Performance Analysis of a Large-Scale Cosmology Application on Three Cluster Systems [*]

Zhiling Lan and Prathibha Deshikachar
Department of Computer Science
Illinois Institute of Technology
{lan, deshpra}@iit.edu

Abstract

A typical cosmological simulation requires a large amount of compute power, which is hard to satisfy with a single machine. Cluster systems provide the opportunity to execute such large-scale applications. In this paper, we investigate and analyze the performance of a large-scale production cosmology application, the ENZO code, on different cluster environments. Three cluster systems, each of them representing a widely-used cluster environment in the area of scientific computing, are used in this work: an IBM SP2 system at SDSC, an IA-64 Linux cluster at NCSA, and a SUN Cluster at IIT. The performance is evaluated from three aspects: overall performance, communication characteristics, and load balancing characteristics. The experimental data shows that the cosmology performance on these clusters depends on the system performance and the application characteristics. The application performance on these clusters does not totally match the NPB measurement. Further, it seems that the IA-64 Linux cluster does not scale past 32 *CPUs for this application.*

1. Introduction

With the availability of powerful microprocessors and high-speed networks as commodity components, the cluster environment has rapidly emerged as a major platform for scientific applications. The modeling of cosmological phenomena entails simulation of the formation and evolution of cosmic structures such as galaxies and clusters of galaxies from shortly after the big bang to the present day. A typical cosmological simulation requires a large amount of compute power, which is hard to satisfy with a single machine. Cluster systems provide the opportunity to execute such large-scale cosmology applications that require vast compute power. In this work, we investigate and analyze a large-scale production cosmology application, the ENZO code, on different cluster systems. The purpose of this paper is to demonstrate the great potential of using cluster environment for large-scale applications, such as cosmology codes.

The cosmology application, ENZO, was developed by Greg Bryan and Michael Norman[4] in the early 1990s. ENZO is currently in use at seven sites. ENZO is based on the Structured Adaptive Mesh Refinement (SAMR) algorithm provided by M. Berger et al.[2] in the 1980s. Basically, SAMR is a type of multiscale algorithm that achieves high spatial resolution in localized regions of dynamic and multidimensional numerical simulations. ENZO is one of the successful implementations of the SAMR algorithm in astrophysics and cosmology.

Two ENZO datasets are analyzed in this work: *AMR64* and *ShockPool3D*. They are chosen due to their different adaptive characteristics as shown in [6]. Three cluster systems are evaluated in this work: the IBM SP2 system *Blue Horizon* at the San Diego Supercomputing Center (SDSC), the IA-64 Linux cluster *Titan* at the National Center for Supercomputing Applications (NCSA), and the SUN cluster *Sunwulf* at the Scalable Computing Software Laboratory of Illinois Institute of Technology. We choose these systems due to their wide usage in the area of scientific computing and their representative configurations in hardware and software.

To evaluate and analyze the performance of ENZO on these clusters, we instrument the ENZO code with performance counters and timers. The performance data is collected and analyzed from three aspects: overall performance, communication characteristics, and load balancing characteristics. Further, NPB2.4 [10] and NetPIPE[9, 11] are used to measure the overall and message-passing performance of these systems. These benchmarking results are

[*] Zhiling Lan is supported by a grant from the National Computational Science Alliance (ACI-9619019) and ERIF grant from Illinois Institute of Technology.

compared with the ENZO performance on the cluster systems.

Our experimental data shows that the cosmology performance on these clusters depends on the system performance and the application characteristics. For example, for most cases, the best performance of the cosmology application is achieved on the IBM SP2 system which is the fastest system according to NPB benchmark; however, there are some cases in which the best cosmology performance is obtained on the IA-64 Linux cluster which is a slower system as compared to the IBM SP2. It is also shown that the relative performance according to the NPB measurement does not exactly match the relative cosmology performance on these clusters. Further, we notice that the underlying interconnected network of *Titan* has scalability problems. Specifically, it is not efficient for small-sized messages when there are more than 32 processes.

The remainder of this paper is organized as follows. Section 2 provides the background information, which includes a description of the SAMR algorithm and the cosmology code ENZO. Section 3 describes three cluster systems used in this work and compares their overall and message-passing performance by using NPB2.4 and NetPIPE respectively. Section 4 presents and analyzes the performance results. Section 5 summarizes the paper and discusses our future work with this project.

2. Cosmology Application

2.1. Structured Adaptive Mesh Refinement Algorithm

Many numerical simulations of multiscale physical phenomena require enormous compute resources, in both memory storage and computing time, because their domains are discretized into high-resolution meshes. However, these resources are often under utilized on subdomains where high resolution is not required. Structured Adaptive Mesh Refinement (SAMR), developed by Marsha Berger et al. in the early 1980's [2], is a class of adaptive strategies that address this problem by performing high spatial resolution only in those required regions. SAMR employs a nested hierarchy of overlapping grids of increasingly fine resolution (in both space and time) permitting high resolution computation in some areas and low resolution in others. The underlying premise of this strategy is that all grids of any given resolution are equivalent in the sense that, given proper boundary information, they can be solved independently by identical means[7].

SAMR represents the grid hierarchy as a tree of grids at any instant of time. The number of levels, the number of grids, and the locations of the grids change with each adaptation. Initially, a uniform mesh covers the entire computational volume, and in regions that require higher resolution, a fine subgrid is added. If the region needs more resolution, an even finer subgrid can be added. This process repeats recursively with each adaptation resulting in a tree of grids like that shown in Figure 1. The top graph in this figure shows the overall structure after several adaptations. The remainder of the figure shows the grid hierarchy for the overall structure with the dotted region identifying the regions requiring further refinement. In this grid hierarchy, there are four levels of grids from level 0 to level 3, and the refinement factor is 2.0.

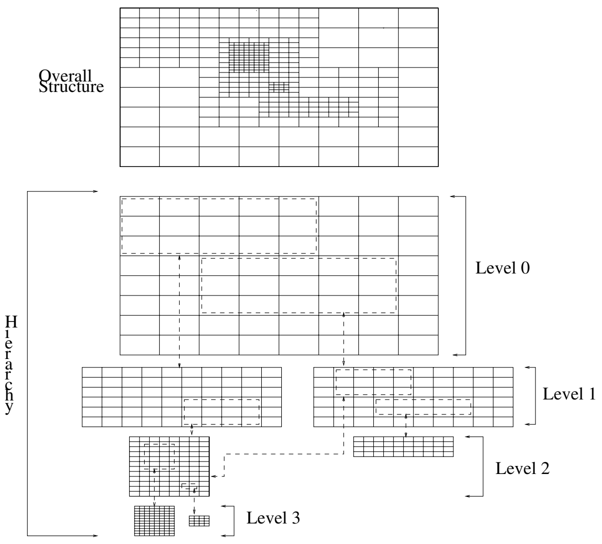

Figure 1. SAMR Grid Hierarchy

2.2. Cosmology Code ENZO

ENZO [3] is one of the successful, parallel implementations of SAMR; ENZO is primarily intended for use in astrophysics and cosmology. This application entails the detailed computations that simulate the formation and evolution of cosmic structures such as galaxies and clusters of galaxies from shortly after the big bang to the present day. Such modeling offers the only practical means to test theory against observations and to rule out incorrect hypotheses. ENZO includes solving the coupled equations of gas dynamics, collisionless dark matter dynamics, self-gravity, and cosmic expansion in three dimensions and at high spatial resolution. The code is written in C++ with FORTRAN routines for computationally-intensive sections and MPI functions for message passing among processors.

The ENZO implementation manages the grid hierarchy globally; that is, each processor stores the grid information of all other processors. In order to save space and reduce communication time, the notation of "real" grid and "fake" grid is used for sharing grid information among processors. Each subgrid in the grid hierarchy resides on one processor and this processor holds the "real" subgrid. All other processors have replicates of this "real" subgrid, called "fake" grids. Usually, the "fake" grid contains the information such as dimensional size of the "real" grid and the processor where the "real" grid resides. The data associated with a "fake" grid is small (usually a few hundred bytes), while the amount of data associated with a "real" grid is large (ranging from several hundred kilobytes to dozens of megabytes).

3. Three Cluster Systems

Three cluster systems are used in this work: the IBM SP2 system *Blue Horizon* at the San Diego Supercomputing Center (SDSC), the IA-64 Linux cluster *Titan* at the National Center for Supercomputing Applications (NCSA), and the SUN cluster *Sunwulf* at the Scalable Computing Software Laboratory of Illinois Institute of Technology. They are chosen due to two reasons. First of all, these systems are widely used in the area of scientific computing. Secondly, they represent different cluster environments. For example, the IBM SP2 system represents a relatively tightly-coupled cluster environment and the underlying hardware and software are optimized for this architecture. *Titan* represents a Linux-based cluster with Myrinet interconnect while *Sunwulf* represents an NOW (Network of Workstations) with fast Ethernet connection.

We have just recently been able to execute ENZO on an IA-32 system, but do not have enough time to fully test the performance of ENZO on it and include the results in this paper. We will include our experimental data on the IA-32 system in our future work.

3.1. IBM SP2 System at SDSC

The first system is an IBM SP2 system called *Blue Horizon* which is located at the San Diego Supercomputing Center (SDSC). With a theoretical peak performance of 1.7 teraflops, *Blue Horizon* is ranked $62nd$ on the current TOP 500 list (June 2003).

Blue Horizon[8] is a teraflop-scale Power3-based clustered SMP system from IBM. The machine contains 1,152 processors and 576 GBytes of main memory, arranged as 144 Symmetric Multiprocessing (SMP) compute nodes. Nodes are connected by the Colony switch, a proprietary IBM interconnect. The application processors run at 375 MHz and are capable of a peak performance of 1.5 GFLOPS. Each Power3 CPU has an L1 (64 KB) cache which is 128-way set associative and L2 (8 MB) cache which is four-way set associative with its own private cache bus.

The operating system on *Blue Horizon* is AIX 4.3.3, which is IBM's proprietary 64-bit version of the Unix OS. IBM's Parallel Operating Environment (POE) (Version 3 Release 2) provides the software that allows users to develop applications which utilize the machine hardware and operating system as a shared resource for effective parallel computing. POE also includes numerical libraries optimized for the *Blue Horizon* architecture.

3.2. IA-64 Linux Cluster at NCSA

The second system is an IA-64 Linux-based cluster called *Titan* which is located at the National Center for Supercomputing Applications (NCSA). It is ranked $111st$ on the current TOP 500 list (June 2003), showing peak performance of 1.024 teraflops and sustained performance of 677.9 gigaflops.

Titan[1] is comprised of 160 IBM IntelliStation Z Pro 6894 servers, each with two 800MHz Intel 64-bit Itanium processors, running Red Hat Linux and Myricom's Myrinet cluster interconnect network. Each node is equipped with 2 GBytes of ECC SDRAM memory shared among its two Itanium processors. Each Itanium CPU has three levels of cache to reduce memory latency: 4 Mbytes L3 cache, 96 Kbytes L2 cache, and 32 Kbytes L1 cache.

The operating system on *Titan* is Linux 2.4.16(Red Hat 7.1). The machine is installed with MPICH 1.2.1 implementation of the MPI standard for message passing libraries and VMI (Virtual Machine Interface) that supports multiple underlying communication devices in a cluster environment. The layering of MPI on top of VMI is accomplished by means of a *ch_vmi* MPI device running on the MPICH distribution. The other libraries required are the *dl* library for dynamic loading, and the *pthreads* library for allowing asynchronous activities within VMI or the various devices to be performed in separate threads of activity.

3.3. SUN Cluster at IIT

The third system is a SUN UNIX-based cluster called *Sunwulf* which is located at the Scalable Computing Software Laboratory of Illinois Institute of Technology.

Sunwulf is composed of a four-processor E450 file server and 63 high-end workstations, with total of 67 CPUs. The Ultra Enterprise 450 server is designed around SUN high-speed Ultra Port Architecture (UPA) cross-bar system interconnect and four modular UltraSPARC-II 480MHz CPUs with clock frequency of 96 Mhz. Each of the 63 high-end workstations is a SUN Blade workstation 100 with one UltraSparc-IIe 500MHz CPU. Each of them comes with

256K L2 cache and 128MB main memory. The underlying interconnect is fast Ethernet.

This system has been loaded with SUNOS 5.8 operating system. The machine is installed with SUN HPC ClusterTools 4.0 which includes a high-performance, thread-safe, and multi-protocol implementation of the Message Passing Interface (MPI).

3.4. NPB Comparison of Three Clusters

Different hardware and software configurations have been used in these cluster systems, which results in different system performance. Therefore, before running ENZO on these clusters, we use NAS Parallel Benchmark (NPB2.4)[10] to calculate the relative performance of these machines. NPB2.4, written and distributed by NAS, includes a set of eight MPI-based source-code programs designed to evaluate the performance of parallel systems. It has gained wide acceptance as a standard indicator for supercomputer performance. The relative performance of these machines is shown in the Figure 2. Here, we use class B with 16 processes for each run. Due to some bug with the BT benchmark, we cannot collect BT performance for *Titan* and *Sunwulf* system.

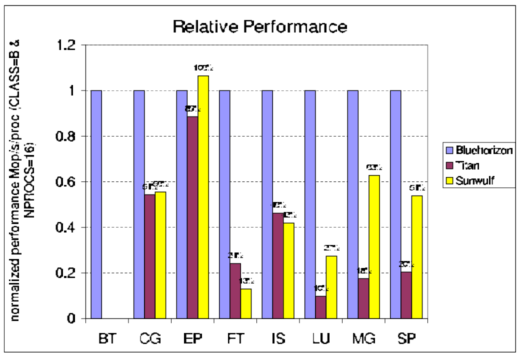

Figure 2. Comparison of Three Systems Using NPB2.4

The performance is measured in terms of *Mop/s/proc* metric and the results are normalized to those of the IBM SP2 system *Blue Horizon*. It is shown that the relative performance of the Linux cluster *Titan* is in the range of $[0.10, 0.89]$ and the relative performance of the SUN cluster *Sunwulf* is in the range of $[0.13, 1.06]$. By using the arithmetic average, we calculate the relative performance of *Blue Horizon*, *Titan* and *Sunwulf* as 1.0, 0.37, and 0.52 respectively.

3.5. NetPIPE Comparison of Three Clusters

In a cluster environment, the inter-processor communication rate is crucial to its overall performance. Therefore, we decide to use the tool *NetPIPE* (Network Protocol Independent Performance Evaluator)[9, 11] to quantify point-to-point latency, bandwidth, and cross-sectional bandwidth of these cluster systems. *NetPIPE* performs simple ping-pong tests, bouncing messages of increasing size between two processes either across a network or within an SMP system.

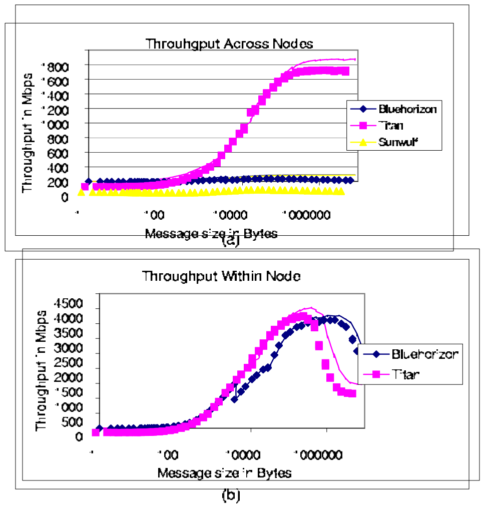

Figure 3. Comparison of Three Systems Using NetPIPE

Figure 3(a) compares cross-sectional throughput of these systems. As we can see, *Bluehorizon* and *Sunwulf* deliver similar message-passing performance. They provide a throughput up to 87 and 84 Mbps respectively. *Titan* can provide a higher throughput, especially when the message size is larger than 400 Bytes. The maximum throughput on *Titan* is about 1681 Mbps.

While both *Bluehorizon* and *Titan* are clustered SMP systems, so their throughput within node is illustrated in Figure 3(b). It is shown that both machines deliver similar throughput when the message size is smaller than 4099 Bytes. When the message size is between 4Kbytes and 0.7Mbytes, *Titan* provides higher performance. Once the message size is larger than 0.7Mbytes, *Bluehorizon* can deliver higher throughput as compared to *Titan*. The maximum throughput is 3776 Mbps and 4043 Mbps respectively for *Bluehorizon* and *Titan*.

Latencies for small messages (smaller than 64 Bytes) are 93ms, 17ms, and 210 ms respectively for *Bluehorizon*, *Ti-*

Dataset	Initial Size	Final Size	# of Adaptations
`AMR64`	$32 \times 32 \times 32$	$4096 \times 4096 \times 4096$	2500
`ShockPool3D`	$50 \times 50 \times 50$	$6000 \times 6000 \times 6000$	600

Table 1. ENZO Datasets

tan, and *Sunwulf*. On *Bluehorizon*, latency within node is reduced to about 7.7ms, while it is about 6.8ms on *Titan*.

NetPIPE results show that in terms of interconnects across nodes, *Sunwulf* has the slowest interconnection. Although the overall message-passing performance across nodes on *Bluehorizon* is slower than that on *Titan*, *Bluehorizon* is a SMP with eight processors per node while *Titan* consists of multiple dual-processor nodes, so we cannot simply say that *Titan* delivers better message-passing performance than *Bluehorizon*.

4. Experimental Data

To evaluate and analyze the performance of ENZO on different clusters, the ENZO code was instrumented with performance counters and timers. The instrumentation data was aggregated during runtime, requiring only one I/O operation to write the results to a file at the end of execution.

Two ENZO datasets (*AMR64 and ShockPool3D*) are used in the experiment. These two datasets are chosen because they have quite different adaptive characteristics, which can be found in [6]. *AMR64* is designed to simulate the formation of a cluster of galaxies. This run tends to create lots of grids randomly distributed across the computation domain. *ShockPool3D* simulates the movement of a plane shock wave which is slightly tilted with respect to the edges of the computational domain. When the shock wave first starts, it enters from one corner of the computational domain; as it progresses, more and more of the shock wave enters the region until at some point the plane spans from one edge to the other. The *ShockPool3D* run creates more and more grids along the moving shock wave plane. *ShockPool3D* solves a purely hyperbolic equation, while *AMR64* uses hyperbolic (fluid) equation and elliptic (Poisson's) equation as well as a set of ordinary differential equations for the particle trajectories. Table 1 shows the problem sizes for both datasets.

In this paper, the performance of ENZO is analyzed by the following features: overall performance, communication characteristics, and load balancing characteristics.

We found it problematic to collect performance data of *AMR64* on *Sunwulf* because more than available memory is trying to be allocated by the application or an unavailable memory segment is being accessed. Therefore, we only present the performance data of *ShockPool3D* on *Sunwulf* in the following subsections.

4.1. Overall Performance

Figure 4 shows the total execution time for *ShockPool3D* and *AMR64* with varying number of processors on three clusters. Each execution time is divided into two parts: *computation* time and *communication* time. First of all, for both datasets, the performance of ENZO on the *Sunwulf* cluster is much worse than on the other two clusters. On *Blue Horizon* and *Titan*, values of the total execution time on the y-axis ranges from 0 to 12,000 seconds, while they range from 0 to 60,000 seconds on *Sunwulf*.

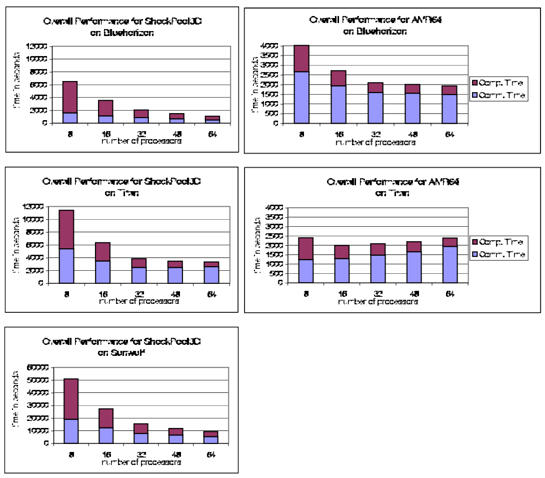

Figure 4. Overall Performance

It is also shown that the overall performance of two datasets is quite different on these systems. According to NPB results shown in the previous section, *Bluehorizon* is the fastest machine among these clusters, which matches the results for the dataset of *ShockPool3D*. However, when the number of processes is less than 32, *Titan* can actually provide better performance for *AMR64*. This indicates that the application performance on these clusters also depends on input datasets. Further, by looking into the data, we notice that the better performance achieved on *Titan* is due to smaller communication cost.

For *ShockPool3D*, the execution time is decreasing as the number of processes increases on all the clusters. However, for *AMR64*, the execution time does not reduce when the number of processes increases from 32 to 64 on both *Blue Horizon* and *Titan*. In particular, the communication cost on *Titan* is increasing dramatically when there are more than 32 processes. It seems that the code has some inherent scalability problems with the *AMR64* dataset.

In the previous section, we used NPB2.4 to calculate the relative performance of these clusters. As we can see, the cosmology performance on these systems does not exactly match the NPB performance. For example, the relative performance of *Titan* as compared to *Blue Horizon* is 0.37 according to NPB result. However, the performance of *AMR64* on *Titan* is actually better than on *Bluehorizon* when the number of processors is less than 32. Further, although the NPB result indicates that *Sunwulf* has better performance as compared to *Titan*, the cosmology performance on *Sunwulf* is much worse than on *Titan*.

4.2. Communication Performance

To analyze communication behavior of ENZO on these clusters, we divide the communication into the *collective* and *point-to-point* communication.

Figure 5 shows the communication results for both datasets on three clusters. For *ShockPool3D*, *Bluehorizon* provides the best communication performance while *Sunwulf* has the worst performance. This result matches the NetPIPE result provided in the previous section. The underlying interconnection on *Sunwulf* is fast Ethernet, which provides the slowest message-passing among these clusters according to NetPIPE measurement. As shown in section 3, *Bluehorizon* is a SMP system with eight processors per node; therefore most of communication occurs within node when using 8 - 64 processors. NetPIPE results show that message-passing performance within node on *Bluehorizon* is much better than message-passing performance across nodes on *Titan*. This explains why *Bluehorizon* delivers better communication performance as compared to *Titan* for the dataset *ShockPool3D*.

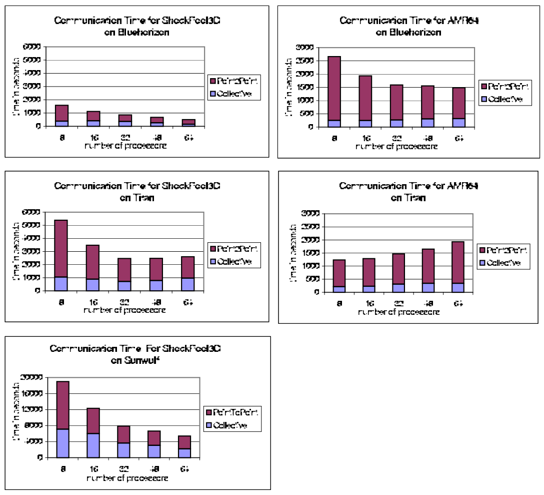

Figure 5. Collective vs. Point-To-Point Communication

For *AMR64*, the figure shows that *Titan* provides better communication performance when using 8 or 16 processors. This indicates that the underlying hardware and software tools for message passing on *Titan* can achieve better communication performance with small number of processors. Further, the figure indicates that on both *Blue Horizon* and *Titan*, the larger portion of communication is point-to-point communication and the collective communication increases with increasing number of processes. On *Blue Horizon*, point-to-point communication decreases when the number of processes is increased from 8 to 32; then it remains the same when we further increase the number of processes from 32 to 64. On *Titan*, point-to-point communication always increases with increasing number of processes.

By looking into the communication performance on *Titan*, we notice that when the number of processes is more than 32, the communication time is increasing for both datasets. This indicates that the underlying interconnected network of *Titan* does not scale past 32 processors. Our experimental data (not shown here) indicates that execution of both datasets tend to create a large amount of small-sized messages (less than 1KBytes), especially when the number of processes is large. Therefore, the underlying interconnection of *Titan* seems not efficient for small-sized messages when there are more than 32 processes.

4.3. Load Balancing Performance

As mentioned earlier, ENZO employs SAMR technique to address the adaptive feature of grid hierarchy. To measure the time spent on adaptation process, we instrumented the ENZO code and measured the time for the subroutine *ReBuildHierachy*. This subroutine is composed of the process of rebuilding the grid hierarchy, sharing the newly constructed grid hierarchy, and performing load balancing among the processes after each adaptation step.

Figure 6 shows the adaptive cost for both datasets on three clusters. For *ShockPool3D*, the performance of *RebuildHierarchy* is quite different. On *Blue Horizon*, it is decreasing from 612.03 seconds to 324.57 seconds as the number of processes increases from 8 to 64. However, on *Titan*, it is first decreasing from 1171.35 seconds to 1037.02 and then increasing to 1311.56 seconds as the number of processes increases. On *Sunwulf*, as compared to the other systems, the cost for *RebuildHierarchy* is much more. The time for this process is decreasing from 9921.54 to 3498.13 with increasing number of processes.

For *AMR64*, the time for this *RebuildHierarchy* process is always increasing as the number of processes increases on both *Blue Horizon* and *Titan*. On *Blue Horizion*, it is increasing from 467.49 to 557.07, while it is increasing from 564.31 to 833.72 on *Titan*.

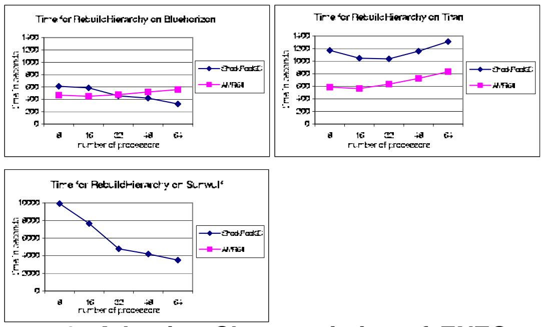

Figure 6. Adaptive Characteristics of ENZO on Three Clusters

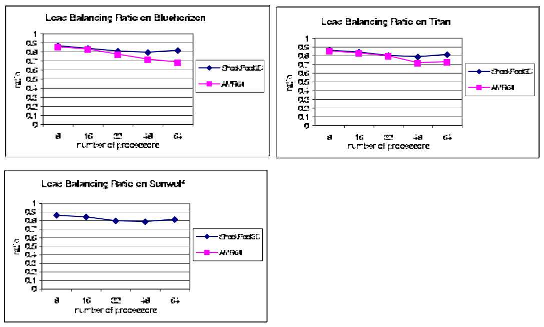

Figure 7. Load Balancing Performance

In [6], we proposed an efficient dynamic load balancing (DLB) scheme for SAMR applications and implemented it in the ENZO code. In this scheme, each load balancing step consists of one or more iterations of two phases: *moving-grid phase* and *splitting-grid phase*. The *moving-grid phase* redistributes grids directly from overloaded processors to underloaded processors by the guidance of the global information; and the *splitting-grid phase* splits a grid into two smaller grids along the longest dimension. For each load balancing step, the *moving-grid phase* is invoked first; then *splitting-grid phase* may be invoked if no more direct movement can occur. If significant imbalance still exists, another round of two phases may be invoked. Experiments show that by using this proposed DLB scheme, the parallel execution time can be reduced by up to 57% and the quality of load-balancing can be improved by a factor of six, as compared to the original DLB scheme used in ENZO.

Three metrics were proposed in [6] to measure the quality of load balancing. Especially, *Load_Balancing_Ratio* is defined as follows:

$$Load\_Balancing\_Ratio = \frac{\sum_{j=1}^{N} \frac{AvgLoad(j)}{MaxLoad(j)}}{N} \qquad (1)$$

Where N is number of adaptations, $MaxLoad(j)$ denotes the maximal amount of load of a processor for the jth adaptation, and $AvgLoad(j)$ denotes the average load of all the processors for the jth adaptation. It is clear that *Load_Balancing_Ratio* is smaller or equal to 1.0. The closer it is to 1.0 the better; the value of 1.0 implies equal load distribution among all processes.

Figure 7 shows the quality of load balancing, represented by *Load_Balancing_Ratio*, achieved on these clusters for both datasets. As we can see, the quality of load balancing achieved on different systems are similar to each other, which indicates that our proposed DLB scheme is not influenced by the underlying platform. Further, for all the cases, the *Load_Balancing_Ratio* is always larger than 0.78. This shows that the proposed DLB scheme can achieve high quality of load balancing for both datasets.

4.4. Analysis Results

Combining the above experimental data, we can draw four conclusions in terms of performance of the ENZO application on these cluster systems:

- First of all, we conclude that the application performance on these cluster systems depend on both the system performance and the application characteristics. For most cases, the best performance of the cosmology application is achieved on the IBM SP2 system *Blue Horizon* which has the best performance according to the NPB measurement. However, we notice that the performance also depends on the input parameters. For example, our experimental data shows that *AMR64* performs better on the IA-64 Linux cluster *Titan* while *ShockPool3D* can achieve better performance on the IBM SP2 system *Blue Horizon*.
- For both datasets, the cosmology performance on the IA-64 Linux cluster *Titan* is not satisfactory when the number of processes is more than 32. For example, for *AMR64*, the execution time on *Titan* increases dramatically when using more than 32 processes. This indicates that the underlying network inter-connection of *Titan* has scalability problems. In particular, it is suspected that it is not efficient for small-sized messages with a large number of processors.
- The performance analysis also indicates that the relative performance according to NPB benchmark does not exactly match the relative performance of the cosmology application on these clusters. According to the NPB measurement, the UNIX-based cluster *Sunwulf* has better performance as compared to the IA-64 Linux cluster *Titan*. However, the performance of *ShockPool3D* is much worse on *Sunwulf* than on *Titan*.

- Lastly, all the data shows that the proposed DLB scheme can achieve the same quality of load balancing on different clusters no matter how different these systems are.

5. Summary and Future Work

In this paper, we provided a detailed performance analysis of a large-scale production cosmology application on three cluster systems (the IBM SP2 system at SDSC, the IA-64 Linux cluster at NCSA, and the SUN cluster at IIT). Each of these systems represents a widely-used cluster environment in the area of scientific computing. The performance is evaluated from three different aspects: overall performance, communication characteristics, and load balancing characteristics. Our experimental data shows that the application performance on these systems depends on both the system performance and the application characteristics. The experimental data also shows that the relative performance according to the NPB benchmark does not exactly match the relative performance of the cosmology application on these clusters. We notice that the underlying interconnected network of *Titan* has scalability problem. Furthermore, the experimental data indicates that the proposed DLB scheme[6] can achieve the same quality of load balancing on different systems no matter how different these systems are.

Currently, we are working on exploring large-scale applications including the cosmology and biology applications on distributed clusters, in particular, the Distributed Terascale Facility (DTF). We are also working on generalizing the dynamic load balancing scheme proposed in [5], with the goal of developing a general and extensible dynamic load balancing tool to be used with large-scale adaptive applications on distributed environment, such as the Computational Grid.

Acknowledgments

The authors would like to thank Valerie Taylor at Texas A&M, Michael Norman at UCSD, and Greg Bryan at Oxford University for numerous comments and suggestions that contributed to this work. We also would like to acknowledge the National Center for Supercomputing Applications (NCSA), San Diego Supercomputing Center (SDSC), and the Scalable Computing Software Lab at Illinois Institute of Technology for the use of their machines.

References

[1] Alliance. *IA-64 Linux Cluster at NCSA*. World Wide Web, http://www.ncsa.uiuc.edu/.

[2] M. Berger and P. Colella. Local adaptive mesh refinement for shock hydrodynamics. *Journal of Computational Physics*, 82(1):64–84, May 1989.

[3] G. Bryan. Fluid in the universe: Adaptive mesh refinement in cosmology. *Computing in Science and Engineering*, 1(2):46–53, March/April 1999.

[4] G. Bryan, T. Abel, and M. Norman. Achieving extreme resolution in numerical cosmology using adaptive mesh refinement: Resolving primordial star formation. In *Proc. of SC2001*, Denver, CO, 2001.

[5] Z. Lan, V. Taylor, and G. Bryan. Dynamic load balancing of samr applications on distributed systems. In *Proc. of SC2001*, Denver, CO, 2001.

[6] Z. Lan, V. Taylor, and G. Bryan. A novel dynamic load balancing scheme for parallel systems. *Journal of Parallel and Distributed Computing*, page 1763 1781, 2002.

[7] H. Neeman. *Autonomous Hierarchical Adaptive Mesh Refinement for Multiscale Simulations*. PhD thesis, UIUC, 1996.

[8] NPACI. *NPACI User Guides*. World Wide Web, http://www.npaci.edu/BlueHorizon/.

[9] NetPIPE Team. *A Network Protocol Independent Performance Evaluator*. World Wide Web, http://www.scl.ameslab.gov/netpipe/.

[10] NPB Team. *Nas Parallel Benchmarks*. World Wide Web, http://www.nas.nasa.gov/Software/NPB/.

[11] Dave Turner and Xuehua Chen. Protocol-dependent message-passing performance on linux clusters. In *IEEE Cluster 2002*, Chicago, IL, 2002.