Redbooks Paper

Amsterdam Density Functional (ADF) Benchmarking on IBM Platforms

Amsterdam Density Functional (ADF) is a quantum chemistry code based on the Density Functional Theory (DFT). It has become widely used for customer benchmarking because it applies to many different fields. In this report, we describe various aspects of installing, compiling and running this application on IBM® platforms.

Although our discussions are focused on IBM AIX® (POWER4™ and POWER3™) and PC Linux 64 (Nocona and Opteron) systems, similar procedures can certainly be applied to other platforms. Some useful tips for troubleshooting and achieving the best performance are also provided.

Author

Qiyou Jiang
HPC specialist, IBM xSeries® and HPC Benchmark Center, Poughkeepsie, NY

Introduction

Amsterdam Density Functional (ADF) is a quantum chemistry Fortran code based on the Density Functional Theory (DFT). It has been developed since the early 1970s (at that time called HFS and later AMOL) mainly by theoretical chemistry groups at Vrije Universiteit in Amsterdam and the University of Calgary, Canada. More recent development is being conducted by the theoretical chemistry group at the University of Groningen, The Netherlands. Maintenance and distribution of the commercial (export) version of the program is done by Scientific Computing & Modelling NV (SCM), a company based in Amsterdam [1].

ADF can be used in diverse fields such as molecular spectroscopy, organic and inorganic chemistry, crystallography and pharmacochemistry.

The ADF package includes the molecular ADF program, BAND (for the study of periodic systems: crystals, surfaces and polymers) and ADF-GUI (Graphical User Interface).

The ADF package is a licensed code and can be freely downloaded for testing from the SCM web site: http://www.scm.com. Before you can do that, you first need to obtain a user ID, password and temporary license by contacting support@scm.com. This package has three different versions: Serial, PVM and MPI. It has been ported to many different platforms, including IBM AIX (POWER4 and POWER3), HP Itanium2 (HP-UX and Linux), SGI Itanium2 Linux (Altix), SGI IRIX 6.5, Mac OS X, PC Linux (RedHat 9 or later), PC Linux 64 (Nocona and Opteron) and others. SCM has prepared binary packages for all these platforms and you only need to download and install the one for your platform. If you want to compile ADF yourself, you also need to download the source code package.

This report provides guidelines for first-time ADF users. We also provide some ADF performance information based on our customer benchmarking experience which IBM teams may find useful.

For detailed instructions, refer to SCF documentation in installing, compiling, running ADF. See “References” on page 9.

Building ADF on IBM AIX systems

1. Set up environment variables:

   Environment variables with typical values using C-shell syntax:

   ```
   setenv ADFHOME $HOME/adf2004.01
   setenv ADFBIN $ADFHOME/bin
   setenv ADFRESOURCES $ADFHOME/atomicdata
   setenv SCMLICENSE $ADFHOME/license
   setenv NSCM 64
   ```

   (The default number of parallel processes to use)

   ```
   setenv SCM_TMPDIR $ADFHOME/tmp
   setenv SCM_USETMPDIR yes
   ```

2. Unpack the binary distribution:

   ```
   gtar xvfz adf2004.01.ibm.mpi.tgz
   ```

3. Unpack source package:

   ```
   gtar xvfz adf2004.01j.src.tgz
   ```

4. Run configuration:

   ```
   cd $ADFHOME
   ```

Install/configure (make the following choices)

- mpi (using MPI)
- 64 (maximum number of processors for your case)
- ibm5 (AIX5 with xlf8.x compiler)

("bin/yam","bin/scu","settings" and "Makeflags" are created.)

5. To compile the program using your own settings:

   Modify Makeflags

```
mkdir tmp
$ADFBIN/yam –c –v -t
(   -v: verbose, print compile, archive and link commands
    -c: clean–remove ALL compilation information and executable
     -t: print timing info)
```

6. To run ADF:

```
$ADFBIN/adf <in >out
```

Building ADF on Linux—(Intel Xeon, EM64T) Nocona/Opteron

Download adf2004.01h.pc64_linux.mpi.bin.tgz and the source package and follow the same building procedure as on AIX. We needed to take some special precautions in running ADF on Linux®.

Notes:

(1) Never set NSCM variable in ~/.cshrc or ~/.bashrc. If you do, then every parallel slave node will get this value, decide that it is running on its own in serial mode, and never call MPI_Init. It will then try to read its input data from stdin. Modify the setting of NSCM every time for different runs with different number of CPUs.

(2) Modify lines 300 and 302 in bin/start. Use `–gm-copy-env` for passing environment variables:

```
mpirun.ch_gm SCM_WD=$SCM_WD SCM_STDIN=$SCM_STDIN –gm-copy-env –gm-eager
65536 –machinefile $SCM_MACHINEFILE –np $NSCM $PROG  "$@"
```

Troubleshooting and performance considerations

XLF8.1 vs. XLF9.1

XLF9.1 uses different naming convention for modules and module procedures: _IMOD_ for intrinsic modules, _NMOD_ for non-intrinsic modules and _mymod_NMOD_myproc for module procedure MYPROC in module MYMOD. If the pre-compiled libraries were compiled with xlf8.1 and you are going to compile/link ADF with xlf9.1, you need to add **`–qmodule=mangle81`** to specify that the compiler should use the XLF8.1 naming convention for non-intrinsic module files.

ADF uses intrinsic function INDEX() a couple of times in Install/adfparser2.f. With OPT on, XLF9.1 inlines INDEX() for performance reasons, but also causes problems due to reuse of some variables. The solution for that is either not turning on OPT or not reusing those variables by using **`–qxflag=rtdindex`**.

Tip: In Makeflags: add **–qmodule=mangle81 –qxflag=rtdindex** to FCPARS and add **–qmodule=mangle81** to FC and FC90–qxflag=rtdindex to FCPARS.

SCM_IOBUFFERSIZE

In some cases you may find that ADF is spending time waiting for I/O. A large difference between elapsed time and CPU time is an indication that this happening. The reason is typically that the I/O bandwidth of the system is insufficient for ADF. In that case, if your machine has enough memory, you can make sure that ADF keeps its files (partially) in memory by setting the environment variable SCM_IOBUFFERSIZE to a value like 512, which refers to the number of Mbytes per CPU reserved for I/O buffering. The default value, specified in $ADFHOME/settings, is rather small: 8 MB.

Keep in mind that this reduces the amount of available memory in the system. This could have an adverse performance effect on your jobs. In such cases it will be better to reduce the value of SCM_IOBUFFERSIZE.

How to interpret Buffered I/O output

The following output is what you would like ADF to print. This example is for a job run on 64 CPUs with 400 MB as SCM_IOBUFFERSIZE value.

```
Buffered I/O statistics:
Memory available:                                   419430400
Number of records fitting in memory:                101409
Input:   __ 2.6%  of                                1033  *4k bytes
Output:     14.7% of                                801   *4k bytes
Records from serial files evicted:                  0
                  Others evicted:                   0
Hash table lookups:  5090 with    0  conflicts  (0.00%)
```

The value of 400 MB returns in the output on the first line “Memory available:” Because each record uses 4 KB, this immediately determines the number of records in the second line. The % number in the 3rd and 4th lines indicates how many read and write requests resulted in a “read from” or “write to” file. Even if everything easily fit in memory, as in this case, these percentages will not be zero. The reason is that ADF has to read fragment files before it can start a calculation and it has to write at least a TAPE21. The number of evicted records in line 5 and 6 are probably the most important numbers. If these are zero, this means that all files fit in memory. In that case, no further improvements can be expected from increasing SCM_IOBUFFERSIZE.

SCM_VECTORLENGTH

Almost all programs within the ADF package use numerical integration and this is normally the most time-consuming part of the code. Numerical integration involves adding results for each “integration point.” By handling a number of points together you can greatly influence the performance. The number of integration points handled together is called the block length. If the block length is too small, you will have a significant overhead and the program will be very slow. If the block length is too large, large amounts of data will not fit in cache memory and again the program is not at optimum speed. The correct block length is somewhere in between, ranging from 32 to 4096, depending on your hardware.

Sometimes it is advantageous to set the block length explicitly NOT to a power of 2, to avoid memory bank conflicts. Try this for yourself with your typical calculation on your production machine to determine the optimal value for your situation. On most machines, 128 is a good compromise value.

Error messages

“iraloc: exceeded maximum memory” or “Program size too small”

Part of ADF uses a static array with a dimension that can be controlled by the input. The error that is printed in the output file is caused by exceeding this array. The dimension of the array can be modified in your input file using the keyword MAXMEMORYUSAGE. For example, code MAXMEMORYUSAGE 256 to use 256 MB. Memory claimed in this way is unavailable for the rest of the program through normal allocate statements, so in principle it should not be made larger than necessary. Using more processors may help reduce the amount of memory needed.

“iraloc: Out of memory”

This means that the program attempted to use more memory than the system allows, for whatever reason. First check the hard and soft limit for the system (limit for csh and ulimit for ksh). Then try to decrease the amount of memory that you allow the program to consume by setting a lower value to the input key maxmemoryusage and the environment variable SCM_IOBUFFERSIZE (set to 0 for testing purposes). If the problem remains and each task seems to be asking for more than 2GB memory, you may need to consider compiling ADF in 64-bit address mode.

“Not all files are created,” “No match atom (type) with fragfile”

Make sure there are no old files (files created in previous runs) under tmp and the working directory.

Different number of iterations for Geometry Optimization, SCF

In some cases, the jobs with different numbers of CPUs may have a different number of steps for geometry optimization and SCF. This can lead to atypical parallel scaling. The reason is that small differences in the results build up during the different geometry updates, so that the jobs start to take different paths. This is a subtle numerical issue and an unfortunate feature of ADF, but not a bug. For this kind of situation, the fairest test would be to fix the number of steps for all numbers of CPUs. If this happens during customer benchmarking, you need to explain and ask for their permission before changing anything in the input file.

32-bit vs. 64-bit

If ADF runs out of the memory limit for 32-bit address mode, you may need to prepare and run the code in 64-bit mode.

Upon our request, the code developer had specifically prepared the precompiled objects on AIX in 64-bit for us with a package – Install_ibm64.tgz.

The procedure for building the 64-bit version is just same as that for the 32-bit.

Notes:

(1) Use **`mpxlf_r`** and **`mpcc_r`** instead of mpxlf and mpcc for 64-bit.

(2) ADF uses 8-byte integers for 64-bit mode. So, link with the “wrappers” to convert 8-byte integers to 4-byte integers for ESSL and MPI.

Structure of the input

ADF has a fragment oriented approach: the poly-atomic system to be computed is conceptually built up from fragments. Obviously there must be a set of fundamental fragments that are not defined in terms of smaller fragments. Therefore, ADF has two modes of execution:

The normal mode—using fragments. A fragment file is simply the standard result file of an ADF calculation on that fragment.

The CREATE mode. In this mode, the fundamental fragments—basic atoms—are created. The basic atoms are artificial objects that represent the smallest building blocks from which any "real" calculations are started. The ADF package is equipped with a database to help you generate basic atoms. Each data file in the database contains a standard basis set and related information for the creation of one basic atom.

Input for ADF is structured by keys. It is not case sensitive. The following are the keys that are frequently used in real calculations:

BASIS	Instructs ADF to run the appropriate create runs automatically, using default values for the basis sets to use.
CREATE	Creates the basic atoms.
FRAGMENTS	Lists the fragment files.
ATOMS	Defines the atomic positions. Each record in the data block contains the chemical symbol of an atom, followed by its Cartesian coordinates in Angstroms (abbreviated Å), each measuring one ten-billionth (1/10000000000) of a meter.
GEOMETRY	Performs a geometry optimization based on a quasi Newton approach using the Hessian for computing changes in the geometry in order to make the gradients vanish. The different run types are characterized by how the geometry is manipulated: SinglePoint (SP), GeometryOptimization (GeoOpt or GO), TransitionState (TS), LinearTransit (LT), IntrinsicReactionCoordinate (IRC) and Frequencies (FREQ). If omitted, the ryb type is GeometryOptimization. If the key GEOMETRY is not used at all, the run type is SinglePoint.
SYMMETRY	Specifies the point group symmetry by a Schonfliess type symbol. If no symmetry is specified, ADF will take the true symmetry of the nuclear frame as the user-specified symmetry.
QM/MM	ADF supports the QM/MM method to handle large systems or environmental effects by treating only part of the atoms quantum-mechanically and the other ones by molecular mechanics.
XC	The Density Functional, also called the exchange-and-correlation (XC) functional, consists of an LDA and a GGA (or equivalently: GRADIENTS) part. If the XC key is not used, the program will apply only the LDA (no GGA terms). The chosen LDA form is then VWN.
RELATIVISTIC	Instructs ADF to take relativistic effects into account. By default, this is suppressed. Recommended use: Relativistic Scalar ZORA or Relativistic SpinOrbit ZORA.
SOLVATION	Turns the solvation calculation on. One can study chemistry in solution, as contrasted to the gas phase, with the implementation in ADF of the Conductor like Screening Model (COSMO) of solvation.
UNITS	Gives the units for lengths and angles.

PRINT Controls the amount of printed output.

MAXMEMORYUSAGE Specifies the total amount of memory (in MBytes.

VECTORLENGTH Specifies the maximum vector length.

For a complete list and explanation of the keys, refer to the ADF User's Guide listed in "References" on page 9. For some real examples of the input files, refer to "Appendix A" on page 10.

Benchmarking results

We present here some performance results from three ADF benchmarking experiences. The version we used is ADF2004.01. The code was compiled and run in 32-bit mode.

System configurations

The systems used were configured as shown in Table1.

Table 1 Hardware and software configuration

System	Single core p575	Dual core p575	p595	Dual core e326
Processor	Power5@1.9GHz smt off	Power5@1.5GHz smt off	Power5@1.9GHz smt off	Dual AMD Opteron 275@2.2GHz
Number of processors	8	16	32	4
Number of cores	8	16	32	4
Memory	31 GB	31 GB	118 GB	8 GB
Large Pages	16 GB	0	0	0
Kernel	64 bit	64 bit	64 bit	2.6.5-7.191-smp
Operating system	AIX 5.2	AIX 5.3	AIX 5.3	SLES 9, SP 2
POE	4.2.0.2	4.2.0.2	4.2.0.2	
Compilers	xlf 9.1.0.2 xlc 7.0.0.1	xlf 9.1.0.2 xlc 7.0.0.1	xlf 9.1.0.1 xlc 7.0.0.1	Intel® 9.0 fortran Intel 9.0 c

Case 1

Case 1 intended to make a performance comparison across the four different IBM platforms: Single core (SC) p575, Dual core (DC) p575, p595 and Dual core e326 clusters. To construct clusters with 32 processors, we used four nodes of SC p575 interconnected with HPS, two nodes of DC p575 with HPS, one node of p595, and four nodes of DC e326 with Myrinet. SMT was disabled for all nodes.

Figure 1 shows the timings and scalabilities for case 1. It seemed that the code scaled well over the number of processors and the performance was not so dependent on communication speed. See "Appendix A" on page 10 for the input file.

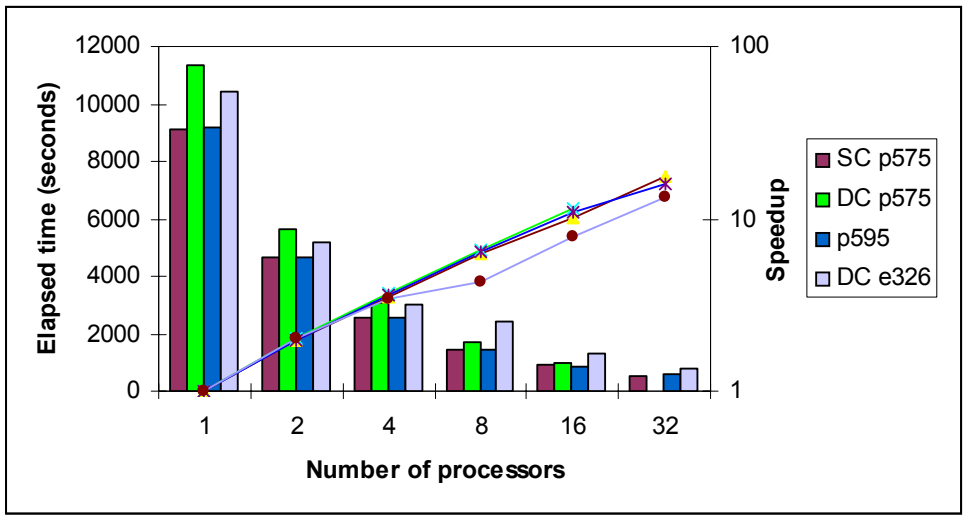

Figure 1 Timings and scalabilities for case 1

Case 2

Case 2 had two different input files: test1 and test2 for SC p575 and DC p575 clusters. Test2 provided more workloads due to more than twice the number of atoms. Figures 2 and 3 show similar performance under different workloads. For both test1 and test2, we experienced the numerical issue previously described: the number of iterations to converge for geometry optimization is different when running with a different number of processors. For example, test1 took 33 steps to converge for 8-way run, 30 steps for 16-way, 29 steps for 32-way, 48-way and 64-way. To make a fair scalability test, we set the number of iterations for geometry optimization to 28 instead of 200 in the input file (plus one final geometry for 29 total steps) for all runs. For the same reason, the number of iterations for geometry was set to 12 instead of 200 for test2. See “Appendix A” on page 10 for both input files.

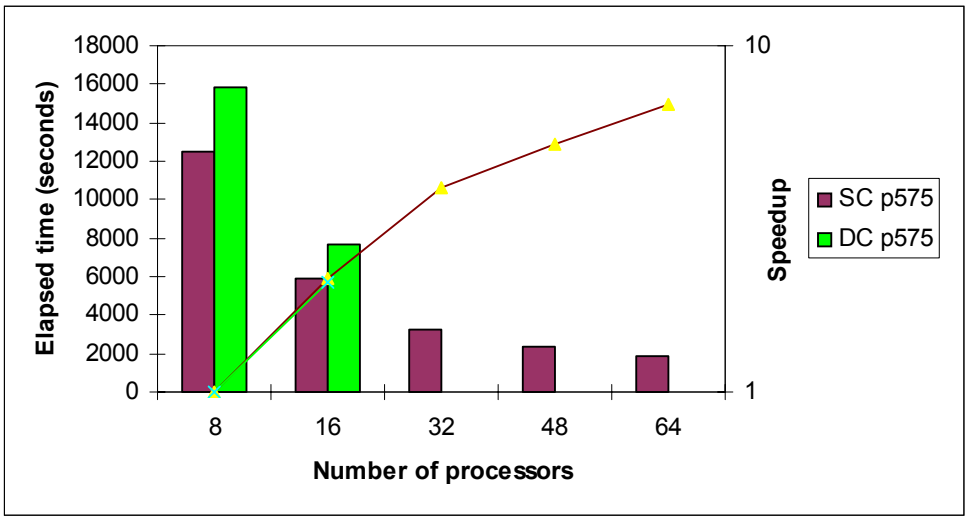

Figure 2 Timings and scalabilities for case 2 with input file –test1

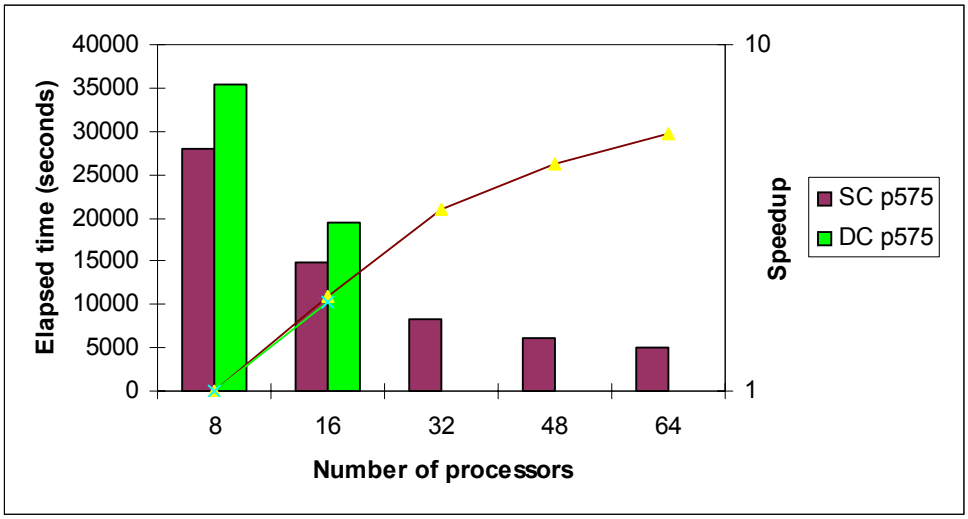

Figure 3 Timings and scalabilities for case 2 with input file–test2

Case 3

Case 3 was indeed a big job due to the combination of many atoms, heavy atoms and spinorbit (SO) option. In the input file, we added: maxmemoryusage 512. We ran the code with 8-way on one node of SC p575 with smt off. Table 2 shows how the setting of SCM_IOBUFFERSIZE can affect the performance of a heavy duty job like this case. See Appendix A for the input file.

Table 2 SCM_IOBUFFERSIZE vs. performance

SCM_IOBUFFERSIZE(MB)	8	1024	1324
Memory used/task (MB)	<1000	1600	1900
Total run time (seconds)	197091 (54.7 hrs)	69833 (19.4 hrs)	63225 (17.6 hrs)

Summary

We described the procedures to build ADF on IBM platforms (AIX and Linux) in this report. Some useful tips for troubleshooting and improved performance from our benchmarking experience were also provided. Our case studies showed that ADF can yield the best performance and excellent scalability on IBM platforms under either light or heavy workloads.

Acknowledgements

The support from SCM staff—Stan van Gisbergen, Alexei Yakovlev and Olivier Visser—our HPC colleagues, system administration team, and management is acknowledged by the author.

References

1. *ADF User's Guide*, Scientific Computing & Modelling, 2005.
2. *Installation Manual*, Scientific Computing & Modelling, 2005.

Appendix A

1. Input file for case 1

```
$ADFBIN/adf -n1 <<eor
Create C $ADFRESOURCES/DZP/C.1s
end input
eor
mv TAPE21 t21.C
rm -f TAPE*
$ADFBIN/adf -n1 <<eor
Create O $ADFRESOURCES/DZP/O.1s
end input
eor
mv TAPE21 t21.O
rm -f TAPE*
$ADFBIN/adf -n1 <<eor
Create Pt $ADFRESOURCES/TZP/Pt.4f
end input
eor
mv TAPE21 t21.Pt
rm -f TAPE*
$ADFBIN/adf -n1 <<eor
Create P $ADFRESOURCES/DZP/P.2p
end input
eor
mv TAPE21 t21.P
rm -f TAPE*
$ADFBIN/adf -n1 <<eor
Create H $ADFRESOURCES/DZP/H
end input
eor
mv TAPE21 t21.H
rm -f TAPE*
$ADFBIN/adf <<eor
title  Benchmark Pt
print timingdetail
symmetry nosym
maxmemoryusage 350
atoms
  (129 atom lines are omitted)
end
fragments
H   t21.H
C   t21.C
O   t21.O
P   t21.P
Pt  t21.Pt
end
xc
gradients becke perdew
end
STOPAFTER GEOPT
GEOMET
ITERATIONS 1
END
end input
eor
rm -f TAPE* logfile
```

2. Input file for case 2

Test1:

```
title
integration  6.0
units
length angstrom
end
symmetry  D(3)
atoms cartesian
  (55 atom lines are omitted)
end
BASIS
Type TZP
Core Small
End
geometry
iterations 200 # 200 is changed to 28.
convergence e=0.0003 grad=0.003 rad=0.003
end
scf
iterations 200
convergens 0.0000007
end
unrestricted
charge 0 5
xc
gga PW91
end
noprint frag sfo
endinput
```

Test2:

```
title
integration 5.5
maxmemoryusage 150
realmemblock 60
integermemblock 5
logicalmemblock 5
stringmemblock 7/2
vectorlength 120
units
length angstrom
end
symmetry D(2D)
atoms cartesian
  (125 atom lines are omitted)
end
BASIS
Type TZP
Core Small
END
geometry
iterations 200    #200 is change to 12
convergence e=0.0003 grad=0.003 rad=0.003
end
scf
iterations 400
convergens 0.000001
```

```
end
unrestricted
charge 1 0
xc
gga PW91
end
::METAGGA
::HFEXCHANGE
::PRINT METAGGA
::noprint frag sfo
endinput
```

3. Input file for case 3

```
TITLE
maxmemoryusage 512
UNITS
      LENGTH Angstrom
 END
 ATOMS Cartesian
  (216 atom lines are omitted)
 END
 BASIS
       S     DZ/S.2p
       K     DZ/K.3p
       Cl    DZ/Cl.2p
 END
 GEOMETRY
       SP
 END
 ALLOW BADCOREINT
 RELATIVISTIC SpinOrbit ZORA SAPA
 SCF
        ITERATIONS 100
        DIIS BFAC=0.6
 END
 XC
      GGA Becke Perdew
 END
 RESTRICTED
 CHARGE 0
 ESR
 END
 QTENS
```

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 give 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 Web sites are provided for convenience only and do not in any manner serve as an endorsement of those Web sites. The materials at those Web sites are not part of the materials for this IBM product and use of those Web sites 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.

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 illustrates 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. 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 IBM's application programming interfaces.

This document created or updated on March 24, 2006.

Send us your comments in one of the following ways:

- Use the online **Contact us** review redbook form found at:
  ibm.com/redbooks
- Send your comments in an email to:
  redbook@us.ibm.com
- Mail your comments to:
  IBM Corporation, International Technical Support Organization
  Dept. HYJ Mail Station P099
  2455 South Road
  Poughkeepsie, NY 12601-5400 U.S.A.

Trademarks

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

@server®	IBM®	Redbooks (logo) ™
AIX®	POWER3™	Redbooks™
@server®	POWER4™	xSeries®

The following terms are trademarks of other companies:

Altix® and IRIX® are trademarks of Silicon Graphics (SGI) in the United and other countries.

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

Itanium® and Xeon® are trademarks of Intel Corporation in the United States and other countries.

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

Mac OS® is a trademark of Apple Computer Inc. in the United States, other countries, or both. Opteron™ is a trademark of Advanced Micro Devices, Inc. in the United States, other countries, or both.

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