Performance Evaluation of Static and Dynamic Load Balancing Schemes for a Parallel Computational Fluid Dynamics Software (CFD) Application (FLUENT) Distributed across Clusters of Heterogeneous Symmetric Multiprocessor Systems

Introduction

Computational Fluid Dynamics (CFD) applications are “highly parallelizable” and can be distributed across a cluster of computers. However, because computation time can vary with the distributed part (mesh), the system loads are unpredictable and processors can have widely different computation speeds. Load balancing (and thus computational efficiency) across a heterogeneous cluster of processors is a significant problem. The goal here is to minimize the execution time of a single application running in parallel on such multicomputer systems. Load balancing can be achieved through static load partitioning and dynamic load balancing policies for clusters of heterogeneous multiprocessors. In static load partitioning policies, the amount of work assigned to a processor is proportional to its processing capacity and remains the same throughout the duration of the job. In the case of dynamic load balancing, the current distribution of load is assessed periodically and adjusted if the resulting distribution results in the reduction in the remaining execution time of the job. Although the static
scheduling policies do provide a good starting point, dynamic load balancing policies can further improve the efficiency of the system by taking into account the changes in the application behavior, as well as system workload. In this paper, we study the performance characteristics of both static load partitioning and dynamic load balancing policies that are available in FLUENT, a popular parallel Computational Fluid Dynamics (CFD) simulation application. The remainder of this paper is organized as follows. Parallel computing contains a brief overview of parallel computing and the IBM product offering to this marketplace. “Clusters” on page 3 discusses the definitions of speed-up and efficiency in parallel systems. “Load imbalance” on page 5 introduces the concepts of load imbalance in parallel systems. “FLUENT” on page 7 gives an overview of the FLUENT application. “Static partitioning and load distribution in FLUENT” on page 10 and “Dynamic load balancing in FLUENT” on page 11, respectively, present static load partitioning and dynamic load balancing facilities available in FLUENT. In “Design of experiments” on page 12, we describe the design of the experiments conducted in this study. In “Results” on page 13, we present the results obtained from these experiments. Finally, in “Conclusion” on page 19, we provide some conclusions.

Parallel computing

A parallel computer is a set of processors that can work collaboratively to solve a single computational problem. This definition is broad enough to include a single computer that has a few tightly coupled microprocessors (also known as Symmetric Multiprocessors or SMP), as well as systems with thousands of such computers clustered together with a high-speed communications network. In the case of SMP systems, the communication between processors is facilitated through a high-speed memory subsystem, and for clusters, the collaboration among the participating processors is carried out through an external communication subsystem. An SMP system is managed by a single copy of an operating system. In clusters, each computer is managed by a separate copy of the operating system. SMP systems can offer better scalability up to a small number of processors due to memory bandwidth constraints, while clusters can scale well into thousands of computers for large-scale problems due to the distribution of memory bandwidth among thousands of processors.¹

The importance of parallelism in obtaining high-performance computing has been recognized by many computer vendors. Most of these vendors offer both SMP-based and cluster-based parallel computers that use both commodity and proprietary microprocessors. For example, IBM offers SMP-based systems using

IBM® POWER4™ architecture, as well as the Intel® Pentium® Xeon family of microprocessors and AMD Opteron family of microprocessors.

**IBM POWER4 offering**

In 2001, IBM announced the POWER4 system, which is a high-performance microprocessor and storage subsystem using IBM’s most advanced semiconductor and packaging technology. It is the building block for the current generation IBM ® pSeries® SMP and clusters of SMP servers. The POWER4 processor chip contains two 64-bit microprocessors, a microprocessor interface controller unit, a level-2 (L2) cache, a level-3 (L3) cache directory, a fabric controller responsible for controlling the flow of data and controls on and off the chip, and chip/system pervasive functions. The clock frequency of the POWER4 processors varies from 1.0 GHz to 1.9 GHz. In certain customer situations, it is possible to have processors with different frequencies within the same cluster, but not within a single SMP.

**Clusters**

SMP-based systems can be clustered together with a high-speed interconnect such as Gigabit Ethernet or some other vendor-provided, high-speed interconnect. Thus, the computer vendors can cater to the needs of users with modest computational requirements, as well as those whose needs demand large-scale computational resources.

Multiprocessor systems and computer networks promise a practical way to speed up solving computation-intensive problems. However, such parallel computation systems pose a number of problems that need to be addressed before acceptable levels of efficiency can be achieved.

One of the more significant problems is the management of computing resources. Speed-up is an important measure of the performance of a parallel computation system. Speed-up is defined as the ratio of the time required to complete a given computation on a single processor to the equivalent computation performed on a parallel computation system.

**Processors of same speed**

Let us consider a parallel computer with N processors of the same speed. Let us define \( T_p(N) \) to be the clock time on a parallel computer with N processors, and the speed-up, \( S \), for a parallel computer system with N processors is given by:

\[
S(N) = \frac{T_p(1)}{T_p(N)}
\]  

---

---

---
Ideally, the speed-up for a computation system with N processors is N. This formula is based on the premise that all processors are homogeneous and equally loaded. The parallel efficiency, $\varepsilon$, of a parallel system is defined by:

$$\varepsilon (N) = \frac{S(N)}{N} \quad (2)$$

The speed-up is reduced from its ideal value of N (and the efficiency from the upper bound of one) due to imbalances that develop while distributing workload among the parallel processors allocated to the job. Other factors that can affect the efficiency of a parallel computation system are communication and software-related overhead and are not considered further in this investigation. In addition, not all work can be distributed, that is, parallelizable.

**Processors of different speeds**

Let us now consider the case where the processors are of different speeds, $R_1$, $R_2$, …, $R_N$, and also that the loading of the processors is proportional to their processing speeds. Then, the parallel time, $T_p(N)$, for the perfectly balanced parallel computer is computed as:

$$T_p(N) = \frac{1}{R_1 + R_2 + \ldots + R_N} \quad (3)$$

When the processors are of different speeds, the serial time, and thus the corresponding parallel speed-up, depend on the processor that is used to determine the serial time. If we define $T_{s_i}(1)$ as the serial time when processor $i$ with processing rate $R_i$ is used and $S_{p_i}(N)$ as the corresponding parallel speed-up, we have following:

$$T_{s_i} = \frac{1}{R_i} \quad (4)$$

$$S_{p_i}(N) = \frac{T_{s_i}}{1/(R_1 + R_2 + \ldots + R_N)}$$

$$= \frac{(R_1 + R_2 + \ldots + R_N)}{R_i} \quad (5)$$

Because this parallel system is perfectly balanced, the efficiency of this system is (1). However, if the loading is not proportional to the processing speeds of the processors, one or more processors can become overloaded, and the resulting completion time can exceed the perfect parallel completion time given in (3). Let:

$$W_k = \text{Percent of total workload allocated to the overloaded processor } k$$

$$R_k = \text{The processing rate for the overloaded processor } k$$

$$R_i = \text{The processing rate for the processor used in serial computation}$$

Then, the parallel completion time, $T_{p_k}(N)$, which is the same as the completion time on the overloaded processor $k$, is:

$$T_{p_k}(N) = \frac{W_k}{R_k} \quad (6)$$
The speed-up for this system is:

\[ Sp_k(N) = \frac{Ts}{TP_k(N)} = \frac{1/R_i}{(W_k/R_k)} \]  \hspace{1cm} (7)

From (5) and (7), we have the efficiency of the unbalanced system:

\[ \varepsilon_k(N) = \frac{Sp_k(N)}{Sp_i(N)} \]

\[ = \frac{((1/R_i)/(W_k/R_k))/((R_1+R_2+\ldots+R_N)/R_i)}{R_k/(W_k * (R_1+R_2+\ldots+R_N))} \]  \hspace{1cm} (8)

**Load imbalance**

Load imbalance can result in parallel systems during the start-up of computation and during the life of the computation. The imbalance that occurs during the startup phase, called here static load imbalance, is due to a distribution of workload that does not match the processing rates of the processors. The load imbalance that develops during the life of the computation, called here dynamic load imbalance, can occur because of several reasons: The computation allocated to a particular processor can increase compared to other processors or external loads might share the processor and network bandwidth.\(^3\)

**Static load imbalance**

If the computational workload is not evenly distributed, load imbalance will result, and processor idling will occur, meaning certain processors must wait for other processors to finish a particular computation. The execution time of a parallel algorithm is determined by the execution time on the processor having the largest amount of work. The following two cases illustrate imbalances in a hypothetical parallel system:

- **Case 1:** Uneven distribution of load on processors of the same speed:

<table>
<thead>
<tr>
<th>Description</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>Number of processors in the</td>
<td>2</td>
</tr>
<tr>
<td>system</td>
<td></td>
</tr>
<tr>
<td>Distribution of workload</td>
<td>60% on processor 1</td>
</tr>
<tr>
<td></td>
<td>40% on processor 2</td>
</tr>
<tr>
<td>Processing rates</td>
<td>1 for both processors</td>
</tr>
<tr>
<td>Bottleneck</td>
<td>Processor 1</td>
</tr>
<tr>
<td>Parallel time</td>
<td>0.6 (time for processor 1 to complete)</td>
</tr>
<tr>
<td>Parallel speed-up</td>
<td>1/0.6 = 1.66</td>
</tr>
<tr>
<td>Parallel time for load balanced system</td>
<td>0.5</td>
</tr>
<tr>
<td>Parallel speed-up of load balanced system</td>
<td>2</td>
</tr>
</tbody>
</table>

---

Efficiency: \( \frac{1.66}{2.0} = 83\% \) versus ideal efficiency of 100

Corrective action: Allocate 50% of workload to each of the two processors

Case 2: Even distribution of load on processors of different speeds:

- Number of processors in the system: 2
- Distribution of workload: 50% on processor 1, 50% on processor 2
- Processing rates: 2/second for processor 1, 1/second for processor 2
- Bottleneck: Processor 2
- Parallel time: 0.5 (time for processor 2 to complete)
- Parallel speed-up: \( \frac{1}{0.5} = 2 \)
- Parallel time for load balanced system: 0.33
- Parallel speed-up of load balanced system: \( \frac{1}{0.33} = 3 \)
- Efficiency: \( \frac{2}{3} = 66\% \) versus ideal efficiency of 100%
- Corrective action: Allocate 67% of the workload to processor 1 and 33% to processor 2

**Dynamic load imbalance**

Clearly, the static load balancing schemes are simpler and less overhead intensive to implement. However, sometimes it is not feasible to predict the run-time behavior of either the evolving computation or the computational environment. This behavior can sometimes invalidate the assumptions under which the initial/static load distribution was made. This can result in unbalanced load conditions and, if left unadjusted, might result in a less efficient system. Monitoring, detection, and correction of such imbalances is very complex and time consuming. Therefore, any benefit that is due to the redistribution of load needs to be weighed against the overhead that is caused by the redistribution mechanism itself.\(^4\)

FLUENT offers functionality to specify both static partitioning and dynamic load balancing schemes. After a brief introduction to the FLUENT application, we describe the static partitioning and dynamic load balancing schemes available in FLUENT and then present some experimental results using these policies.\(^5\)

---


Computational Fluid Dynamics (CFD) is the science of predicting fluid flow, heat and mass transfer, chemical reactions, and related phenomena by solving numerically the set of governing mathematical equations (conservation of mass, momentum, energy, and so forth). The results of CFD analyses are relevant in:

- Conceptual studies of new designs
- Detailed product development
- Troubleshooting
- Redesign
- CFD analysis complementing testing and experimentation.
- Reducing of the total effort required in the experiment design and data acquisition

FLUENT is a leading computer program for modeling fluid flow and heat transfer in complex geometries. FLUENT provides complete mesh flexibility, solving flow problems with unstructured meshes that can be generated about complex geometries with relative ease. Supported mesh types include 2D triangular/quadrilateral, 3D tetrahedral/hexahedral/pyramid/wedge, and mixed (hybrid) meshes. FLUENT also enables you to refine or coarsen your grid based on the flow solution, as well as offering dynamic meshes that deform and conform to geometries with components in relative motion.

The basic steps in solving a CFD problem using FLUENT are:

1. Problem identification and pre-processing:
   a. Define your modeling goals.
   b. Identify the domain you will model.
   c. Design and create the grid.

2. Solver execution:
   a. Set up the numerical model.
   b. Compute and monitor the solution.

3. Post-processing:
   a. Examine the results.
   b. Consider revisions to the model.

Of these steps, the solver execution phase is computationally intensive and is a prime candidate for productivity improvement through parallel processing. Parallel processing allows for reduced time to solution, larger problem sizes,
handling of more complicated physics and geometry of the model, and better grid resolution.

In order to solve the CFD problem in parallel, the grid generated is partitioned and distributed among a set of processes started on a given set of processors. Then during the solution phase, these processes perform iterative computation and cooperate with each other in arriving at the final solution.

For our discussion, we use the size of the grid measured as the numbers of cells to represent the amount of computation. It is important to distribute the workload among the number of processors allocated to the FLUENT job such that the job can be completed as fast as possible. Parallel FLUENT splits up the grid and data into multiple partitions and then assigns each grid partition to a different compute process. The number of partitions (or processes) is then mapped onto processors within a computer system/network. Several allocation choices exist. For example, all the FLUENT partitions can be collocated on the same node if the number of free processors available on that node is greater than or equal to the number of FLUENT processes. Another choice is to distribute the FLUENT processes among more than one node in a cluster. In this scenario, the processes that are not collocated on the same system use the external communication network to exchange information. In this paper, we use the phrases FLEUNT partition, process, task, and processor interchangeably because the mapping is one-to-one. The term node, however, has a different significance. It is used to identify a system in the network where each of these FLUENT processes is started. It is possible that more than one process can be started on the same node if the node has more than one processor.

Running parallel FLUENT jobs

The FLUENT jobs are submitted using the following syntax:

```
fluent <version> -p<comm> -t<N> -cnf=<host list> -i <journal file>
```

Where:

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>version</td>
<td>2d or 3d, meaning 2-dimensional or 3-dimensional modeling</td>
</tr>
<tr>
<td>N</td>
<td>Number of processors used to solve the FLUENT job</td>
</tr>
<tr>
<td>comm</td>
<td>The communicator used to facilitate the communication between processes</td>
</tr>
<tr>
<td></td>
<td>vmpi - when vendor specified communication library is used</td>
</tr>
<tr>
<td></td>
<td>smpi - when shared memory version of MPICH is used</td>
</tr>
<tr>
<td></td>
<td>nmpi - when distributed version of MPICH is used</td>
</tr>
<tr>
<td></td>
<td>gmpi - when Myrinet is used on LINUX based clusters</td>
</tr>
<tr>
<td>host list</td>
<td>A text file containing a list of hosts when vmpi or nmpi or gmpi is used for the communications</td>
</tr>
</tbody>
</table>

---

journal file  A text file containing the commands to drive the FLUENT program

The host list contains the nodes on which the FLUENT job will be running. A sample host list with four nodes, where each line specifies that a CPU from that node be allocated to one FLUENT process (partition), is shown here:

v60n257
v60n273
v60n321
v60n321

Because each node can be an SMP-type multiprocessor, it can accommodate more than one FLUENT process, and this is specified by replicating the host name of that particular node several times in the host list.

A journal file will contain commands to read a file containing the definition of the grid, partition the grid into N partitions using one of the many methods that are available in FLUENT, distribute these partitions to the FLUENT processes running on different nodes, specify to the solver how many time steps the simulation needs to be carried out, and finally take any snapshot that needs to be taken for post processing. Example 1 shows a listing of a sample journal file.

Example 1   Sample journal file

```plaintext
# Read the grid stored in file bench_mak.cas
/file/read-case
bench_mark.cas
# Partition the mesh using metis method
/pallel/partition/method metis 4
# Simulate for 100 time steps
/solve/iterate 100
# Print timing statistics such as average iteration time per iteration
/para/timer/print
# Write the final state of the system for later visualization purposes
/file/write-data
bench_mark_final.dat
/exit yes
```

A sample invocation of the FLUENT command is:

```
fluent 3d  -pvmpi -t4 -cnf=host.list -I fluent.jou
```

FLUENT reads the data, builds the mesh, and distributes the mesh (cells) among the FLUENT processes. A sample distribution from FLUENT run is shown in Table 1.
Table 1  Cell distribution in the partitions

<table>
<thead>
<tr>
<th>Process or partition</th>
<th>Cells</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>89 K</td>
</tr>
<tr>
<td>1</td>
<td>88 K</td>
</tr>
<tr>
<td>2</td>
<td>88 K</td>
</tr>
<tr>
<td>3</td>
<td>88 K</td>
</tr>
<tr>
<td>Total</td>
<td>353 K</td>
</tr>
</tbody>
</table>

As shown in Table 1, the distribution is very even among the four partitions (processes). After the mesh is distributed, FLUENT runs the simulation for 100 time steps and prints the time statistics. The statistic that is of interest is the average time per iteration. For example, let us assume that the time reported is 20 seconds/iteration. In an ideal parallel system, the time/iteration for the same FLUENT run made on a single CPU will be 4X20 = 80 seconds.

Static partitioning and load distribution in FLUENT

If the speeds of the processors vary significantly, equal distribution of the workload can result in uneven completion times for the processors. Consequently, it is more efficient to distribute the load such that allocation to each processor is proportional to its capacity relative to that of the other processor in the system. For example, if processor A is twice as fast as processor B, processor A should be allocated twice as much load as that assigned to processor B.

If the speeds of the processors that are used for a parallel calculation differ significantly, you can specify a load distribution for partitioning, using the load-distribution text command in the journal file:

```
/parallel/partition/set/load-distribution (17, 17, 17, 10)
```

For example, if you are allocating four POWER4 processors to a FLUENT job, and the clock frequencies for these four processors are 1.7 GHz, 1.7 GHz, 1.7 GHz, and 1.0 GHz, the above text command instructs the FLUENT system to normalize the four values entered to a total load of 1.0 and allocate to each of the first three processors workload (cells) that is 1.7 times the workload that is allocated to the last processor. If FLUENT is run on a cluster, it is important that the faster nodes on which the first three FLUENT tasks are started are entered as the first three lines in the host list. Alternatively, as described later, you could enable the load balancing feature to have FLUENT automatically attempt to discern any difference in load among the compute nodes.
Dynamic load balancing in FLUENT

Subsequent to the initial allocation, sometimes load imbalances develop while the computation is in progress due to the following:

- Variations in computational performance of processors
- Variations in network performance for clusters
- Changes in solution behavior

These changes render the initial allocation inferior and, if left unadjusted, might result in less efficient use of the resources. For example, while using four CPUs to solve a problem, you would expect to reduce the turnaround time by a factor of four. This is, of course, the ideal situation and assumes that there is very little communication needed among the CPUs, that the CPUs are all of equal speed, and that the CPUs are dedicated to your job. In practice, this is often not the case. For example, loads in individual partitions can vary due to local mesh adaptation, CPU speeds can differ if you are solving in parallel on a heterogeneous collection of workstations, other jobs might be competing for use of one or more of the CPUs, and network traffic either from within the parallel solver or generated from external sources might delay some of the necessary communication among the CPUs.

Load balancing strategies involve the adjustment of the distribution of the workload among the participating processors if the distribution is expected to result in a reduction of the total execution time. Load balancing is not inexpensive. These potential reductions due to load balancing need to be weighed against the overhead involved in monitoring the progress of the computation, as well as the redistribution of workload among the processors.

A dynamic load balancing capability is available in FLUENT. If the dynamic load balancing in FLUENT is enabled, the load across the computational and networking resources is monitored periodically. If the load balancer determines that performance can be improved by redistributing the cells among the compute nodes, it will automatically do so. There is a time penalty associated with load balancing itself, and so it is disabled by default. If you are using a dedicated homogeneous resource, or if you are using a heterogeneous resource but have accounted for differences in CPU speeds during partitioning by specifying a load distribution (as described in “Static partitioning and load distribution in FLUENT” on page 10), there is no need to use dynamic load balancing.
In order to enable dynamic load balancing in FLUENT, the following three commands are entered into the journal file (see “Running parallel FLUENT jobs” on page 8 for a description of the journal file):

(rp-var-value-set! 'parallel/enable-balance #t)
(rp-var-value-set! 'parallel/niter-balance 25)
/parallel/partition/auto/method metis

FLUENT also offers a graphical user interface (GUI) to specify these parameters. The switch <enable-balance> is used to toggle between enabling/disabling the dynamic load balancer. The variable <niter-balance> indicates the desired Balancing Interval for load balancing. If specified, FLUENT attempts to perform load balancing after every <initer-balance>. You should be careful to select an interval that is large enough to outweigh the cost of performing the load balancing operations. Alternatively, you can let the FLUENT system determine the frequency for load balancing. This approach is particularly helpful in situations where the behavior is totally unpredictable. The last command is used to select the method used to create new grid partitions. As part of the automatic load balancing procedure, the grid will be repartitioned into several small partitions using the specified method. The resulting partitions will then be distributed among the compute processes to achieve a more balanced load.

**Design of experiments**

Several experiments were conducted to evaluate the benefits of both the static load partitioning and dynamic load balancing functions currently available in FLUENT.

**Measures of performance**

Average (elapsed) time per iteration and speed-up are used to measure the performance of static partitioning and dynamic loading balancing functions available in FLUENT.

**Parameters**

We use the following parameters:

- SMP systems and clusters
  
  Both stand-alone SMP systems and clusters of SMP systems are used in these experiments. It is expected that in an SMP-type system the dynamic load balancing will incur less overhead due to faster interprocessor communications and thus contributes to better overall performance.
Number of processors and processing speeds

For both SMP systems and the clusters, the number of processors used in this study is 1, 2, and 4. In the case of the clustered system, in order to demonstrate the imbalance introduced due to variation in the speeds of the processors, processors of different clock speeds are used to run a single FLUENT job. For this study, the mix consisted of the processors with the following clock speeds: POWER4 1.7 GHz and POWER4 1.0 GHz systems.

Distribution of workload

In this study, we hypothesize that significant load imbalances in a parallel system would undermine its efficiency. In order to test this, we have introduced some load imbalance in the parallel/distributed system by instructing the system to distribute the workload according to a load distribution specified at the system startup time. Then during the load balancing phase, these induced imbalances are used to study the effectiveness of FLUENT’s dynamic load balancing scheme.

Results

We obtained the following results.

Static load balancing

To study static load balancing, both SMP and clustered environments are used. For SMP systems, a POWER4 1.7 GHz (model IBM @server pSeries 655 with single CPU/chip) system with four processors is used. The number of CPUS is varied from 1 to 4 and the load distribution from a perfectly balanced state to a completely skewed distribution. The system contained 16 GB of memory. Because the FLUENT model used in this study needed only 1 GB of memory, the results are not expected to vary much when a system with less memory is used. If the experiments done in this study on SMP systems are repeated on systems with two CPUs/chip, the results can vary. Typically, the results for systems with two CPUs/chip tend to take longer because of L2-cache contention. Table 2 lists all the configurations and load allocations that were used in the experiments where a single SMP system is used. Lines 2, 5, and 6 represent perfectly balanced loading with an equal number of cells that are distributed among the CPUs. As expected, for a configuration of a given size, the balanced allocations can result in much better parallel efficiency. For the problem solved in this experiment, the average clock time per simulation time step is 6 seconds on a pSeries 655 POWER4 technology-based system operating at 1.7 GHz (resulting in a speed-up of 3.7 and efficiency of 93%) when the processors are equally loaded. While for an unbalanced system where a disproportionately large number of the cells are allocated to some processors, the speed-up and thus the efficiency suffer. Table 2 shows that the speed-up and efficiency of the system...
drops to 1.6 and 40% when 70% of the cells are moved to one of the four processors. Even in the case where the processors are equally loaded, the efficiency of the system falls short of the ideal efficiency level of 100% (see lines 5 and 6 in Table 2). This is due to minor variation in the actual work that is performed among the processes, synchronization between processes, and parts of the code that can not be parallelized.

Table 2  Part 1: Static load partitioning

<table>
<thead>
<tr>
<th>L#</th>
<th>CPUs</th>
<th>Speeds (GHz)</th>
<th>Load distribution (%)</th>
<th>Load distribution (KCells)</th>
<th>Avg. time/iter (sec)</th>
<th>Speed-up (a)</th>
<th>Ideal speed-up (b)</th>
<th>Efficiency (%) (a)/(b)</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>1</td>
<td>1.7</td>
<td>100</td>
<td>353</td>
<td>22</td>
<td>1.0</td>
<td>1</td>
<td>100</td>
</tr>
<tr>
<td>2a</td>
<td>2</td>
<td>1.7-1.7</td>
<td>50-50</td>
<td>176-177</td>
<td>11</td>
<td>2.0</td>
<td>2</td>
<td>100</td>
</tr>
<tr>
<td>3</td>
<td>2</td>
<td>1.7-1.7</td>
<td>70-30</td>
<td>247-106</td>
<td>16</td>
<td>1.4</td>
<td>2</td>
<td>70</td>
</tr>
<tr>
<td>4</td>
<td>2</td>
<td>1.7-1.7</td>
<td>97-30</td>
<td>342-11</td>
<td>21</td>
<td>1.0</td>
<td>2</td>
<td>50</td>
</tr>
<tr>
<td>5a</td>
<td>3</td>
<td>1.7-1.7-1.7</td>
<td>34-33-33</td>
<td>121-116-116</td>
<td>8</td>
<td>2.8</td>
<td>3</td>
<td>93</td>
</tr>
<tr>
<td>6a</td>
<td>4</td>
<td>1.7-1.7-1.7-1.7</td>
<td>25-25-25-25</td>
<td>89-88-88-88</td>
<td>6</td>
<td>3.7</td>
<td>4</td>
<td>93</td>
</tr>
<tr>
<td>7</td>
<td>4</td>
<td>1.7-1.7-1.7-1.7</td>
<td>33-33-33-01</td>
<td>116-116-116-5</td>
<td>8</td>
<td>2.8</td>
<td>4</td>
<td>70</td>
</tr>
<tr>
<td>8</td>
<td>4</td>
<td>1.7-1.7-1.7-1.7</td>
<td>70-10-10-10</td>
<td>248-35-35-35</td>
<td>14</td>
<td>1.6</td>
<td>4</td>
<td>40</td>
</tr>
<tr>
<td>9</td>
<td>4</td>
<td>1.7-1.7-1.7-1.7</td>
<td>97-01-01-01</td>
<td>341-4-4-4</td>
<td>21</td>
<td>1.0</td>
<td>4</td>
<td>25</td>
</tr>
</tbody>
</table>

a. Highlighted lines represent initial distribution of workload that is proportional to the speeds of the processors.

For clusters, two configurations are used. First, a cluster of four POWER4 1.7 GHz systems each containing four processors is used. All the configurations and workload allocations used for this cluster are shown in Table 3. Only one processor from each system is used. Next, a cluster of three POWER4 1.7 GHz nodes and one POWER4 1.0 GHz node are used. All the configurations and workload allocations used for this cluster are shown in Table 4. Again, for this cluster, only one processor from each system is used.
Table 3  Part 2: Static load partitioning

<table>
<thead>
<tr>
<th>L#</th>
<th>CPUs</th>
<th>Speeds (GHz)</th>
<th>Load distribution (%)</th>
<th>Load distribution (KCells)</th>
<th>Avg. time/iter (sec)</th>
<th>Speed-up (a)</th>
<th>Ideal speed-up (b)</th>
<th>Efficiency (%) (a)/(b)</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>1</td>
<td>1.7</td>
<td>100</td>
<td>353</td>
<td>22</td>
<td>1.0</td>
<td>1</td>
<td>100</td>
</tr>
<tr>
<td>2a</td>
<td>2</td>
<td>1.7-1.7</td>
<td>50-50</td>
<td>176-177</td>
<td>12</td>
<td>1.8</td>
<td>2</td>
<td>90</td>
</tr>
<tr>
<td>3</td>
<td>2</td>
<td>1.7-1.7</td>
<td>70-30</td>
<td>247-106</td>
<td>17</td>
<td>1.3</td>
<td>2</td>
<td>65</td>
</tr>
<tr>
<td>4</td>
<td>2</td>
<td>1.7-1.7</td>
<td>97-03</td>
<td>342-11</td>
<td>22</td>
<td>1.0</td>
<td>2</td>
<td>50</td>
</tr>
<tr>
<td>5a</td>
<td>3</td>
<td>1.7-1.7-1.7</td>
<td>34-33-33</td>
<td>121-116-116</td>
<td>12</td>
<td>1.8</td>
<td>3</td>
<td>90</td>
</tr>
<tr>
<td>6a</td>
<td>4</td>
<td>1.7-1.7-1.7-1.7</td>
<td>25-25-25-25</td>
<td>89-88-88-88</td>
<td>9</td>
<td>2.4</td>
<td>4</td>
<td>60</td>
</tr>
<tr>
<td>7</td>
<td>4</td>
<td>1.7-1.7-1.7-1.7</td>
<td>33-33-33-01</td>
<td>116-116-116-5</td>
<td>11</td>
<td>2.0</td>
<td>4</td>
<td>50</td>
</tr>
<tr>
<td>8</td>
<td>4</td>
<td>1.7-1.7-1.7-1.7</td>
<td>70-10-10-10</td>
<td>248-35-35-35</td>
<td>15</td>
<td>1.5</td>
<td>4</td>
<td>38</td>
</tr>
<tr>
<td>9</td>
<td>4</td>
<td>1.7-1.7-1.7-1.7</td>
<td>97-01-01-01</td>
<td>341-4-4-4</td>
<td>21</td>
<td>1.0</td>
<td>4</td>
<td>25</td>
</tr>
</tbody>
</table>

a. Highlighted lines represent initial distribution of workload that is proportional to the speeds of the processors.

Referring to Table 3, lines 2, 5, and 6 represent a perfectly balanced loading with an equal number of cells distributed among the processors using static load balancing. As expected, for a configuration of a given size, the balanced allocations resulted in the best parallel efficiency. For example, in the case of the four-processor configuration, the average clock time per simulation time step is measured at 9 seconds (resulting in a speed-up of 2.4 and efficiency of 60%). While for an unbalanced system where a disproportionately large number of the cells are allocated to some processors, the speed-up and thus the efficiency suffer. For example, line 8 in Table 3 shows that the speed-up and efficiency of the system drop to 1.5 and 38%, respectively, when 70% of the cells are moved to one of the four processors. As expected, the performance of the clustered system, presented in Table 3, trails that of an SMP system presented in Table 2 for small configurations used in this study (for example, see line 2 in Table 2 and line 2 in Table 3). This is because the processes that are collocated in the same node communicate with each other at a higher speed using shared memory than those that reside on different nodes and use slower communication networks for communication.

Referring to Table 4, lines 3, 6, and 8 represent a perfectly balanced loading where the distribution of cells among the CPUs is proportional to their processing capacity. That means faster processors are allocated more cells. As expected, for a configuration of a given size, the perfectly balanced allocations result in the best parallel efficiency. For example, line 8 in Table 4 shows that for a perfectly
balanced four-processor system, the average clock time per simulation time step is 10 seconds (speed-up of 2.2 and efficiency of 55%). While for an unbalanced system where a disproportionately large number of the cells are allocated to some processors, the speed-up and thus the efficiency suffer. For example, line 10 in Table 4 shows that the speed-up (or slowdown) and efficiency of such an allocation of the system are 0.8 and 20%, respectively, when 70% of the cells are moved to one of the four processors. Because the cluster used for the results presented Table 4 contains one slower (POWER4 1.0 GHz) processor, the performance of this cluster trails the one shown in Table 3.

**Table 4 Part 3: Static load partitioning**

<table>
<thead>
<tr>
<th>L#</th>
<th>CPUs</th>
<th>Speeds (GHz)</th>
<th>Load Distribution (%)</th>
<th>Load Distribution (KCells)</th>
<th>Avg. time/iter (sec)</th>
<th>Speed-upa</th>
<th>Ideal speed-upa</th>
<th>Efficiency (%)</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>1</td>
<td>1.7</td>
<td>100</td>
<td>353</td>
<td>22</td>
<td>1.0</td>
<td>1</td>
<td>100</td>
</tr>
<tr>
<td>2</td>
<td>2</td>
<td>1.7-1.0</td>
<td>50-50</td>
<td>176-177</td>
<td>19</td>
<td>1.2</td>
<td>1.6</td>
<td>75</td>
</tr>
<tr>
<td>3b</td>
<td>2</td>
<td>1.7-1.0</td>
<td>63-37</td>
<td>222-131</td>
<td>15</td>
<td>1.5</td>
<td>1.6</td>
<td>94</td>
</tr>
<tr>
<td>4</td>
<td>2</td>
<td>1.7-1.0</td>
<td>03-97</td>
<td>11-342</td>
<td>37</td>
<td>0.6</td>
<td>1.6</td>
<td>38</td>
</tr>
<tr>
<td>5</td>
<td>3</td>
<td>1.7-1.0-1.7</td>
<td>34-33-33</td>
<td>121-116-116</td>
<td>17</td>
<td>1.3</td>
<td>2.6</td>
<td>50</td>
</tr>
<tr>
<td>6b</td>
<td>3</td>
<td>1.7-1.0-1.7</td>
<td>39-22-39</td>
<td>138-77-138</td>
<td>11</td>
<td>2.0</td>
<td>2.6</td>
<td>77</td>
</tr>
<tr>
<td>7</td>
<td>4</td>
<td>1.7-1.0-1.7-1.7</td>
<td>25-25-25-25</td>
<td>89-88-88-88</td>
<td>11</td>
<td>2.0</td>
<td>3.6</td>
<td>56</td>
</tr>
<tr>
<td>8b</td>
<td>4</td>
<td>1.7-1.0-1.7-1.7</td>
<td>28-16-28-28</td>
<td>99-56-99-99</td>
<td>10</td>
<td>2.2</td>
<td>3.6</td>
<td>61</td>
</tr>
<tr>
<td>9</td>
<td>4</td>
<td>1.7-1.0-1.7-1.7</td>
<td>01-97-01-01</td>
<td>4-341-4-4</td>
<td>38</td>
<td>0.6</td>
<td>3.6</td>
<td>17</td>
</tr>
<tr>
<td>10</td>
<td>4</td>
<td>1.7-1.0-1.7-1.7</td>
<td>70-10-10-10</td>
<td>248-35-35-35</td>
<td>14</td>
<td>1.6</td>
<td>3.6</td>
<td>44</td>
</tr>
</tbody>
</table>

a. Serial time for POWER4 1.7 GHz system is used to compute speed-up.
b. Highlighted lines represent initial distribution of workload that is proportional to the speeds of the processors.

Using parallel systems containing both homogeneous and heterogeneous processors, we have demonstrated that the speed-up and efficiency of the system can be improved for real applications such as FLUENT by an allocation that is proportional to the speeds of the processors. FLUENT provides the capability to allocate different workloads to different processors. In situations where the system environment does not change significantly during the life of the computation, the static workload allocation schemes offer an excellent opportunity to improve the efficiency of the system.
**Dynamic load balancing**
To study dynamic load balancing, we used the same configurations and initial loading patterns that were used to evaluate the static partitioning schemes (see columns (a) and (b) in Table 5). The grid is partitioned across the four processors using the specification shown in column (c) in Table 5. The highlighted rows represent the partitioning that matches the processing speeds of the CPUs. At this point, the automatic load balancing is turned off, and the system is allowed to progress for 60 iterations. In this study, two measures of performance were observed: elapsed time and average time per iteration (see columns (d) and (e) in Table 5). Then, each of the test configurations is restarted with the same initial load, and the simulation is run with the load balancing option turned on. The resulting final distribution of cells, elapsed time, and time/iteration are shown in columns (f), (g), and (h) in Table 5. The improvement or slowdown due to the load balancing scheme compared to not invoking the load balancing is shown in column (c) of Table 5.

Lines 2, 5, and 9 in Table 5 represent systems with extreme imbalance. These scenarios are used to illustrate the benefits of using load balancing schemes. For example, referring to line 5 in Table 5, where initially most of the load (97%) is on the first CPU, the elapsed time and time per iteration are reduced from 1571 seconds and 21 seconds, respectively, to 890 seconds and 10 seconds, respectively, resulting in a two-fold improvement. Also, the final distribution of the workload is more balanced and very closely matches the relative processing speeds of the four CPUs used in this configuration. In real-life situations, such imbalances are rare, and the actual benefits might be more limited.
An efficient load balancing scheme should not impose excessive burden on the system. In order to assess this, using a system with a balanced initial load distribution, the performance of the FLUENT application is measured with and without the load balancing feature turned on. For example, referring to line 1 in Table 5, the elapsed time and time per iteration have increased from 467 seconds and 6 seconds, respectively, when no load balancing is done, to 515 seconds and 7 seconds, respectively, when load balancing is used. That amounts to about a 10% penalty. This penalty seems to be more in the case of clustered systems (see line 4 in Table 5). In order to benefit from load balancing schemes, the level of imbalance should be severe enough to offset any overhead due to the load balancing scheme.

Finally, dynamic load balancing schemes have to contend with an undesirable phenomenon called “oscillation” or “thrashing” that occurs when the data used to make the decisions to shift the grid cells between partitions is either inaccurate or outdated, or the load balancing scheme is too sensitive to temporary fluctuations in system/application loads. Based on testing done so far, the load balancing function in FLUENT seems to minimize the movement of cells between partitions after the system has reached a balanced state.
Conclusion

In this study, we explained the adverse impact of load imbalances on the speed-up and efficiency of parallel systems. For situations where the hardware and network parameters, such as speed and bandwidth and application behavior, are known a priori and do not change, static load partitioning schemes are preferred due to their simplicity and lack of run-time overhead.

Otherwise, during the life of the computation, if the loads on the processors and application behavior tend to shift dramatically from the values assumed during the application invocation, run-time monitoring and redistribution of workload among the processors can improve the efficiency of the system. However, the level of benefits that can be derived are reduced due to overhead associated with monitoring, redistribution, and potential disturbances induced by the rebalancing algorithm itself. Such overhead can vary widely with different hardware networks and external load conditions. We have shown that the dynamic load balancing function in FLUENT is very effective in detecting any load imbalances that are introduced and restoring the system to a stable and balanced state. Future investigations will focus on the impact of various types and capacities of network, processor architectures, and external load conditions on the performance of dynamic load balancing schemes in high-performance applications. In some situations, simple static partitioning of work to match the processing speeds of processors might not result in better performance due to complex interactions between the application, processors, and networks. In such situations, it might be useful to use dynamic load balancing in early studies for a given problem type and hardware configuration and use the information about the resulting load distribution in static scheduling.

Notes on benchmarks and values

The benchmarks and values shown here were derived using particular, well-configured, development-level computer systems. Unless otherwise indicated for a system, the values were derived using 64-bit applications and external cache, if external cache is supported on the system. All benchmark values are provided “AS IS" and no warranties or guarantees are expressed or implied by IBM. Actual system performance may vary and is dependent upon many factors, including system hardware configuration software design and configuration. Buyers should consult other sources of information to evaluate the performance of systems they are considering buying and should consider conducting application oriented testing.

Unless otherwise indicated for a system, the performance benchmarks were conducted using AIX® 5L™ Version 5.1, IBM C Set++ for AIX/6000 Version 6.1.0.3, and AIX XL FORTRAN Version 7.1.0.3 with optimization where the compilers were used in the benchmark tests.
The author that wrote this Redpaper

Hari Reddy
High Performance Computing Solutions Development
Systems and Technology Group, IBM
6609 Carriage Drive
Colleyville, TX 76034

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

We are grateful to Fluent, Inc. for their permission to use FLUENT 6.1 software and to publish the results obtained in this study. We are thankful to Prasad Alavilli at Fluent, Inc. for his help during the course of this investigation and providing valuable critique of the paper.

We would like to thank Joel Tendler for his numerous valuable suggestions, which resulted in significant enhancements to the paper. We thank Elisabeth Stahl for proofing the paper and providing several suggestions that improved the readability of this paper. We thank Bruce Hurley, Matthew Cali, and the Solutions Enablement Management team at IBM for their support for this study. We are also grateful for the ample hardware resources made available by the Poughkeepsie pSeries Benchmark Center.
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.

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 measurement 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 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.

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. JN9B Building 905 Internal Mail Drop 9053D005, 11501 Burnet Road
   Austin, Texas 78758-3493 U.S.A.

Trademarks

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

- AIX 5L™
- AIX®
- @server®
- ibm.com®
- IBM®
- POWER4™
- pSeries®
- Redbooks (logo) ™
- Redbooks™

The following terms are trademarks of other companies:

Intel, Intel Inside (logos), MMX, and Pentium are trademarks of Intel Corporation in the United States, other
countries, or both.

FLEUNT, FLUENT 6.1, and Fluent Inc. are registered trademarks of Fluent, Inc.

AMD and Opteron are trademarks or registered trademarks of Advanced Micro Devices, Inc.

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