POWER7 based distro considerations

For distros that do not recognize the POWER8 processor core, the processor appears as a POWER7 processor, and normal SMT=4 rules apply. This includes the RHEL 6.5 and SLES 11 SP3 releases.

Applications running in this mode still benefit from the efficiencies of the newer processor core, improved cache and memory characteristics, and I/O performance improvements.

6.3.5 Microthreading considerations

Microthreading is an POWER8 feature that enables each POWER8 core with eight hardware threads to be split into four subcores (each with two hardware threads per subcore). This feature is supported only on Linux systems running under the PowerKVM hypervisor.

The individual subcores can run as separate virtual cores in the same VM (guest) or in different VMs at the same time. Each virtual core can run in single-threaded or SMT2 mode. PowerKVM requires that all cores on the system be either full cores or split into four smaller subcores each. Therefore, only VMs that are configured to run with a maximum of two threads per virtual core can be activated with split-core enabled.

The key benefits of microthreading are that it improves CPU resource usage and increases the number of virtual machines that can be concurrently supported per physical core.

The PowerKVM host runs in single-threaded (or SMT off) mode. Enabling microthreading requires first switching to SMT on mode on the host, then setting the number of subcores per core, and finally switching back to SMT off mode with the following commands in this specific order:

```
ppc64_cpu --smt=on
ppc64_cpu --subcores-per-core=4
ppc64_smt --smt=off
```

Important: No guests must be active when running these commands.

CPU numbering on a POWER8 host is usually 0, 8, 16, 24, and so on (in multiples of 8) because the other seven threads of each core (1 - 7, 9 - 15, 17 - 23, and so on) are disabled in SMT off mode. When switching to multithreading mode, CPUs 2, 4, and 6 (of the first core), CPUs 10, 12, and 14 (of the second core), and so on, also become active.

For more information about multithreading, see 5.3.2, “Microthreading”, in *IBM PowerKVM Configuration and Use*, SG24-8231.

6.4 Little Endian

Endianness is an operating mode of the processor that determines how multi-byte objects are arranged in memory. When memory objects are stored on physical media, this arrangement is also reflected there. In *Big Endian* systems, the most significant byte is stored first in memory, and in a *Little Endian* system, the least significant byte is stored first.

Power Architecture processors can support both Big Endian and Little Endian modes, but have used the Big Endian mode.

Applications that are written in an interpreted language (such as Java, Python, and Ruby) usually work on Little or Big Endian platforms with minimal to no changes required. However, applications that are compiled for one endian mode will not run on the other endian mode without at least a recompile. More work is needed if the code has made any assumptions about the order in which data has been stored (for example, code that manipulates data through pointer casting or bit fields, code that passes data to network without converting it to network byte order, or devices that require or assume a specific endian mode).

The Linux distributions on Power historically supported Big Endian mode, and recent releases run in Little Endian mode (ppc64le), as shown in the following list:

- ▶ SLES 11 - Big Endian only
- ▶ SLES 12 - Little Endian only
- ▶ RHEL 7.0 - Big Endian only
- ▶ RHEL 7.1 - Big Endian and Little Endian
- ▶ Ubuntu 14.04, 14.10, 15.04 - Little Endian only
- ▶ All Little Endian versions of Linux support only POWER8 processor-based systems

Open source applications now support Little Endian mode also on Power Systems. Many third-party and most IBM applications have migrated to Little Endian and work continues in optimizing them to run efficiently.

A new more efficient ABI has also been introduced for Little Endian mode, which is described in the next section.

6.4.1 Application binary interface

The POWER8 platform supports two application binary interfaces (ABIs) for processors running in 64-bit mode. The traditional ABI that is used with previous POWER processors remains in use with POWER8 processor-based systems operating in Big Endian mode (providing full compatibility with existing applications). POWER8 processor-based systems operating in Little Endian mode use the new Power Architecture 64-bit Executable and Linking Format (ELF) V2 ABI. Because POWER8 processor-based systems are the first to use the Little Endian capability, there are no existing applications with which compatibility must be maintained.

The new ELF V2 ABI differs from the traditional ABI in many ways. Some of those highlights are noted in the following list:

- ▶ The ELF V2 ABI improves the efficiency of function call sequences in the following ways:
 - It streamlines passing and returning of homogeneous aggregates of floating-point and vector values. For example, a small array of floating-point values may now be passed in registers rather than memory.
 - It removes a level of indirection from the calculation of the address of the target function.
 - It defines local entry points within functions so that calls occurring within a single compilation unit are more efficient.
- ▶ The ELF V2 ABI provides explicit support for Little Endian operation in the following ways:
 - It specifies the layout of Little Endian data in memory, including data that is aggregated into structures and unions.
 - It defines the layout of vector elements within vector registers on Little Endian systems.
 - It defines a unified vector programming interface that is appropriate for use on both Big Endian and Little Endian systems.

In addition, the thread-local storage (TLS) ABI has now been integrated into the ELF V2 ABI, rather than remaining a separate document.

The ELF V2 ABI is available from the OpenPOWER Connect website:

<https://www-03.ibm.com/technologyconnect/tgcm/TGCMServlet.wss?alias=OpenPOWER>

6.5 Related publications

The publications that are listed in this section are considered suitable for a more detailed description of the topics that are covered in this chapter:

- ▶ *Getting started with OProfile on PowerLinux* (contains references to the **operf** command):
<http://www-01.ibm.com/support/knowledgecenter/api/redirect/lxinfo/v3r0m0/topic/liacf/oprofgetstart.htm>
- ▶ *IBM PowerKVM Configuration and Use*, SG24-8231
- ▶ *POWER6 Decimal Floating Point (DFP)*, found at:
<https://www.ibm.com/developerworks/community/wikis/home?lang=en#!/wiki/Power+Systems/page/POWER6+Decimal+Floating+Point+%28DFP%29>
- ▶ *Power ISA Version 2.07*, found at:
<https://www.power.org/documentation/power-isa-v-2-07b/>

Refer to the following sections:

- Section 3.1: Program Priority Registers
 - Section 3.2: “or” Instruction
 - Section 4.3.4: Program Priority Register
 - Section 4.4.3: OR Instruction
 - Section 5.3.4: Program Priority Register
 - Section 5.4.2: OR Instruction
 - Book I – 4 Floating Point Facility
 - Book I – 5 Decimal Floating Point
 - Book I – 6 Vector Facility
 - Book I – 7 Vector-Scalar Floating Point Operations (VSX)
 - Book I – Chapter 5 Decimal Floating-Point.
 - Book II – 4.2 Data Stream Control Register
 - Book II – 4.3.2 Data Cache Instructions
 - Book II – 4.4 Synchronization Instructions
 - Book II – A.2 Load and Reserve Mnemonics
 - Book II – A.3 Synchronize Mnemonics
 - Book II – Appendix B. Programming Examples for Sharing Storage
 - Book III – 5.7 Storage Addressing
- *Red Hat Enterprise Linux 6 Performance Tuning Guide, Optimizing subsystem throughput in Red Hat Enterprise Linux 6, Edition 4.0*, found at:
http://docs.redhat.com/docs/en-US/Red_Hat_Enterprise_Linux/6/html-single/Performance_Tuning_Guide/index.html
- *SMT settings*, found at:
<http://www.ibm.com/support/knowledgecenter/POWER7/p7hc3/iphc3attributes.htm?cp=POWER7%2F1-8-3-7-2-0-3>
- *Simultaneous multithreading*, found at:
<http://www-01.ibm.com/support/knowledgecenter/api/redirect/lxinfo/v3r0m0/index.jsp?topic=%2F1aai.hpctune%2Fsmtsetting.htm>
- *SUSE Linux Enterprise Server System Analysis and Tuning Guide (Version 11 SP3)*, found at:
http://www.suse.com/documentation/sles11/pdfdoc/book_sle_tuning/book_sle_tuning.pdf



Compilers and optimization tools for C, C++, and Fortran

This chapter describes the optimization and tuning of the POWER8 processor-based servers by using compilers and tools. It covers the following topics:

- ▶ 7.1, “Compiler versions and optimization levels” on page 142
- ▶ 7.2, “Advanced compiler optimization techniques” on page 143
- ▶ 7.3, “Capitalizing on POWER8 features with the XL and GCC compilers” on page 148
- ▶ 7.4, “IBM Feedback Directed Program Restructuring” on page 160
- ▶ 7.5, “Using the Advance Toolchain with IBM XLC and XLF” on page 169
- ▶ 7.7, “Related publications” on page 171

7.1 Compiler versions and optimization levels

The IBM XL compilers are updated periodically to improve application performance and add processor-specific tuning and capabilities. The XLC13 and XLF15 compilers for AIX and Linux are the first versions to include the capabilities of the POWER8 processor, and are the preferred version for projects that target current generation systems.

For the GNU GCC, G++, and gfortran compilers on Linux, the IBM Advance Toolchain V7.0 (GCC 4.8) has the POWER8 processor enabled. The normal distribution version of GCC 4.8 does not have POWER8 support. Recent Enterprise Linux distributions with GCC 4.8 or later compilers are fully enabled for the POWER8 processor. GCC and POWER8 support are continuously being improved, so additional POWER8 optimizations are expected in later versions of GCC (for example, the IBM Advance Toolchain V8.0 based on GCC 4.9). XLF is preferred over gfortran for its high floating point performance characteristics.

For all production codes, it is imperative to enable a minimum level of compiler optimization by adding the **-O2** option for the XL compilers (**-O** is equivalent to **-O2**) and for the GNU compilers (**-O3** is the preferred option for GNU). Without optimization, the focus of the compiler is on faster compilation and debug ability, and it generates code that performs poorly at run time. In practice, many projects set up a dual build environment, with a development build without optimization for use during development and debugging, and a production build with optimization to be used for performance verification and production delivery.

For projects with increased focus on runtime performance, take advantage of the more advanced compiler optimization. For numerical or compute-intensive codes, the XL compiler options **-O3** or **-qhot -O3** enable loop transformations, which improve program performance by restructuring loops to make their execution more efficient by the target system. These options perform aggressive transformations that can sometimes cause minor differences on precision of floating point computations. If that is a concern, the original program semantics can be fully recovered with the **-qstrict** option.

For GCC, the minimum suggested level of optimization is **-O3**. The GCC default is a strict mode, but the **-ffast-math** option disables strict mode. The **-Ofast** option combines **-O3** with **-ffast-math** in a single option. Other important options include **-fpeel-loops**, **-funroll-loops**, **-ftree-vectorize**, **-fvect-cost-model**, and **-mcmmodel=medium**.

By default, these compilers generate code that run on various Power Systems. Options should be added to exclude older processor chips that are not supported by the target application. This configuration might enable better code generation as the compiler takes advantage of capabilities not available on those older systems.

There are two major XL compiler options to control this support:

- ▶ **-qarch**: Indicates the oldest processor chip generation that the binary file supports.
- ▶ **-qtune**: Indicates the processor chip generation of most interest for performance. It allows specification of a target SMT mode to direct optimizations for best performance in that mode.

For example, for an application that must run on POWER7 processor-based systems, but for which most users are on a POWER8 processor-based system, the appropriate combination is **-qarch=pwr7 -qtune=pwr8**. For an application that must run well across both POWER7 and POWER8 processor-based systems in current common usage, consider using **-qtune=balanced**.

On GCC, the equivalent options are **-mcpu** and **-mtune**. So, for an application that must run on a POWER7 processor-based system, but that is usually run on a POWER8 processor-based system, the options are **-mcpu=power7** and **-mtune=power8**.

The XLC13 and XLF15 compilers for AIX and Linux introduce an extension to **-qtune** to indicate the SMT mode in which the application will most often run. For example, for an application that must run on POWER8 processor-based systems and will most often run in SMT4 mode (for example, four hardware threads per core), use **-qarch=pwr8 -qtune=pwr8:smt4**. If the same application might run in several different SMT modes, consider using **-qarch=pwr8 -qtune=pwr8:balanced**.

The POWER8 processor supports the Vector Scalar eXtension (VSX) instruction set, which improves performance for numerical applications over regular data sets. These performance features can increase the performance of some computations, and can be accessed manually by using the AltiVec vector extensions, or automatically by the XL compiler by using **-O3** or above with **-qarch=pwr7** or **-qarch=pwr8**. By default, these options implicitly enable **-qsimd**, which allows the XL compilers to transform loops in an application to use VSX instructions. The POWER8 processor includes several extensions to the Vector Multimedia eXtension (VMX) and VSX instruction sets, which can improve performance of applications by using 64-bit integer types and single-precision floating point.

The GCC compiler equivalents are the **-maltivec** and **-mvsx** options, which you should combine with **-ftree-vectorize** and **-fvect-cost-model**. On GCC, the combination of **-O3** and **-mcpu=power7** or **-mcpu=power8** implicitly enables AltiVec and VSX code generation with auto-vector (**-ftree-vectorize**) and **-mpopcntd**. Other important options include **-mrecip=rsqrt** and **-mveclibabi=mass** (which *require* **-ffast-math** or **-Ofast** to be effective). If the compiler uses optimizations that depend on the MASS libraries, the link command must explicitly name the MASS library directories and library names.

For more information about this topic, see 7.7, “Related publications” on page 171.

7.2 Advanced compiler optimization techniques

This section describes some of the more advanced compiler optimization techniques.

7.2.1 Common prerequisites

Compiler analysis and transformations improve runtime performance by changing the translation of the program source into assembly code. Changes in these translations might cause the application to behave differently, possibly even causing it to produce incorrect results.

Compilers follow rules and assumptions that are part of the programming language to perform this transformation. If the programmer breaks some of these rules, it is possible for the application to malfunction, and it might do so only at higher optimization levels, where it is more difficult for the problem to be diagnosed.

To put this situation into perspective, imagine a C program with three variables: “int a[4], b, c;”. These variables are normally placed contiguously in memory. If the user runs a statement of the form `a[5]=0`, this statement breaks the language rules, but if variable `b` is unused, the statement might overwrite variable `b` and the program might continue to behave correctly. However, if, at a higher optimization level, variable `b` is eliminated, as the compiler determines it is unused, the incorrect statement might overwrite variable `c`, triggering a runtime failure.

It is critical, then, to eliminate programming errors as higher optimization is applied. Testing the application thoroughly without optimization is a good initial step, but it is not required or sufficient. The application must be tested at the optimization level to be used in production.

7.2.2 XL compiler family

The XL compilers provide static analysis and runtime checking to allow users to detect and correct source code problems.

Prerequisites

The XL compilers assist with identifying certain programming errors that are outlined 7.2.1, “Common prerequisites” on page 143:

- ▶ **Static analysis/warnings:** The XL compilers can identify suspicious code constructs, and provide some information about these constructs through the **-qinfo=all** option. Examine the output of this option to identify suspicious code constructs and validate that the constructs are correct.
- ▶ **Runtime analysis or warning:** The XL compilers can cause the application to perform runtime checks to validate program correctness by using the **-qcheck** option. This option triggers a program abort when an error condition (such as a null pointer dereference or out-of-bounds array access) is run, identifying a problem and making it easier for you to identify it. This option has a significant performance cost, so use it only during functional verification, not on a production environment.
- ▶ **Aliasing compliance:** The C, C++, and Fortran languages specify rules that govern the access of data through overlapping pointers. These rules are brought into play aggressively by optimization techniques, but they can lead to incorrect results if they are broken. The compiler can be instructed not to take advantage of these rules, at a cost of runtime performance. This situation can be useful for older code that is written without following these rules. The options to request this optimization are **-qalias=noansi** for C/C++ and **-qalias=nostd** for Fortran.

The XLC13 and XLF15 compilers include enhancements to **-qinfo** and **-qcheck** to detect accesses to uninitialized variables and stack corruption or stack clobbering.

High-order transformations

The XL compilers have sophisticated optimizations to improve the performance of numeric applications. These applications often contain regular loops that process large amounts of data. The high-order transformation (HOT) optimizations in these compilers analyze these loops, identify opportunities for restructuring them to improve cache usage, improve data reuse, and expose more instruction-level parallelism to the hardware. For these types of applications, the performance impact of this option can be substantial.

There are two levels of aggressiveness to the HOT optimization framework in these compilers:

- ▶ **Level 0**, which is the default at optimization level **-O3**, performs a minimal amount of loop optimization, focusing on simple opportunities and minimizing compilation time.
- ▶ **Level 1**, which is the default at optimization levels **-O4** and up, performs full loop analysis and transformation of loops.

The HOT optimizations can be explicitly requested through the **-qhot=level=0** and **-qhot=level=1** options. The **-qhot** option alone enables **-qhot=level=1**. The **-O3 -qhot** options are preferred for numerical applications.

OpenMP

The OpenMP API is an industry specification for shared-memory parallel programming. The latest XL Compilers provide a full implementation of the OpenMP 3.1 specification and partial support of the OpenMP4.0 specification in C, C++, and Fortran. You can program with OpenMP to capitalize on the incremental introduction of parallelism in an existing application by adding pragmas or directives to specify how the application can be parallelized.

For applications with available parallelism, OpenMP can provide a simple solution for parallel programming without requiring low-level thread manipulation. The OpenMP implementation on the XL compilers is available by using the **-qsmp=omp** option.

Whole-program analysis

Traditional compiler optimizations operate independently on each application source file. Inter-procedural optimizations operate at the whole-program scope by using the interaction between parts of the application on different source files. It is often effective for large-scale applications that are composed of hundreds or thousands of source files.

On the XL compilers, these capabilities are accessed by using the **-qipa** option. It is also implied when you use optimization levels **-O4** and **-O5**. In this phase, the compiler saves a high-level representation of the program in the object files during compilation, and reoptimizes it at the whole-program scope during the link phase. For this situation to occur, the compiler driver must be used to link the resulting binary file instead of starting the system linker directly.

Whole-program analysis (IPA) is effective on programs that use many global variables, overflowing the default AIX limit on global symbols. If the application requires the use of the **-bbigtoc** option to link successfully on AIX, it is likely a good candidate for IPA optimization.

There are three levels of IPA optimization on the XL compilers (0, 1, and 2). By default, **-qipa** implies **ipa=level=1**, which performs basic program restructuring. For more aggressive optimization, apply **-qipa=level=2**, which performs full program restructuring during the link step. The time that it takes to complete the link step can increase significantly.

Optimization that is based on Profile Directed Feedback

Profile-based optimization allows the compiler to collect information about the program behavior and use that information when you make code generation decisions. It involves compiling the program twice: first, to generate an *instrumented* version of the application that collects program behavior data when run, and a second time to generate an optimized binary file by using information that is collected by running the instrumented binary file through a set of typical inputs for the application.

Profile-based optimization in the XL compiler is accessed through the **-qpdf1** and **-qpdf2** options, on top of **-O** or higher optimization levels. The instrumented binary file is generated by using **-qpdf1** on top of all other options, and the resulting binary file generates the profile data on a file, named **._pdf** by default.

The Profile Directed Feedback (PDF) framework on the XL compilers is built on top of the IPA infrastructure, with **-qpdf1** and **-qpdf2** implying **-qipa=level=0**. For the PDF2 step, it is possible to reuse the object files from the **-qpdf1** compilation step, and relink only the application with the **-qpdf2** option.

For PDF optimizations to be successful, the instrumented workload must be run with common workloads that reflect common usage of the application. Use multiple workloads that can exercise the program in different ways. The data for all instrumentation runs is aggregated into a single PDF file and used during optimization.

For the PDF profile data to be written out at the end of execution, the program must either implicitly or explicitly call the `exit()` library subroutine. Using `exit()` causes code that is introduced as part of the PDF instrumentation to be run and write out the PDF profile data. In contrast, running the `_exit()` system call skips the writing of the PDF profile data file, which results in inaccurate profile data being recorded.

7.2.3 GCC compiler family

The information in this section applies specifically to the GCC compiler family.

Prerequisites

The GCC compiler assists with identifying certain programming errors that are outlined in 7.2.1, “Common prerequisites” on page 143:

- ▶ Static analysis and warnings. The `-pedantic` and `-pedantic-errors` options warn of violations of ISO C or ISO C++ standards.
- ▶ The language standard to enforce and the aliasing compliance requirements are specified by the `-std`, `-ansi`, and `-fno-strict-aliasing` options. For example:
 - ISO C 1990 level: `-std=c89`, `-std=iso9899:1990`, and `-ansi`
 - ISO C 1998 level: `-std=c99` and `-std=iso9899:1999`
 - Do not assume strict aliasing rules for the language level: `-fno-strict-aliasing`

The GCC compiler documentation contains more details about these options.^{1, 2, 3}

High-order transformations (HOTs)

The GCC compilers have sophisticated additional optimizations beyond `-O3` to improve the performance of numeric applications. These applications often contain regular loops that process large amounts of data. These optimizations, when enabled, analyze these loops, identify opportunities for restructuring them to improve cache usage, improve data reuse, and expose more instruction-level parallelism to the hardware. For these types of applications, the performance impact of this option can be substantial. The key compiler options include:

- ▶ `-fpeel-loops`
- ▶ `-funroll-loops`
- ▶ `-ftree-vectorize`
- ▶ `-fvect-cost-model`
- ▶ `-mcmmodel=medium`

Specifying the `-mveclibabi=mass` option and linking to the MASS libraries enables more loops for `-ftree-vectorize`. The MASS libraries support only static archives for linking, and so they require explicit naming and library search order for each platform/mode:

- ▶ POWER8 32-bit: `-L<MASS-dir>/lib -lmassvp8 -lmass_simdp8 -lmass -lm`
- ▶ POWER8 64-bit: `-L<MASS-dir>/lib64 -lmassvp8_64 -lmass_simdp8_64 -lmass_64 -lm`
- ▶ POWER7 32-bit: `-L<MASS-dir>/lib -lmassvp7 -lmass_simdp7 -lmass -lm`
- ▶ POWER7 64-bit: `-L<MASS-dir>/lib64 -lmassvp7_64 -lmass_simdp7_64 -lmass_64 -lm`
- ▶ POWER6 32-bit: `-L<MASS-dir>/lib -lmassvp6 -lmass -lm`
- ▶ POWER6 64-bit: `-L<MASS-dir>/lib64 -lmassvp6_64 -lmass_64 -lm`

¹ *Language Standards Supported by GCC*, found at:

<http://gcc.gnu.org/onlinedocs/gcc-3.4.2/gcc/Standards.html#Standards>

² *Options Controlling C Dialect*, found at:

<http://gcc.gnu.org/onlinedocs/gcc-3.4.2/gcc/C-Dialect-Options.html#C-Dialect-Options>

³ *Options That Control Optimization, and specifically the discussion of -fstrict-aliasing*, found at:

<http://gcc.gnu.org/onlinedocs/gcc-3.4.2/gcc/Optimize-Options.html#Optimize-Options>

ABI improvements

The `-mmodel={medium|large}` option implements important ABI improvements that are further optimized in hardware for future generations of the POWER processor. This optimization extends the Table-Of-Content (TOC) to 2 GB and eliminates the previous requirement for `-minimal-toc` or multi-TOC switching within a single program or library. The default for newer GCC compilers (including Advance Toolchain V4.0 and later) is `-mmodel=medium`. This model logically extends the TOC to include local static data and constants and allows direct data access relative to the TOC pointer.

OpenMP

The OpenMP API is an industry specification for shared-memory parallel programming. The current GCC compilers, starting with GCC- 4.4 (Advance Toolchain V4.0 and later), provide a full implementation of the OpenMP 3.0 specification in C, C++, and Fortran. Programming with OpenMP allows you to benefit from the incremental introduction of parallelism in an existing application by adding pragmas or directives to specify how the application can be parallelized.

For applications with available parallelism, OpenMP can provide a simple solution for parallel programming, without requiring low-level thread manipulation. The GNU OpenMP implementation on the GCC compilers is available under the `-fopenmp` option. GCC also provides auto-parallelization under the `-ftree-parallelize-loops` option.

Whole-program analysis

Traditional compiler optimizations operate independently on each application source file. Inter-procedural optimizations operate at the whole-program scope, by using the interaction between parts of the application on different source files. It is often effective for large-scale applications that are composed of hundreds or thousands of source files.

Starting with GCC- 4.6 (Advance Toolchain V5.0), there is the Link Time Optimization (LTO) feature. LTO allows separate compilation of multiple source files but saves additional (an abstract program description) information in the resulting object file. Then, at application link time, the linker can collect all the objects (with additional information) and pass them back to the compiler (GCC) for whole program whole-program analysis (IPA) and final code generation.

The GCC LTO feature is enabled during the compile and link phases by the `-flto` option. A simple example follows:

```
gcc -flto -O3 -c a.c
gcc -flto -O3 -c b.c
gcc -flto -o program a.o b.o
```

Additional options that can be used with `-flto` include:

- ▶ `-flto-partition={lto1|balanced|none}`
- ▶ `-flto-compression-level=n`

Detailed descriptions about `-flto` and its related options are in *Options That Control Optimization*, found at:

<http://gcc.gnu.org/onlinedocs/gcc-4.6.3/gcc/Optimize-Options.html#Optimize-Options>

Profiled-based optimization

Profile-based optimization allows the compiler to collect information about the program behavior and use that information when you make code generation decisions. It involves compiling the program twice: first, to generate an *instrumented* version of the application that collects program behavior data when run, and a second time to generate an optimized binary by using information that is collected by running the instrumented binary file through a set of typical inputs for the application.

Profile-based optimization in the GCC compiler is accessed through the **-fprofile-generate** and **-fprofile-use** options on top of **-O2** optimization levels. The instrumented binary file is generated by using **-fprofile-generate** on top of all other options, and the resulting binary file generates the profile data in a file, named `._pdf` by default. For example:

```
gcc -fprofile-generate -O3 -c a.c
gcc -fprofile-generate -O3 -c b.c
gcc -fprofile-generate -o program a.o b.o
program < sample1
program < sample2
program < sample3
gcc -fprofile-use -O3 -c a.c
gcc -fprofile-use -O3 -c b.c
gcc -fprofile-use -o program a.o b.o
```

Additional options that are related to GCC PDF include:

- | | |
|--------------------------------|--|
| -fprofile-correction | Corrects for missing counter-samples from multi-threaded applications. |
| -fprofile-dir=PATH | Specifies the directory for generating and using profile data. |
| -fprofile-generate=PATH | Combines -fprofile-generate and -fprofile-dir . |
| -fprofile-use=PATH | Combines -fprofile-use and -fprofile-dir . |

Detailed descriptions about **-fprofile-generate** and its related options can be found in *Options That Control Optimization*, found at:

<http://gcc.gnu.org/onlinedocs/gcc-4.6.3/gcc/Optimize-Options.html#Optimize-Options>

For more information about this topic, see 7.7, “Related publications” on page 171.

7.3 Capitalizing on POWER8 features with the XL and GCC compilers

This section describes built-in functions that are provided by the XL and GCC compiler families for high-level language access to new POWER8 features and instructions.

7.3.1 In-core cryptography

The GCC, XL C/C++, and XL Fortran compilers provide built-in functions for the in-core cryptography instructions. For GCC, the following built-in functions require **-mcpu=power8** or **-mcrypto**. For the XL compiler family, **-qarch=pwr8** is required.

AES

The following built-in functions are provided for the implementation of the AES algorithm:

► **vsbox**

- GCC: vector unsigned long long **__builtin_crypto_vsbox** (vector unsigned long long)
- XL C/C++: vector unsigned char **__vsbox** (vector unsigned char) XLF: **VSBOX** (ARG1), where ARG1 and result are unsigned vector types of kind 1

► **vcipher**

- GCC: vector unsigned long long **__builtin_crypto_vcipher** (vector unsigned long long, vector unsigned long long)
- XL C/C++: vector unsigned char **__vcipher** (vector unsigned char, vector unsigned char)
- XLF: **VCIPHER** (ARG1,ARG2), where ARG1, ARG2, and result are unsigned vector types of kind 1

► **vcipherlast**

- GCC: vector unsigned long long **__builtin_crypto_vcipherlast** (vector unsigned long long, vector unsigned long long)
- XL C/C++: vector unsigned char **__vcipherlast** (vector unsigned char, vector unsigned char)
- XLF: **VCIPHERLAST** (ARG1,ARG2), where ARG1, ARG2, and result are unsigned vector types of kind 1

► **vncipher**

- GCC: vector unsigned long long **__builtin_crypto_vncipher** (vector unsigned long long, vector unsigned long long)
- XL C/C++: vector unsigned char **__vncipher** (vector unsigned char, vector unsigned char)
- XLF: **VNCIPHER** (ARG1,ARG2), where ARG1, ARG2, and result are unsigned vector types of kind 1

► **vncipherlast**

- GCC: vector unsigned long long **__builtin_crypto_vncipherlast** (vector unsigned long long, vector unsigned long long)
- XL C/C++: vector unsigned char **__vncipherlast** (vector unsigned char, vector unsigned char)
- XLF: **VNCIPHERLAST** (ARG1,ARG2), where ARG1, ARG2, and result are unsigned vector types of kind 1

For more information, see “AES” on page 47.

AES Galois Counter Mode

The following built-in functions are provided for the implementation of the Galois Counter Mode (GCM) of AES:

vpmsumd :

- GCC: vector unsigned long long **__builtin_crypto_vpmsum** (vector unsigned long long, vector unsigned long long)
- XL C/C++: vector unsigned long long **__vpmsumd** (vector unsigned long long, vector unsigned long long)

- XLF: **VPMSUMD** (ARG1, ARG2), where ARG1, ARG2, and result are unsigned vector types of kind 8

For more information, see “AES special mode of operation: Galois Counter Mode” on page 48.

SHA-2

The following built-in functions are provided for the implementation of SHA-2 hash functions:

- **vshasigmad**
 - GCC: vector unsigned long long **__builtin_crypto_vshasigmad** (vector unsigned long long, int, int)
 - XL C/C++: vector unsigned long long **__vshasigmad** (vector unsigned long long, int, int)
 - XLF: **VSHASIGMAD** (ARG1,ARG2,ARG3), where ARG1 and result are unsigned vector types of kind 8, and ARG2 and ARG3 are integer types
- **vshasigmaw**
 - GCC: vector unsigned int **__builtin_crypto_vshasigmaw** (vector unsigned int, int, int)
 - XL C/C++: vector unsigned int **__vshasigmaw** (vector unsigned int, int, int)
 - XLF: **VSHASIGMAW** (ARG1,ARG2,ARG3), where ARG1 and result are unsigned vector types of kind 4, and ARG2 and ARG3 are integer types

For more information, see “SHA-2” on page 48.

CRC

The following built-in functions are provided for the implementation of the CRC algorithm:

- **vpmsumd**
 - GCC: vector unsigned long long **__builtin_crypto_vpmsum** (vector unsigned long long, vector unsigned long long)
 - XL C/C++: vector unsigned long long **__vpmsumd** (vector unsigned long long, vector unsigned long long)
 - XLF: **VPMSUMD** (ARG1, ARG2), where ARG1, ARG2, and result are unsigned vector types of kind 8
- **vpmsumw**
 - GCC: vector unsigned int **__builtin_crypto_vpmsum** (vector unsigned int, vector unsigned int)
 - XL C/C++: vector unsigned int **__vpmsumw** (vector unsigned int, vector unsigned int)
 - XLF: **VPMSUMW** (ARG1, ARG2), where ARG1, ARG2, and result are unsigned vector types of kind 4
- **vpmsumh**
 - GCC: vector unsigned short **__builtin_crypto_vpmsum** (vector unsigned short, vector unsigned short)
 - XL C/C++: vector unsigned short **__vpmsumh** (vector unsigned short, vector unsigned short)
 - XLF: **VPMSUMH** (ARG1, ARG2), where ARG1, ARG2, and result are unsigned vector types of kind 2

- ▶ **vpmsumb**
 - GCC: vector unsigned char **__builtin_crypto_vpmsum** (vector unsigned char, vector unsigned char)
 - XL C/C++: vector unsigned char **__vpmsumb** (vector unsigned char, vector unsigned char)
 - XLF: **VPMSUMB** (ARG1, ARG2), where ARG1, ARG2, and result are unsigned vector types of kind 1

For more information about the topic of in-core cryptography, from the processor and OS perspectives, see:

- ▶ 2.2.7, “In-core cryptography and integrity enhancements” on page 47 (*processor*)
- ▶ 4.2.7, “On-chip encryption accelerator” on page 94 (*AIX*)

7.3.2 Compiler support for Vector Scalar eXtension

XLC supports vector processing technologies through language extensions on both AIX and Linux. GCC supports using the VSX engine on Linux. XL, and GCC. C implements and extends the AltiVec Programming Interface specification. In the extended syntax, type qualifiers and storage class specifiers can precede the keyword vector (or its alternative spelling, **__vector**) in a declaration.

Also, the XL compilers can automatically generate VSX instructions from scalar code when they generate code that targets the POWER7 processor or later at **-O3** or higher. GCC also automatically generates VSX instructions from scalar code when generating code for a POWER7 or later processor. This is accomplished by specifying **-O3** (or **-ftree-vectorize**).

Table 7-1 lists the supported vector data types and the size and possible values for each type.

Table 7-1 Vector data types

Type	Interpretation of content	Range of values
vector unsigned char	16 unsigned char	0..255
vector signed char	16 signed char	-128..127
vector bool char	16 unsigned char	0, 255
vector unsigned short	8 unsigned short	0..65535
vector unsigned short int		
vector signed short	8 signed short	-32768..32767
vector signed short int		
vector bool short	8 unsigned short	0, 65535
vector bool short int		
vector unsigned int	4 unsigned int	0..2 ³² -1
vector unsigned long	4 unsigned int (32-bit)	0..2 ³² -1
vector unsigned long int	2 unsigned long int (64-bit)	0..2 ⁶⁴ -1
vector signed int	4 signed int	-2 ³¹ ..2 ³¹ -1

vector signed long	4 signed int (32-bit)	$-2^{31}..2^{31}-1$
vector signed long int	2 signed long int (64-bit)	$-2^{63}..2^{63}-1$
vector bool int	4 unsigned int	0, $2^{32}-1$
vector bool long	4 unsigned int (32-bit)	0, $2^{32}-1$
vector bool long int	2 unsigned long int (64-bit)	0, $2^{64}-1$
vector float	4 float	IEEE-754 single (32-bit) precision floating point values
vector double	2 double	IEEE-754 double (64-bit) precision floating point values
vector pixel	8 unsigned short	1/5/5/5 pixel

Vector types: The *vector double* type requires architectures that support the VSX instruction set extensions, such as the POWER7 processor. You must specify the XL `-qarch=pwr7 -qaltivec` compiler options when you use this type, or the GCC `-mcpu=power7` or `-mvsx` options.

The hardware does not have instructions for supporting vector unsigned long long, vector bool long long, or vector signed long long. In GCC, you can declare these types, but the only hardware operation that you can use these types for is vector floating point convert. In 64-bit mode, vector long is the same as vector long long. In 32-bit mode, these types are not permitted.

All vector types are aligned on a 16-byte boundary. An aggregate that contains one or more vector types is aligned on a 16-byte boundary, and padded, if necessary, so that each member of vector type is also 16-byte aligned. Vector data types can use some of the unary, binary, and relational operators that are used with primitive data types. All operators require compatible types as operands unless otherwise stated. For more information about the operator's usage, see the XLC online publications.^{4, 5, 6}

Individual elements of vectors can be accessed by using the VMX or the VSX built-in functions. For more information about the VMX and the VSX built-in functions, see the "Built-in functions" section of *Vector Built-in Functions*.⁷

⁴ Support for POWER7 processors, found at:

<http://www-01.ibm.com/support/knowledgecenter/api/redirect/comphelp/v111v131/index.jsp?topic=/com.ibm.xlc111.aix.doc/getstart/architecture.html>

⁵ Vector built-in functions, found at:

http://www-01.ibm.com/support/knowledgecenter/api/redirect/comphelp/v111v131/index.jsp?topic=/com.ibm.xlc111.aix.doc/compiler_ref/vec_intrin_cpp.html

⁶ Initialization of vectors (IBM extension), found at:

http://www-01.ibm.com/support/knowledgecenter/api/redirect/comphelp/v111v131/index.jsp?topic=%2Fcom.ibm.xlc111.aix.doc%2Flanguage_ref%2Fvector_init.html

⁷ Vector built-in functions, found at:

http://www-01.ibm.com/support/knowledgecenter/api/redirect/comphelp/v111v131/index.jsp?topic=%2Fcom.ibm.xlc111.aix.doc%2Fcompiler_ref%2Fvec_intrin_cpp.html

Vector initialization

A vector type is initialized by a vector literal or any expression that has the same vector type. For example:⁸

```
vector unsigned int v1;  
vector unsigned int v2 = (vector unsigned int)(10); // XL only, not GCC  
v1 = v2;
```

The number of values in a braced initializer list must be less than or equal to the number of elements of the vector type. Any uninitialized element is initialized to zero.

Here are examples of vector initialization that use initializer lists:

```
vector unsigned int v1 = {1}; // initialize the first 4 bytes of v1 with 1  
                        // and the remaining 12 bytes with zeros  
vector unsigned int v2 = {1,2}; // initialize the first 8 bytes of v2 with 1 and 2  
                        // and the remaining 8 bytes with zeros  
vector unsigned int v3 = {1,2,3,4}; // equivalent to the vector literal  
                        // (vector unsigned int) (1,2,3,4)
```

How to use vector capability in the POWER8 processor

When you target a POWER processor that supports VMX or VSX, you can request the compiler to transform code into VMX or VSX instructions. These machine instructions can run up to 16 operations in parallel. This transformation mostly applies to loops that iterate over contiguous array data and perform calculations on each element. You can use the NOSIMD directive to prevent the transformation of a particular loop:⁹

- ▶ Using a compiler: Compiler versions that recognize the POWER8 architecture are XL C/C++ 13.1 and XLF Fortran 15.1 or recent versions of GCC, including the Advance Toolchain, and the SLES 11SP1 or Red Hat RHEL6 GCC compilers:
 - For C:
 - `xlc -qarch=pwr8 -qtune=pwr8 -O3 -qhot`
 - `gcc -mcpu=power8 -mtune=power8 -O3`
 - For Fortran
 - `xlf -qarch=pwr8 -qtune=pwr8 -O3 -qhot`
 - `gfortran -mcpu=power8 -mtune=power8 -O3`
- ▶ Using Engineering and Scientific Subroutine (ESSL) libraries with vectorization support:
 - Select routines have vector analogs in the library.
 - Key FFT, BLAS routines.

For more information about the topic of VSX, from the processor and OS perspectives, see:

- ▶ 2.2.5, “Vector Scalar eXtension” on page 45 (*processor*)
- ▶ 4.2.5, “Vector Scalar eXtension” on page 91 (*AIX*)
- ▶ 5.2.3, “Vector Scalar eXtension” on page 113 (*IBM i*)
- ▶ 6.2.5, “Vector Scalar eXtension” on page 125 (*Linux*)

⁸ *Vector types (IBM extension)*, found at:
http://www-01.ibm.com/support/knowledgecenter/api/redirect/comphelp/v111v131/index.jsp?topic=%2Fcom.ibm.xlcl111.aix.doc%2Flanguage_ref%2Faltivec_types.html

⁹ Ibid

7.3.3 Built-in functions for storage synchronization

The XL C/C++ compiler provides built-in functions for direct usage of the storage synchronization load/store operations, as described in 2.2.9, “Storage synchronization (sync, lwsync, lwarx, stwcx., and eieio)” on page 49. New functions for the POWER8 processor are indicated and require **-qarch=pwr8**.

Each pair of built-ins is used to implement a read-modify-write operation on a memory location of a given size, where the load built-in (beginning with `__l`) returns the loaded value, and the store built-in (beginning with `__st`) returns 1 if the store succeeds, 0 otherwise. The functions work together to ensure that if the store succeeds, then no other processor or mechanism can modify the target between the time of the load execution and store completion.

- ▶ `char __lbarx(volatile char* addr)`
- ▶ `int __stbcx(volatile char* addr, char data)`

Load and store, respectively, the byte value at **addr**. Requires **-qarch=pwr8**.

- ▶ `short __lharx(volatile short* addr)`
- ▶ `int __sthcx(volatile short* addr, short data)`

Load and store, respectively, the halfword value at **addr**. **addr** must be aligned on a halfword boundary. Requires **-qarch=pwr8**.

- ▶ `int __lwarx(volatile int* addr)`
- ▶ `int __stwcx(volatile int* addr, int data)`

Load and store, respectively, the word value at **addr**. **addr** must be aligned on a word boundary.

- ▶ `long __ldarx(volatile long* addr)`
- ▶ `long __stdcx(volatile long* addr, long data)`

Load and store, respectively, the doubleword value at **addr**. **addr** must be aligned on a doubleword boundary. Only valid in 64-bit mode.

- ▶ `void __lqarx(volatile long* addr, long data[2])`
- ▶ `long __stqcx(volatile long* addr, long data[2])`

`__lqarx` loads the quadword value at **addr** into the quadword location that is specified by **data**. `__stqcx` stores the quadword value in **data** to the quadword location that is specified by **addr**. Both **addr** and **data** must be aligned on a quadword boundary. Only valid in 64-bit mode, and requires **-qarch=pwr8**.

7.3.4 Data Streams Control Register controls

The XL C/C++ and XL Fortran compilers provide the following built-in functions to modify the DSCR setting. The **-qarch=pwr8** option is required to use the following built-in functions. For descriptions of the DSCR fields, see Table 2-4 on page 40.

- ▶ C/C++: `void __software_transient_enable(int)`
- ▶ Fortran: `SOFTWARE_TRANSIENT_ENABLE(flag)`, where **flag** is a scalar of type logical
Set the SWTE bit to the provided value (0 or 1).
- ▶ C/C++: `void __hardware_transient_enable(int)`
- ▶ Fortran: `HARDWARE_TRANSIENT_ENABLE(flag)`, where **flag** is a scalar of type logical
Set the HWTE bit to the provided value (0 or 1).

- ▶ C/C++: **void __store_transient_enable(int)**
- ▶ Fortran: **STORE_TRANSIENT_ENABLE(flag)**, where **flag** is a scalar of type logical
Set the STE bit to the provided value (0 or 1).
- ▶ C/C++: **void __load_transient_enable(int)**
- ▶ Fortran: **LOAD_TRANSIENT_ENABLE(flag)**, where **flag** is a scalar of type logical
Set the LTE bit to the provided value (0 or 1),
- ▶ C/C++: **void __software_unit_count_enable(int)**
- ▶ Fortran: **SOFTWARE_UNIT_COUNT_ENABLE(flag)**, where **flag** is a scalar of type logical
Set the SWUE bit to the provided value (0 or 1).
- ▶ C/C++: **void __hardware_unit_count_enable(int)**
- ▶ Fortran: **HARDWARE_UNIT_COUNT_ENABLE(flag)**, where **flag** is a scalar of type logical
Set the HWUE bit to the provided value (0 or 1).
- ▶ C/C++: **void __set_prefetch_unit_count(int)**
- ▶ Fortran: **SET_PREFETCH_UNIT_COUNT(cnt)**, where **cnt** is a scalar of type integer
Set the UNITCNT field to the provided value (in range [0,1023]).
- ▶ C/C++: **void __depth_attainment_urgency(int)**
- ▶ Fortran: **DEPTH_ATTAINMENT_URGENCY(cnt)**, where **cnt** is a scalar of type integer
Set the URG field to the provided value (in range [0,7]).
- ▶ C/C++: **void __load_stream_disable(int)**
- ▶ Fortran: **LOAD_STREAM_DISABLE(flag)**, where **flag** is a scalar of type logical
Set the LSD bit to the provided value (0 or 1).
- ▶ C/C++: **void __stride_n_stream_enable(int)**
- ▶ Fortran: **STRIDE_N_STREAM_ENABLE(flag)**, where **flag** is a scalar of type logical
Set the SNSE bit to the provided value (0 or 1).
- ▶ C/C++: **void __default_prefetch_depth(int)**
- ▶ Fortran: **DEFAULT_PREFETCH_DEPTH(cnt)**, where **cnt** is a scalar of type integer
Set the DPFDP field to the provided value (in range [0,7]).
- ▶ C/C++: **unsigned long long __prefetch_get_dscr_register(void)**
- ▶ Fortran: **PREFETCH_GET_DSCR_REGISTER()**, with return type integer*8
Get the current 64-bit DSCR register value.
- ▶ C/C++: **void __prefetch_set_dscr_register(void)**
- ▶ Fortran: **PREFETCH_SET_DSCR_REGISTER(val)**, where **value** is a scalar of type integer*8
Set the DSCR value to the provided 64-bit value.

The topic of DSCR is described from a processor perspective in “Data prefetching using d-cache instructions and the Data Streams Control Register (DSCR)” on page 39.

7.3.5 Transactional memory

Transactional memory (TM) is a new approach that simplifies concurrent programming, specifically in the accessing of shared data across multiple threads. Previous to TM, accesses to shared data were synchronized by the use of locks. Threaded code that needed access to shared data first had to acquire the data lock, then access the shared data, and then release the lock. On many systems, acquiring locks can be expensive, making accessing shared data vastly more expensive than accessing non-shared data. This additional locking can be especially burdensome when the shared data has low contention between the multiple threads.

Using TM, shared data accesses are placed into blocks of code called *transactions*. When using hardware transactional memory (HTM), these transactions are run without locking, and the results are seen by other threads atomically.

The POWER8 processor supports the HTM instructions that are defined in *Power ISA Version 2.07*, found at:

<https://www.power.org/documentation/power-isa-v-2-07b/>

Users have three options when writing code to use the POWER8 HTM features:

- The first option to use HTM is through the low-level GCC built-in functions, which are enabled with the GCC `-mcpu=power8` or `-mhtm` compiler options. The HTM built-in functions (with the exception of `__builtin_tbegin`) return the full 4-bit condition register value that is set by their associated hardware instruction. The header, `htmintrin.h`, defines some macros that can be used to decipher the return value. The `__builtin_tbegin` built-in returns a simple true or false value, depending on whether a transaction was successfully started or unsuccessful. The arguments to the HTM built-in functions match exactly the type and order of the associated hardware instruction operands (except for the `__builtin_tcheck` built-in, which does not take any input arguments), as shown in Example 7-1.

Example 7-1 GCC HTM built-in functions

```
unsigned int __builtin_tbegin (unsigned int)
unsigned int __builtin_tend (unsigned int)

unsigned int __builtin_tabort (unsigned int)
unsigned int __builtin_tabortdc (unsigned int, unsigned int, unsigned int)
unsigned int __builtin_tabortdci (unsigned int, unsigned int, int)
unsigned int __builtin_tabortwc (unsigned int, unsigned int, unsigned int)
unsigned int __builtin_tabortwci (unsigned int, unsigned int, int)

unsigned int __builtin_tcheck (void)
unsigned int __builtin_treclaim (unsigned int)
unsigned int __builtin_trechkt (void)
unsigned int __builtin_tsr (unsigned int)

unsigned long __builtin_get_texasr (void)
unsigned long __builtin_get_texasru (void)
unsigned long __builtin_get_tfhar (void)
unsigned long __builtin_get_tfiar (void)

void __builtin_set_texasr (unsigned long);
void __builtin_set_texasru (unsigned long);
```

```
void __builtin_set_tfhar (unsigned long);
void __builtin_set_tfiar (unsigned long);
```

In addition to Example 7-1 on page 156, in this book we have added built-in functions for some common extended mnemonics of the HTM instructions, as shown in Example 7-2.

Example 7-2 GCC HTM built-in functions for extended mnemonics

```
unsigned int __builtin_tendall (void)
unsigned int __builtin_tresume (void)
unsigned int __builtin_tsuspend (void)
```

Common usage of these HTM built-in functions might produce results similar to those shown in Example 7-3.

Example 7-3 Simple use of HTM built-in functions

```
#include <htmintrin.h>
if (__builtin_tbegin (0))
{
    /* Transaction State Initiated. */
    if (is_locked (lock))
        __builtin_tabort (0);
    a = b + c;
    __builtin_tend (0);
}
else
{
    /* Transaction State Failed, Use Locks. */
    acquire_lock (lock);
    a = b + c;
    release_lock (lock);
}
```

A slightly more complicated example is shown in Example 7-4. This example shows an attempt to retry the transaction a specific number of times before falling back to using locks.

Example 7-4 Complex use of HTM built-in functions

```
#include <htmintrin.h>
int num_retries = 10;
while (1)
{
    if (__builtin_tbegin (0))
    {
        /* Transaction State Initiated. */
        if (is_locked (lock))
            __builtin_tabort (0);
        a = b + c;
        __builtin_tend (0);
        break;
    }
    else
    {
        /* Transaction State Failed. Use locks if the transaction
           failure is "persistent" or we've tried too many times. */
        if (num_retries-- <= 0
            || _TEXASRU_FAILURE_PERSISTENT (__builtin_get_texasru ()))
        {
            acquire_lock (lock);
            a = b + c;
            release_lock (lock);
        }
    }
}
```

```

        {
            acquire_lock (lock);
            a = b + c;
            release_lock (lock);
            break;
        }
    }
}

```

In some cases, it can be useful to know whether the code that is being run is in the transactional state or not. Unfortunately, that cannot be determined by analyzing the HTM Special Purpose Registers (SPRs). That specific information is contained only within the Machine State Register (MSR) Transaction State (TS) bits, which are not accessible by user code. To allow access to that information, we have added one final built-in function and some associated macros to help the user to determine what the transaction state is at a particular point in their code:

```
unsigned int __builtin_ttest (void)
```

Usage of the built-in function and its associated macro might look like the code that is shown in Example 7-5.

Example 7-5 Determine the transaction state

```

#include <htmintrin.h>

unsigned char tx_state = __builtin_ttest ();

if (_HTM_STATE (tx_state) == _HTM_TRANSACTIONAL)
{
    /* Code to use in transactional state. */
}
else if (_HTM_STATE (tx_state) == _HTM_NONTRANSACTIONAL)
{
    /* Code to use in non-transactional state. */
}
else if (_HTM_STATE (tx_state) == _HTM_SUSPENDED)
{
    /* Code to use in transaction suspended state. */
}

```

- A second option for using HTM is by using the slightly higher-level inline functions that are common to GCC and the IBM XL compilers on both POWER and System z®. These sets of common HTM built-in functions are defined in the `htmxlintrin.h` header file and can be used to write code that can be compiled on POWER or System z by using either the IBM XL or GCC compilers. See Example 7-6.

Example 7-6 HTM intrinsic functions common to IBM XL and GCC compilers

```

long __TM_simple_begin (void)
long __TM_begin (void* const TM_buff)
long __TM_end (void)
void __TM_abort (void)
void __TM_named_abort (unsigned char const code)
void __TM_resume (void)
void __TM_suspend (void)

long __TM_is_user_abort (void* const TM_buff)
long __TM_is_named_user_abort (void* const TM_buff, unsigned char *code)

```

```

long __TM_is_illegal (void* const TM_buff)
long __TM_is_footprint_exceeded (void* const TM_buff)
long __TM_nesting_depth (void* const TM_buff)
long __TM_is_nested_too_deep(void* const TM_buff)
long __TM_is_conflict(void* const TM_buff)
long __TM_is_failure_persistent(void* const TM_buff)
long __TM_failure_address(void* const TM_buff)
long long __TM_failure_code(void* const TM_buff)

```

Using these built-in functions, you can create a more portable version of the code that is shown in Example 7-4 on page 157 so that it works on POWER and on System z, by using either GCC or the XL compilers. This more portable version is shown in Example 7-7.

Example 7-7 Complex HTM usage using portable HTM intrinsics

```

#ifdef __GNUC__
# include <htmxlintrin.h>
#endif

int num_retries = 10;
TM_buff_type TM_buff;

while (1)
{
    if (__TM_begin (TM_buff) == _HTM_TBEGIN_STARTED)
    {
        /* Transaction State Initiated. */
        if (is_locked (lock))
            __TM_abort ();
        a = b + c;
        __TM_end ();
        break;
    }
    else
    {
        /* Transaction State Failed. Use locks if the transaction
           failure is "persistent" or we've tried too many times. */
        if (num_retries-- <= 0
            || __TM_is_failure_persistent (TM_buff))
        {
            acquire_lock (lock);
            a = b + c;
            release_lock (lock);
            break;
        }
    }
}

```

- The third and most portable option uses a high-level language interface that is implemented by GCC and the GNU Transactional Memory Library (LIBTM), which is described at the following website:

<http://gcc.gnu.org/wiki/TransactionalMemory>

This high-level language option is enabled by using the `-fgnu-tm` option (`-mcpu=power8` and `-mhtm` are not needed), and it provides a common transactional model across multiple architectures and multiple compilers by using the `__transaction_atomic {...}` language construct. The LIBITM library, which is included with the GCC compiler, can determine, at run time, whether it is running on a processor that supports HTM instructions, and, if so, it uses them in running the transaction. Otherwise, it automatically falls back to using software TM, which relies on locks. LIBITM also can retry a transaction by using HTM if the initial transaction **begin** failed, similar to the complicated example (Example 7-4 on page 157). An example of the third option that is equivalent to the complicated examples (Example 7-4 on page 157 and Example 7-7 on page 159) is simple and is shown in Example 7-8.

Example 7-8 GNU Transactional Memory Library (LIBITM) Usage

```
__transaction_atomic
{
    a = b + c;
}
```

Support for the HTM built-in functions, the XL HTM built-in functions, and LIBITM support will be in an upcoming Free Software Foundation (FSF) version of GCC. However, it is also available in the GCC 4.8-based compiler that is shipped in Advance Toolchain (AT) V7.0.

For more information about the topic of TM, from the processor, OS, and compiler perspectives, see:

- ▶ 2.2.4, “Transactional memory” on page 42 (*processor*)
- ▶ 4.2.4, “Transactional memory” on page 89 (*ALX*)
- ▶ 6.2.4, “Transactional memory” on page 124 (*Linux*)
- ▶ 8.4.2, “Transactional memory” on page 182 (*Java*)

7.4 IBM Feedback Directed Program Restructuring

Feedback Directed Program Restructuring (FDPR) is a feedback-based, directed, and post-link optimization tool.

7.4.1 Introduction

FDPR optimizes the executable binary file of a program by collecting information about the behavior of the program while the program is used for a typical workload, and then creates a new version of the program that is optimized for that workload. Both main executable and dynamically linked libraries (DLLs) are supported.

FDPR performs global optimizations at the level of the entire executable library, including statically linked library code. Because the executable library to be optimized by FDPR is not relinked, the compiler and linker conventions do not need to be preserved, thus allowing aggressive optimizations that are not available to optimizing compilers.

The main advantage that is provided by FDPR is the reduced footprint of both code and data, resulting in more effective cache usage. The principal optimizations of FDPR include global code reordering, global data reordering, function inlining, and loop unrolling, along with various tuning options that are tailored for the specific POWER target. The effectiveness of the optimization depends largely on how representative the collected profile is regarding the true workload.

FDPR runs on both AIX and Linux and produces optimized code for all versions of the Power Architecture. The POWER7 processor is its default target architecture.

Figure 7-1 shows how FDPR is used to optimize executable programs.

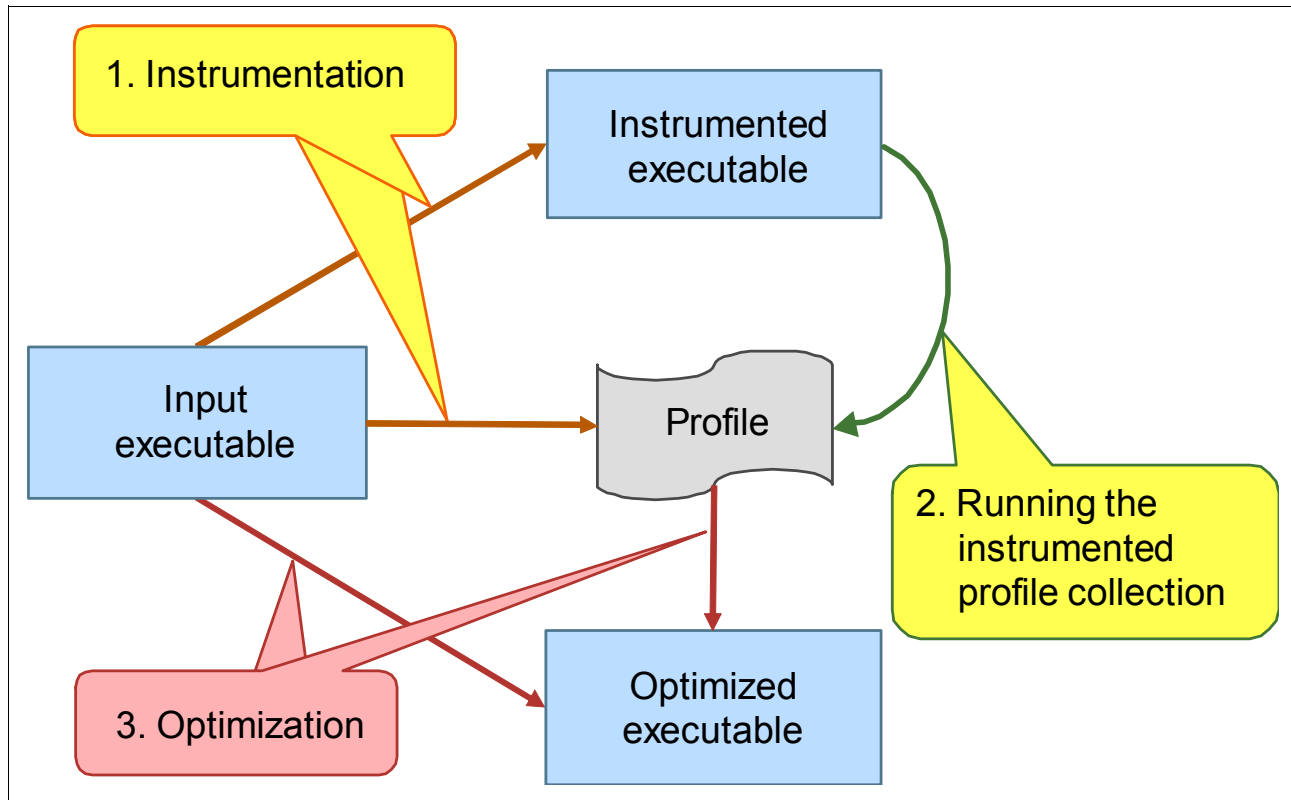


Figure 7-1 FDPR operation

FDPR builds an optimized executable program in three distinct phases:

1. Instrumentation (Yellow)
 - Creates an instrumented version of the input program and an empty profile file.
 - The input program can be an executable file or a dynamically linked shared library.
2. Profiling (Green)
 - Runs the instrumented program on a representative workload.
 - The profile file is filled with count data at run time.
3. Optimization (Red)

FDPR receives the original input program along with the filled profile file to create an optimized version of the input.

7.4.2 Feedback Directed Program Restructuring supported environments

FDPR is available on the following platforms:

- ▶ AIX and Power Systems: Part of the AIX 5L V5 operating system and later for both 32-bit and 64-bit applications. For more information, see *AIX 5L Performance Tools Handbook*, SG24-6039.
- ▶ Software Development Toolkit for Linux on Power: Available for use through the IBM SDK for Linux on Power. Linux distributions of RHEL5 and later, Ubuntu 14.04.2 and later, and SLES10 and later are supported. For more information, see the following website:

<http://www14.software.ibm.com/webapp/set2/sas/f/lopdiags/sdklop.html>

In these resources, detailed online help, including manuals, is provided for each of these environments.

7.4.3 Acceptable input formats

The input binary can be a main executable program or a shared library, originally written in any language (for example, C, C++, or Fortran), if it is statically compiled. Thus, Java byte code is not acceptable. Code that is written in assembly language is acceptable, but must follow the Power ABI convention. For more information, see *64-bit PowerPC ELF Application Binary Interface Supplement 1.9*, found at:

<http://refspecs.linuxfoundation.org/ELF/ppc64/PPC-elf64abi-1.9.pdf>

It is important that the file includes relocation information. Although this is the default in AIX, on Linux you must add **-Wl,-q**, or **-Wl,--emit-relocs** to the command that is used for linking the program (or **-q** if the **ld** command is used directly).

The input binary can include debug information. FDPR correctly processes line number information so that the optimized output can be debugged.

7.4.4 General operation

FDPR is started by running the **fdprpro** program as follows:

```
$ fdprpro -a action [-p] in -o out -f prof [opt ...]
```

The **action** indicates the specific processing that is requested. The most common ones are **instr** for the instrumentation step and **opt** for the optimization step.

The **in**, **out**, and **prof** items indicate the input and output binary files and profile files.

FDPR comes also with a wrapper command, named **fdpr**, which performs the instrumentation, profiling, and optimization under one roof. Run **man fdpr** for more information about this wrapper.

Special input and output files

FDPR has a number of options that control input and output files. One option that controls the input files is **--ignored-function-list *file* (-ifl *file*)**.

In some cases, the structure of some functions confuses FDPR, which can result in bad code generation. The file that is specified by **--ignored-function-list *file* (-ifl *file*)** contains a list of functions that are considered unsafe for optimization. This configuration prevents the potential bad code generation that might otherwise occur.

In addition to the profile and the instrumented and output optimized files, FDPR can optionally produce various secondary files to help you understand the static and dynamic nature of the input binary program. These secondary files have the same base name as the output file and a special extension. The options that control important output files are:

- ▶ **--disassemble_text (-d)** and **--dump-mapper (-dm)**: The **-d** option creates a disassembly of (the code segment) of the program (extension `.dis_text`). The disassembly is useful to understand the structure of program as analyzed or created by FDPR. The **-dm** option produces a mapping of basic-blocks from their original address to their address in the optimized code. This mapping can be used, for example, to understand how a specific piece of code was broken, or for user-specific post-processing tools.
- ▶ **--dump-ascii-profile (-dap)**: This option dumps the profile file in a human readable ASCII format (extension `.aprof`). The `.aprof` file is useful for manual inspection or user-defined post-processing of the collected profile.
- ▶ **--verbose *n* (-v *n*)**, **--print-inlined-funcs (-pif)**, and **--journal *file* (-j *file*)**: These options generate different analyses of the optimized file. **-v *n*** generates general and optimization-specific statistics (`.stat` extension). The amount of verbosity is set by *n*. Basic statistics are provided by **-v 1**. Optimization-specific statistics are added in level 2 and the instruction mix is added in level 3. The list of inlining and inlined functions is produced with the **-pif** option (`.inl_list` extension). The **-j *file*** produces a journal of the main optimizations, in an XML format, with detailed information about each optimization site, including the corresponding source file and line information. This information can be used by GUI tools to display optimizations in the context of the source code.

Controlling output to the console

The amount of progress information that is printed to the console can be controlled by two options. The default progress information is as follows:

```
fdprpro (FDPR) Version vvv for IBM PowerLinux™
fdprpro -a opt -O3 in -o out -f prof
> reading_exe ...
> adjusting_exe ...
> analyzing ...
> building_program_infrastructure ...
> building_profiling_cfg ...
> add_profiling ...
>> reading_profile ...
>> building_control_flow_transfer_profiling ...
> pre_reorder_optimizations ...
>> derat_optimization ...
...
```

This information might also be interspersed with warning and debugging messages. Use the **-quiet (-q)** option to avoid progress information. To limit the warning information, use the **-warning 1 (-w 1)** option.

7.4.5 Instrumentation and profiling

FDPR instrumentation is performed by running the following command:

```
$ fdprpro -a instr in [-o out] [-f prof] [opts...]
```

If **out** is not specified, the output file is in `in.instr`. If the profile is not specified, `in.nprof` is used.

Two files are created: the instrumented program and an empty profile. The instrumented program (or shared library), when run on a representative workload, fills the profile with execution counts of nodes and edges of the binary control flow graph (CFG). A node in this CFG is a basic block (piece of code with single entry and exit points). An edge indicates a control transfer between two basic blocks through a branch (regular branch, call, or return instruction).

To run the instrumented program, use the same command parameters as with the original program. As indicated in 7.4.1, “Introduction” on page 160, the workload that is exercised during the instrumented run should be representative, making the optimization step more effective. Because of the instrumentation code, the program is slower.

Successive runs of the instrumented program accumulate the counts in the same profile. Similarly, if the instrumented program is a shared library, each time the shared library participates in a process, the corresponding profile is updated with added counts.

Profiling shared libraries

When the dynamic linker searches for and links a shared library during execution, it looks for the original name that is used for the command that is used for linking the program. To ensure that the instrumented library is run, ensure that the following items are true:

1. The instrumented library should have the same name as the original library. The user can rename the original or place the libraries in different folders.
2. The folder that contains the library must be in the library search path: `LIBPATH` on AIX and `LD_LIBRARY_PATH` on Linux.

Moving and renaming the profile file

The location of the profile file is specified in the instrumented program, as indicated by the `-f` option. However, the profile file might be moved, or if its original specification is relative, the real location can change before execution.

Use the `-fdir` option to set the profile directory if it is known at instrumentation time and is different from the one implied or specified by the `-f` option.

Use the `FDPR_PROF_DIR` environment variable to specify the profile directory if the profile file is not present in the relative or absolute location where it was created in the instrumentation step (or where specified originally by `-fdir`).

Use the `FDPR_PROF_NAME` environment variable to specify the profile file name if the profile file name changed.

Profile file descriptor

When the instrumented binary file is run, the profile file is mapped to shared memory. The process is using a default file descriptor (FD) number (1023 on Linux and 1999 on AIX) for the mapping. If the application uses this specific FD, an error can occur during the profiling phase because of this conflict of use. Use the **-fd** option to change the default FD that is used by FDPR:

```
$ fdprpro -a instr my_prog -fd <fd num>
```

The FD can also be controlled by using the **FDPR_PROF_FD** environment variable by changing the FD at run time:

```
$ export FDPR_PROF_FD=fd_num
```

FDPR can be used to profile several binary executable files in a single run of an application. If so, you must specify a different FD for each binary. For example:

- ▶ **\$ fdprpro -a instr in/libmy_lib1 -o out/libmy_lib1 -f out/libmy_lib1.prof -fd 1023**
- ▶ **\$ fdprpro -a instr in/libmy_lib2 -o out/libmy_lib2 -f out/libmy_lib2.prof -fd 1022**

Because environment variables are global in nature, when profiling several binary files at the same time, use explicit instrumentation options (**-f**, **-fd**, and **-fdir**) to differentiate between the profiles rather than using the environment variables (**FDPR_PROF_FD** and **FDPR_PROF_NAME**).

Instrumentation stack

The instrumentation is using the stack for saving registers by dynamically allocating space on the stack at a default location below the current stack pointer. On AIX, this default is at offset -10240, and on Linux it is -1800. In some cases, especially in multi-threaded applications where the stack space is divided between the threads, following a deep calling sequence, the application can be quite close to the end of the stack, which can cause the application to fail. To allocate the instrumentation closer to the current stack pointer, use the **-iso** option:

```
$ fdprpro -a instr my_prog -iso -300
```

7.4.6 Optimization

The optimization step is performed by running the following command:

```
$ fdprpro -a opt in [-o out] -f prof [opts...]
```

If **out** is not specified, the output file is **in.fdpr**. No profile is provided by default. If **none** is specified or if the profile is empty, the resulting output binary file is not optimized.

Code reordering

Global code reordering works in two phases: making chains and reordering the chains.

The initial chains are sequentially ordered basic blocks, with branch conditions inverted where necessary, so that branches between the basic blocks are mostly not taken. This configuration makes instruction prefetching more efficient. Chains are terminated when the heat (that is, execution count) goes below a certain threshold relative to the initial heat.

The second phase orders chains by successively merging the more strongly linked two chains, based on how frequent the calls between the chains are. Combining chains crosses function boundaries. Thus, a function can be broken into multiple chunks in which different pieces of different functions are placed closely if there is a high frequency of call, branch, and return between them. This approach improves code locality and thus i-cache and page table efficiency.

You use the following options for code reordering:

- ▶ **--reorder-code (-RC)**: This component is the hard-working component of the global code reordering. Use **--rcaf** to determine the aggressiveness level:
 - 0: no change
 - 1: Standard (default)
 - 2: Most aggressive.

Use **--rcctf** to lower the threshold for terminating chains. Use **-pp** to preserve function integrity and **-pc** to preserve CSECT integrity (AIX only). These two options limit global code reordering and might be requested for ease of debugging.
- ▶ **--branch-folding (-bf)** and **--branch-prediction (-bp)**: These options control important parts of the code reordering process. The **-bf** folds branch to branch into a single branch. The **-bp** sets the static branch prediction bit when taken or not taken statistics justify it.

Function inlining

FDPR performs function inlining of function bodies into their respective calling sites if the call site is selected by one of a number of user-selected filters:

- ▶ Dominant callers (**--selective-inlining (-si)**, **-sidf f**, and **-siht f**): The filter criteria here is that the site is dominant regarding other callers of the called function (the callee). It is controlled by two attributes. The **-sidf** option sets the domination percentage threshold (default 80). The **-siht** option further restricts the selection to functions hotter than the threshold, which is specified in percents relative to the average (default 100).
- ▶ Hot functions (**--inline-hot-functions f (-ihf f)**): This filter selects inlining for all call sites where the call is hotter than the heat threshold (in percent, relative to the average).
- ▶ Small functions (**--inline-small-functions f (-isf f)**): This filter selects for inlining all functions whose size, in bytes, is smaller than or equal to the parameter.
- ▶ Selective hot code (**--selective-hot-code-inline f (-shci f)**): The filter computes how much execution count is saved if the function is inlined at a call site and selects those sites where the relative saving is above the percentage.

De-virtualization

De-virtualization is addressed by the **--ptrgl-optimization (-pto)** option. Its call by a pointer mechanism (ptrgl) sets a new TOC anchor, loads the function address, moves it to the counter register (CTR), and jumps indirectly through the CTR. The **-pto** option optimizes this mechanism in cases where there are few hot targets from a calling site. In terms of C++, it de-virtualizes the virtual method calls by calling the actual targets directly. The optimized code compares the address of the function descriptor, which is used for the indirect call, against the address of a hot candidate, as identified in the profile, and conditionally calls such a target directly. If none of the hot targets match, the code starts the original indirect call mechanism. The idea is that most of the time the conditional direct branches are run instead of the ptrgl mechanism. The impact of the optimization on performance depends heavily on the function call profile.

The following thresholds can help to tune the optimization and to adjust it to different workloads:

- Use **-ptoh *thres*** to set the frequency threshold for indirect calls that will be optimized (*thres* can be 0 - 1, with 0.8 by default).
- Use **-ptos1 *n*** to set the limit of the number of hot functions to optimize in a given indirect call site (the default for *n* is 3).

Loop-unrolling

Most programs spend their time in loops. This statement is true regardless of the target architecture or application. FDPR has one option to control the unrolling optimization for loops: **--loop-unrolling *factor*** (**-lu *factor***).

FDPR optimizes a loop by using a technique that is called *loop-unrolling*. By unrolling a loop *n* times, the number of back branches is reduced *n* times, so code prefetch efficiency can be improved. The downside with loop-unrolling is code inflation, which results in increased code footprint and increased i-cache misses. Unlike traditional loop-unrolling, FDPR can mitigate this problem by unrolling only the hottest paths in the loop. The **factor** parameter determines the aggressiveness of the optimization. With **-03**, the optimization is started with **-lu 9**.

By default, loops are unrolled two times. Use **-lu *factor*** to change that default.

Architecture-specific optimizations

Here are some architecture-specific optimizations:

- **--machine *tgt*** (**-m *tgt***): FDPR optimizations include general optimizations that are based on a high-level program representation as a control and data flow, in addition to peephole optimizations, relying on different architecture features. Those optimizations can perform better when they are tuned for specific platforms. The **-m** flag allows the user to specify the target machine model cases where the program is not intended for use on multiple target platforms. The default target is the POWER7 processor.
- **--align-code *code*** (**-A *code***): Optimizing the alignment and the placement of the code is crucial to the performance of the program. Correct alignment can improve instruction fetching and dispatching. The alignment algorithm in FDPR uses different techniques that are based on the target platform. Some techniques are generic for the Power Architecture, and others are considered dispatch rules of the specific machine model. If **code** is 1 (the default), FDPR applies a standard alignment algorithm that is adapted for the selected target machine (see **-m** in the previous bullet point). If **code** is 2, FDPR applies a more advanced version, by using dispatch rules and other heuristics to decide how the program code chunks are placed relatively to i-cache sectors, again based on the selected target. A value of 0 disables the alignment algorithm.

Function optimization

FDPR includes a number of function level optimizations that are based on detailed data flow analysis (DFA). With DFA, optimizations can determine the data that is contained in each register at each point in the function and whether this value is used later.

Here are the function optimizations:

- ▶ **--killed-regs (-kr)**: A register is considered killed at a point (in the function) if its value is not used in any ensuing path. FDPR uses the Power ABI convention that defines which registers are non-volatile (NV) across function calls. NV registers that are used inside a function are saved in its prologue and restored in its epilogue. The **-kr** optimization analyzes called functions that are looking for save and restore instructions of killed NV registers. If the register is killed at the calling site, then the save and restore instructions for this register are removed. The optimization considers all calls to this function because an NV might be alive when the function is called. When needed, the optimization might also reassign (rename) registers at the calling side to ensure that an NV is indeed killed and can be optimized.
- ▶ **--hco-reschedule (-hr)**: The optimization analyzes the flow through hot basic blocks and looks for instructions that can be moved to dominating colder basic blocks (basic block b1 dominates b2 if all paths to b2 first go through b1). For example, an instruction that loads a constant to a register is a candidate for such a motion.
- ▶ **--simplify-early-exit *factor* (-see *factor*)**: Sometimes a function starts with an early exit condition so that if the condition is met, the whole body of the function is ignored. If the condition is commonly taken, it makes sense to avoid saving the registers in the prologue and restoring them in the epilogue. The **-see** optimization detects such a condition and provides a reduced epilogue that restores only registers that are modified by computing the condition. If **factor** is 1, a more aggressive optimization is performed where the prologue is optimized.

Peephole optimization

Peephole optimizations require a small context around the specific site in the code, which is problematic. The more important optimizations that FDPR performs are **-las**, **-tlo**, and **-nop**.

- ▶ **--load-after-store (-las)**: In recent Power Architectures, when a load instruction from address A closely follows a store to that address, it can cause the load to be rejected. The instruction is then tried in a slower mode, which produces a large performance penalty. This behavior is also called *Load-Hit-Store (LHS)*. With the **-las** optimization, the load is pushed further from the store, thus avoiding the reject condition.
- ▶ **--toc-load-optimization (-tlo)**: The TOC is a data section in programs where pointers are kept to avoid the lengthy address computation at run time. Loading an address (a pointer) is a costly operation and FDPR can reduce the amount of processing if the address is close enough to the TOC anchor (R2). In such cases, the load from TOC is replaced by `addi Rt,R2,offset`, where `R2+offset` equals a loaded address. The optimization is performed after data is reordered so that commonly accessed data is placed closer to R2, increasing the potential of this optimization. A TOC is used in 32-bit and 64-bit programs on AIX, and in 64-bit programs on Power Systems running Linux. Linux 32-bit uses a GOT, but this optimization is not relevant here.
- ▶ **--nop-removal (-nop)**: The compiler (or the linker) sometimes inserts no-operation (NOP) instructions in various places to create some necessary space in the instruction stream. The most common place is following a function call in code. Because the call might have modified the TOC anchor register (R2), the compiler inserts a load instruction that resets R2 to its correct value for the current function. Because FDPR has a global view of the program, the optimization can remove the NOP if the called function uses the same TOC (the TOC anchor is used in AIX and in Linux 64-bit).

Data reordering

The profile that is collected by FDPR provides important information about the running of branch instructions, thus enabling efficient code reordering. The profile does not provide direct information about whether to put specific objects one after the other. Nevertheless, FDPR can infer such a placement by using the collected profile.

Here are the relevant options:

- ▶ **--reorder-data (-RD)**: This optimization reorders data by placing pointers and data closer to the TOC anchor, depending on their hotness. FDPR uses a heuristic where the hotness is computed as the total count of basic blocks where the pointer to the data was retrieved from the TOC.
- ▶ **--reduce-toc *thres* (-rt *thres*)**: The optimization removes from the TOC entries that are colder than the threshold. Their access, if any, is replaced by computing the address (see **-tlo** optimization in “Peephole optimization” on page 168). Typically, you use **-rt 0**, which removes only the entries that are never accessed.

Combination optimizations

FDPR has predefined optimization sets that provide a good starting point for performance tuning:

- ▶ **-0**: Performs code reordering (**-RC**) with the branch prediction bit setting (**-bp**), branch folding (**-bf**), and NOOP instructions removal (**-nop**).
- ▶ **-02**: Adds to **-0** function de-virtualization (**-pto**), TOC-load optimization (**-tlo**), function inlining (**-isf 8**), and some function optimizations (**-hr**, **-see 0**, and **-kr**).
- ▶ **-03**: Turns on data reordering (**-RD** and **-rt 0**), loop-unrolling (**-lu**), more aggressive function optimization (**-see 1** and **-vro**), and employs more aggressive inlining (**-lro** and **-isf 12**). This set provides an aggressive but still stable set of optimizations that are beneficial for many benchmarks and applications.
- ▶ **-04**: Essentially turns on more aggressive inlining (**-sidf 50**, **-ihf 20**, and **-shci 90**). As a result, the number of branches is reduced, but at the cost of increasing the code footprint. This option works well with large i-caches or with small to medium programs/threads.

7.5 Using the Advance Toolchain with IBM XLC and XLF

For XLC13 and XLF15, there is a new feature in the existing `new_install` script, which is shipped with the Linux package.

Run this script with one option, and it detects whether AT has been installed in the environment. If yes, it automatically generates a configuration file with the AT information specified, and generates a new invocation that is named `xlc_at`, which uses the generated configuration file. Then, you can use this `xlc_at` invocation to get the **XLC + AT** usage.

7.6 Using GPU accelerators with C/C++

One way to speed up in a C/C++ program is to offload large computations to an onboard graphics processing unit (GPU). NVIDIA makes a set of GPUs specifically for POWER8 processor-based systems that are enabled with a Linux kernel in LE mode. Using the GPU from C/C++ has never been easier with the introduction of NVIDIA Compute Unified Device Architecture (CUDA). CUDA is a programming model that uses GPU devices that are produced by NVIDIA.

For more information about CUDA, go to this website:

<https://developer.nvidia.com/cuda-zone>

Functions that run on the GPU are called *kernels*, and are written in C syntax. The CUDA development kit includes a precompiler that can automatically start the C/C++ compiler.

Here is some example C/C++ and CUDA code:

```
#include <cuda.h>
__global__ void grayscaleKernel(uchar3* bgr, int width, int height)
{
    int row = (blockIdx.y * blockDim.y) + threadIdx.y; // builtin CUDA thread-
    int col = (blockIdx.x * blockDim.x) * threadIdx.x; // specific indexes
    if( (row < height) && (col < width) )
    {
        int idx = (row * width) + col; // each call addresses a single pixel
        float blue = (float)bgr[idx].x; // a pixel is 3 bytes: blue,green,red
        float green = (float)bgr[idx].y;
        float red = (float)bgr[idx].z;
        float gray = (.299f * red) + (.587 * green) + (.114f * blue);
        if( gray > 255.0 )
            gray = 255.0;
        bgr[idx].x = gray;
        bgr[idx].y = gray;
        bgr[idx].z = gray;
    }
}

void grayscale( unsigned char* bgr, int width, int height )
{
    uchar3* d_bgr; // 3-byte structure: x=Blue, y=Green, z=Red
    int bytes = width * height * sizeof(uchar3);
    cudaMalloc( &d_bgr, bytes ); // allocate GPU device memory
    cudaMemcpy( d_bgr, bgr, bytes, cudaMemcpyHostToDevice ); // copy data to GPU
    dim3 block(16,16,1), grid(((width-1)/16)+1,((height-1)/16)+1,1);
    grayscaleKernel<<<block,grid>>>(d_bgr,width,height); // invoke kernel(s)
    cudaMemcpy( bgr, d_bgr, bytes, cudaMemcpyDeviceToHost ); // result from GPU
    cudaFree( d_bgr );
}
```

This example converts an uncompressed color (BGR) image into gray scale. The kernel function is identified by the `__global__` specifier. The function works on a single 3-byte pixel. However, the GPU's many cores can run this kernel on many pixels simultaneously. The C-function below the kernel function shows an example of how to run the kernel on the default GPU device. In this example, the image must be copied completely into the GPU's device memory. The kernel function replaces each pixel with its gray-scale equivalent in the device memory. When control is returned from the kernel executions, the device memory is then copied back into regular host CPU memory.

These two functions can be coded in the same source file. The CUDA precompiler (nvcc) processes the kernel functions and then calls the C/C++ compiler automatically.

Here is an example of calling the precompiler from a command line interface:

```
nvcc -c grayscale.cu -o grayscale.o
```

NVIDIA also provides a set of CUDA libraries that include highly optimized kernels for various purposes. Some of these libraries and tools can be found at the following website:

<https://developer.nvidia.com/gpu-accelerated-libraries>

One of these libraries is NVBLAS, which is a CPU implementation that automatically uses the cuBLAS library GPU kernels to accelerate some BLAS calls. For more information about NVBLAS, go to the following website:

<http://docs.nvidia.com/cuda/nvblas/index.html>

CUDA kernels are written as C functions. NVIDIA also provides a C++ library that is called Thrust to better integrate with existing C++ applications. Here is a Thrust example:

```
void sortVector( int* values, int count )
{
    thrust::device_vector<int> d_vec(count); // create device memory
    thrust::copy(values, values+count, d_vec.begin()); // copy data to GPU
    thrust::sort(d_vec.begin(), d_vec.end()); // call builtin sort kernel
    thrust::copy(d_vec.begin(), d_vec.end(), values); // copy result from GPU
}
```

The Thrust library also supports functions and integrates well with the C++ standard template libraries. For more information about the Thrust library, see the following website

<https://developer.nvidia.com/thrust>

For more information about CUDA and POWER8 processor-based systems, see *NVIDIA CUDA on IBM POWER8: Technical Overview, Software Installation, and Application*, REDP-5169.

7.7 Related publications

The publications that are listed in this section are considered suitable for a more detailed description of the topics that are covered in this chapter:

- ▶ *C/C++ Cafe* (IBM Rational®), found at:
<http://www.ibm.com/rational/cafe/community/ccpp>
- ▶ *FDPR, Post-Link Optimization for Linux on Power*, found at:
<https://www.ibm.com/developerworks/mydeveloperworks/groups/service/html/communityview?communityUuid=5a116d75-b560-4152-9113-7515fa73e67a>
- ▶ *Feedback Directed Program Restructuring (FDPR)*, found at:
<https://www.research.ibm.com/haifa/projects/systems/cot/fdpr/>
- ▶ GCC online documentation
 - All versions: <http://gcc.gnu.org/onlinedocs/>
 - Advance Toolchain V6.0: <https://gcc.gnu.org/onlinedocs/gcc-4.7.4/gcc/>
 - Advance Toolchain V7.0 and V7.1: <https://gcc.gnu.org/onlinedocs/gcc-4.8.4/gcc/>
 - Advance Toolchain V8.0: <https://gcc.gnu.org/onlinedocs/gcc-4.9.2/gcc/>

- ▶ XL Compiler Documentation:
 - C and C++ Compilers
 - *C and C++ Compilers family*, found at:
<http://www.ibm.com/software/awdtools/xlcpp/>
 - *Optimization and Programming Guide - XL C/C++ for AIX, V12.1*, found at:
<http://www.ibm.com/support/docview.wss?uid=swg27024208>
 - Fortran compilers
 - *Fortran Compilers family*, found at:
<http://www.ibm.com/software/awdtools/fortran/>
 - *Optimization and Programming Guide - XL Fortran for AIX, V14.1*, found at:
<http://www.ibm.com/support/docview.wss?uid=swg27024219>



Java

This chapter describes the optimization and tuning of Java based applications that are running on a POWER8 processor-based system. It covers the following topics:

- ▶ 8.1, “Java levels” on page 174
- ▶ 8.2, “32-bit versus 64-bit Java” on page 174
- ▶ 8.3, “Memory and page size considerations” on page 175
- ▶ 8.4, “Capitalizing on POWER8 features with IBM Java” on page 181
- ▶ 8.5, “Java garbage collection tuning” on page 183
- ▶ 8.6, “Application scaling” on page 186
- ▶ 8.8, “Related publications” on page 192

8.1 Java levels

For POWER8 processor-based systems, the preferred Java level is Java 8 where possible, or the latest service refresh of Java 7.1 if Java 7 compatibility is still required. Java 7.1 is the only release of Java 7 that is optimized for POWER8 processor-based systems, and takes advantage of POWER8 specific hardware features for performance. Java 8 contains the latest performance improvements for POWER8 processor-based systems and is therefore preferred. For Java 7 compatibility, any version of Java 7.1 is acceptable, but SR1 and later contain additional POWER8 usage and tuning and are therefore preferred.

For POWER7 processor-based systems, Java 8 is also preferred where possible, but it is acceptable to use Java 6 SR7 or later on POWER7. As of Java 6 SR7, the Java virtual machine (JVM) defaults to using 64 KB pages on AIX. Earlier versions defaulted to 4 KB pages, which is the default page size on AIX. For more information, see “Tuning to capitalize on hardware performance features” on page 14 and 8.3.1, “Medium and large pages for Java heap and code cache” on page 175.

The JIT compiler automatically detects on what platform it is running and generates binary code most suitable to, and performing best on, that platform. Java 7.1 and later can recognize the POWER8 processor and best use its hardware features.

Java 8 is the latest major release of IBM Java. This release brings with it improved runtime performance, improved compilation heuristics (to reduce compilation time and improve application ramp-up time), and additional exploitation of POWER8 hardware.

8.2 32-bit versus 64-bit Java

64-bit applications that do not require large amounts of memory typically run slower than 32-bit applications. This situation occurs because of the larger data types, such as 64-bit pointers instead of 32-bit pointers, which increase the demand on memory throughput.

The exception to this situation is when the processor architecture has more processor registers in 64-bit mode than in 32-bit mode and 32-bit application performance is negatively impacted by this configuration. Because of fewer registers, the demand on memory throughput can be higher in 32-bit mode than in 64-bit mode. In such a situation, running an application in 64-bit mode is required to achieve the best performance.

The Power Architecture does not require running applications in 64-bit mode to achieve best performance because 32-bit and 64-bit modes have the same number of processor registers.

Consider the following items:

- ▶ Applications with a small memory requirement typically run faster as 32-bit applications than as 64-bit applications.
- ▶ 64-bit applications have a larger demand on memory because of the larger data types, such as pointers being 64-bit instead of 32-bit, which leads to the following circumstances:
 - The memory foot print increases because of the larger data types.
 - The memory alignment of application data contributes to memory demand.
 - More memory bandwidth is required.

For best performance, use 32-bit Java unless the memory requirement of the application requires running in 64-bit mode. For more information, see 8.3.4, “Compressed references” on page 177 and “32-bit or 64-bit JDK” on page 240.

For more information about this topic, see 8.8, “Related publications” on page 192.

8.2.1 Little Endian support

A Little Endian version of the SDK and runtime environment is available for Java 7.1 SR1 or later. Unlike the Big Endian version, which has a 32-bit version, the Little Endian version of Java has only 64-bit versions. Otherwise, the Little Endian version of Java maintains all of the same capabilities as the Big Endian version. All of the optimization and tuning techniques in the Java section apply to the Little Endian version of Java and can be used to accelerate the performance of Java based applications running on a Little Endian system.

8.3 Memory and page size considerations

IBM Java can take advantage of medium (64 KB) and large (16 MB) page sizes that are supported by the current AIX versions and POWER processors. Using medium or large pages instead of the default 4 KB page size can improve application performance. The performance improvement of using medium or large pages is a result of a more efficient use of the hardware translation caches, which are used when you translate application page addresses to physical page addresses. Applications that are frequently accessing a vast amount of memory benefit most from using pages sizes that are larger than 4 KB.

Table 8-1 shows the hardware and software requirements for 4 KB, 64 KB, and 16 MB pages.

Table 8-1 Page sizes that are supported by AIX and Linux on POWER processors

Page size	Platform	Linux version	AIX version	Requires user configuration
4 KB	All	RHEL 5, SLES 10 and earlier	All	No
64 KB	POWER5+ processor-based systems or later	RHEL 6, SLES 11	AIX 5L V5.3 and later	No
16 MB	POWER4 processor-based system or later	RHEL 5, SLES11	AIX 5L V5.3 and later	Yes

8.3.1 Medium and large pages for Java heap and code cache

Medium and large pages can be enabled for the Java heap and JIT code cache independently of other memory areas. IBM JVM supports at least three page sizes, depending on the platform:

- ▶ 4 KB (default)
- ▶ 64 KB
- ▶ 16 MB

Large pages, specifically 16 MB pages, do have some processing impact and are best suited for long-running applications with large memory requirements. The **-Xlp64k** option provides many of the benefits of 16 MB pages with less impact and can be suitable for workloads that benefit from large pages but do not take full advantage of 16 MB pages.

Starting with IBM Java 6 SR7, the default page size is 64 KB.

Starting with IBM Java 7 SR4 (and Java 6.2.6 SR5), there are more command-line options to specify pagesize for java heap and code cache. The `-Xlp:objectheap:pagesize=<size>` and `-Xlp:codecache:pagesize=<size>` options are supported. To obtain the large page sizes available and the current setting, use the `-verbose:sizes` option. The current settings are the requested sizes and not the sizes that are obtained.

8.3.2 Configuring large pages for Java heap and code cache

In an AIX environment, to use large pages with Java requires both configuring the large pages and setting the `v_pinshm` tunable to a value of one by running `vmo`. The following example demonstrates how to configure dynamically 1 GB of 16 MB pages and set the `v_pinshm` tunable:

```
# vmo -o lgpg_regions=64 -o lgpg_size=16777216 -o v_pinshm=1
```

To configure permanently large pages, the `-r` option must be specified with the `vmo` command. Run `bosboot` to configure the large pages at boot time:

```
► # vmo -r -o lgpg_regions=64 -o lgpg_size=16777216 -o v_pinshm=1
► # bosboot -a
```

Non-root users must have the `CAP_BYPASS_RAC_VMM` capability on AIX enabled to use large pages. The system administrator can add this capability by running `chuser`:

```
# chuser capabilities=CAP_BYPASS_RAC_VMM,CAP_PROPAGATE <user_id>
```

On Linux, 1 GB of 16 MB pages are configured by running `echo`:

```
# echo 64 > /proc/sys/vm/nr_hugepages
```

8.3.3 Prefetching

Prefetching is an important strategy to reduce memory latency and take full advantage of on-chip caches. The `-XtlhPrefetch` option can be specified to enable aggressive prefetching of thread-local heap memory shortly before objects are allocated. This option ensures that the memory that is required for new objects that are allocated from the TLH is fetched into the cache ahead of time if possible, reducing latency and increasing overall object allocation speed.

A POWER8 processor-based system has increased cache sizes compared to POWER7 and POWER7+ processor-based systems, and also features an additional L4 cache. Therefore, it is important to conduct thorough performance evaluations with TLH prefetching to determine whether it is beneficial to the application being run. The `-Xnot1hPrefetch` option can be used to disable explicitly TLH prefetching if it is enabled by default. This option can provide noticeable gains for workloads that frequently allocate objects, such as transactional workloads, but it can also hurt performance if prefetching causes more important data to be thrown out of the cache.

In addition to the TLH prefetching, POWER processors feature a hardware prefetching engine that can detect certain memory allocation patterns and effectively prefetch memory. Applications that access memory in a linear, predictable fashion can benefit from enabling hardware prefetching, but this must be done in cooperation with the operating system. For a brief description of the `dscrct1` and `ppc64_cpu` commands that can be used to affect hardware prefetching on AIX and Linux respectively, see 1.5.1, “Lightweight tuning and optimization guidelines” on page 7.

8.3.4 Compressed references

For huge workloads, 64-bit IBM JVMs might be necessary to meet application needs. The 64-bit processes primarily offer a much larger address space, allowing for larger Java heaps, JIT code caches, and reducing the effects of memory fragmentation in the native heap. However, 64-bit processes also must deal with the increased processing impact. The impact comes from the increased memory usage and decreased cache usage. This impact is present with every object allocation, as each object must now be referred to with a 64-bit address rather than a 32-bit address.

To alleviate this impact, use the **-Xcompressedrefs** option. When this option is enabled, IBM JVM uses 32-bit references to objects instead of 64-bit references wherever possible. Object references are compressed and extracted as necessary at minimal cost. The need for compression and decompression is determined by the overall heap size and the platform on which IBM JVM is running; smaller heaps can do without compression and decompression, eliminating even this impact. To determine the compression and decompression impact for a heap size on a particular platform, run the following command:

```
java -Xcompressedrefs -verbose:gc -version ...
```

The resulting output has the following content:

```
<attribute name="compressedRefsDisplacement" value="0x0" />
<attribute name="compressedRefsShift" value="0x0" />
```

Values of 0 for the named attributes essentially indicate that no work must be done to convert between 32-bit and 64-bit references for the invocation. Under these circumstances, 64-bit IBM JVMs running with **-Xcompressedrefs** can reduce the impact of 64-bit addressing even more and achieve better performance.

With **-Xcompressedrefs**, the maximum size of the heap is much smaller than the theoretical maximum size that is allowed by a 64-bit IBM JVM, although greater than the maximum heap under a 32-bit IBM JVM. Currently, the maximum heap size with **-Xcompressedrefs** is around 31 GB on both AIX and Linux.

Linux problem: Combination of compressed references and large pages

The prelink utility can interfere with using large pages. The problem is reflected as large pages not being used to back the heap if compressed references are used. The request for backing the heap of a size approximately 3.5 GB or larger (such as **-Xmx4g** **-Xms4g** options) with large pages (**-Xlp** option) might not be accepted with compressed references (**-Xcompressedrefs**; default on Java 7.0 SR4 and later). This problem is seen on some Linux distributions with the prelink utility (for example, RHEL 6). This is expected to cause an impact in performance (the actual extent depends on the workload).

The problem occurs because the prelink utility can relink shared libraries that are dynamically linked by the JVM to the particular virtual memory segment that is required by compressed references for the memory mapping of the heap segment. The memory mapping of shared libraries occurs earlier than that of the heap segment, and uses conventional pages, which restrict its entire virtual memory segment to conventional pages. Later, during the heap memory initialization by the JVM, the memory mapping of the heap segment with a different page size (large pages) on that virtual memory segment fails because of that restriction. Then, when the JVM cannot accept the request for backing the heap with large pages, it thus starts falling back to conventional pages.

To verify the problem, you can compare the values of the **requestedPageSize** and **pageSize** attributes in the verbose garbage collection (GC) log.

Example 8-1 depicts the failure. For a heap size of 4 GB, large pages are requested (**requestedPageSize** is 0x1000000, or 16 MB), but conventional pages are obtained (**pageSize** is 0x10000, or 64 kB); the problem does not occur without compressed references (the **-Xnocompressedrefs** option).

Example 8-1 The requestPageSize and pageSize attributes in the verbose GC log (failure)

```
# java -Xlp -Xmx4g -Xms4g -verbose:gc -version 2>&1 | grep -i pageSize
  <attribute name="pageSize" value="0x10000" />
  <attribute name="requestedPageSize" value="0x1000000" />

# java -Xnocompressedrefs -Xlp -Xmx4g -Xms4g -verbose:gc -version 2>&1 | grep -i pageSize
  <attribute name="pageSize" value="0x1000000" />
  <attribute name="requestedPageSize" value="0x1000000" />
```

To resolve the problem, you can choose one of the following methods:

1. Disable prelink completely.

This can be accomplished by removing the prelink settings and package. Consider its applicability and requirement in your environment before proceeding. To accomplish this task, for example, on RHEL 6, run the following commands:

```
# prelink --undo --all
# yum remove prelink
```

2. Disable prelink selectively for the affected shared libraries.

This can be accomplished by discovering the shared libraries that are relinked to the conflicting virtual memory segment, then reverting, and then disabling the prelink setting for those shared libraries in the configuration files of the prelink utility. Use the following steps to perform this action:

- a. Create and compile a Java program that simply waits for some time (5 minutes). It allows you to examine the shared libraries in the memory map of the JVM. See Example 8-2.

Example 8-2 Java program for waiting 5 minutes

```
# cat <<EOF > SleepFiveMinutes.java
public class SleepFiveMinutes {
    public static void main(String[] args)
        throws InterruptedException {
        Thread.sleep(5 * 60 * 1000);
    }
}
EOF

# javac SleepFiveMinutes.java

# date; java SleepFiveMinutes; date
Mon May  4 08:41:29 EDT 2015
Mon May  4 08:46:29 EDT 2015
```

- b. Run the Java program and extract from its memory map the shared libraries in the virtual memory segment between 4 GB (nine hexadecimal digits, inclusive) and 1 TB (11 hexadecimal digits, exclusive). Example 8-3 on page 179 demonstrates those steps, and lists the relevant shared libraries and their load address as defined by the prelink utility (which matches that list of virtual memory addresses that are present in the memory map).

Example 8-3 Discover shared libraries in the JVM 4 GB - 1 TB virtual memory segment

```
# java SleepFiveMinutes &
[1] 3154

# libs="$(grep '[0-9a-z]\{9,10\}-' /proc/3154/smmaps | sort -u -k6 | awk '{ print $6 }')"

# grep '[0-9a-z]\{9,10\}-' /proc/3154/smmaps | sort -u -k6
8001230000-8001240000 rw-p 00000000 00:00 0
8001000000-8001030000 r-xp 00000000 fd:00 524686 /lib64/ld-2.12.so
8001050000-8001210000 r-xp 00000000 fd:00 524687 /lib64/libc-2.12.so
8001240000-8001250000 r-xp 00000000 fd:00 524689 /lib64/libdl-2.12.so
8001470000-8001490000 r-xp 00000000 fd:00 524696 /lib64/libgcc_s-4.4.7-20120601.so.1
8001330000-8001410000 r-xp 00000000 fd:00 524695 /lib64/libm-2.12.so
8001270000-8001290000 r-xp 00000000 fd:00 524697 /lib64/libpthread-2.12.so
8001e00000-8001e20000 r-xp 00000000 fd:00 524704 /lib64/libresolv-2.12.so
80012b0000-80012c0000 r-xp 00000000 fd:00 524386 /lib64/librt-2.12.so

# for lib in $libs; do objdump -p $lib | grep -m1 LOAD | awk '{ printf $5 }'; echo " $lib";
done
0x0000008001000000 /lib64/ld-2.12.so
0x0000008001050000 /lib64/libc-2.12.so
0x0000008001240000 /lib64/libdl-2.12.so
0x0000008001470000 /lib64/libgcc_s-4.4.7-20120601.so.1
0x0000008001330000 /lib64/libm-2.12.so
0x0000008001270000 /lib64/libpthread-2.12.so
0x0000008001e00000 /lib64/libresolv-2.12.so
0x00000080012b0000 /lib64/librt-2.12.so

# kill -9 3154
```

- c. Revert the prelink setting to the shared libraries (which has an immediate effect), and configure the **ibm-java.conf** prelink configuration file so that it does not relink those shared libraries anymore (by using the **-b** option). These steps are described in Example 8-4.

Example 8-4 Revert and disable the prelink setting to the shared libraries

```
# for lib in $libs; do prelink --undo $lib; echo "-b $lib" >>
/etc/prelink.conf.d/ibm-java.conf; done

# for lib in $libs; do objdump -p $lib | grep -m1 LOAD | awk '{ printf $5
}'; echo " $lib"; done
0x0000000000000000 /lib64/ld-2.12.so
0x0000000000000000 /lib64/libc-2.12.so
0x0000000000000000 /lib64/libdl-2.12.so
0x0000000000000000 /lib64/libgcc_s-4.4.7-20120601.so.1
0x0000000000000000 /lib64/libm-2.12.so
0x0000000000000000 /lib64/libpthread-2.12.so
0x0000000000000000 /lib64/libresolv-2.12.so
0x0000000000000000 /lib64/librt-2.12.so

# cat /etc/prelink.conf.d/ibm-java.conf
-b /lib64/ld-2.12.so
-b /lib64/libc-2.12.so
-b /lib64/libdl-2.12.so
-b /lib64/libgcc_s-4.4.7-20120601.so.1
-b /lib64/libm-2.12.so
-b /lib64/libpthread-2.12.so
```

```
-b /lib64/libresolv-2.12.so
-b /lib64/librt-2.12.so
```

After performing one of these methods, the problem should be resolved.

You can verify the equality between the values of the **requestedPageSize** and **pageSize** attributes, and inspect the virtual memory segment between 4 GB - 1 TB to verify that there are no shared libraries. Example 8-5 describes that verification for equal values and lists the memory map of the heap segment of 4 GB size (0x800000000 - 0x700000000 = 0x100000000 bytes = 4 GB) with 16 MB pages.

Example 8-5 The requestPageSize and pageSize attributes in the verbose GC log (success)

```
# java -Xlp -Xmx4g -Xms4g -verbose:gc -version SleepFiveMinutes 2>&1 | grep -i
pagesize &
  <attribute name="pageSize" value="0x1000000" />
  <attribute name="requestedPageSize" value="0x1000000" />
[1] 4072

# grep '^([0-9a-z])\|^KernelPageSize:' /proc/4072/smaps | grep -A1
'^([0-9a-z])\{9,10\}-'
700000000-800000000 rw-s 00000000 00:0c 983040
/SYSV00000000 (deleted)
KernelPageSize:    16384 kB
```

8.3.5 JIT code cache

JIT compilation is an important factor in optimizing performance. Because compilation is carried out at run time, it is complicated to estimate the size of the program or the number of compilations that are carried out. The JIT compiler has a cap on how much memory it can allocate at run time to store compiled code. For most applications, the default cap is more than sufficient.

However, certain programs, especially those programs that take advantage of certain language features, such as reflection, can produce a number of compilations and use up the allowed amount of code cache. After the limit of code cache is consumed, no more compilations are performed. This situation can have a negative impact on performance if the program calls many interpreted methods that cannot be compiled as a result. The **-Xjit:codetotal=<nnn>** (where *nnn* is a number in KB units) option can be used to specify the cap of the JIT code cache. The default is 64 MB or 128 MB for 32-bit and 64-bit IBM JVMs.

As of Java 7 SR6, the **-Xcodecachetotal<size>** option is the preferred way to specify the total amount of memory that is dedicated to the compiled code.

Another consideration is how the code caches are allocated. If they are allocated far apart from each other (more than 32 MB away), calls from one code cache to another carry a higher processing impact. The **-Xcodecache<size>** option can be used to specify how large each allocation of code cache is. For example, **-Xcodecache4m** means 4 MB is allocated as code cache each time the JIT compiler needs a new one until the cap is reached. Typically, there are multiple pieces (for example, four) of code cache that are found at start time to support multiple compilation threads. It is important to alter the default code cache size only if it is insufficient, as a large but empty code cache needlessly consumes resources.

-Xcodecachetotal<size> is the preferred option in Java 7.1. Java 7 SR6 and later, and Java 8 SR15 and later and are fully documented and supported.

Two techniques can be used to determine whether the code cache allocation sizes or total limit must be altered. First, a Java core file can be produced by running `kill -3 <pid>` at the end/stable state of your application. The core file shows how many pieces of code cache are allocated. The active amount of code cache can be estimated by summing all of the pieces.

For example, if 20 MB is needed to run the application, `-Xcodecache5m` (four pieces of 5 MB each) typically allocates 20 MB code caches at start time, and they are likely close to each other and have better performance for cross-code cache calls. Second, to determine whether the total code cache is sufficient, the `-Xjit:verbose` option can be used to print method names as they are compiled. If compilation fails because the limit of code cache is reached, an error to that effect is printed.

8.3.6 Shared classes

IBM JVM supports class data sharing between multiple IBM JVM instances. The `-Xshareclasses` option can be used to enable class data sharing, and the `-Xscmx<size>` option can be used to specify the maximum cache size of the stored data, where `<size>` can be `<nnn>K`, `<nnn>M`, or `<nnn>G` for sizes in KB, MB, or GB.

The shared class data is stored in a memory-mapped cache file on disk. Sharing reduces the overall virtual storage consumption when more than one IBM JVM shares a cache. Sharing also reduces the start time for an IBM JVM after the cache is created. The shared class cache is independent of any running IBM JVM and persists until it is deleted.

A shared cache can contain:

- ▶ Bootstrap classes
- ▶ Application classes
- ▶ Metadata that describes the classes
- ▶ Ahead-of-time (AOT) compiled code

8.4 Capitalizing on POWER8 features with IBM Java

The following sections describe how to maximize POWER8 features for encryption, transactional memory (TM), and runtime instrumentation by using IBM Java.

8.4.1 In-core Advanced Encryption Standard and Secure Hash Algorithm acceleration and instructions

Ensuring confidentiality through encryption is a computationally intensive aspect of workloads that is becoming increasingly important. POWER8 processor-based systems introduce in-core Advanced Encryption Standard (AES) and Secure Hash Algorithm (SHA) instructions that are compliant with the FIPS 197: AES Specification, and FIPS 180: Secure Hash Standard.

Starting with IBM Java 7.1, AES is accelerated by using POWER8 in-core AES instructions by specifying `-Dcom.ibm.crypto.provider.doAESInHardware=true` on the JVM command line. In-core AES instructions can increase speed, as compared with equivalent JIT-generated code.

Starting with IBM Java 8, SHA2 (for example, SHA224, SHA256, SHA384, and SHA512) is accelerated by using POWER8 in-core SHA instructions. SHA2 is enabled by default and no command-line parameter is required. In-core SHA instructions can increase speed, as compared with equivalent JIT-generated code.

8.4.2 Transactional memory

POWER8 Hardware Transaction Memory (HTM) is used by IBM JVM in two aspects, both of which are intended to be transparent to Java application programmers and JVM users:

- ▶ Transactional Lock Elision (TLE)
- ▶ Targeted class exploitation

Starting with IBM Java V7.1, and while HTM is enabled on the platform (AIX or Linux), this usage can occur transparently.

The JIT compiler automatically chooses particular Java synchronization blocks to transform into HTM regions. Only the blocks that are deemed to benefit from the transformation in terms of performance are chosen. When those blocks behave synergistically with HTM, application scalability and performance can be improved. Because the transformation is automatic, TLE is transparent to programmers and users. Concurrently, certain classes, including `ConcurrentHashMap` and `ConcurrentLinkedQueue`, were rewritten to take advantage of HTM in IBM JVM. These classes work on processors that do not support HTM, but they can take advantage of HTM running on POWER8 processor-based systems transparently.

However, application programmers can modify applications to take advantage of TLE or HTM by targeting the applications specifically for POWER8 processor-based systems. Because they are data structures of dispersing nature, which are less likely to be accessed in the same cache line and therefore conflict with each other, `HashTable/HashMap/Map` behaves well with HTM and TLE. When the application is modified to use more of the classes that are mentioned (`ConcurrentHashMap` and `ConcurrentLinkedQueue`), the application is more likely to benefit from TLE.

Note: Using more of the classes that are mentioned (`ConcurrentHashMap` and `ConcurrentLinkedQueue`) might adversely affect performance on POWER7 or older processors that do not support HTM.

For more information about the topic of transactional memory, from the processor, OS, and compiler perspectives, see:

- ▶ 2.2.4, “Transactional memory” on page 42 (*processor*)
- ▶ 4.2.4, “Transactional memory” on page 89 (*AIX*)
- ▶ 6.2.4, “Transactional memory” on page 124 (*Linux*)
- ▶ 7.3.5, “Transactional memory” on page 156 (*XL and GCC compiler families*)

8.4.3 Runtime instrumentation

IBM Java 7 SR1 and later uses the POWER8 event-based branching facility and enhanced performance monitoring unit (PMU) to enable runtime instrumentation of compiled code. Runtime instrumentation allows the JIT compiler to collect detailed performance information directly from the hardware, without any kernel or system call impact, and in turn, use this information to further optimize compiled code. The PMU is the same unit that is used by external profiling tools, such as **tprof** and **hpmcount** on AIX, and **perf** and **OProfile** on Linux. The POWER8 processor allows the PMU to be used for application self-profiling. JIT profiling, and optimization focus on collecting information that otherwise is difficult to collect without hardware assistance and better utilization of cache and TLB resources, and on reducing function call impact and branch mispredicts, among others.

For Java 8, runtime instrumentation is used to help guide compilation heuristics, which reduces warm-up time and improves ramp-up by allowing the JIT compiler to be more selective. This selectivity is with regard to the Java methods it chooses to compile and the optimization level at which it compiles those methods. This results in less CPU time being spent on compilation and faster compilation turnaround times, which allows the application to reach peak performance sooner.

On current Linux kernels for each thread, the PMU can be used only by one party at any given moment. This means that if the JVM is using the PMU to profile one or more application threads, a system profiler such as **perf** or **OProfile** cannot be used to profile the same thread by using the PMU. To work around this limitation, the system profiler can be configured to use a timer-based profiling mechanism rather than the PMU. Another option is to disable the runtime instrumentation on the JVM by using the **-XX:-RuntimeInstrumentation** option. This limitation might be fixed in future Linux kernels.

8.5 Java garbage collection tuning

The IBM Java VM supports multiple garbage collection (GC) strategies to allow software developers an opportunity to prioritize various factors. Throughput, latency, and scaling are the main factors that are addressed by the different collection strategies. Understanding how an application behaves regarding allocation frequencies, required heap size, expected lifetime of objects, and other factors can make one or more of the non-default GC strategies preferable. The GC strategy can be specified with the **-Xgcpolicy:<policy>** option.

8.5.1 GC strategy: Optthruput

This strategy prioritizes throughput at the expense of maximum latency by waiting until the last possible time to do a GC. A global GC of the entire heap is performed, creating a longer pause time at the expense of latency. After GC is triggered, the GC stops all application threads and performs the three GC phases:

- ▶ Mark
- ▶ Sweep
- ▶ Compact (if necessary)

Each phase is parallelized to perform GC as quickly as possible.

The optthruput strategy is the default in the original Java 6 that uses the IBM JVM V2.4 J9.

8.5.2 GC strategy: Optavgpause

This strategy prioritizes latency and response time by performing the initial mark phase of GC concurrently with the running of the application. The application is halted only for the sweep and compact phases, minimizing the total time that the application is paused. Performing the mark phase concurrently with the running of the application might affect throughput because the CPU time that perform out the mark phase. This situation can be acceptable on machines with many processor cores and relatively few application threads, as idle processor cores can be put to good use otherwise.

8.5.3 GC strategy: Gencon

This strategy employs a generational GC scheme that attempts to deal with many varying workloads and memory usage patterns. In addition, gencon also uses concurrent marking to minimize pause times. The gencon strategy works by dividing the heap into two categories:

- ▶ New space
- ▶ Old space

The new space is dedicated to short-lived objects that are created frequently and unreferenced shortly thereafter. The old space is for long-lived objects that survived long enough to be promoted from the new space. This GC policy is suited to workloads that have many short-lived objects, such as transactional workloads, because GC in the new space (carried out by the *scavenger*) is cheaper per object overall than GC in the old space. By default, up to 25% of the heap is dedicated to the new space. The division between the new space and the old space can be controlled with the `-Xmn` option, which specifies the size of the new space; the remaining space is then designated as the old space. Alternatively, `-Xmns` and `-Xmnx` can be used to set the starting and maximum new space sizes if a non-constant new space size is wanted. For more information about constant versus non-constant heaps in general, see 8.5.5, “Optimal heap size” on page 185.

The gencon strategy is the default in the updated Java 7.1 that uses the IBM JVM V2.6 J9, and in the later Java 7 version.

8.5.4 GC strategy: Balanced

This strategy evens out pause times across GC operations that are based on the amount of work that is being generated. This strategy can be affected by object allocation rates, object survival rates, and fragmentation levels within the heap. This smoothing of pause times is a best effort rather than a real-time guarantee. A fundamental aspect of the balanced collector's architecture, which is critical to achieving its goals of reducing the impact of large collection times, is that it is a region-based garbage collector. A region is a clearly delineated portion of the Java object heap that categorizes how the associated memory is used and groups related objects together.

During the IBM JVM start, the garbage collector divides the heap memory into equal-sized regions, and these region delineations remain static for the lifetime of the IBM JVM. Regions are the basic unit of GC and allocation operations. For example, when the heap is expanded or contracted, the memory that is committed or released corresponds to a number of regions.

Although the Java heap is a contiguous range of memory addresses, any region within that range can be committed or released as required. This situation enables the balanced collector to contract the heap more dynamically and aggressively than other garbage collectors, which typically require the committed portion of the heap to be contiguous. Java heap configuration for the `-Xgcpolicy:balanced` strategy can be specified through the `-Xmn`, `-Xmx`, and `-Xms` options.

8.5.5 Optimal heap size

By default, the IBM JVM provides a considerably flexible heap configuration that allows the heap to grow and shrink dynamically in response to the needs of the application. This configuration allows the IBM JVM to claim only as much memory as necessary at any time, thus cooperating with other processes that are running on the system. The starting and maximum size of the heap can be specified with the `-Xms` and `-Xmx` options.

This flexibility comes at a cost, as the IBM JVM must request memory from the operating system whenever the heap must grow and return memory whenever it shrinks. This behavior can lead to various unwanted scenarios. If the application heap requirements oscillate, this situation can cause excessive heap growth and shrinkage.

If the IBM JVM is running on a dedicated machine, the processing impact of heap resizing can be eliminated by requesting a constant sized heap. This situation can be accomplished by setting `-Xms` equal to `-Xmx`. Choosing the correct size for the heap is highly important, as GC impact is directly proportional to the size of the heap. The heap must be large enough to satisfy the application's maximum memory requirements and contain extra space. The GC must work much harder when the heap is near full capacity because of fragmentation and other issues, so 20 - 30% of extra space above the maximum needs of the application can lower the overall GC impact.

If an application requires more flexibility than can be achieved with a constant sized heap, it might be beneficial to tune the sizing parameters for a dynamic heap. One of the most expensive GC events is *object allocation failure*. This failure occurs when there is not enough contiguous space in the current heap to satisfy the allocation, and results in a GC collection and a possible heap expansion. If the current heap size is less than the `-Xmx` size, the heap is expanded in response to the allocation failure if the amount of free space is below a certain threshold. Therefore, it is important to ensure that when an allocation fails, the heap is expanded to allow the failed allocation and many future allocations to succeed, or the next failed allocation might trigger yet another GC collection. This situation is known as *heap thrashing*.

The `-Xminf`, `-Xmaxf`, `-Xmine`, and `-Xmaxe` group of options can be used to affect when and how the GC resizes the heap. The `-Xminf<factor>` option (where factor is a real number 0 - 1) specifies the minimum free space in the heap; if the total free space falls below this factor, the heap is expanded. The `-Xmaxf<factor>` option specifies the maximum free space; if the total free space rises above this factor, the heap is shrunk. These options can be used to minimize heap thrashing and excessive resizing. The `-Xmine` and `-Xmaxe` options specify the minimum and maximum sizes by which to shrink and grow the heap. These options can be used to ensure that the heap has enough free contiguous space to allow it to satisfy a reasonable number of allocations before failure.

Regardless of whether the heap size is constant, it should never be allowed to exceed the physical memory that is available to the process; otherwise, the operating system might have to swap data in and out of memory. An application's memory behavior can be determined by using various tools, including verbose GC logs. For more information about verbose GC logs and other tools, see "Java (either AIX or Linux)" on page 239.

8.6 Application scaling

Large workloads using many threads on multi-CPU machines face extra challenges regarding concurrency and scaling. In such cases, steps can be taken to decrease contention on shared resources and reduce the processing impact.

8.6.1 Choosing the correct simultaneous multithreading mode

AIX and Linux represent each SMT thread as a logical CPU. Therefore, the number of logical CPUs in an LPAR depends on the SMT mode. For example, an LPAR with four virtual processors that are running in SMT4 mode has 16 logical CPUs; an LPAR with that same number of virtual processors that are running in SMT2 mode has only eight logical CPUs.

Table 8-2 shows the number of SMT threads and logical CPUs that are available in ST, SMT2, SMT4, and SMT8 modes.

Table 8-2 ST, SMT2, SMT4, and SMT8 modes - SMT threads and CPUs available

SMT mode	Number of SMT threads	Number of logical CPUs
ST	1	1
SMT2	2	2
SMT4	4	4
SMT8	8	8

The default SMT mode on a POWER7 and later processor depends on the AIX version and the compatibility mode with which the processor cores are running. Table 8-3 shows the default SMT modes.

Table 8-3 SMT mode on the POWER8 processor depends on the AIX and compatibility mode

AIX version	Compatibility mode	Default SMT mode
AIX V7.1 TL3 SP3	POWER8	SMT4
AIX V6.1	POWER7	SMT4
AIX V6.1	POWER6/POWER6+	SMT2
AIX 5L V5.3	POWER6/POWER6+	SMT2

Most applications benefit from SMT. However, some applications do not scale with an increased number of logical CPUs on an SMT-enabled system. One way to address such an application scalability issue is to make a smaller LPAR or use processor binding, as described in 8.6.2, “Using resource sets” on page 187.

Additionally, if you need improved performance from your larger new system, there is a potential alternative. If your application semantics support it, you might be able to run multiple smaller instances of your application, each bound to exclusive processors/cores. In this way, the aggregate performance from the multiple instances of your application might be able to meet your performance expectations. This alternative is one of the WebSphere preferred practices. For more information about selecting an appropriate SMT mode, see “Scalability challenges when moving from a POWER5 or POWER6 processor-based system to a POWER7 or POWER8 processor-based system” on page 208.

For applications that might benefit from a lower SMT mode with fewer logical CPUs, experiment with using SMT2 or ST modes. For more information, from the processor and OS perspectives, see:

- ▶ “Simultaneous multithreading” on page 29 (*processor*)
- ▶ “Simultaneous multithreading” on page 73 (*AIX*)
- ▶ “Simultaneous multithreading” on page 112 (*IBM i*)
- ▶ “Simultaneous multithreading” on page 119 (*Linux*)

Java application scaling on Linux

Java applications scale better on Linux in some cases if the `sched_compat_yield` scheduler tunable is set to 1 by running the following command:

```
sysctl -w kernel.sched_compat_yield=1
```

For more information about this topic, see:

- ▶ “Deployment guidelines” on page 15 (*Linux*)
- ▶ “Simultaneous multithreading” on page 119 (*Linux*)

8.6.2 Using resource sets

This section describes the use of resource sets (RSETs) in AIX and Linux environments.

AIX environment

In an AIX environment, RSETs allow specifying on which logical CPUs an application can run. They are useful when an application that does not scale beyond a certain number of logical CPUs is run on a large LPAR, for example, an application that scales up to eight logical CPUs but is run on an LPAR that has 64 logical CPUs.

For more information, see “The POWER8 processor and affinity performance effects” on page 16. An example is included in “Partition sizes and affinity” on page 16.

RSETs can be created with the `mkrset` command and attached to a process by using the `attachrset` command. An alternative way is creating an RSET and attaching it to an application in a single step by using the `execrset` command.

The following example demonstrates how to use `execrset` to create an RSET with CPUs 4 - 7 and run an application that is attached to it:

```
execrset -c 4-7 -e <application>
```

In addition to running the application that is attached to an RSET, set the `MEMORY_AFFINITY` environment variable to MCM to assure that the application’s private and shared memory is allocated from memory that is local to the logical CPUs of the RSET:

```
MEMORY_AFFINITY=MCM
```

In general, RSETs are created on core boundaries. For example, a partition with four POWER8 cores that are running in SMT4 mode has 16 logical CPUs. Create an RSET with four logical CPUs by selecting four SMT threads that belong to one core. Create an RSET with eight logical CPUs by selecting eight SMT threads that belong to two cores. The `smtctl` command can be used to determine which logical CPUs belong to which core, as shown in Example 8-6.

Example 8-6 Use the `smtctl` command to determine which logical CPUs belong to which core

```
# smtctl
This system is SMT capable.
This system supports up to 4 SMT threads per processor.
SMT is currently enabled.
SMT boot mode is not set.
SMT threads are bound to the same physical processor.

proc0 has 4 SMT threads.
Bind processor 0 is bound with proc0
Bind processor 1 is bound with proc0
Bind processor 2 is bound with proc0
Bind processor 3 is bound with proc0

proc4 has 4 SMT threads.
Bind processor 4 is bound with proc4
Bind processor 5 is bound with proc4
Bind processor 6 is bound with proc4
Bind processor 7 is bound with proc4
```

The `smtctl` output in Example 8-6 shows that the system is running in SMT4 mode with bind processors (logical CPU) 0 - 3 belonging to `proc0` and bind processors 4 - 7 belonging to `proc1`. Create an RSET with four logical CPUs either for CPUs 0 - 3 or for CPUs 4 - 7.

To achieve the best performance with RSETs that are created across multiple cores, all cores of the RSET must be from the same chip and in the same scheduler resource allocation domain (SRAD). The `lssrad` command can be used to determine which logical CPUs belong to which SRAD, as shown in Example 8-7:

Example 8-7 Use the `lssrad` command to determine which logical CPUs belong to which SRAD

```
# lssrad -av
REF1 SRAD      MEM      CPU
0
      0 22397.25    0-31
1
      1 29801.75   32-63
```

The output in Example 8-7 shows a system that has two SRADs. CPUs 0 - 31 belong to the first SRAD, and CPUs 32 - 63 belong to the second SRAD. In this example, create an RSET with multiple cores either by using the CPUs of the first or second SRAD.

Authority for RSETs: A user must have root authority or have `CAP_NUMA_ATTACH` capability to use RSETs.

Linux environment

In a Linux environment, the equivalent to **execrset** is the **taskset** command. The following example demonstrates how to use **taskset** to create a taskset with CPUs 4 - 7 and run an application that is attached to it:

```
Linux: taskset -c 4-7 <application>
```

There is no equivalent environment variable to **MEMORY_AFFINITY** on Linux; however, there is a command, **numactl**, that can accomplish the same task as **MEMORY_AFFINITY** and the **execrset** and **taskset** commands. For example:

```
numactl [-l | --localalloc] -C 4-7 <application>
```

The `-l | --localalloc` option is analogous to **MEMORY_AFFINITY=MCM**.

8.6.3 Java lock reservation

Synchronization and locking are an important part of any multi-threaded application. Shared resources must be adequately protected by monitors to ensure correctness, even if some resources are only infrequently shared. If a resource is primarily accessed by a single thread at any time, that thread is frequently the only thread to acquire the monitor that is guarding the resource. In such cases, the cost of acquiring the monitor can be reduced by using the **-XlockReservation** option. With this option, it is assumed that the last thread to acquire the monitor is also likely to be the next thread to acquire it. The lock is, therefore, said to be reserved for that thread, minimizing its cost to acquire and release the monitor.

This option is suited to workloads that use many threads and many shared resources that are infrequently shared in practice.

8.6.4 Java GC threads

The GC that is used by IBM JVM takes every opportunity to use parallelism on multi-CPU machines. All phases of the GC can be run in parallel with multiple helper threads dividing up the work to complete the task as quickly as possible. Depending on the GC strategy and heap size in use, it can be beneficial to adjust the number of threads that the GC uses. The number of GC threads can be specified with the **-Xgcthreads<number>** option. The default number of GC threads is equal to the number of logical processors on the partition, and it is not helpful to exceed this value. Reducing it, however, reduces the GC impact and might be wanted in some situations, such as when RSETs are used. The number of GC threads is capped at 64 starting in IBM JVM V2.6 J9.

8.6.5 Java concurrent marking

The gencon policy combines concurrent marking with generational GC. If generational GC is wanted but the impact of concurrent marking, regarding both the impact of the marking thread and the extra book-keeping that is required when you allocate and manipulate objects, is not wanted, then concurrent marking can be disabled by using the **-Xconcurrentlevel0** option. This option is appropriate for workloads that benefit from the gencon policy for object allocation and lifetimes, but also require maximum throughput and minimal GC impact while the application threads are running.

In general, for both the gencon and optavgpause GC policies, concurrent marking can be tuned with the `-Xconcurrentlevel<number>` option, which specifies the ratio between the amount of heap that is allocated and heap that is marked. The default value is 8. The number of low-priority mark threads can be set with the `-Xconcurrentbackground<number>` option. By default, one thread is used for concurrent marking.

For more information about this topic, see 8.8, “Related publications” on page 192.

8.7 Using GPU accelerators with IBM Java

GPU accelerators can be used to increase substantially Java software performance when certain conditions are met. The GPU can be accessed from Java by using several different techniques that are fully compatible and supported by the IBM Java platform.

Regardless of the choice of the Java technology (to be detailed later) that is used to use the GPU, the software should have some or all of these conditions met:

1. Key code segments (hot spots) where the Java program spends the majority of its time should exist. For example, processing large data sets and complex mathematical operations.
2. These code segments can be expressed as parallel operations or as a short sequence of parallel operations. For example, sorting, linear algebra operations, and other operations that are highly parallel.
3. The parallelism in these operations should be of a fine grain type, which means that several thousand threads can operate concurrently to complete the overall operation.
4. Sufficient computation per data item. Each data item is used several times.

These items are only rules of thumb and there might be situations where they either fail or other situations where the GPU still provides performance benefit even if some conditions fail.

Ultimately, one must experiment with GPU acceleration and observe whether there are indeed performance advantages in any given situation.

8.7.1 Automatic GPU compilation

The IBM Java Just-In-Time (JIT) compiler is able to offload certain processing tasks to a GPU without any user involvement or special knowledge. The only requirement on the user is to express the computation as a parallel loop by using Java Lambda expressions.

The JIT determines at run time whether a parallel loop is suitable to be offloaded to the GPU by using certain performance heuristics.

Here are the two Java constructs that can be accelerated by the GPU:

- ▶ `IntStream.range(<range>).parallel().forEach(<lambda>)`
- ▶ `IntStream.rangeClosed(<range>).parallel().forEach(<lambda>)`

Where `<range>` defines upper and lower bounds and `<lambda>` is a correctly defined lambda expression.

The lambda expression may use variables and one-dimensional arrays of all Java primitive types and automatic, parameter, and instance variables. Also, you may use all standard Java exceptions.

The following items are not supported:

- ▶ Method invocations
- ▶ Intermediate operations, such as map or filter
- ▶ User-defined exceptions
- ▶ New/delete statements

To enable GPU processing of the parallel loops, set the `-Xjit:enableGPU` option on the command line when you start your Java application.

For more information, see the Java 8 SDK documentation, found at:

https://www-01.ibm.com/support/knowledgecenter/SSYKE2_8.0.0/com.ibm.java.lnx.80.doc/diag/understanding/gpu_jit.html

8.7.2 Accessing the GPU through the CUDA4J application programming interface

You can use the NVIDIA Compute Unified Device Architecture 4J (CUDA4J) API to develop applications that can specify exactly when to use the GPU for application processing. Unlike the JIT automatic GPU compilation, you have explicit control over the GPU operations and under which conditions they are started. However, you must write and maintain the GPU source code by using the CUDA4J API.

There are many classes that are available in the CUDA4J API, which are described in the Java 7 API reference (or later), found at:

https://www-01.ibm.com/support/knowledgecenter/SSYKE2_8.0.0/com.ibm.java.lnx.80.doc/user/gpu_developing_cuda4j.html?lang=en

8.7.3 The com.ibm.gpu application programming interface

You can use the `com.ibm.gpu` API to develop applications that sort arrays of primitive types (int, long, float, or double) on the GPU. Parallel processing of sort operations on data arrays can improve performance if the array is large enough to justify the impact of moving the data. Testing indicates that moving the data from the CPU to the GPU is cost neutral at 20,000 entries, but for larger arrays, you see a reduced sort time when you use the GPU rather than the CPU.

For more information, see the Java 7 API reference (or later), found at:

https://www-01.ibm.com/support/knowledgecenter/SSYKE2_8.0.0/com.ibm.java.lnx.80.doc/user/gpu_developing_sort.html?lang=en

8.7.4 NVIDIA Compute Unified Device Architecture: Java Native interface

You can always use the Java Native Interface (JNI) to call CUDA kernels directly, which is done by using standard JNI techniques. For example:

```
// Java source code
import java.nio.ByteBuffer;
class CudaTest1
{
    static { System.loadLibrary("cudatest1"); } // native shared-obj or DLL
    private void grayscale( byte[] bgr, int width, int height )
    {
```

```

        ByteBuffer buffer = ByteBuffer.allocateDirect(data.length);
        buffer.put(data,0,data.length);
        buffer.position(0);
        grayscale0(buffer,width,height);
        buffer.get(data);
    }
    private static native void grayscale0(ByteBuffer buffer, int width, int
height);
...
}

// C source code
#include <jni.h>
JNIEXPORT void JNICALL Java_CudaTest1_grayscale0(JNIEnv* env, jclass,
        jobject buffer, jint width, jint height)
{
    unsigned char* bytes = (unsigned char*)env->GetDirectBufferAddress(buffer);
    int length = width * height * 3; // 3 bytes per pixel
    grayscale(bytes,width,height); // see C/C++ section on example implementation
}

```

8.8 Related publications

The publications that are listed in this section are considered suitable for a more detailed description of the topics that are covered in this chapter:

- *Java Performance on POWER7 – Best Practice*, found at:
http://www.ibm.com/systems/power/hardware/whitepapers/java_perf.htm
- *Java Performance on POWER7*, found at:
<https://www.ibm.com/developerworks/wikis/display/LinuxP/Java+Performance+on+POWER7>
- *Top 10 64-bit IBM WebSphere Application Server FAQ*, found at:
ftp://public.dhe.ibm.com/software/webservers/appserv/WAS_64-bit_FAQ.pdf



IBM DB2

This chapter describes the optimization and tuning of DB2 running on POWER processor-based servers. It covers the following topics:

- ▶ 9.1, “DB2 and the POWER processor” on page 194
- ▶ 9.2, “Taking advantage of the POWER processor” on page 194
- ▶ 9.3, “Capitalizing on the compilers and optimization tools for POWER” on page 197
- ▶ 9.4, “Capitalizing on POWER virtualization” on page 198
- ▶ 9.5, “Capitalizing on the AIX system libraries” on page 199
- ▶ 9.6, “Capitalizing on performance tools” on page 201
- ▶ 9.7, “Conclusion” on page 202
- ▶ 9.8, “Related publications” on page 202

9.1 DB2 and the POWER processor

IBM DB2 is positioned to take full advantage of the Power Architecture. This chapter refers to DB2 10, including DB2 10.1, DB2 10.5, and all subsequent updates. References to *POWER* are to the POWER7, POWER7+, and the new POWER8 processors.

DB2 offers many capabilities that are tailored to use POWER processor features. The DB2 self-tuning memory manager (STMM) feature is one of many features that can help you efficiently consolidate DB2 workloads on Power Systems. Additionally, DB2 is one of the most optimized software applications on Power Systems. During the DB2 development cycles, IBM evaluates new POWER processor capabilities and tests and characterizes the performance of DB2 on the latest Power Systems servers. So far, in the earlier chapters of this book, you read detailed descriptions about most of these POWER guidelines and technologies. The focus of this chapter is to showcase how DB2 uses various POWER features and preferred practices from this guide during its own software development cycle, which is done to maximize performance on the Power Architecture. General DB2 tuning and preferred practices of DB2 are covered extensively in many other places, some of which are listed in 9.8, “Related publications” on page 202.

Most of the Power Systems exploitation capabilities of DB2 extend to the POWER7, POWER7+, and POWER8 processors. A number of the new POWER8 capabilities are evolutionary in nature, and no new externals are required in DB2 (just verification and some adjustment of the internal data structures and algorithms in DB2 to take full advantage of the new capabilities). Similarly, DB2 evolves, so maximizing the benefits of POWER processors in DB2 10.1 has been extended and enhanced in DB2 10.5.

However, there are also new capabilities in DB2 10.5 that take advantage of new POWER processor capabilities. For example, the new columnar in-memory analytic processing capability that is known as BLU Acceleration uses the POWER VSX engine on all of the POWER7 and later processors.

As of DB2 10.5 FP5, DB2 supports the Linux on Power Little Endian platform on POWER8 processors running under RHEL 7.1 and Ubuntu 14.4. Most of the material in this chapter specifically refers to DB2 running under AIX. Where appropriate, specific details are provided about Linux on Power Little Endian.

9.2 Taking advantage of the POWER processor

Methods for taking advantage of the inherent power of the POWER processor include affinization, page size, decimal arithmetics, and the usage of simultaneous multithreading (SMT) priorities for internal lock implementation. New in DB2 10.5 is the ability to use Single Instruction Multiple Data (SIMD) processing with the VSX engine.

9.2.1 Affinization

A simple way to achieve affinization on POWER7 and POWER8 processor-based systems is through the DB2 registry variable DB2_RESOURCE_POLICY. In general, this variable defines a policy that outlines which operating system resources are available for DB2 databases. When this variable is set to AUTOMATIC, the DB2 database system automatically detects the POWER hardware topology and computes the best way to assign engine dispatchable units (EDUs) to various hardware modules. The goal is to determine the most efficient way to share memory between multiple EDUs that need access to the same regions of memory.

On AIX, the AUTOMATIC setting uses Scheduler Resource Allocation Domain Identifier (SRADID) attachments for affinity purposes. On Linux, the AUTOMATIC setting uses NUMA nodes (as exposed by libnuma) for affinity purposes.

The AUTOMATIC setting can be used on POWER7 and POWER8 processor-based systems running the following or later releases:

- ▶ AIX V6.1 Technology Level (TL) 5 with DB2 10.1
- ▶ Linux (Little Endian) with DB2 10.5 FP5

This setting is intended for multi-socket SCM and all DCM Power Systems. It is best to run a performance analysis of the workload before and after you set this variable to AUTOMATIC to validate the performance improvement.

For more information about other usages of DB2_RESOURCE_POLICY other memory-related DB2 registry variables, see Chapter 2, “AIX configuration”, in *Best Practices for DB2 on AIX 6.1 for POWER Systems*, SG24-7821.

9.2.2 Page sizes

DB2 objects, such as tables and indexes, are stored in pages on disk and in memory. DB2 supports 4 KB, 8 KB, 16 KB, and 32 KB page sizes.

DB2 buffer pools, which are memory regions that are used to store pages from disk, use memory that is allocated from the OS. Depending on the OS, various page sizes can be used to improve the performance of the virtual memory manager (VMM). The default page size on AIX is 4 KB, and on Linux it is 64 KB, but other page sizes are available.

To achieve increased performance on Power Systems, DB2 10.1 by default uses 64 KB (“medium pages”) on AIX.

Using large pages

For some workloads, particularly ones that require intensive memory access, there are performance benefits for using large pages. (A “large page” is 16 MB on Power Systems.) However, certain drawbacks must be considered. When large pages support is enabled through DB2, all the memory that is set for large pages is pinned. It is possible to allocate too much memory with large pages and not enough for 4 KB and 64 KB pages, which can result in heavy paging activities. Furthermore, enabling large pages prevents the STMM from automatically tuning overall database memory consumption. Consider using the **DB2_LARGE_PAGE_MEM** variable only for defined workloads that have a relatively static database memory requirement.

To enable large page support on AIX operating systems, complete the following steps:¹

1. Configure AIX server for large pages support by running **vmo**:

```
vmo -r -o lpgg_size=<LargePageSize> -o lpgg_regions=<LargePages>
```

<LargePageSize> is the size in bytes of the hardware-supported large pages, and <LargePages> specifies the number of large pages to reserve.

2. Run **bosboot** to pick up the changes (made by running **vmo**) for the next system start.
3. After restart, run **vmo** to enable memory pinning:

```
vmo -o v_pinshm=1
```

¹ *Enabling large page support (AIX)* (for DB2 Version 10.1 for Linux, UNIX, and Windows), found at: <http://www-01.ibm.com/support/knowledgecenter/api/redirect/db2luw/v10r1/index.jsp?topic=%2Fcom.ibm.db2.luw.admin.dboobj.doc%2Fdoc%2Ft0010405.html>

To enable large page support on Linux operating systems, complete the following steps:

Note: Do not be confused by the Linux OS terminology of *huge pages*.

1. Configure Linux server for large page support by running the following command:

```
echo "vm.nr_hugepages=<LargePages>" >> /etc/sysctl.conf
```

2. Restart the server.

In both cases, after the server is configured for large pages and restarted, use the following steps to enable large page support in DB2:

1. Set the DB2_LARGE_PAGE_MEM registry variable by running **db2set**:

```
db2set DB2_LARGE_PAGE_MEM=DB
```

2. Start the DB2 database manager by running **db2start**:

```
db2start
```

9.2.3 Decimal arithmetic

DB2 for AIX uses the hardware DFP unit in POWER6, POWER7, and POWER8 processors in its implementation of decimal-encoded formats and arithmetic. One example of a data type that uses this hardware support is the DECFLOAT data type that is introduced in DB2 9.5. This decimal-floating point data type supports business applications that require exact decimal values, with a precision of 16 or 34 digits. When the DECFLOAT data type is used for a DB2 database that is on a POWER6, POWER7, or POWER8 processor, the native hardware support for decimal arithmetic is used. In comparison to other platforms, where such business operations can be achieved only through software emulation, applications that run on POWER6, POWER7, or POWER8 processors can use the hardware support to gain performance improvements.

DECFLOAT: The Data Type of the Future describes this topic in more detail. This paper is found at the following website:

<http://www.ibm.com/developerworks/data/library/techarticle/dm-0801chainani/>

9.2.4 Using simultaneous multithreading priorities for internal lock implementation

DB2 uses SMT and hardware priorities in its internal lock implementation. Internal locks are short duration locks that are required to ensure consistency of various values in highly concurrent applications such as DB2. In certain cases, it is beneficial to prioritize different DB2 agent threads to maximize system resource utilization.

For more information about this topic, see 9.8, “Related publications” on page 202.

9.2.5 Single Instruction Multiple Data

DB2 10.5 with BLU Acceleration uses SIMD processing to speed up analytic query processing. On POWER processors, this is called the VSX engine. The VSX engine has been part of the POWER architecture for a time, was improved in the POWER7 processor, and has received further improvements in the POWER8 processor.

SIMD instructions are low-level CPU instructions that enable you to perform the same operation on multiple data points at the same time.

DB2 10.5 with BLU Acceleration auto-detects whether it is running on an SIMD-enabled CPU, and automatically uses SIMD to effectively multiply the power of the CPU. In particular, BLU Acceleration can use a single SIMD instruction to get results from multiple data elements.

Figure 9-1 is an example of a scan operation that involves a predicate evaluation.

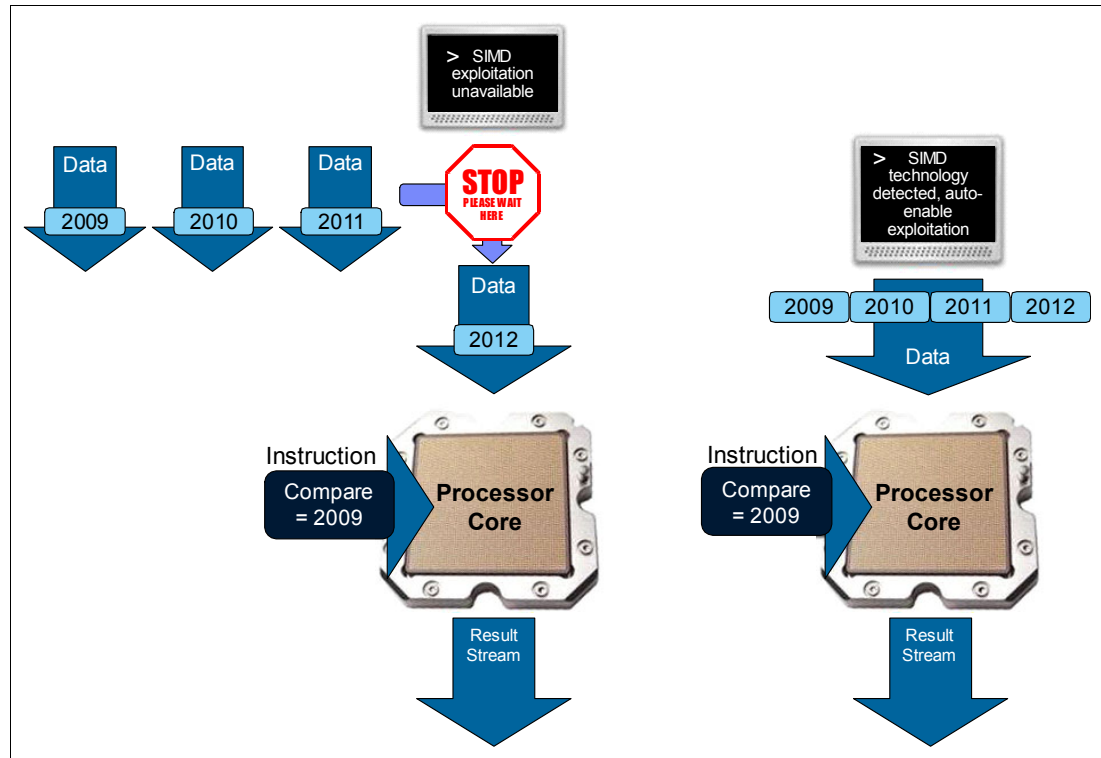


Figure 9-1 Compare predicate evaluation with and without SIMD on POWER using DB2 10.5 with BLU Acceleration

The left side of Figure 9-1 illustrates a typical operation, namely, that each data element (or column value) is evaluated, one after another. The right side of the figure shows how BLU Acceleration processes four columns at a time by using SIMD. Think of it as CPU power multiplied by four. Although Figure 9-1 shows a predicate evaluation, BLU Acceleration can also take advantage of SIMD processing for join operations, arithmetic, and more.

9.3 Capitalizing on the compilers and optimization tools for POWER

DB2 is built by using the latest IBM XL C/C++ compiler found at the start of the development cycle. For DB2 10.1 for AIX, IBM uses the Version 11 compiler, and for DB2 10.5 for AIX, IBM uses the Version 12 compiler. The latest XLC compiler technology was used for DB2 10.5 on Linux on Power Little Endian. IBM uses various compiler optimization flags along with optimization techniques based on the common three steps of software profiling:

- ▶ Application preparation/instrumentation
- ▶ Application profiling
- ▶ Application optimization

9.3.1 Whole-program analysis and profile-based optimizations

On the AIX platform, whole-program analysis (IPA) and profile-directed feedback (PDF) compiler options are used to optimize DB2 by using a set of customer representative workloads. This technique produces a highly optimized DB2 executable file that is targeted at the best usage of the Power Architecture.

For more information, see “Whole-program analysis” and “Optimization that is based on Profile Directed Feedback” on page 145.

9.3.2 IBM Feedback Directed Program Restructuring

In addition to IPA and PDF optimizations that use IBM XL C/C++ compiler, a post-link optimization step provides further performance improvement on the AIX platform. The particular tool that is used is Feedback Directed Program Restructuring (FDPR). Similar to IPA and PDF, a set of DB2 customer representative workloads is employed in this step, and IBM FDPR-Pro profiles and ultimately creates an optimized version of the DB2 product. For more information, see 7.4, “IBM Feedback Directed Program Restructuring” on page 160.

For more information about this topic, see 9.8, “Related publications” on page 202.

9.4 Capitalizing on POWER virtualization

DB2 supports and fully draws upon the virtualization technologies that are provided by the Power Architecture. These technologies include PowerVM for AIX and Linux, PowerKVM for Linux, and System Workload Partitioning (WPAR) for AIX. Many of the DB2 performance preferred practices for a non-virtualized environment also extend to a virtualized environment.

Furthermore, DB2 offers IBM SubCapacity Licensing, which enables customers to consolidate more effectively their infrastructure and reduce their overall total cost of ownership (TCO). DB2 also provides a flexible software licensing model that supports advanced virtualization capabilities, such as shared processor pools, Micro-Partitioning, virtual machines, and dynamic reallocation of resources. To support this type of licensing model, a tool is provided that allows customers to track and manage their own software license usage.

9.4.1 DB2 virtualization

DB2 is engineered to take advantage of the many benefits of virtualization on Power Systems and therefore allows various types of workload to be deployed in a virtualized environment. One key DB2 feature that enables workloads to run efficiently in virtualized environments is the STMM. STMM automatically adjusts the values of several memory configuration parameters in DB2. When enabled, it dynamically evaluates and redistributes available memory resources among the buffer pools, lock memory, package cache, and sort memory to maximize performance. The changes are applied dynamically and can simplify the task of manual configuration of memory parameters. This feature is useful in a virtualized environment because STMM can respond to dynamic changes in partition memory allocation.

By default, most DB2 parameters are set to automatic to enable STMM. As a preferred practice, leave the **instance_memory** parameter and other memory parameters as automatic, especially when you are running in a virtualized environment because DB2 is designed to allow STMM to look for available memory in the system when **instance_memory** is set to automatic.

DB2 also supports the PowerVM Live Partition Mobility (LPM) feature when virtual I/O is configured. LPM allows an active database to be moved from a system with limited memory to one with more memory without disrupting the operating system or applications. When coupling dynamic LPAR (DLPAR) with STMM, the newly migrated database can automatically adjust to the additional memory resource for better performance.

DB2 Virtualization, SG24-7805 describes in considerable detail the concept of DB2 virtualization, in addition to setup, configuration, and management of DB2 on IBM Power Systems with PowerVM technology. That book follows many of the preferred practices for Power Systems virtualization and has a list of preferred practices for DB2 on PowerVM.

9.4.2 DB2 in an AIX workload partition

DB2 supports product installation on system WPARs. DB2 can be installed either within a local file system on a system WPAR or in a global environment under either the `/usr` or `/opt` directory with each instance created on the local WPARs. In both cases, each DB2 instance is only visible and managed by the system WPAR in which it is created. If DB2 is installed in a global environment, different instances on different WPARs share the globally installed DB2 copy to improve i-cache efficiency and memory usage. WPAR mobility is also supported where a DB2 instance that is running on a system WPAR can migrate to a remote WPAR on a different physical machine.

There are certain restrictions and considerations to keep in mind when you install DB2 in a global environment:

- ▶ Certain DB2 installation features cannot be installed on a system WPAR. These features are IBM Tivoli® System Automation for Multiplatforms and IBM Data Studio Administration Console.
- ▶ When you uninstall a DB2 copy in a global environment, all associated instances must be dropped or updated to another DB2 copy and its corresponding system WPARs must be active.
- ▶ When you apply fix packs to a DB2 copy in a global environment, all associated instances must be stopped and its corresponding system WPARs must be active.

For information about installing a DB2 copy on a WPAR, see Chapter 8, “Workload Partitioning”, in *Best Practices for DB2 on AIX 6.1 for POWER Systems*, SG24-7821.

For more information about this topic, see 9.8, “Related publications” on page 202.

9.5 Capitalizing on the AIX system libraries

This section describes methods for capitalizing on the AIX system libraries.

9.5.1 Using the `thread_post_many` API

DB2 uses `thread_wait` and `thread_post_many` to improve the efficiency of DB2 threads running on multi-processor Power Systems. DB2 takes advantage of the `thread_post_many` function. The availability of such an API on AIX directly impacts the efficiency of DB2 processing, as it allows for waking many EDUs with a single function call, which in other operating systems requires many individual function calls (typically as many as the number of EDUs being woken up).

9.5.2 File systems

DB2 uses most of the advanced features within the AIX file systems. These features include Direct I/O (DIO), Concurrent I/O (CIO), Asynchronous I/O, and I/O Completion Ports (IOCP).

Non-buffered I/O

By default, DB2 uses CIO or DIO for newly created table space containers because non-buffered I/O provides more efficient underlying storage access over buffered I/O on most workloads, with most of the benefit realized by bypassing the file system cache. Non-buffered I/O is configured through the `NO FILE SYSTEM CACHING` clause of the table space definition. To maximize the benefits of non-buffered I/O, a correct buffer pool size is essential. This size can be achieved by using STMM to tune the buffer pool sizes. (The default buffer pool is always tuned by STMM, but user-created buffer pools must specify the **automatic** keyword for the size to allow STMM to tune them.) When STMM is enabled, it automatically adjusts the buffer pool size for optimal performance.

For file systems that support CIO, such as AIX JFS2, DB2 automatically uses this I/O method because of its performance benefits over DIO.

The DB2 log file by default uses DIO, which brings similar performance benefits as avoiding file system cache for table spaces.

Asynchronous I/O

In general, DB2 users cannot explicitly choose synchronous or asynchronous I/O. However, to improve the overall response time of the database system, minimizing synchronous I/O is preferred and can be achieved through correct database tuning. Consider the following items:

- ▶ Synchronous read I/O can occur when a DB2 agent needs a page that is not in the buffer pool to process an SQL statement. In addition, a synchronous write I/O can occur if no clean pages are available in the buffer pool to make room to bring another page from disk into that buffer pool. This situation can be minimized by having sufficiently large buffer pools or setting the buffer pool size to automatic to allow STMM to find its optimal size, in addition to tuning the page cleaning (by using the `chnpggs_thresh` database parameter).
- ▶ Not all pages read into buffer pools are done synchronously. Depending on the SQL statement, DB2 can prefetch pages of data into buffer pools through asynchronous I/O. When prefetching is enabled, two parallel activities occur during query processing: data processing and data page I/O. The latter is done through the I/O servers that wait for prefetch requests from the former. These prefetch requests contain a description of the I/O that must satisfy the query. The number of I/O servers for a database is specified through the `num_ioservers` configuration parameter. By default, this parameter is automatically tuned during database start.

For more information about how to monitor and tune AIO for DB2, see *Best Practices for DB2 on AIX 6.1 for POWER Systems*, SG24-7821.

I/O Completion Port

Configure the AIX I/O Completion Port for performance purposes, even though it is not mandatory, as part of the DB2 10 installation process. For more information, see *Configuring IOCP (AIX)*, found at:

<http://www-01.ibm.com/support/knowledgecenter/api/redirect/db2luw/v10r1/index.jsp?topic=/com.ibm.db2.luw.admin.perf.doc/doc/t0054518.html>

After IOCP is configured on AIX, then DB2, by default, capitalize on this feature for all asynchronous I/O requests. With IOCP configured, AIO server processes from the AIX operating system manage the I/O requests by processing many requests in the most optimal way for the system.

For more information about this topic, see 9.8, “Related publications” on page 202.

9.6 Capitalizing on performance tools

Correct performance tools are crucial for maximizing DB2 performance. Zoning in on potential performance bottlenecks is impossible without a strong performance tool set, such as the ones on Power Systems.

9.6.1 High-level investigation

During the general analysis of any performance investigation, the identification of the system resource bottlenecks is key to determining the root cause of the bottleneck. System resource bottlenecks can be classified into several categories, such as CPU bound, IO bound, network bound, or excessive idling, all of which can be identified with AIX system commands.

9.6.2 Low-level investigation

Various system level tools are essential in drilling down to find a potential root cause for the type of the bottlenecks that are listed in 9.6, “Capitalizing on performance tools” on page 201. Profiling tools are especially invaluable for identifying CPU-bound issues and are available on AIX and Linux on Power platforms.

AIX tprof

tprof is a powerful profiling tool on the AIX platform that does program counter-sampling in clock interrupts. It can work on any binary without recompilation and is a great tool for codepath analysis.

For instructions about using the **tprof** command, go to the following website:

<http://www-01.ibm.com/support/knowledgecenter/api/redirect/aix/v7r1/topic/com.ibm.aix.prftools/doc/prftools/tprofcommand.htm>

AIX tprof microprofiling

Beyond the high-level **tprof** profiling, DB2 also uses the microprofiling option of **tprof** during development. Microprofiling allows DB2 to perform instruction-level profiling to apportion the total CPU time to individual source program lines.

Linux OProfile and perf

OProfile and **perf** are system profiling tools, similar to **tprof**, which are popular on the Linux platform. **OProfile** and **perf** use hardware counters to provide functional-level profiling in both the kernel and user space. Similar to **tprof**, these tools are useful during DB2 development for codepath analysis.

For more information about this topic, see 9.8, “Related publications” on page 202.

9.7 Conclusion

DB2 is positioned to capitalize on many POWER processor features to maximize the return on investment (ROI) of the full IBM stack. During the entire DB2 development cycle, there is a targeted effort to take advantage of POWER processor features and ensure that the highest level of optimization is employed on this platform. With every new POWER processor generation, DB2 ensures that the key features are supported and brought into play at the POWER processor launch by working on such features well in advance of general availability. This type of targeted effort ensures that DB2 is at the forefront of optimization for POWER processor applications.

9.8 Related publications

The publications that are listed in this section are considered suitable for a more detailed description of the topics that are covered in this chapter:

- ▶ *Best Practices for DB2 on AIX 6.1 for Power Systems*, SG24-7821
- ▶ *Best practices for DB2 for Linux, UNIX, and Windows*, found at:
<http://www.ibm.com/developerworks/data/bestpractices/db2luw/>
- ▶ DB2 documentation about its many variations is found at:
http://www-01.ibm.com/support/knowledgecenter/SSEPGG_10.5.0/com.ibm.db2.luw.admin.regvars.doc/doc/r0005665.html?cp=SSEPGG_10.5.0%2F2-4-5-4-7
- ▶ *DB2 database products in a workload partition (AIX)*, found at:
<http://www-01.ibm.com/support/knowledgecenter/api/redirect/db2luw/v10r1/index.jsp?topic=/com.ibm.db2.luw.qb.server.doc/doc/c0053344.html>
- ▶ DB2 performance registry variables, including **DB2_LOGGER_NON_BUFFERED_IO** and **DB2_USE_IOCP**, are described in *Performance variables*, found at:
<http://www-01.ibm.com/support/knowledgecenter/api/redirect/db2luw/v10r1/index.jsp?topic=/com.ibm.db2.luw.admin.regvars.doc/doc/r0005665.html>
- ▶ *DB2 Version 10.1 for Linux, UNIX, and Windows, Performance variables* describes DB2 performance registry variables, including **DB2_RESOURCE_POLICY** and **DB2_LARGE_PAGE_MEM**:
<http://www-01.ibm.com/support/knowledgecenter/api/redirect/db2luw/v10r1/index.jsp?topic=/com.ibm.db2.luw.admin.regvars.doc/doc/r0005665.html>
- ▶ *DB2 Virtualization*, SG24-7805
- ▶ *DECFLOAT: The data type of the future*, found at:
<http://www.ibm.com/developerworks/data/library/techarticle/dm-0801chainani/>
- ▶ *DECFLOAT scalar function* (for DB2 10.1 for Linux, UNIX, and Windows), found at:
<http://www-01.ibm.com/support/knowledgecenter/api/redirect/db2luw/v10r1/index.jsp?topic=/com.ibm.db2.luw.sql.ref.doc/doc/r0050508.html>
- ▶ *Feedback Directed Program Restructuring (FDPR)*, found at:
<https://www.research.ibm.com/haifa/projects/systems/cot/fdpr/>
- ▶ *FDPR-Pro - Usage: Feedback Directed Program Restructuring*, found at:
http://www.research.ibm.com/haifa/projects/systems/cot/fdpr/papers/fdpr_pro_usage_cs.pdf

- ▶ IBM DB2 10.1 IBM Knowledge Center, found at:
<http://www-01.ibm.com/support/knowledgecenter/api/redirect/db2luw/v10r1/index.jsp?topic=/com.ibm.db2.luw.welcome.doc/doc/welcome.html>
- ▶ *Smashing performance with OProfile*, found at:
<http://www.ibm.com/developerworks/library/l-oprof/>
- ▶ *tprof Command*, found at:
<http://www-01.ibm.com/support/knowledgecenter/api/redirect/aix/v7r1/index.jsp?topic=/com.ibm.aix.cmds/doc/aixcmds5/tprof.htm>



IBM WebSphere Application Server

This chapter describes the optimization and tuning of the POWER8 processor-based server running WebSphere Application Server. The topic 10.1, “IBM WebSphere” on page 206 is the highlight of this chapter:

10.1 IBM WebSphere

This chapter is intended to provide you with performance and functional considerations for running WebSphere Application Server middleware on Power Systems. It primarily describes POWER7 and POWER8 processor-based systems. Even though WebSphere Application Server is designed to run on many operating systems and platforms, some specific capabilities of Power Systems are used by WebSphere Application Server as a part of platform optimization efforts.

The intent of this chapter is to explain WebSphere Application Server installation, deployment, and migration topics when WebSphere Application Server is running on Power Systems. This chapter also describes preferred practices for performance when you run enterprise Java applications on Power Systems. This chapter also highlights some of the known WebSphere Application Server topics and solutions for Power Systems.

10.1.1 Installation

As there are multiple versions of WebSphere Application Server, there are also multiple versions of AIX that are supported by POWER7 and POWER8 processor-based systems. Table 10-1 shows some of the installation considerations. Use the most currently available code, including the latest installation binary files. For the most current AIX installation and configuration details, see 4.4.1, “AIX preferred practices that are applicable to all Power Systems generations” on page 105.

Important: If running on a POWER7 processor-based system, use the versions of WebSphere Application Server that run in *POWER7 mode with performance enhancements*. Similarly, for POWER8 processor-based systems, use the versions of WebSphere Application Server that run in *POWER8 mode with performance enhancements*.

Table 10-1 Installation considerations

Document	Associated website	Information provided
IBM WebSphere Application Server support on POWER7 hardware	http://www.ibm.com/support/docview.wss?uid=swg21422150	Various fix pack levels and 64-bit considerations for running in POWER7 mode

10.1.2 Deployment

When you start the WebSphere Application Server, there is an option to bind the Java processors to specific CPU processor cores to circumvent the operating system scheduler to send the work to available processors in the pool. In certain cases, using RSETs and binding the JVM to stay within core/socket boundaries improves the performance. Table 10-2 on page 207 lists some of the deployment considerations.

Table 10-2 Deployment considerations

Consideration	Associated website	Information provided
<i>Workload partitioning (WPAR) in AIX V6.1</i>	http://www.ibm.com/developerworks/aix/library/au-wpar6laix/	Determining when it is useful to move from LPAR deployment to WPAR deployment
<i>Troubleshooting and performance analysis of different applications in versioned WPARs</i>	http://www.ibm.com/developerworks/aix/library/au-wpars/	The benefits of moving from old hardware to the new POWER7 hardware in the form of versioned WPARs

Processor affinity benefits for WebSphere applications

When an application that is running on top of WebSphere Application Server is deployed on a large LPAR, it might not use all the cores in that LPAR, resulting in less than optimum application performance. If this situation occurs, performance improvements to these applications can be obtained by binding the application server to certain cores. This task can be accomplished by creating the resource sets and attaching them to the application server that is running **excerset**. For an example of using the **taskset** and **numactl** commands in a Linux environment, see “Partition sizes and affinity” on page 16.

10.1.3 Performance

When you run WebSphere Application Server on POWER7 and POWER8 processor-based systems, end-to-end performance depends on many subsystems. This includes the network, memory, disk, and CPU subsystems of POWER7 and POWER8 processor-based systems; a crucial consideration is Java configuration and tuning. Topology also plays a major role in the performance of the enterprise application that is being deployed. The architecture of the application must be considered when you determine the best deployment topology. Table 10-3 includes links to preferred practices documents, which target each of these major areas.

Table 10-3 Performance considerations

Document	Associated website	Information provided
<i>Java Performance on POWER7 - Best practice</i>	http://www.ibm.com/common/ssi/cgi-bin/ssialias?infotype=SA&subtype=WH&htmlfid=POW03066USEN	This white paper highlights key preferred practices for all Java applications that are running on Power Systems and simultaneous multithreading (SMT) considerations when you are migrating from POWER5 or POWER6 processor-based systems to a POWER7 processor-based system.
<i>Optimizing AIX 7 network performance: Part 1, Network overview - Monitoring the hardware</i>	http://www.ibm.com/developerworks/aix/library/au-aix7networkoptimize1/index.html	This three-part white paper reviews AIX V7.1 networking and includes suggestions for achieving the best network performance.

<i>Optimizing AIX V7 memory performance: Part 1, Memory overview and tuning memory parameters</i>	http://www.ibm.com/developerworks/aix/library/au-aix7memoryoptimize1/index.html	Memory optimization is essential for running WebSphere Application Server faster on a POWER7 processor-based systems.
<i>Optimizing AIX V7 performance: Part 2, Monitoring logical volumes and analyzing the results</i>	http://www.ibm.com/developerworks/aix/library/au-aix7optimize2/index.html	Optimizing the disk and troubleshooting the I/O bottlenecks is crucial for I/O-intensive applications.

WebSphere channel framework degradation on POWER7 processor-based systems

Certain applications that run on WebSphere Application Server on POWER7 processor-based systems can experience performance degradation because of asynchronous I/O (AIO). AIO can be disabled to improve the performance of these applications. For instructions about how to accomplish this task, see *Disabling AIO (Asynchronous Input/Output) native transport in WebSphere Application Server*, found at:

<http://www.ibm.com/support/docview.wss?uid=swg21366862>

Scalability challenges when moving from a POWER5 or POWER6 processor-based system to a POWER7 or POWER8 processor-based system

By default, the POWER7 processor runs in SMT4 mode, the POWER8 processor for AIX runs in SMT4 mode, and the POWER8 processor for Linux runs in SMT8 mode. As such, there are either four or eight hardware threads (logical CPUs) per core that provide tremendous concurrency for applications. If the enterprise applications are migrated to POWER7 or POWER8 processor-based systems from an earlier version of POWER hardware (POWER5 or POWER6 processor-based systems), you might experience scalability issues because the default SMT mode on POWER8 is SMT8, and on the POWER7 processor is SMT4, but on POWER5 and POWER6 processor-based systems, the default is SMT and SMT2 mode, respectively. As some of these applications might not be designed for the massive parallelism of POWER7 or POWER8 processors, performance and scalability can be improved by using smaller partitions or processor binding. Processor binding is described in “Processor affinity benefits for WebSphere applications” on page 207.

Memory affinity benefits for WebSphere applications

In addition to the processor affinity that is described in “Processor affinity benefits for WebSphere applications” on page 207, applications can benefit from avoiding remote memory accesses by setting the environment variable **MEMORY_AFFINITY** to MCM (AIX only; this does not apply to Linux). This variable allocates application private and shared memory from processor local memory.

These three tuning techniques (SMT scalability, CPU affinity, and memory affinity) can improve the performance of WebSphere Application Server on POWER7 and POWER8 processor-based systems. For an example of using the **taskset** and **numactl** commands in a Linux environment, see “Partition sizes and affinity” on page 16 and “Processor affinity benefits for WebSphere applications” on page 207.

For more information about these topics, see *Java Performance on POWER7 - Best practice*, found at the following website:

<http://www.ibm.com/common/ssi/cgi-bin/ssialias?infotype=SA&subtype=WH&htmlfid=POW03066USEN>

10.1.4 Performance analysis, problem determination, and diagnostic tests

Resources for addressing issues regarding performance analysis, problem determination, and diagnostic tests are listed in Table 10-4.

Table 10-4 Performance analysis and problem determination

Document	Associated website	Information provided
<i>Java Performance Advisor (JPA)</i>	https://www.ibm.com/developerworks/wikis/display/WikiPtype/Java+Performance+Advisor	The JPA tool provides suggestions for improving the performance of Java/WebSphere Application Server applications that are running on Power Systems.
<i>The performance detective: Where does it hurt?</i>	http://www.ibm.com/developerworks/aix/library/au-performancedetective/index.html	Describes how to isolate performance problems.
<i>MustGather: Performance, hang, or high CPU issues with WebSphere Application Server on AIX</i>	http://www.ibm.com/support/docview.wss?uid=swg21052641	MustGather assists with collecting the data that is necessary to diagnose and resolve issues with hanging or CPU usage issues.

For more information about addressing performance analysis, see:

- ▶ 8.2, “32-bit versus 64-bit Java” on page 174
- ▶ 8.3, “Memory and page size considerations” on page 175
- ▶ 8.5, “Java garbage collection tuning” on page 183
- ▶ 8.6, “Application scaling” on page 186



A

Analyzing malloc usage under IBM AIX

This appendix describes the optimization and tuning of the memory usage of an application by using the AIX malloc subroutine. It covers the following topics:

- ▶ “Introduction” on page 212
- ▶ “How to collect malloc usage information” on page 212

Introduction

There is a simple methodology on AIX to collect useful information about how an application uses the C heap. That information can then be used to choose and tune the appropriate malloc settings. The type of information that typically must be collected is:

- ▶ The distribution of malloc allocation sizes that are used by an application, which shows whether AIX **MALLOCOPTIONS**, such as pool and buckets, are expected to perform well. This information can be used to fine-tune bucket sizes.
- ▶ The steady state size of the heap, which shows how to size the pool option.

Additional information about thread counts, malloc usage per thread, and so on, can be useful, but the information that is presented here presents a basic view.

This appendix does not apply to the watson2 allocator (see “Memory allocators” on page 95), which autonomically adjusts to the memory usage of an application and does not require specific tuning.

How to collect malloc usage information

To discover the distribution of allocation sizes, set the following environment variable:

```
export MALLOCOPTIONS=buckets,bucket_statistics:stdout
```

Run an application. When the application completes, a summary of the malloc activity is output. Example A-1 shows a sample output from a simple test program.

Example A-1 Output from a simple test program

```
=====
Malloc buckets statistical summary
=====
Configuration values:
  Number of buckets: 16
  Bucket sizing factor: 32
  Blocks per bucket: 1024
Allocation request totals:
  Buckets allocator: 118870654
  Default allocator: 343383
  Total for process: 119214037
Allocation requests by bucket
Bucket      Maximum      Number of
Number      Block Size   Allocations
-----
  0           32      104906782
  1           64       9658271
  2           96       1838903
  3          128        880723
  4          160        300990
  5          192        422310
  6          224        143923
  7          256        126939
  8          288        157459
  9          320         72162
 10          352         87108
```

11	384	56136
12	416	63137
13	448	66160
14	480	45571
15	512	44080

Allocation requests by heap

Heap Number	Buckets Allocator	Default Allocator
-----	-----	-----
0	118870654	343383

This environment variable causes the program to produce a histogram of allocation sizes when it terminates. The number of allocation requests that are satisfied by the default allocator indicates the fraction of requests that are too large for the buckets allocator (larger than 512 bytes, in this example). By modifying some of the malloc buckets configuration options, you can, for example, obtain more information about larger allocation sizes.

To discover the steady state size of the heap, set the following environment variable:

```
export MALLOCDDEBUG=log
```

Run an application to a steady state point, attach it by running **dbx**, and then run **malloc**. Example A-2 shows a sample output.

Example A-2 Sample output from the malloc subroutine

```
(dbx) malloc
The following options are enabled:
    Implementation Algorithm..... Default Allocator (Yorktown)
    Malloc Log
        Stack Depth..... 4
Statistical Report on the Malloc Subsystem:
    Heap 0
        heap lock held by..... pthread ID 0x20023358
        bytes acquired from sbrk()..... 5309664
        bytes in the freespace tree..... 334032
        bytes held by the user..... 4975632
        allocations currently active..... 76102
        allocations since process start.. 20999785
The Process Heap
    Initial process brk value..... 0x20013850
    current process brk value..... 0x214924c0
    sbrk()s called by malloc..... 78
```

The bytes held by the user value indicates how much heap space is allocated. By stopping multiple times when you run **dbx** and then running **malloc**, you can get a good estimate of the heap space that is needed by the application.

For more information, see *System memory allocation using the malloc subsystem*, found at:

http://www-01.ibm.com/support/knowledgecenter/api/redirect/aix/v7r1/topic/com.ibm.aix.genprogc/doc/genprogc/sys_mem_alloc.htm



Performance tools and empirical performance analysis

This appendix describes the optimization and tuning of the POWER8 processor-based system from the perspective of performance tools and empirical performance analysis. It covers the following topics:

- ▶ “Introduction” on page 216
- ▶ “Performance advisors” on page 216
- ▶ “IBM Power Virtualization Performance” on page 223
- ▶ “AIX” on page 223
- ▶ “Linux” on page 233
- ▶ “Java (either AIX or Linux)” on page 239

Introduction

This appendix includes a general description about performance advisors, and descriptions that are specific to the three performance advisors that are referenced in this book:

- ▶ AIX
- ▶ Linux
- ▶ Java (either AIX or Linux)

Performance advisors

IBM developed four new performance advisors that empower users to address their own performance issues to best use their Power Systems server. These performance advisors can be run by a broad class of users.

The first three of these advisors are tools that run and analyze the configuration of a system and the software that is running on it. They also provide advice about the performance implications of the current configuration and suggestions for improvement. These three advisors are documented in “Expert system advisors” on page 216.

The fourth advisor is part of the IBM Rational Developer for Power Systems Software. It is a component of an integrated development environment (IDE), which provides a set of features for performance tuning of C and C++ applications on AIX and Linux. That advisor is documented in “IBM Rational Performance Advisor” on page 221.

Expert system advisors

The expert system advisors are three new tools that are developed by IBM. What is unique about these applications is that they collect and interpret performance data. In one step, they collect performance metrics, analyze data, and provide a one-page visual report. This report summarizes the performance health of the environment, and includes instructions for alleviating detected problems. The performance advisors produce advice that is based on the expertise of IBM performance analysts, and IBM documented preferred practices. These expert systems focus on AIX Partition Virtualization, VIOS, and Java performance.

All of the advisors follow the same reporting format, which is a single page XML file you can use to assess quickly conditions by visually inspecting the report and looking at the descriptive icons, as shown in Figure B-1.






ICON	DEFINITION
	Informative: Context relevant data helpful in making adjustments.
	Optimal: Current condition likely to deliver best performance.
	Warning: Current condition deviates from best practices. Opportunity likely exists for better performance.
	Critical: Current condition likely causing negative impacts.
	Investigate: Further investigation or information required by user to determine if observation is impacting performance.

Figure B-1 Descriptive icons in expert system advisors (AIX Partition Virtualization, VIOS Advisor, and Java Performance Advisor)

The XML reports that are generated by all of the advisors are interactive. If a problem is detected, three pieces of information are shared with the user:

1. What is this?
This section explains why a particular topic was monitored, and provides a definition of the performance metric or setting.
2. Why is it important?
This report entry explains why the topic is relevant and how it impacts performance.
3. How do I modify it?
Instructions for addressing the problem are listed in this section.

VIOS Performance Advisor

The VIOS Performance Advisor provides guidance about various aspects of VIOS:

- ▶ CPU
- ▶ Shared processing pool
- ▶ Memory
- ▶ Fibre Channel performance
- ▶ Disk I/O subsystem
- ▶ Shared Ethernet adapter

The output is presented on a single page, and copies of the report can be saved, making it easy to document the settings and performance of VIOS over time. The goal of the advisor is for you to be able to self-assess the health of your VIOS and act to attain optimal performance.

Figure B-2 shows a window of the VIOS Performance Advisor, focusing on the FC adapter section of the report, which attempts to guide the user in determining whether any of the FC ports are being saturated, and, if so, to what extent. An investigate image was displayed next to the idle FC port to confirm that the idle adapter port is intentional and because of an administrative configuration design choice.

The *VIOS Advisor* can be found at the following website:

<https://www.ibm.com/developerworks/community/wikis/home?lang=en#!/wiki/Power%20Systems/page/VIOS%20Advisor>

Figure B-2 shows a window from the VIOS Advisor.










VIOS - DISK ADAPTERS							
	Name	Measured Value	Recommended Value	First Observed	Last Observed	Risk 1=lowest 5=highest	Impact 1=lowest 5=highest
	FC Adapter Count	2	-	06/12 20:32:25	-	n/a	n/a
	FC Avg IOps	avg: 7134 iops @ 4KB	 show/hide details	06/12 20:32:25	06/12 20:37:25	n/a	n/a
	FC Avg IOps(fcs2)	idle	-	06/12 20:32:25	06/12 20:37:25	n/a	n/a
	FC Avg IOps(fcs3)	avg: 7134 iops @ 4KB peak: 7251 iops @ 4KB	-	06/12 20:32:25	06/12 20:37:25	n/a	n/a
	FC Adapter Utilization		 show/hide details	-	-	n/a	n/a
	FC Adapter Utilization(fcs2)	idle	-	06/12 20:32:44	06/12 20:37:16	4	4
	FC Adapter Utilization(fcs3)	high:14.3% util. avg:14.1%	-	06/12 20:32:44	06/12 20:37:16	4	4

Figure B-2 The VIOS Advisor

Virtualization Performance Advisor

The Virtualization Performance Advisor provides guidance for various aspects of a logical partition (LPAR), both dedicated and shared:

- ▶ LPAR physical memory domain allocation
- ▶ Physical CPU entitlement and virtual CPU optimization
- ▶ SMT effectiveness
- ▶ Processor folding effectiveness
- ▶ Shared processing pool
- ▶ Memory optimization
- ▶ Physical Fibre Channel adapter optimization
- ▶ Virtual disk I/O optimization (virtual small computer system interface (vSCSI) and N_Port ID Virtualization (NPIV))

The output is presented in a single window, and copies of the report can be saved, making it easy for the user to document the settings and performance of their LPAR over time. The goal of the advisor is for the user to be able to self-assess the health of their LPAR and act to attain optimal performance.

Figure B-3 is a snapshot of the LPAR Virtualization Performance Advisor, focusing on the LPAR optimization section of the report, which applies virtualization preferred practice guidance to the LPAR configuration, resource usage of the LPAR, and shared processor pool, and determines whether the LPAR configuration is optimized. If the advisor finds that the LPAR configuration is not optimal for the workload, it guides the user in determining the best possible configuration. The *LPAR Performance Advisor* can be found at the following website:

https://www.ibm.com/developerworks/community/blogs/simplyaix/entry/lpar_performance_advisor?lang=en

LPAR PROCESSOR OPTIMIZATION										
	Name	Current Value				Recommended Value	First Observed	Last Observed	Risk 1=lowest 5=highest	Impact 1=lowest 5=highest
✖	Lpar Placement Optimization	Placement				Memory is allocated from multiple domains, containing memory in one or fewer domains will improve performance. However, changing this allocation requires assistance from IBM. Also, refer to P7 Virtualization Performance best practice document.	Tue Mar 6 20:14:28 2012	-	NA	NA
		Global Domain	Chip Domain	Memory	CPU					
		0	0	63228.560	0-11					
		0	6	60756.000						
		1	1	63246.000	12-15					
		1	2	63246.000	16-19					
		2	3	63223.000	20-23					
		2	4	63478.690	24-27					
		3	5	61503.000	28-31					
		3	7	52539.000						
		Only memory assigned from 0 Global Domain 6 Chip Domain Only memory assigned from 3 Global Domain 7 Chip Domain								
✖		Local Memory access - 0.00 and Remote memory access - 1210.84 Remote memory access is high. Local Memory access - 0.00 and Distant memory access - 9275.13 Distant memory access is high.				Memory is allocated from multiple domains, containing memory in one or fewer domains will improve performance. However, changing this allocation requires assistance from IBM. Also, refer to P7 Virtualization Performance best practice document.	Tue Mar 6 20:14:28 2012	-	NA	NA
✔	SMT Effectiveness	SMT-4 is set				-	Tue Mar 6 20:14:28 2012	-	NA	NA
✔	Virtual Processor Folding Optimization	Virtual Processor Folding Threshold - 49%				Rerun when lpar is busy	Tue Mar 6 20:14:33 2012	-	NA	NA

Figure B-3 LPAR Virtualization Performance Advisor

Java Performance Advisor

The Java Performance Advisor provides recommendations to improve the performance of a stand-alone Java or WebSphere Application Server application that is running on an AIX machine. The guidance that is provided is categorized into four groups:

- ▶ Hardware and LPAR-related parameters: Processor sharing, SMT levels, memory, and so on
- ▶ AIX specific tunables: Process RSET, TCP buffers, memory affinity, and so on

- ▶ JVM tunables: Heap sizing, garbage collection (GC) policy, page size, and so on
- ▶ WebSphere Application Server related settings for a WebSphere Application Server process

The guidance is based on Java tuning preferred practices. The criteria that are used to determine the guidance include the relative importance of the Java application, machine usage (test and production), and the user's expertise level.

Figure B-4 on page 221 is a snapshot of Java and WebSphere Application Server recommendations from a sample run, indicating the best JVM optimization and WebSphere Application Server settings for better results, per Java preferred practices. Details about the metrics can be obtained by expanding each of the metrics. The output of the run is a simple XML file that can be viewed by using the supplied XSL viewer and any browser. The *Java Performance Advisor (JPA)* can be found at the following website:

<https://www.ibm.com/developerworks/wikis/display/WikiPtype/Java+Performance+Advisor>













Java					
	Name	Current Value	Recommended Value	Risk 1=lowest 5=highest	Impact 1=lowest 5=highest
	JVM Version	1.6.0 SR2	More Details...	4	4
	JVM Type	64 bit	32 bit	4	4
	Initial Heap Size	100 MB	400 MB to 1.5625 GB	2	3
	Maximum Heap Size	1.5625 GB	1.5625 GB	5	5
	JVM Debug	Off	Off	1	5
	Verbose Class Loading	Off	Off	1	2
	Verbose Garbage Collection	Off	On	1	1
WebSphere					
	Name	Current Value	Recommended Value	Risk 1=lowest 5=highest	Impact 1=lowest 5=highest
	WebSphere Version	7.0.0.0	More Details...	3	4
	WebSphere PMI	On	Off	3	5
	Session Time Out	30 Minutes	5 Minutes to 30 Minutes	3	1
	Minimum Web Container Threads	50 Threads	10 Threads to 72 Threads	3	3
	Maximum Web Container Threads	50 Threads	24 Threads to 144 Threads	4	5

Figure B-4 Java Performance Advisor

IBM Rational Performance Advisor

IBM Rational Developer for AIX and Linux IDE V9.1.1.1 and later includes a component that is called Rational Performance Advisor, which provides a rich set of features for performance tuning C and C++ applications on IBM AIX and Linux on Power Systems, including support for POWER8 processor-based systems and Linux on Power Little Endian. Although not directly related to the tools that are described in “Expert system advisors” on page 216, Rational Performance Advisor has the same goal of helping users to best use Power Systems hardware with tools that offer simple collection, management, and analysis of performance data.

Rational Performance Advisor gathers data from several sources. The raw application performance data comes from the same expert-level **tprof** and **OProfile** CPU profilers that are described in “AIX” on page 223 and “Linux” on page 233, and other low-level operating system tools. The debug information that is generated by the compiler allows this data to be matched back to the original source code. XLC compilers can generate XML report files that provide information about optimizations that were performed during compilation. Finally, the application build and runtime systems are analyzed to determine whether there are any potential environmental problems.

All of this data is automatically gathered, correlated, analyzed, and presented in a way that is quick to access and easy to understand (Figure B-5).

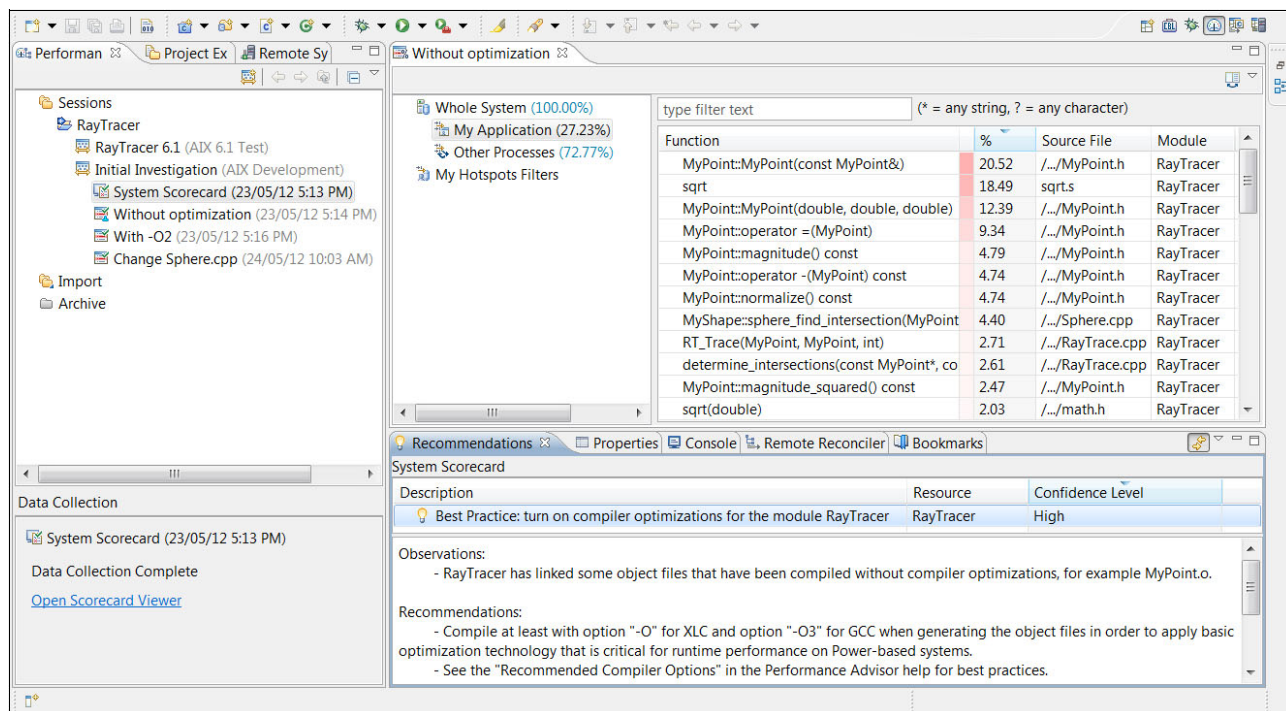


Figure B-5 Rational Performance Advisor

Key features include:

- ▶ Performance Explorer organizes your performance tuning sessions and data.
- ▶ System Scorecard reports on your Power Systems build and runtime environments.
- ▶ Hotspots Browser shows CPU profiling results for your application and its functions.
- ▶ Hotspots Comparison Browser compares runs for regression analysis or fix verification.
- ▶ The Performance Source Viewer and Outline view gives precise line-level profiling results.
- ▶ Invocations Browser displays dynamic call information from your application
- ▶ The Recommendations view offers expert-system guidance.

For more information about Rational Performance Advisor, including a trial download, see Rational Developer for AIX and Linux C/C++ Edition, found at:

<http://www.ibm.com/software/products/en/dev-c-cpp>

IBM Power Virtualization Performance

IBM Power Virtualization Performance (IBM PowerVP™) is a performance monitoring solution that provides detailed and real-time information about virtualized workloads that are running on Power Systems. PowerVP is a licensed program that is offered as part of PowerVM Enterprise Edition, but is also available separately for clients without PowerVM Enterprise Edition. You can use PowerVP to understand how virtual workloads use resources, to analyze performance bottlenecks, and to make informed choices about resource allocation and virtualized machine placement. PowerVP V1.1.2 supports the POWER8 hardware.

The PowerVP tool has the following features:

- ▶ Monitors the performance of an entire system (or frame).
- ▶ Is supported on AIX, IBM i, Linux, and VIOS operating systems.
- ▶ Provides a GUI for monitoring virtualized workloads.
- ▶ Includes a system-level monitoring agent that collects data from the PowerVM hypervisor, which provides a complete view of virtualized machines that are running on the server.
- ▶ Displays the data that is collected at the system level, at the hardware node level, and at the partition level. You can optimize performance by using the PowerVP performance metrics, which provide information about balancing and improving affinity and application efficiency.
- ▶ Provides an illustration of the Power Systems hardware topology along with resource usage metrics.
- ▶ Provides a mapping between real and virtual processor resources.
- ▶ Provides a recording feature for storing performance information with digital video recorder- (DVR-)like functions, such as play, fast forward, rewind, jump, pause, and stop. You can find performance bottlenecks by playing back the recorded data at any point in time.

For more information about PowerVP, go to the following website:

http://www-01.ibm.com/support/knowledgecenter/api/redirect/powersys/v3r1m5/index.jsp?topic=%2Fp7ecu1%2Fp7ecu_intro_powervp.htm

AIX

The section introduces tools and techniques that are used for optimizing software for a combination of Power Systems and AIX. The intended audience for this section is software development teams. As such, this section does not address performance topics that are related to capacity planning, and system-level performance monitoring and tuning.

To download Java for AIX, go to the following website:

<http://www.ibm.com/developerworks/java/jdk/aix/>

For capacity planning, see the IBM Systems Workload Estimator, found at the following website:

<http://www-912.ibm.com/estimator>

For system-level performance monitoring and tuning information for AIX, see *Performance management*, found at:

http://www-01.ibm.com/support/knowledgecenter/api/redirect/aix/v7r1/topic/com.ibm.aix.prftungd/doc/prftungd/performance_management-kickoff.htm

The bedrock of any empirically based software optimization effort is a suite of repeatable benchmark tests. To be useful, such tests must be representative of the manner in which users interact with the software. For many commercial applications, a benchmark test simulates the actions of multiple users that drive a prescribed mix of application transactions. Here, the fundamental measure of performance is throughput (the number of transactions that are run over a period) with an acceptable response time. Other applications are more *batch-oriented*, where few jobs are started and the time that is taken to completion is measured. Whichever benchmark style is used, it must be repeatable. Within some small tolerance (typically a few percent), running the benchmark several times on the same setup yields the same result.

Tools and techniques that are employed in software performance analysis focus on pinpointing aspects of the software that inhibit performance. At a high level, here are the two most common inhibitors to application performance:

- ▶ Areas of code that consume large amounts of CPU resources. This code is caused by using inefficient algorithms, poor coding practices, or inadequate compiler optimization
- ▶ Waiting for locks or external events. Locks are used to serialize execution through critical sections, that is, sections of code where the need for data consistency requires that only one software thread run at a time. An example of an external event is the system that is waiting for a disk I/O to complete. Although the amount of time that an application must wait for external events might be outside of the control of the application (for example, the time that is required for a disk I/O depends on the type of storage employed), simply being aware that the application is having to wait for such an event can open the door to potential optimizations.

CPU profiling

A CPU profiler is a performance tool that shows in which code CPU resources are being consumed. **tprof** is a powerful CPU profiler that encompasses a broad spectrum of profiling functions:

- ▶ It can profile any program, library, or kernel extension that is compiled with C, C++, Fortran, or Java compilers. It can profile machine code that is created in real time by the JIT compiler.
- ▶ It can attribute time to processes, threads, subroutines (user mode, kernel mode, shared library, and Java methods), source statements, and even individual machine instructions.
- ▶ In most cases, no recompilation of object files is required.

Usage of **tprof** typically focuses on generating subroutine-level profiles to pinpoint code hotspots, and to examine the impact of an attempted code optimization. A common way to run **tprof** is as follows:

```
$ tprof -E -skeuz -x sleep 10
```

The **-E** flag instructs **tprof** to employ the performance monitoring unit (PMU) as the sampling mechanism to generate the profile. Using the PMU as the sampling mechanism provides a more accurate profile than the default time-based sampling mechanism, as the PMU sampling mechanism can accurately sample regions of kernel code where interrupts are disabled. The **s**, **k**, **e**, and **u** flags instruct **tprof** to generate subroutine-level profiles for shared library, kernel, kernel extension, and user-level activity. The **z** flag instructs **tprof** to report CPU time in the number of *ticks* (that is, samples) instead of percentages. The **-x sleep 10** argument instructs **tprof** to collect profiling data during the running of the **sleep 10** command. This command collects profile data over the entire system (including all running processes) over a period of 10 seconds.

Excerpts from a **tprof** report are shown in Example B-1, Example B-2, and Example B-3 on page 226.

Example B-1 is a breakdown of samples of the processes that are running on the system. When multiple processes have the same name, they have only one line in this report: the number of processes with that name is in the “Freq” column. “Total” is the total number of samples that are accumulated by the process, and “Kernel”, “User”, and “Shared” are the number of samples that are accumulated by the processes in kernel (including kernel extensions), user space, and shared libraries. “Other” is a catchall for samples that do not fall in the other categories. The most common scenario where samples wind up in “Other” is because of CPU resources that are being consumed by machine code that is generated in real time by the JIT compiler. The **-j** flag of **tprof** can be used to attribute these samples to Java methods.

Example: B-1 Excerpt from a tprof report - breakdown of samples of processes running on the system

Process	Freq	Total	Kernel	User	Shared	Other	
=====		=====	=====	=====	=====	=====	=====
wait		4	5810	5810	0	0	0
./version1	1	1672	35	1637	0	0	0
/usr/bin/tprof	2	15	13	0	2	0	0
/etc/syncd	1	2	2	0	0	0	0
/usr/bin/sh	2	2	2	0	0	0	0
swapper	1	1	1	0	0	0	0
/usr/bin/trcstop	1	1	1	0	0	0	0
rmcd	1	1	1	0	0	0	0
=====		=====	=====	=====	=====	=====	=====
Total		13	7504	5865	1637	2	0

Example B-2 is a breakdown of samples of the threads that are running on the system. In addition to the columns that are described in Example B-1, this report has *PID* and *TID* columns that detail the process IDs and thread IDs.

Example: B-2 Excerpt from a tprof report - breakdown of threads that are running on the system

Process	PID	TID	Total	Kernel	User	Shared	Other	
=====		=====	=====	=====	=====	=====	=====	=====
wait	16392	16393	1874	1874	0	0	0	0
wait	12294	12295	1873	1873	0	0	0	0
wait	20490	20491	1860	1860	0	0	0	0
./version1	245974	606263	1672	35	1637	0	0	0
wait	8196	8197	203	203	0	0	0	0
/usr/bin/tprof	291002	643291	13	13	0	0	0	0
/usr/bin/tprof	274580	610467	2	0	0	2	0	0
/etc/syncd	73824	110691	2	2	0	0	0	0
/usr/bin/sh	245974	606263	1	1	0	0	0	0
/usr/bin/sh	245976	606265	1	1	0	0	0	0

/usr/bin/trcstop	245976	606263	1	1	0	0	0
swapper	0	3	1	1	0	0	0
rmcd	155876	348337	1	1	0	0	0
=====	===	===	=====	=====	=====	=====	=====
Total			7504	5865	1637	2	0

Total Samples = 7504 Total Elapsed Time = 18.76s

Example B-3 from the report gives the subroutine-level profile for the Version1 program. In this simple example, all of the time is spent in **main()**.

Example: B-3 Excerpt from a tprof report - subroutine-level profile for the version1 program with all time spent in main()

Profile: ./version1

Total Ticks For All Processes (./version1) = 1637

	Subroutine	Ticks	%	Source	Address	Bytes
=====	=====	=====	=====	=====	=====	=====
.main	1637	21.82	version1.c	350	536	

For more information about using AIX **tprof** for Java programs, see “Hot method or routine analysis” on page 241.

The functions of **tprof** are rich. As such, it cannot be fully described in this guide. For complete **tprof** documentation, see *tprof Command*, found at:

<http://www-01.ibm.com/support/knowledgecenter/api/redirect/aix/v7r1/index.jsp?topic=/com.ibm.aix.cmds/doc/aixcmds5/tprof.htm>

AIX trace-based analysis tools

Trace¹ is a powerful utility that is provided by AIX for collecting a time-sequenced log of operating system events on a Power Systems server. The AIX kernel and kernel extensions are richly instrumented with trace *hooks* that, when trace is activated, append trace records with context-relevant data, to a pinned, kernel-resident trace buffer. These records can be later read from that buffer and logged to a disk-resident file. Further utilities are provided to interpret and summarize trace logs and generate human-readable reports. The **tprof** CPU profiler is one such utility. Besides **tprof**, two of the most-commonly used trace-based utilities are **curt**² and **splat**.^{3,4}

¹ *trace Daemon*, found at:

<http://www-01.ibm.com/support/knowledgecenter/api/redirect/aix/v7r1/index.jsp?topic=/com.ibm.aix.cmds/doc/aixcmds5/trace.htm>

² *CPU Utilization Reporting Tool (curt)*, found at:

http://www-01.ibm.com/support/knowledgecenter/api/redirect/aix/v7r1/topic/com.ibm.aix.prftools/doc/prftools/idprftools_cpu.htm

³ *Simple performance lock analysis tool (splat)*, found at:

http://www-01.ibm.com/support/knowledgecenter/api/redirect/aix/v7r1/topic/com.ibm.aix.prftools/doc/prftools/idprftools_splat.htm

⁴ *splat Command*, found at:

<http://www-01.ibm.com/support/knowledgecenter/api/redirect/aix/v7r1/index.jsp?topic=/com.ibm.aix.cmds/doc/aixcmds5/splat.htm>

The **curt** command takes as its input a trace that is collected by using the AIX trace facility, and generates a report that breaks down how CPU time is consumed by various entities, including:

- ▶ Processes (grouped by process name)
- ▶ Individual processes
- ▶ Individual threads
- ▶ System calls (either on a system-wide or per-thread basis)
- ▶ Interrupts

One of the most useful reports from **curt** is the *System Calls Summary*. This report provides a system-wide summary of the system calls that are run while the trace is collected. For each system call, the following information is provided:

- ▶ Count: The number of times the system call was run during the monitoring interval
- ▶ Total Time: Amount of CPU time (in milliseconds) consumed in running the system call
- ▶ % sys time: Percentage of overall CPU capacity that is spent in running the system call
- ▶ Avg Time: Average CPU time that is consumed for each execution of the system call
- ▶ Min Time: Minimum CPU time that is consumed during an execution of the system call
- ▶ Max Time: Maximum CPU time that is consumed during an execution of the system call
- ▶ SVC: Name and address of the system call

An excerpt from a System Calls Summary report is shown in Example B-4.

Example: B-4 System Calls Summary report (excerpt)

System Calls Summary						
Count	Total Time (msec)	% sys time	Avg Time (msec)	Min Time (msec)	Max Time (msec)	SVC (Address)
-----	-----	-----	-----	-----	-----	-----
123647	3172.0694	14.60%	0.0257	0.0128	0.9064	kpread(2a2d5e8)
539	1354.6939	6.24%	2.5133	0.0163	4.1719	listio64(516ea40)
26496	757.6204	3.49%	0.0286	0.0162	0.0580	_esend(2a29f88)
26414	447.7029	2.06%	0.0169	0.0082	0.0426	_erecv(2a29e98)
9907	266.1382	1.23%	0.0269	0.0143	0.5350	kpwrite(2a2d588)
34282	167.8132	0.77%	0.0049	0.0032	0.0204	_thread_wait(2a28778)

As a first step, compare the mix of system calls to the expectation of how the application is expected to behave. Is the mix aligned with expectations? If not, first confirm that the trace is collected while the wanted workload runs. If the trace is collected at the correct time and the mix still differs from expectations, then investigate the application logic. Also, examine the list of system calls for potential optimizations. For example, if **select** or **poll** is used frequently, consider employing the pollset facility (see 4.3.3, “pollset” on page 98).

As a further breakdown, **curt** provides a report of the system calls that are run by each thread. An example report is shown in Example B-5.

Example: B-5 System calls run by each thread

Report for Thread Id: 549305 (hex 861b9) Pid: 323930 (hex 4f15a)						
Process Name: proc1						

Total Application Time (ms): 89.010297						
Total System Call Time (ms): 160.465531						
Total Hypervisor Call Time (ms): 18.303531						
Thread System Call Summary						

Count	Total Time	Avg Time	Min Time	Max Time	SVC (Address)	

	(msec)	(msec)	(msec)	(msec)	
=====	=====	=====	=====	=====	=====
492	157.0663	0.3192	0.0032	0.6596	listio64(516ea40)
494	3.3656	0.0068	0.0002	0.0163	GetMultipleCompletionStatus(549a6a8)
12	0.0238	0.0020	0.0017	0.0022	_thread_wait(2a28778)
6	0.0060	0.0010	0.0007	0.0014	thread_unlock(2a28838)
4	0.0028	0.0007	0.0005	0.0008	thread_post(2a288f8)

Another useful report that is provided by **curt** is the *Pending System Calls Summary*. This summary shows the list of threads that are in an unfinished system call at the end of the trace. An example report is given in Example B-6.

Example: B-6 Threads that are in an unfinished system call at the end of the trace

Pending System Calls Summary				

Accumulated	SVC (Address)	Procname	(Pid	Tid)
Time (msec)				
=====	=====	=====	=====	=====
0.0082	GetMultipleCompletionStatus(549a6a8)	proc1	(323930	532813)
0.0089	_nsleep(2a28d30)	proc2	(270398	545277)
0.0054	_thread_wait(2a28778)	proc1	(323930	549305)
0.0088	GetMultipleCompletionStatus(549a6a8)	proc1	(323930	561437)
3.3981	listio64(516ea40)	proc1	(323930	577917)
0.0130	kpwrite(2a2d588)	proc1	(323930	794729)

For each thread in an unfinished system call, the following items are provided:

- ▶ The accumulated time in the system call
- ▶ The name of the system call (followed by the system call address in parentheses)
- ▶ The process name, followed by the Process ID and Thread ID in parentheses

This report is useful in determining what system calls are blocking threads from proceeding. For example, threads appearing in this report with an unfinished **recv** call are waiting on data to be received over a socket.

Another useful trace-based tool is **splat**, which is the Simple Performance Lock Analysis Tool. The **splat** tool provides reports about the usage of kernel and application (pthread-level) locks. At the pthread level, **splat** can report about the usage of pthread synchronizers: mutexes, read/write locks, and condition variables. Importantly, **splat** provides data about the degree of contention and blocking on these objects, an important consideration in creating highly scalable and pthread-based applications.

The pthread library instrumentation does not provide names or classes of synchronizers, so the addresses are the only way that you have to identify them. Under certain conditions, the instrumentation can capture the return addresses of the function call stack, and these addresses are used with the output of the **gensyms** tool to identify the call chains when these synchronizers are created. The creation and deletion times of the synchronizer can sometimes be determined as well, along with the ID of the pthread that created them.

An example of a mutex report from **splat** is shown in Example B-7.

Example: B-7 Mutex report from splat

```
[pthread MUTEX]  ADDRESS:      00000000F0154CD0
Parent Thread:  0000000000000001  creation time:    26.232305
Pid: 18396      Process Name: trcstop
Creation call-chain =====
00000000D268606C      .pthread_mutex_lock
```

00000000D268EB88	.pthread_once
00000000D01FE588	._libs_init
00000000D01EB2FC	dne_callbacks
00000000D01EB280	._libc_declare_data_functions
00000000D269F960	._pth_init_libc
00000000D268A2B4	.pthread_init
00000000D01EAC08	._modinit
000000001000014C	._start

Acqui- sitions	Miss Rate	Spin Count	Wait Count	Busy Count	Secs Held		Percent Held (26.235284s)			
					CPU	Elapsed	Real CPU	Real Elapsed	Comb Spin	Real Wait
1	0.000	0	0	0	0.000006	0.000006	0.00	0.00	0.00	0.00

Depth	Min	Max	Avg
SpinQ	0	0	0
WaitQ	0	0	0
Recursion	0	1	0

PThreadID	Acqui- sitions	Miss Rate	Spin Count	Wait Count	Busy Count	Percent Held of Total Time			
						CPU	Elapse	Spin	Wait
1	1	0.00	0	0	0	0.00	0.00	0.00	0.00

Function Name Offset	Acqui- sitions	Miss Rate	Spin Count	Wait Count	Busy Count	Percent Held of Total Time				Return Address	Start Address
						CPU	Elapse	Spin	Wait		
.pthread_once	0	0.00	0	0	0	99.99	99.99	0.00	0.00	00000000D268EC98	00000000D2684180
.pthread_once	1	0.00	0	0	0	0.01	0.01	0.00	0.00	00000000D268EB88	00000000D2684180

In addition to the common header information and the [pthread MUTEX] identifier, this report lists the following lock details:

Parent thread	Pthread ID of the parent pthread
Creation time	Elapsed time in seconds after the first event recorded in trace (if available)
Deletion time	Elapsed time in seconds after the first event recorded in trace (if available)
PID	Process identifier
Process Name	Name of the process that uses the lock
Call-chain	Stack of called methods (if available)
Acquisitions	The number of times the lock was acquired in the analysis interval
Miss Rate	The percentage of attempts that failed to acquire the lock
Spin Count	The number of unsuccessful attempts to acquire the lock
Wait Count	The number of times a thread is forced into a suspended wait state while waiting for the lock to come available
Busy Count	The number of trylock calls that returned busy
Seconds Held	This field contains the following subfields:
CPU	The total number of processor seconds the lock is held by a running thread.
Elapse(d)	The total number of elapsed seconds the lock is held, whether the thread was running or suspended.

Percent Held	This field contains the following subfields:	
	Real CPU	The percentage of the cumulative processor time the lock was held by a running thread.
	Real Elapsed	The percentage of the elapsed real time the lock is held by any thread, either running or suspended.
	Comb(ined) Spin	The percentage of the cumulative processor time that running threads spend spinning while it tries to acquire this lock.
	Real Wait	The percentage of elapsed real time that any thread was waiting to acquire this lock. If two or more threads are waiting simultaneously, this wait time is charged only one time. To learn how many threads are waiting simultaneously, look at the WaitQ Depth statistics.
Depth	This field contains the following subfields:	
	SpinQ	The minimum, maximum, and average number of threads that are spinning on the lock, whether running or suspended, across the analysis interval
	WaitQ	The minimum, maximum, and average number of threads that are waiting on the lock, across the analysis interval
	Recursion	The minimum, maximum, and average recursion depth to which each thread held the lock

Finding alignment issues

Improperly aligned code or data can cause performance degradation. By default, the IBM compilers and linkers correctly align code and data, including stack and statically allocated variables. Incorrect typecasting can result in references to storage that are not correctly aligned. There are two types of alignment issues with which to be concerned:

- ▶ Alignment issues that are handled by Licensed Internal Code in the POWER7 processor
- ▶ Alignment issues that are handled through alignment interrupts.

Examples of alignment issues that are handled by Licensed Internal Code with a performance penalty in the POWER7 processor are loads that cross a 128-byte boundary and stores that cross a 4 KB page boundary. To give an indication of the penalty for this type of misalignment, on a 4 GHz processor, a nine-instruction loop that contains an 8-byte load that crosses a 128-byte boundary takes double the time of the same loop with the load correctly aligned.

Alignment issues that are handled by Licensed Internal Code can be detected by running **hpmcount** or **hpmstat**. The **hpmcount** command is a command-line utility that runs a command and collects statistics from the POWER7 PMU while the command runs. To detect alignment issues that are handled by Licensed Internal Code, run **hpmcount** to collect data for group 38. An example is provided in Example B-8.

Example: B-8 Example of the results of the hpmcount command

```
# hpmcount -g 38 ./unaligned
Group: 38
Counting mode: user
Counting duration: 21.048874056 seconds
PM_LSU_FLUSH_ULD (LRQ unaligned load flushes)      :      4320840034
PM_LSU_FLUSH_UST (SRQ unaligned store flushes)      :              0
```

PM_LSU_FLUSH_LRQ (LRQ flushes)	:	450842085
PM_LSU_FLUSH_SRQ (SRQ flushes)	:	149
PM_RUN_INST_CMPL (Run instructions completed)	:	19327363517
PM_RUN_CYC (Run cycles)	:	84219113069
Normalization base: time		
Counting mode: user		
Derived metric group: General		
[] Run cycles per run instruction	:	4.358

The **hpmstat** command is similar to **hpmcount**, except that it collects performance data on a system-wide basis, rather than just for the running of a command.

Generally, scenarios in which the ratio of (*LRQ unaligned load flushes* + *SRQ unaligned store flushes*) divided by *Run instructions completed* is greater than 0.5% must be further investigated. The **tprof** command can be used to further pinpoint where in the code the unaligned storage references are occurring. To pinpoint unaligned loads, the **-E PM_MRK_LSU_FLUSH_ULD** flag is added to the **tprof** command line, and to pinpoint unaligned stores, the **-E PM_MRK_LSU_FLUSH_UST** flag is added. When these flags are used, **tprof** generates a profile where unaligned loads and stores are sampled instead of time-based sampling.

Examples of alignment issues that cause an alignment interrupt include execution of a **lmmw** or **lwarx** instruction on a non-word-aligned boundary. These issues can be detected by running **alstat**. This command can be run with an interval, which is the number of seconds between each report. An example is presented in Example B-9.

Example: B-9 Alignment issues can be addressed with the alstat command

```
> alstat 5
Alignment  Alignment
SinceBoot   Delta
    2016      0
    2016      0
    2016      0
    2016      0
    2016      0
    2016      0
```

The key metric in the **alstat** report is the Alignment Delta. This metric is the number of alignment interrupts that occurred during the interval. Nonzero counts in this column merit further investigation with **tprof**. Running **tprof** with the **-E ALIGNMENT** flag generates a profile that shows where the unaligned references are occurring.

For more information, see *alstat Command*, found at:

<http://www-01.ibm.com/support/knowledgecenter/api/redirect/aix/v7r1/index.jsp?topic=/com.ibm.aix.cmds/doc/aixcmds1/alstat.htm>

Finding emulation issues

Over the 20+ year evolution of the POWER instruction set, a few instructions were removed. Instead of trapping programs that run these instructions, AIX emulates them in the kernel, although with a significant processing impact. Generally, programs that are written in a third-generation language (for example, C and C++) and compiled with an up-to-date compiler do not contain these emulated instructions. However, older binary files or older hand-written assembly language might contain such instructions, and because they are silently emulated by AIX, the performance penalty might not be readily apparent.

The **emstat** command detects the presence of these instructions. Like **alstat**, it is run with an interval, which is the number of seconds between reports. An example is shown in Example B-10.

Example: B-10 The emstat command detects the presence of emulated instructions

```
> emstat 5
  Emulation  Emulation
  SinceBoot   Delta
           0      0
           0      0
           0      0
           0      0
           0      0
```

The key metric is the **Emulation Delta** (the number of instructions that are emulated during each interval). Nonzero values merit further investigation. Running **tprof** with the **-E EMULATION** flag generates a profile that shows where the emulated instructions are.

For more information, see *emstat Command*, found at:

<http://www-01.ibm.com/support/knowledgecenter/api/redirect/aix/v7r1/index.jsp?topic=/com.ibm.aix.cmds/doc/aixcmds2/emstat.htm>

hpmstat, hpmcount, and tprof -E

The POWER processor provides a powerful on-chip PMU that can be used to count the number of occurrences of performance-critical processor events. A rich set of events is countable; examples include level 2 and level 3 d-cache misses, and cache reloads from local, remote, and distant memory. *Local memory* is memory that is attached to the same POWER processor chip that the software thread is running on. *Remote memory* is memory that is attached to a different POWER processor that is in the same central electronic complex (CEC) (that is, the same node or building block in the case of a multi-CEC system, such as a Power 780) on which the software thread is running. *Distant memory* is memory that is attached to a POWER processor that is in a different CEC from the CEC on which the software thread is running.

Two commands exist to count PMU events: **hpmcount** and **hpmstat**. The **hpmcount** command is a command-line utility that runs a command and collects statistics from the PMU while the command runs. The **hpmstat** command is similar to **hpmcount**, except that it collects performance data on a system-wide basis, rather than just for the execution of a command.

Further documentation about **hpmcount** and **hpmstat** can be found at:

- ▶ <http://www-01.ibm.com/support/knowledgecenter/api/redirect/aix/v7r1/index.jsp?topic=/com.ibm.aix.cmds/doc/aixcmds2/hpmcount.htm>
- ▶ <http://www-01.ibm.com/support/knowledgecenter/api/redirect/aix/v7r1/index.jsp?topic=/com.ibm.aix.cmds/doc/aixcmds2/hpmstat.htm>

In addition to simply counting processor events, the PMU can be configured to sample instructions based on processor events. With this capability, profiles can be generated that show which parts of an application are experiencing specified processor events. For example, you can show which subroutines of an application are generating level 2 or level 3 cache misses. The **tprof** profiler includes these functions through the **-E** flag, which allows a PMU event name to be provided to **tprof** as the sampled event. The list of PMU events can be generated by running **pm1ist -c -1**. Whenever possible, perform profiling by using *marked* events, as profiling that uses marked events is more accurate than profiling that uses unmarked events. The marked events begin with the prefix **PM_MRK_**.

For more information about using the **-E** flag of **tprof**, go to the following website:

<http://www-01.ibm.com/support/knowledgecenter/api/redirect/aix/v7r1/index.jsp?topic=/com.ibm.aix.cmds/doc/aixcmds5/tprof.htm>

Linux

The section introduces tools and techniques that are used for optimizing software on the combination of Power Systems and Linux. The intended audience for this section is software development teams.

To download Java for Linux, go to the following website:

<http://www.ibm.com/developerworks/java/jdk/linux/>

Empirical performance analysis by using the IBM Software Development Kit for Linux on Power

After you apply the best high-level optimization techniques, a deeper level of analysis might be required to gain more performance improvements. You can use the IBM Software Development Kit (SDK) for Linux on Power to help you gain these improvements.

The IBM SDK for Linux on Power is a set of tools that support:

- ▶ Hot spot analysis
- ▶ Analysis of ported code for missed platform-specific optimization
- ▶ Whole program analysis for coding issues, for example, pipeline hazards, inlining opportunities, early exits and hidden path length, devirtualization, and branch prediction hints
- ▶ Lock contention and IO delay analysis

The *IBM SDK for PowerLinux* can be found at:

<http://www.ibm.com/support/customer/sas/f/lopdiags/sdklop.html>

The SDK provides an Eclipse C/C++ IDE with Linux tools integration. The SDK provides graphical presentation and source code view integration with Linux execution profiling (**gprof/OProfile/Perf**), malloc and memory usage (**valgrind**), pthread synchronization (**helgrind**), SystemTap tapsets, and tapset development.

Hotspot analysis

You should profile the application and look for hotspots. When you run the application under one or more representative workloads, use a hardware-based profiling tool such as **OProfile**. **OProfile** can be run directly as a command-line tool or under the IBM SDK for Linux on Power.

Note: You might find it useful to use the Linux **perf** tool in its *top* mode. This **perf top** works similarly to old **top** (however, instead of showing processes, it shows hot methods).

The **OProfile** tools can monitor the whole system (LPAR), including all the tasks and the kernel. This action requires root authority, but is the preferred way to profile the kernel and complex applications with multiple cooperating processes. **OProfile** is fully enabled to take samples by using the full set of the PMU events (run **ophelp** for a complete list of events). **OProfile** can produce text file reports that are organized by process, program and libraries, function symbols, and annotated source file and line number or machine code disassembly.

The IBM SDK for Linux on Power can profile applications that are associated with Eclipse projects. The SDK automates the setup and running of the profile, but is restricted to a single application, its libraries, and direct kernel calls. The SDK is easier to use, as it is hierarchically organized by percentage with program, function symbol, and line number. Clicking the line number in the profile pane *jumps* the source view pane to the matching source file and line number. This action simplifies edit, compile, and profile tuning activities.

The whole system profile is a good place to start. You might find that your application is consuming most of the CPU cycles, and deeper analysis of the application is the next logical step. The IBM SDK for Linux on Power provides a number of helpful tools, including integrated application profiling (**OProfile** and **valgrind**), Migration Assistant, and the Source Code Advisor.

High kernel usage

If the bulk of the CPU cycles is consumed in the kernel or runtime libraries that are not part of your application, then a different type of analysis is required. If the kernel is consuming significant cycles, then the application might be I/O or lock contention bound. This situation can occur when an application moves to larger systems (higher core count) and fails to scale up.

I/O bound applications can be constrained by small buffer sizes or a poor choice of an access method. One issue to look for is applications that use local loopback sockets for interprocess communications (IPC). This situation is common for applications that are migrating from early scale-out designs to larger systems (and core-count). The first application change is to choose a lighter weight form of IPC for in-system communications.

Excessive locking or poor lock granularity can also result in high kernel usage (in the kernel's `spin_lock`, `futex`, and scheduler components) when applications move to larger system configurations. This situation might require adjusting the application lock strategy and possibly the type of lock mechanism that is used as well:

- ▶ POSIX `pthread_mutex` and `pthread_rwlock` locks are complex and heavy, and POSIX semaphores are simpler and lighter.
- ▶ Use `trylock` forms to spin in user mode for a limited time when appropriate. Use this technique when there is normally a finite lock hold time and limited contention for the resource. This situation avoids context switch and scheduler impact in the kernel.
- ▶ Reserve POSIX `pthread_spinlock` and `sched_yield` for applications that have exclusive use of the system and with carefully designed thread affinity (assigning specific threads to specific cores).
- ▶ The compiler provides inline functions (`__sync_fetch_and_add`, `__sync_fetch_and_or`, and so on) that are better suited for simple atomic updates than POSIX lock and unlock. Use thread local storage, where appropriate to avoid locking for thread safe code.

Using the IBM SDK for Linux on Power Trace Analyzer

The IBM SDK for Linux on Power provides tools, including the SystemTap and `pthread` monitor, for tracking I/O and lock usage of a running application. The higher-level Trace Analyzer tools can target a specific application for a combined SystemTap syscall trace and Lock Trace. The resulting trace information is correlated for time strip display and analysis within the tool.

High library usage

If libraries are consuming significant cycles, then you must determine whether the following items are true:

- ▶ Those libraries are part of your application, provided by a third party, or the Linux distribution
- ▶ There are alternative libraries that are better optimized
- ▶ You can recompile those libraries at a higher optimization

Libraries that are part of your application require the same level of empirical analysis as the rest of your application (by using source profiling and the Source Code Advisor (SCA)). When you have libraries that are used by, but not part of your application, this implies a number of options and strategies:

- ▶ Most open source packages in the Linux environment are compiled with optimization level `-O2` and tend to avoid additional (higher-level GCC) compiler options. This configuration might be sufficient for a CISC processor with limited register resources, but not sufficient for a RISC-based register-rich processor, such as POWER7 and POWER8 processors.
- ▶ A RISC-based, superscalar, out-of-order execution processor chip, such as the POWER8 processor, requires more aggressive inlining and loop-unrolling to capitalize on the larger register set and superscalar design point. Also, automatic vectorization is not enabled at this lower (`-O2`) optimization level, and so the vector registers and ISA feature go unused.

- In GCC, you must specify the **-O3** optimization level and inform the compiler that you are running on a newer processor chip with the Vector ISA extensions. In fact, with GCC, you need both **-O3** and **-mcpu=power7** for the compiler to generate code that capitalizes on the new VSX feature of the POWER7 processor. You need both **-O3** and **-mcpu=power8** for the compiler to take advantage of the latest VSX instructions that are implemented on the POWER8 processor.

One source of optimized libraries is the IBM Advance Toolchain for Linux on Power. The Advance Toolchain provides alternative runtime libraries for all the common POSIX C language, Math, and pthread libraries that are highly optimized (**-O3** and **-mcpu=**) for multiple Power Systems platforms (including POWER7 and POWER8 processor-based systems). The IBM Advance Toolchain runtime RPM provides multiple CPU tuned library instances and automatically selects the specific library version that is optimized for the specific POWER5, POWER6, POWER7, or POWER8 processor-based system.

If there are specific open source or third-party libraries that are dominating the execution profile of your application, you must ask the distribution or library product owner to provide a build that uses higher optimization. Alternatively, for open source library packages, you can build your own optimized binary version of those packages.

Deeper empirical analysis

If simple recompilation with higher optimization options or even a more capable compiler does not provide acceptable performance, then deeper analysis is required. The IBM SDK for Linux on Power integrates the following analysis tools:

- Migration Assistant analysis, non-performing codes, and data types
- Application-specific hotspot profiling
- SCA analysis for non-performing code idioms and induced execution hazards

The Migration Assistant analyzes the source code directly and does not require a running binary application for analysis. Profiling and the SCA *do* require compiled application binary files and an application-specific benchmark or repeatable workload for analysis.

The Migration Assistant

For applications that originate on another platform, the Migration Assistant (MA) can identify non-portable code that must be addressed for a successful port to Power Systems. The MA uses the Eclipse infrastructure to analyze the following items:

- Data-endian-dependent unions and structures
- Casts with potential endian issues
- Non-portable data types
- Non-portable inline assembly language code
- Non-portable or arch-dependent compiler built-ins
- Proprietary or architectural-specific APIs

Program usage of non-portable data types and inline assembly language can cause poor performance on the POWER processor, which always must be investigated and addressed.

For example, the long double data type is supported for both Intel x86 and POWER, but has a different size, data range, and implementation. The x86 80-bit Floating Point format is implemented in hardware and is faster than (although not compatible with) the AIX long double, which is implemented as an algorithm that uses two 64-bit doubles. Neither one is fully IEEE-compliant, and both must be avoided in cross-platform application codes and libraries.

Another example is small Intel specific optimization that uses inline x86 assembly language and conditionally providing a generic C implementation for other platforms. In most cases, GCC provides an equivalent built-in function that generates the optimal code for each platform. Replacing inline assembly language with GCC built-in functions makes the application more portable and provides equivalent or better performance on all platforms.

To use the MA tool, complete the following steps:

1. Import your project into the SDK.
2. Select the Project's **Properties**.
3. Select the **Linux/x86 to PowerLinux application Migration** check box under C/C++ General/Code Analysis.
4. Right-click the project name, and select **Run Migration Advisor**.

Hotspot profiling

IBM SDK for Linux on Power integrates the Linux **OProfile** hardware event profiling with the application source code view. This configuration is a convenient way to do hotspot analysis. The integrated Linux Tools profiler focuses on an application that is selected from the current SDK project.

After you run the application, the SDK opens an **OProfile** tab in a console window. This window shows a nested set of *twisties*, starting with the *event* (cycles by default), then *program/library*, *function*, and *source line* (within function). The developer drills down by opening the twisties in the profile window, opening the next level of detail. Items are ordered by profile frequency with highest frequency first. Clicking the function or line number entries in the profile window causes the source view to *jump* to the corresponding source file or line number.

This process is a convenient way to do hotspot analysis, focusing only on the top three to five items at each level in the profile. Examine the source code for algorithmic problems, excess conversions, unneeded debug code, and so on, and make the appropriate source code changes.

With your application code (or subset) imported in to the SDK, it is easy to edit, compile, and profile code changes and verify improvements. As the developer makes code improvements, the hotspots in the profile change. Repeat this process until performance is satisfactory or all the profile entries at the function level are in the low single digits.

To use the integrated profiler, right-click the project and select **Profile As** → **Profile with OProfile**. If your project contains multiple applications or the application needs setup or inputs to run the specific workload, then create profile configurations as needed.

Detailed analysis with the Source Code Advisor

Hotspot analysis might not find all of the latent performance problems, especially coding style and some machine-specific hazards. These problems tend to be diffused across the application, and do not show up in hotspot analysis. Common examples of machine hazards include address translation, cache misses, and branch miss-predictions.

Complex C++ applications or C programs that use object-based techniques might see performance issues that are related to using many small functions of indirect calls. Unless the compiler or optimizer can see the whole program or library, it cannot prove that it is safe to optimize these cases. However, it is possible for the developer to optimize manually at the source level, as the developer knows the original intent or actual usage in context.

The SCA can find and recommend solutions for many of these coding style and machine hazards. The process generates a journal that associates performance problems (including hazards) with specific source file and line numbers.

The SCA window has a drill-down hierarchy similar to the profile window that is described in “Hotspot profiling” on page 237. The SCA window is organized as a list of problem categories, and then nested twisties, for affected functions and source line numbers within functions. Functions and lines are ordered by the percent of overall contribution to execution time. Associated with each problem is a plain language description and suggested solution that describes a source change or compiler or linker options that are expected to resolve the problem. Clicking the line number item *jumps* the source display to the associated source file and line number for editing.

SCA uses the Feedback Directed Program Restructuring (FDPR) tool to instrument your application (or library) for code and data flow trace when you run a workload. The resulting FDPR journal is used to drive the SCA analysis. Running FDPR and retrieving the journal is automated by clicking **Profile as** → **Profile with Source Code Advisor**.

Pipeline stall analysis with the cycles per instruction breakdown tool

The cycles per instruction (CPI) metric is a measure of the average processor clock cycles that are needed to complete an instruction. The CPI value is a measure of processor performance and, in a modern processor such as the POWER processor, a high value can indicate poor performance because of a high ratio of stalls in the execution pipeline. By collecting information from the processor's PMU, those events and derived metrics can be mapped to the CPU functional units (for example, branch, load or store, or floating point), where they occurred. These events and metrics can be represented in a hierarchical breakdown of cycles, called the CPI breakdown model (CBM). For more information about CPI metric and pipeline analysis, see *Commonly Used Metrics for Performance Analysis*, found at (registration required):

<https://www.power.org/documentation/commonly-used-metrics-for-performance-analysis/>

The IBM SDK for Linux on Power delivers with the CPI breakdown tool for automating the collection of PMU stall events and for building a CBM representation of application execution. After you run the application, the CPI breakdown tool opens a CBM view in the default Eclipse perspective. This view shows a breakdown of stall events and metrics, along with their contribution percentage and description. A sample is shown in Figure B-6 on page 239.

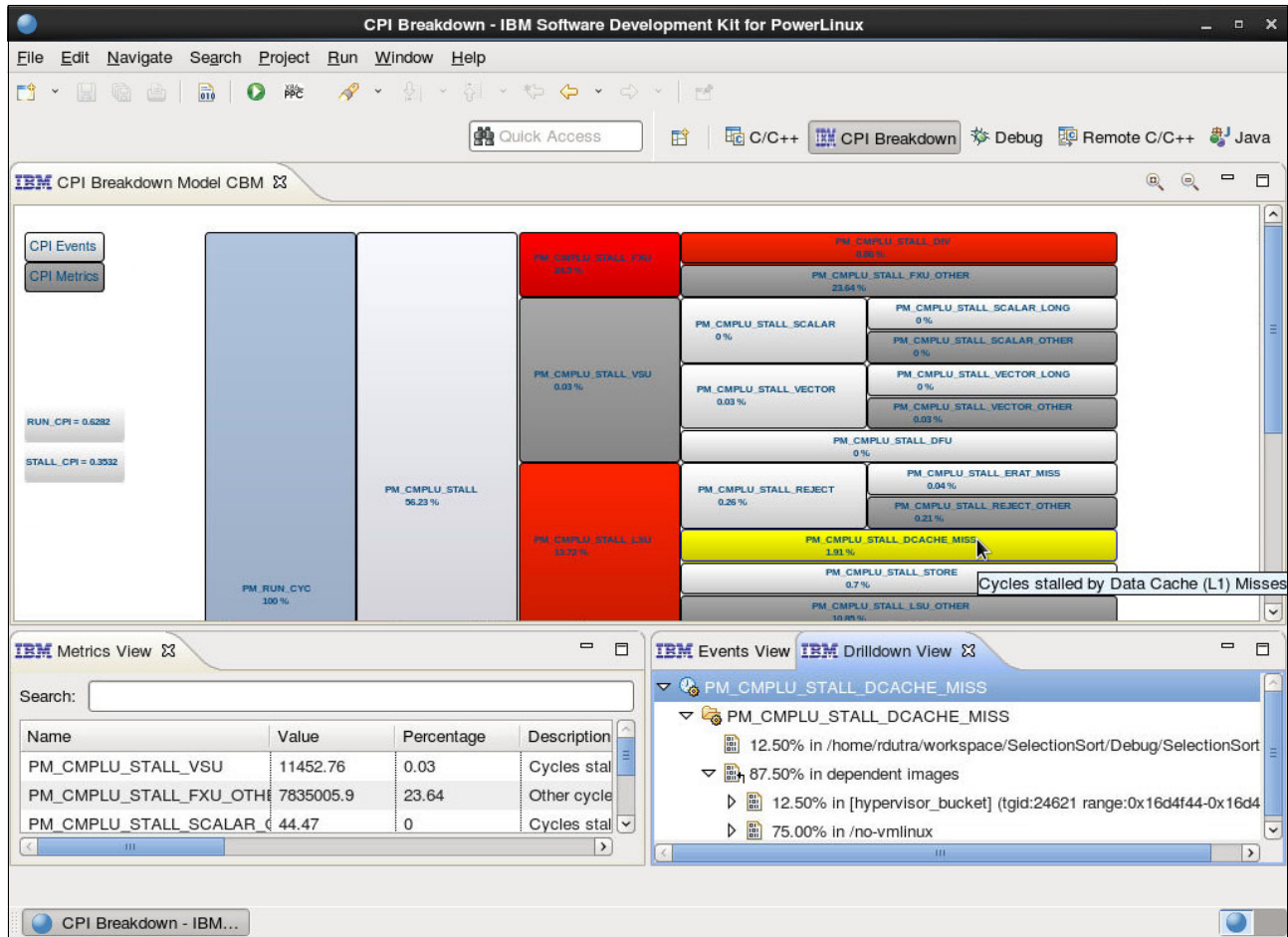


Figure B-6 The CPI Breakdown tool perspective

In the CBM view (see Figure B-6), click any of the squares to open the drill-down menu that shows a nested set of twists, including the event, program or library, function, and source line, as shown in the lower right of Figure B-6. As you drill down, the items are ordered by profile frequency, with highest frequency first. Click a function or line number in the profile window to open the source view and jump to the corresponding source file and line number.

Use the CPI Breakdown tool to measure application behavior in the POWER processor and for hotspot analysis. The tool assists in finding the CPU functional units with a high ratio of stalls and the corresponding chunks of source code that are likely to be the cause of performance degradation of your application.

Java (either AIX or Linux)

Focused empirical analysis of Java applications involves gathering specific types of performance information, making and assessing changes, and repeating the process. The specific areas to consider, the types of performance information to gather, and the tools to use, are described in this section.

To download Java for AIX or Linux, go to the following websites:

- ▶ For AIX: <http://www.ibm.com/developerworks/java/jdk/aix/>
- ▶ For Linux: <http://www.ibm.com/developerworks/java/jdk/linux/>

32-bit or 64-bit JDK

All other things being equal, a 32-bit JDK has about 5% higher performance than a 64-bit JDK that uses **-Xcompressedrefs**. Without the **-Xcompressedrefs** options for a 64-bit JDK, a 32-bit JDK might have 10% higher performance compared to a 64-bit JDK. Give careful consideration to the choice of a 32-bit or 64-bit JVM. It is *not* a good choice to take an application that suffers from excessive object allocation rates and switch to a 64-bit JVM simply to allow a larger heap size.

For more information about this topic, see the following sections:

- ▶ “Verbose GC Log” on page 240
- ▶ Section 8.2, “32-bit versus 64-bit Java” on page 174.

Java heap size, and garbage collection policies and parameters

The performance of Java applications is often influenced by the heap size, GC policy, and GC parameters. Try different combinations that are guided by appropriate data gathering and analysis. Various tools and diagnostic options are available that can provide detailed information about the state of the JVM. The information that is provided can be used to guide tuning decisions to maximize performance for an application or workload.

Verbose GC Log

The verbose GC log is a keytool to understanding the memory characteristics of a particular workload. The information that is provided in the log can be used to guide tuning decisions to minimize GC impact and improve overall performance. Logging can be activated with the **-verbose:gc** option and is directed to the command terminal. Logging can be redirected to a file with the **-Xverbosegclog:<file>** option.

Verbose logs capture many types of GC events, such as regular GC cycles, allocation failures, heap expansion and contraction, events that are related to concurrent marking, and scavenger collections. Verbose logs also show the approximate length of time many events take, the number of bytes processed (if applicable), and other relevant metrics. Information relevant to many of the tuning issues for GC can be obtained from the log, such as appropriate GC policies, optimal constant heap size, optimal min and max free space factors, and growth and shrink sizes. For a detailed description of verbose log output, see *Diagnostics Guide for IBM SDK and Runtime Environment Java Technology Edition, Version 6*, found at:

<http://www-01.ibm.com/support/knowledgecenter/api/redirect/javasdk/v6r0/topic/com.ibm.java.doc.diagnostics.60/homepage/plugin-homepage-java6.html>

Garbage collection and memory visualizer

For large, long-running workloads, verbose logs can quickly grow in size, making them difficult to work with and to analyze an application's behavior over time. The GC and memory visualizer is a tool that can parse verbose GC logs and present them in a visual manner by using graphs and other diagrams, allowing trends and totals to be recognized easily and quickly. The graphs can be used to determine the minimum and maximum heap usage, growth and shrink rates over time, and identify oscillating behaviors. This information can be especially helpful when you choose optimal GC parameters. The GC and memory visualizer can also compare multiple logs side by side, which can aid in testing various options in isolation and determining their effects.

For more information about the GC and memory visualizer, see *Java diagnostics, IBM style, Part 2: Garbage collection with the IBM Monitoring and Diagnostic Tools for Java – Garbage Collection and Memory Visualizer*, found at:

<http://www.ibm.com/developerworks/java/library/j-ibmtools2>

Java Health Center

The Java Health Center is the successor to both the GC and memory visualizer and the Java Lock Monitor. It is an all-in-one tool that provides information about GC activity, memory usage, and lock contention. The Health Center also functions as a profiler, providing sample-based statistics on method execution. The Health Center functions as an agent of the JVM being monitored and can provide information throughout the life of a running application.

For more information about the Java Health Center, see *Java diagnostics, IBM style, Part 5: Optimizing your application with the Health Center*, found at:

<https://www.ibm.com/developerworks/java/library/j-ibmtools5>

For more information, see 8.5, “Java garbage collection tuning” on page 183.

Hot method or routine analysis

A CPU profile shows a breakdown of the time that is spent in Java methods and JNI or system routines. Investigate any hot methods or routines to determine whether the concentration of execution time in them is warranted or whether there is poor coding or other issues.

Here are some tools and techniques for this analysis:

- ▶ AIX **tprof** profiling. For more information, see *tprof Command*, found at:

<http://www-01.ibm.com/support/knowledgecenter/api/redirect/aix/v7r1/index.jsp?topic=/com.ibm.aix.cmds/doc/aixcmds5/tprof.htm>

- ▶ Linux **OProfile** profiling. For more information about **OProfile**, see the following resources:

- *Getting started with OProfile on PowerLinux* (resource page), found at:

<http://www-01.ibm.com/support/knowledgecenter/api/redirect/lxinfo/v3r0m0/index.jsp?topic=%2Fliacf%2Foprofgetstart.htm>

- *Getting started with OProfile on PowerLinux*, found at:

http://www-01.ibm.com/support/knowledgecenter/api/redirect/lxinfo/v3r0m0/topic/liacf/oprofile_pdf.pdf

- *OProfile results with JIT samples*, found at:

<http://oprofile.sourceforge.net/doc/getting-jit-reports.html>

- *Java Performance on POWER7*, found at:

https://www.ibm.com/developerworks/community/wikis/home?lang=en#!/wiki/W51a7ffcf4dfd_4b40_9d82_446ebc23c550/page/Java%20Performance%20on%20POWER7

- *OProfile manual*, found at:

<http://oprofile.sourceforge.net/doc/index.html>

General information about running the profiler and interpreting the results are in “AIX” on page 223 and “Linux” on page 233. For Java profiling, additional Java options are required to profile the machine code that is generated for methods by the JIT compiler:

- ▶ AIX 32-bit: **-agentlib:jpa=instructions=1**
- ▶ AIX 64-bit: **-agentlib:jpa64=instructions=1**
- ▶ Linux OProfile: **-agentlib:jvmti_oprofile**

The entire execution of a Java program can be profiled, for example, on AIX by running the following command:

```
tprof -ujeskl -A -I -E -x java ...
```

However, it is more common to profile Java after a warm-up period so that JIT compilation activity has completed. To profile after a warm-up, start Java and wait an appropriate interval until a steady-state performance is reached, which is anywhere from a few seconds to a few minutes for large applications. Then, start the profiler, for example, on AIX, by running the following command:

```
tprof -ujeskl -A -I -E -x sleep 60
```

On Linux, **OProfile** and **perf** can be used in a similar fashion; for more information, see “Java profiling example”.

Java profiling example

Example B-11 contains a sample Java program that is profiled on AIX and Linux. This program does some meaningless work and is purposely poorly written to illustrate lock contention and GC impact in the profile. The program creates three threads, but serializes their execution by having them attempt to lock the same object. One thread at a time acquires the lock, forcing the other two threads to wait until they can get the lock and run the code that is protected by the synchronized statement in the **doWork** method. While they wait to acquire the lock, the threads initially use *spin locking*, repeatedly checking whether the lock is free. After a suitable amount of spinning, the threads block rather than continuing to use CPU resources.

Example: B-11 Sample Java program

```
public class ProfileTest extends Thread {

    static Object o; /* used for locking to serialize threads */
    static Double A[], B[], C[];
    static int Num=1000;

    public static void main(String[] args) {
        o = new Object();
        new ProfileTest().start(); /* start 3 threads */
        new ProfileTest().start(); /* each thread executes the "run" method */
        new ProfileTest().start();
    }

    public void run() {
        double sum = 0.0;
        for (int i = 0; i < 50000; i++) {
            sum += doWork(); /* repeatedly do some work */
        }
        System.out.println("sum: "+sum); /* use the results of the work */
    }
}
```

```

public double doWork() {
    double d;
    synchronized (o) { /* serialize the threads to create lock contention */
        A = new Double [Num];
        B = new Double [Num];
        C = new Double [Num];
        initialize();
        calculate();
        d = C[0].doubleValue();
    }
    return(d); /* use the calculated values */
}

public static void initialize() {
    /* Initialize A and B. */
    for (int i = 0; i < Num; i++) {
        A[i] = new Double(Math.random()); /* use new to create objects */
        B[i] = new Double(Math.random()); /* to force garbage collection */
    }
}

public static void calculate() {
    for (int i = 0; i < Num; i++) {
        C[i] = new Double(A[i].doubleValue() * B[i].doubleValue());
    }
}
}

```

The program also uses the Double class, creating many short-lived objects by using **new**. By running the program with a small Java heap, GC is frequently required to free the Java heap space that is taken by the Double objects that are no longer in use.

Example B-12 shows how this program was run and profiled on AIX. 64-bit Java was used with the options **-Xms10m** and **-Xmx10m** to specify the size of the Java heap. The profile that is generated appears in the `java.prof` file.

Example: B-12 Results of running tprof on AIX

```
# tprof -ujeskl -A -I -E -x java -Xms10m -Xmx10m -agentlib:jpa64=instructions=1 ProfileTest
```

```
Starting Command java -Xms10m -Xmx10m -agentlib:jpa64=instructions=1 ProfileTest
```

```

sum: 12518.481782746869
sum: 12507.63528674597
sum: 12526.320955364286
stopping trace collection.
Sun Oct 30 15:04:21 2011
System: AIX 6.1 Node: e19-90-28 Machine: 00F603F74C00
Generating java.trc
Generating java.syms

```

```
Generating java.prof
```

Example B-13 and Example B-14 contain excerpts from the `java.prof` file that is created on AIX. Here are the notable elements of the profile:

- **Lock contention impact:** The impact of spin locking is shown in Example B-13 as ticks in the `libj9jit24.so` helper routine `jitMonitorEntry`, in the AIX pthreads library `libpthreads.a`, and in the AIX kernel routine `_check_lock`. This Java program clearly has excessive lock contention with `jitMonitorEntry` consuming 26.66% of the ticks in the profile. `jitMonitorEntry` and other routines, such as `jitMethodMonitorEntry`, indicate spin locking at the Java language level, and the impact in the pthreads library or `_check_lock` is locking at the system level, which might be associated with Java locks. For example, `libpthreads.a` and `_check_lock` are active for lock contention that is related to `malloc` on AIX.

Example: B-13 AIX profile excerpt showing kernel and shared library ticks

Total Ticks For All Processes (KERNEL) = 690					
Subroutine	Ticks	%	Source	Address	Bytes
=====	=====	=====	=====	=====	=====
._check_lock	240	5.71	low.s	3420	40
Shared Object	Ticks	%	Address	Bytes	
=====	=====	=====	=====	=====	
libj9jit24.so	1157	27.51	900000003e81240	5c8878	
libj9gc24.so	510	12.13	900000004534200	91d66	
/usr/lib/libpthreads.a[shr_xpg5_64.o]	175	4.16	900000000b83200	30aa0	

Profile: `libj9jit24.so`

Total Ticks For All Processes (`libj9jit24.so`) = 1157

Subroutine	Ticks	%	Source	Address	Bytes
=====	=====	=====	=====	=====	=====
._jitMonitorEntry	1121	26.66	nathelp.s	549fc0	cc0

- **GC impact:** The impact of initializing new objects and of GC is shown in Example B-13 as 12.13% of ticks in the `libj9gc24.so` shared object. This high GC impact is related to the excessive creation of `Double` objects in the sample program.
- **Java method execution:** In Example B-14, the profile shows the time that is spent in the `ProfileTest` class, which is broken down by method. Some methods appear more than one time in the breakdown because they are compiled multiple times at increasing optimization levels by the JIT compiler. Most of the ticks appear in the final highly optimized version of the `doWork()D` method, into which the `initialize()V` and `calculate()V` methods are inlined by the JIT compiler.

Example: B-14 AIX profile excerpt showing Java classes and methods

Total Ticks For All Processes (JAVA) = 1450		
Class	Ticks	%
=====	=====	=====
ProfileTest	1401	33.32
java/util/Random	38	0.90

java/lang/Float	5	0.12
java/lang/Double	3	0.07
java/lang/Math	3	0.07

Profile: ProfileTest

Total Ticks For All Processes (ProfileTest) = 1401

Method	Ticks	%	Source	Address	Bytes
=====	=====	=====	=====	=====	=====
doWork()D	1385	32.94	ProfileTest.java	1107283bc	b54
doWork()D	6	0.14	ProfileTest.java	110725148	464
doWork()D	4	0.10	ProfileTest.java	110726e3c	156c
initialize()V	3	0.07	ProfileTest.java	1107262dc	b4c
calculate()V	2	0.05	ProfileTest.java	110724400	144
initialize()V	1	0.02	ProfileTest.java	1107255c4	
d04					

Example B-15 contains a shell program that collects a profile on Linux by using **OProfile**. The resulting profile might be similar to the previous example profile on AIX, indicating substantial time in spin locking and in GC. Depending on some specifics of the Linux system, however, the locking impact can appear in routines in the `libj9thr24.so` shared object, as compared to the AIX spin locking seen in `libj9jit24.so`.

In some cases, an environment variable setting might be necessary to indicate the location of the JVMTI library that is needed for running **OProfile** with Java:

- ▶ Linux 32-bit: **LD_LIBRARY_PATH=/usr/lib/oprofile**
- ▶ Linux 64-bit: **LD_LIBRARY_PATH=/usr/lib64/oprofile**

Alternatively, you can specify the full path to the JVMTI library on the Java command line, such as:

```
java -agentpath:/usr/lib/oprofile/libjvmti_oprofile.so
```

Example: B-15 Linux shell to collect a profile by using OProfile

```
#!/bin/bash

# Note -
# Oprofile version 0.9.8 (~2010) deprecated the opcontrol interfaces
# in favor of the Linux kernel perf-events interface, available through
# the perf utility. The opcontrol interface is no longer used.

# Select the performance counter that counts non-idle cycles and
# generate a sample after 500,000 such events.
# this becomes "-e PM_RUN_CYC:500000 "
```

```
# Specify the jvmti_oprofile library to allow the perf tools to resolve
# the jitted methods.
# this becomes "-agentlib:jvmti_oprofile"
```

```
perf -e PM_RUN_CYC:500000 java -Xms10m -Xmx10m -agentlib:jvmti_oprofile ProfileTest
```

```
opreport > ProfileTest_summary.log
```

```
opreport -l > ProfileTest_long.log
```

Locking analysis

Locking bottlenecks are fairly common in Java applications. Collect locking information to identify any bottlenecks, and then take the appropriate steps to eliminate the problems. A common case is when older Java/util classes, such as Hashtable, do not scale well and cause a locking bottleneck. An easy solution is to use Java/util/concurrent classes instead, such as ConcurrentHashMap.

Locking can be at the Java code level or at the system level. Java Lock Monitor is an easy to use tool that identifies locking bottlenecks at the Java language level or in internal JVM locking. A profile that is slowing a significant fraction of time in kernel locking routines indicates that system level locking that might be related to an underlying Java locking issue. Other AIX tools, such as **splat**, are helpful in diagnosing locking problems at the system level.

Always evaluate locking in the largest required scalability configuration (the largest number of cores).

Java Lock Monitor

The Java Lock Monitor (JLM) is a valuable tool to deal with concurrency and synchronization in multi-threaded applications. The JLM can provide detailed information, such as how contested every monitor in the application is, how often a particular thread acquires a particular monitor, and how often a monitor is reacquired by a thread that already owns it. The locks that are surveyed by the JLM include both application locks and locks that are used internally by the JVM, such as GC locks. These statistics can be used to make decisions about GC policies, lock reservation, and so on, to make optimal usage of processing resources. For more information about the JLM, see *Java diagnostics, IBM style, Part 3: Diagnosing synchronization and locking problems with the Lock Analyzer for Java*, found at:

<http://www.ibm.com/developerworks/library/j-ibmtools3/>

Also, see “Hot method or routine analysis” on page 241.

Thread state analysis

Multi-threaded Java applications, especially applications that are running on top of WebSphere Application Server, often have many threads that might be blocked or waiting on locks, database operations, or file system operations. A powerful analysis technique is to look at the state of the threads to diagnose performance issues.

Always evaluate thread state analysis in the largest required scalability configuration (the largest number of cores).

IBM Whole-system Analysis of Idle Time

IBM Whole-system Analysis of Idle Time (WAIT) is a lightweight tool to assess various performance issues that range from GC to lock contention to file system bottlenecks and database bottlenecks, to client delays and authentication server delays, and more, including traditional performance issues, such as identifying hot methods.

WAIT was originally developed for Java and Java Platform, Enterprise Edition workloads, but a beta version that works with C/C++ native code is also available. The WAIT diagnostic capabilities are not limited to traditional Java bottlenecks such as GC problems or hot methods. WAIT employs an expert rule system to look at how Java code communicates with the wider world to provide a high-level view of system and application bottlenecks.

WAIT is also agentless (relying on `javacores`, `ps`, `vmstat`, and similar information, all of which are subject to availability). For example, WAIT produces a report with whatever subset of data can be extracted on a machine. Getting `javacores`, `ps`, and `vmstat` data almost never requires a change to command lines, environment variables, and so on.

The output is viewed in a browser such as Firefox, Chrome, Safari, and Internet Explorer, and assuming one has a browser, no additional installation is needed to view the WAIT output. Reports are interactive, and clicking different elements reveals more information. Manuals, animated demonstrations, and sample reports are also available on the WAIT website.

For more information about WAIT, go to the following website:

<http://wait.researchlabs.ibm.com>

This website also has sample input files for WAIT, so users can try out the data analysis and visualization aspects without collecting any data.

Redbooks

Performance Optimization and Tuning Techniques for IBM Power Systems Processors Including IBM POWER8

SG24-8171-01

ISBN 0738440922



(0.5" spine)

0.475" <-> 0.873"

250 <-> 459 pages



SG24-8171-01

ISBN 0738440922

Printed in U.S.A.

Get connected

