Scientific Programming 10 (2002) 319–328
IOS Press

Dynamic load balancing of SAMR applications on distributed systems[1]

Zhiling Lan[a], Valerie E. Taylor[b] and Greg Bryan[c]
[a]*Department of Computer Science, Illinois Institute of Technology, Chicago, IL 60616, USA*
E-mail: lan@iit.edu
[b]*Department of Electrical and Computer Engineering, Northwestern University, Evanston, IL 60208, USA*
E-mail: taylor@ece.nwu.edu
[c]*Nuclear and Astrophysics Laboratory, Oxford University, Oxford, OX13RH, UK*
E-mail: gbryan@astro.ox.ac.uk

Abstract. Dynamic load balancing(DLB) for parallel systems has been studied extensively; however, DLB for distributed systems is relatively new. To efficiently utilize computing resources provided by distributed systems, an underlying DLB scheme must address both heterogeneous and dynamic features of distributed systems. In this paper, we propose a DLB scheme for Structured Adaptive Mesh Refinement(SAMR) applications on distributed systems. While the proposed scheme can take into consideration (1) the heterogeneity of processors and (2) the heterogeneity and dynamic load of the networks, the focus of this paper is on the latter. The load-balancing processes are divided into two phases: *global load balancing* and *local load balancing*. We also provide a heuristic method to evaluate the computational gain and redistribution cost for global redistribution. Experiments show that by using our distributed DLB scheme, the execution time can be reduced by 9%–46% as compared to using parallel DLB scheme which does not consider the heterogeneous and dynamic features of distributed systems.

Keywords: Dynamic load balancing, distributed systems, adaptive mesh refinement, heterogeneity, dynamic network loads

1. Introduction

Structured Adaptive Mesh Refinement (SAMR) is a type of multiscale algorithm that dynamically achieves high resolution in localized regions of multidimensional numerical simulations. It shows incredible potential as a means of expanding the tractability of a variety of numerical experiments and has been successfully applied to model multiscale phenomena in a range of disciplines, such as computational fluid dynamics, computational astrophysics, meteorological simulations, structural dynamics, magnetic, and thermal dynamics. A typical SAMR application may require a large amount of computing power. For example, simulation of the first star requires a few days to execute on four processors of a SGI Origin2000 machine; however, the simulation is not sufficient, for which there are some unresolved scales that would result in longer execution time [3]. Simulation of the galaxy formation requires more than one day to execution on 128 processors of the SGI Origin2000 and requires more than 10GB of memory. Distributed systems provide an economical alternative to traditional parallel systems. By using distributed systems, researchers are no longer limited by the computing power of a single site, and are able to execute SAMR applications that require vast computing power (e.g., beyond that available at any single site). A number of national technology grids are being developed to provide access to many compute resources regardless of the location, e.g., *GUSTO*

[1]Zhiling Lan is supported by a grant from the National Computational Science Alliance (ACI-9619019), Valerie Taylor is supported in part by a NSF NGS grant (EIA-9974960), and Greg Bryan is supported in part by a NASA Hubble Fellowship grant (HF-01104.01-98A).

testbed [11], *NASA's Information Power Grid* [13], *National Technology Grid* [25]; several research projects, such as *Globus* [11] and *Legion* [12], are developing software infrastructures for ease of use of distributed systems.

Execution of SAMR applications on distributed systems involves dynamically distributing the workload among the systems at runtime. A distributed system may consist of heterogeneous machines connected with heterogeneous networks; and the networks may be shared. Therefore, to efficiently utilize the computing resources provided by distributed systems, the underlying dynamic load balancing (DLB) scheme must take into consideration the heterogeneous and dynamic features of distributed systems. DLB schemes have been researched extensively, resulting in a number of proposed approaches [14,7,8,17,21,22,24,26,27]. However, most of these approaches are inadequate for distributed systems. For example, some schemes assume the multiprocessor system to be homogeneous, (e.g., all the processors have the same performance and the underlying networks are dedicated and have the same performance). Some schemes consider the system to be heterogeneous in a limited way (e.g., the processors may have different performance but the networks are dedicated). To address the heterogeneity of processors, a widely-used mechanism is to assign a relative weight which measures the relative performance to each processor. For example, Elsasser et al. [9] generalize existing diffusive schemes for heterogeneous systems. Their scheme considers the heterogeneity of processors, but does not address the heterogeneity and dynamicity of networks. In [5], a parallel partitioning tool *ParaPART* for distributed systems is proposed. ParaPART takes into consideration the heterogeneity of both processors and networks; however, it is a static scheme and does not address the dynamic features of the networks or the application. Similar to PLUM [21], our scheme also use some evaluation strategies; however, PLUM addresses the issues related to homogeneous systems while our work is focused on heterogeneous systems. KeLP [14] is a system that provides block structured domain decomposition for SPMD. Currently, the focus of KeLP is on distributed memory parallel computers, with future focus on distributed systems.

In this paper, we proposed a dynamic load balancing scheme for distributed systems. This scheme takes into consideration (1) the heterogeneity of processors and (2) the heterogeneity and dynamic load of the networks. Our DLB scheme address the heterogeneity of processors by generating a relative performance weight for each processor. When distributing workload among processors, the load is balanced proportional to these weights. To deal with the heterogeneity of network, our scheme divides the load balancing process into *global load balancing phase* and *local load balancing phase*. The primary objective is to minimize remote communication as well as to efficiently balance the load on the processors. One of the key issues for global redistribution is to decide when such an action should be performed and whether it is advantageous to do so. This decision making process must be fast and hence based on simple models. In this paper, a heuristic method is proposed to evaluate the computational gain and the redistribution cost for global redistributions. The scheme addresses the dynamic features of networks by adaptively choosing an appropriate action based on the current observation of the traffic on the networks.

While our DLB takes into consideration the two features, the experiments presented in this paper focus on the heterogeneity and dynamic load of the networks due to the limited availability of distributed system testbeds. The compute nodes used in the experiments are dedicated to a single application and have the same performance. Experiments show that by using this distributed DLB scheme, the total execution time can be reduced by 9%–46% and the average improvement is more than 26%, as compared with using parallel DLB scheme which does not consider the heterogeneous and dynamic features of distributed systems. While the distributed DLB scheme is proposed for SAMR applications, the techniques can be easily extended to other applications executed on distributed systems.

The remainder of this paper is organized as follows. Section 2 introduces SAMR algorithm and its parallel implementation ENZO code. Section 3 identifies the critical issues of executing SAMR applications on distributed systems. Section 4 describes our proposed dynamic load balancing scheme for distributed systems. Section 5 presents the experimental results comparing the performance by this distributed DLB scheme with parallel DLB scheme which does not consider the heterogeneous and dynamic features of distributed systems. Finally, Section 6 summarizes the paper and identifies our future work.

2. Structured adaptive mesh refinement applications

This section gives an overview of the SAMR method, developed by M. Berger et al., and ENZO, a parallel

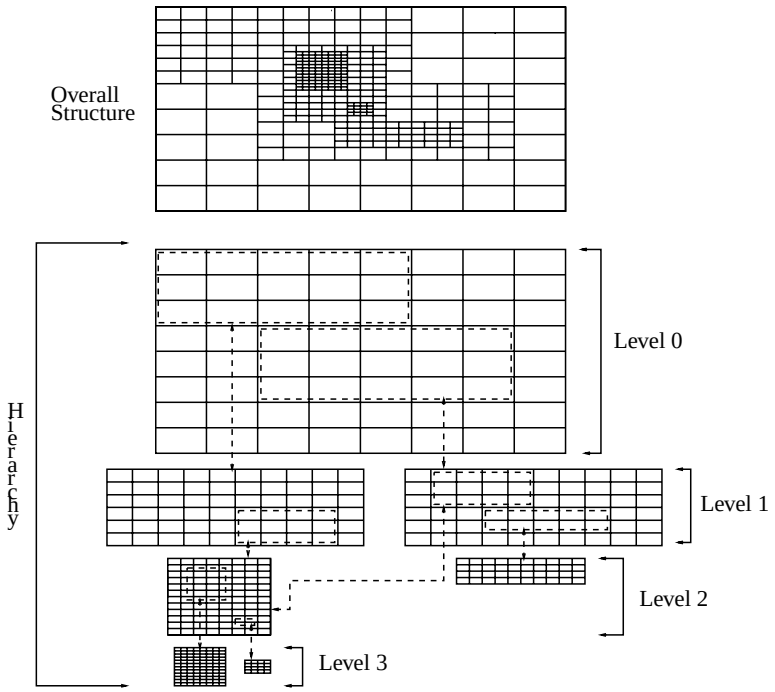

Fig. 1. SAMR grid hierarchy.

implementation of this method for astrophysical and cosmological applications. Additional details about ENZO and the SAMR method can be found in [2,1,19, 3,20].

2.1. Layout of grid hierarchy

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. That is, a uniform mesh covers the entire computational volume and in regions that require higher resolution, a finer subgrid is added. If a region needs still more resolution, a even finer subgrid is added. This process repeats recursively with each adaptation resulting in a tree of grids like that shown in Fig. 1 [19]. 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 regions corresponding to those that underwent further refinement. In this grid hierarchy, there are four levels of grids going from level 0 to level 3. Throughout execution of an SAMR application, the grid hierarchy changes with each adaptation.

2.2. Integration execution order

The SAMR integration algorithm goes through the various adaptation levels advancing each level by an appropriate time step, then recursively advancing to the next finer level at a smaller time step until it reaches the same physical time as that of the current level. Figure 2 illustrates the execution sequence for an application with four levels and a refinement factor of 2. First we start with the first grid on level 0 with time step dt. Then the integration continues with one of the subgrids, found on level one, with time step $dt/2$. Next, the integration continues with one of the subgrids on level 2, with time step $dt/4$, followed by the analysis of the subgrids on level 3 with time step $dt/8$. The figure illustrates the order for which the subgrids are analyzed with the integration algorithm.

2.3. ENZO: A parallel implementation of SAMR

Although the SAMR strategy shows incredible potential as a means for simulating multiscale phenomena and has been available for over two decades, it is still not widely used due to the difficulty with implementation. The algorithm is complicated because of dynamic nature of memory usage, the interactions between different subgrids and the algorithm itself. ENZO [3] is one

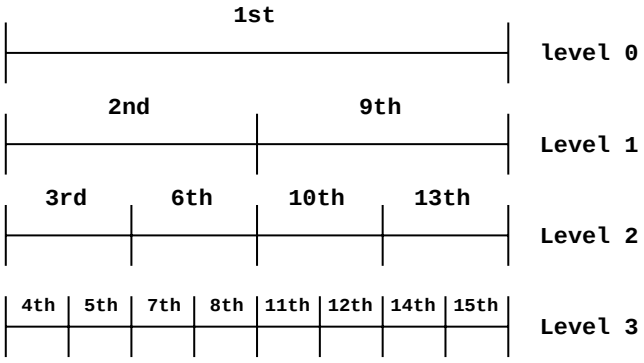

Fig. 2. Integrated execution order (refinement factor = 2).

of the successful parallel implementations of SAMR, which is primarily intended for use in astrophysics and cosmology. It is written in C++ with Fortran routines for computationally intensive sections and MPI functions for message passing among processors. ENZO was developed as a community code and is currently in use on at least six different sites.

In [15,16], a DLB scheme was proposed for SAMR on parallel systems. It was designed for efficient execution on homogeneous systems by considering some unique adaptive characteristics of SAMR applications. In the remainder of this paper, we denote this scheme as *parallel DLB scheme*.

3. Issues and motivations

In this section we compare the performance of ENZO executed on a parallel machine with that executed on a distributed system. It is well-known that the distributed system will have a larger execution time than the parallel system with the same number of processors because of the performance of the WANs used to interconnect the machines in a distributed system. WANs generally have much larger latency than the interconnects found in parallel machines. However, the comparison is given in this paper to illustrate the amount of overhead introduced by the WAN in the distributed system, which is the focus of this paper. Our DLB attempts to reduced this overhead to make distributed systems more efficient. The experiments use small numbers of processors to illustrate the concepts, but it is assumed that in practice the distributed system will have a large number of processors to provide the needed compute power, which is beyond any single, available parallel machine.

The experiment used for the comparison uses the *parallel DLB scheme* on both the parallel and distributed systems. The parallel system consists of a 250 MHz R10000 SGI Origin2000 machine at Argonne National Lab (ANL); the parallel executable was compiled with SGI implemented MPI. The distributed system consists of two geographically distributed 250 MHz R10000 SGI Origin2000 machines: one located at ANL and the other located at National Center for Supercomputing Applications(NCSA). The machines are connected by the MREN network consisting of ATM OC-3 networks. The distributed executable was compiled with the grid-enabled implementation MPICH-G2 provided by ANL [18]. We used *Globus* [11] tool to run the application on the distributed system. The experiments used the dataset *ShockPool3D*, which is described in detail in Section 5.

Five configurations ($1+1, 2+2, 4+4, 6+6$, and $8+8$) are tested. For the distributed system, the configuration $4+4$ implies four processors at ANL and four processor at NCSA; for the parallel system this configuration implies eight processors at ANL. The results are given in Fig. 3. For all the configurations, the times for parallel computation and distributed computation are similar as expected since the ANL and NCSA processors have the same performance. However, since the distributed system consists of two remotely connected machines and the connection is a shared network, times for distributed communication are much larger than those for parallel communication. Therefore, in order to get higher performance from distributed systems, the key issues are how to reduce remote communication and how to adaptively adjust to the dynamic feature of networks. These results motivate us to design a distributed DLB scheme that considers the heterogeneity in processors and the heterogeneity and dynamic load of the networks.

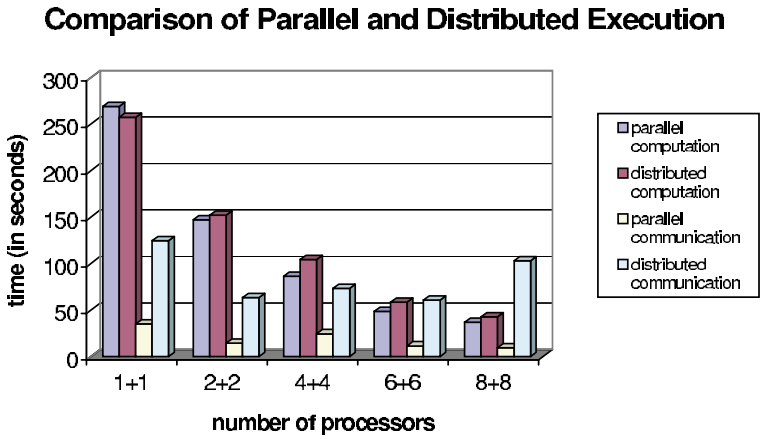

Fig. 3. Comparison of parallel and distributed execution.

4. Distributed dynamic load balancing scheme

In this section, we present a DLB scheme for SAMR applications on distributed systems. To address the heterogeneity of processors, each processor is assigned a relative weight. To deal with the heterogeneity of networks, the scheme divides the load balancing process into two steps: *global load balancing phase* and *local load balancing phase*. Further, the proposed scheme addresses dynamic feature of networks by adaptively choosing an appropriate action according to the traffic on them. The details are given in the following subsections.

4.1. Description

First, we define a "group" as a set of processors which have the same performance and share an intra-connected network; a group is a homogeneous system. A group can be a shared-memory parallel computer, a distributed-memory parallel computer, or a cluster of workstations. Communications within a group are referred as local communication, and those between different groups are remote communications. A distributed system is composed of two or more groups. This terminology is consistent with that given in [6]

Our distributed DLB scheme entails two steps to redistribute the workload: *global load balancing phase* and *local load balancing phase*, which are described in detail below.

– *Global Load Balancing Phase*
  After each time-step at level 0 only, the scheme evaluates the load distribution among the groups by considering both heterogeneous and dynamic features of the system. If imbalance is detected, a heuristic method described in the following subsections is invoked to calculate the computational gain of removing the imbalance and the overhead of performing such a load redistribution among groups. If the computational gain is larger than the redistribution overhead, this step will be invoked. All the processors will be involved in this process, and both global and local communications are considered. Workload will be redistributed by considering the heterogeneity of number of processors and processor performance of each group.
– *Local Load Balancing*
  After each time-step at the finer levels, each group entails a balancing process within the group. The *parallel DLB scheme* as mentioned in section 2.3 is invoked, that is, the workload of each group is evenly and equally distributed among the processors. However, load balancing is only allowed within the group. An overloaded processor can migrate its workload to an underloaded processor of the same group only. In this manner, children grids are always located at the same group as their parent grids; thus no remote communication is needed between parent and children grids. There may be some boundary information exchange between sibling grids which usually is very small. During this step, load imbalance may be detected among groups at the finer levels, however, the *global balancing* process will not be invoked until the next time-step at level 0.

The flow control of this scheme is shown in Fig. 4. Here, $time(i)$ denotes the iterative time for level i, and $dt(i)$ is the time-step for level i. The left part of the

figure illustrates the global load balancing step, while the right part represents the local load balancing step. Following each time-step $dt(0)$ at level 0, there are several smaller time-steps $dt(i)$ at a finer level i until the finer level i reaches the same physical time as that of level 0. Figure 5 illustrates the executing points of *global balancing* and *local balancing* processes in terms of the execution order given in Fig. 2. It is shown that the *local balancing* process may be invoked after each smaller time-step while the *global balancing* process may be invoked after each time-step of the top level only. Therefore, there are fewer *global balancing* processes during the run-time as compared to *local balancing* processes.

4.2. Cost evaluation

To determine if a global redistribution is invoked, an efficient evaluation model is required to calculate the redistribution cost and the computational gain. The evaluation should be very fast to minimize the overhead imposed by the DLB. Basically, the redistribution cost consists of both communicational and computational overhead. The communicational overhead includes the time to migrate workload among processors. The computational overhead includes the time to partition the grids at the top level, rebuild the internal data structures, and update boundary conditions.

We propose a heuristic method to evaluate the redistribution cost as follows. First, the scheme checks the load distribution of the system. If imbalance exists, the scheme calculates the amount of load needed to migrate between groups. In order to adaptively calculate communication cost, the network performance is modeled by the conventional model, that is $T_{comm} = \alpha + \beta \times L$. Here T_{comm} is the communication time, α is the communication latency, β is the communication transfer rate, and L is the data size in bytes. Then the scheme sends two messages between groups, and calculates the network performance parameters α and β. If the amount of workload need to be redistributed is W, the communication cost would be $\alpha + \beta \times W$. This communication model is very simple so little overhead is introduced.

To estimate the computational cost, the scheme uses history information, that is, recording the computational overhead of the previous iteration. We denote this portion of cost as δ. Therefore, the total cost for redistribution is:

$$Cost = (\alpha + \beta \times W) + \delta \tag{1}$$

4.3. Gain evaluation

SAMR allows local refining and coarsening of the grid hierarchy during the execution to capture the phenomena of interest, resulting in dynamically changing workload. The total amount of workload at the time t may not be the same as that at the time $t + dt$. However, the difference is usually not very much between time steps. Thus, the scheme predicts the computational gain by the following heuristic method. Between two iterations at level 0, the scheme records several performance data, such as the amount of load each processor has for all levels, the number of iterations for each finer level, and the execution time for one time-step at level 0. Note that there are several iterative steps for each finer level between two iterations at level 0 (see Fig. 5). For each group, the total workload (including all the levels) is calculated for one time-step at level 0 using this recorded data. Then the difference of total workload between groups is estimated. Lastly, the computational gain is estimated by using the difference of total workload and the recorded execution time of one iteration at the top level. The detailed formula are as follows:

$$W^i_{group}(t) = \sum_{proc \in group} w^i_{proc}(t) \tag{2}$$

$$W_{group}(t) = \sum_{0 \leqslant i \leqslant maxlevel} W^i_{group}(t) \times N^i_{iter}(t) \tag{3}$$

$$Gain = T(t) \times \frac{max(W_{group}(t)) - min(W_{group}(t))}{Number\_Groups \times max(W_{group}(t))} \tag{4}$$

Here, $Gain$ denotes the estimated computational gain for global load balancing at time t; $w^i_{proc}(t)$ is the workload of processor $proc$ at level i for time t; $W^i_{group}(t)$ is the total amount of load of $group$ at level i for time t; $N^i_{iter}(t)$ is the number of iterative steps of level i from the time t to $t + dt$; $W_{group}(t)$ denotes the total workload of $group$ at time t; and $T(t)$ is the execution time from the time $(t - dt)$ to t, a time step. Hence, the gain provides a very conservative estimate of the amount of decrease in execution time that will occur from the redistribution of load resulting from the DLB.

4.4. Global load redistribution

The global load redistribution is invoked when the computational gain is larger than some factor times the

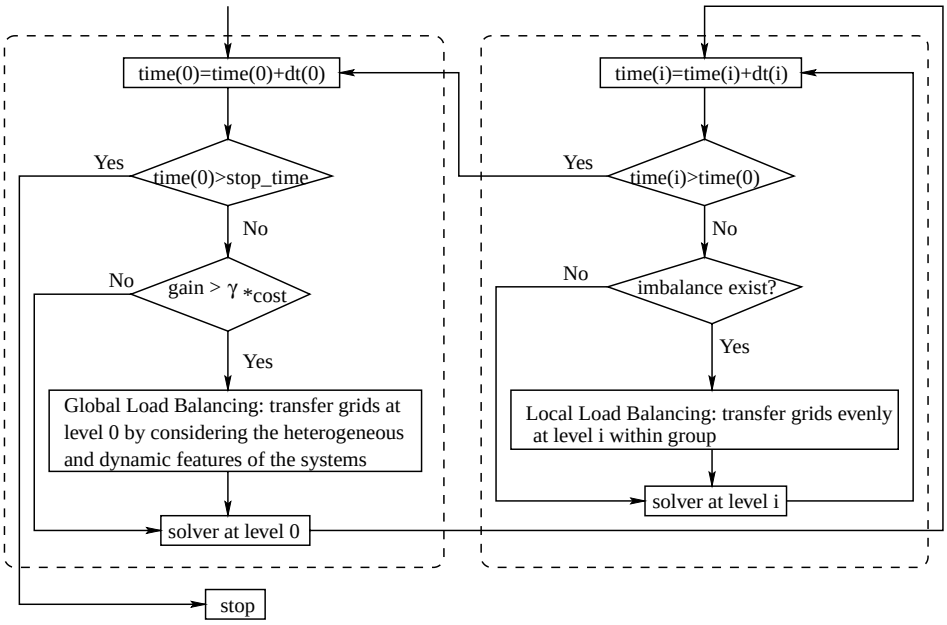

Fig. 4. Flowchart of distributed DLB.

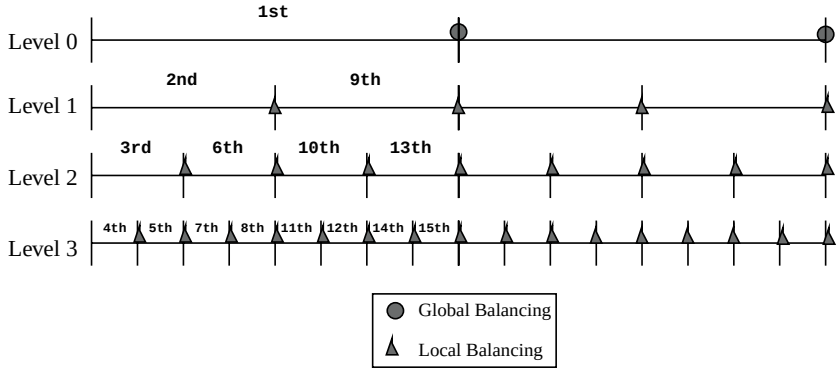

Fig. 5. Integrated execution order showing points of load balancing.

redistribution cost, that is, when $Gain > \gamma \times Cost$. Here, γ is a user-defined parameter (default is 2.0) which identifies how much the computational gain must be for the redistribution to be invoked. The detailed sensitivity analysis of this parameter will be included in our future work. During the global redistribution step, the scheme redistributes the workload by considering the heterogeneity of processors. For example, suppose the total workload is W, which needs to be partitioned into two groups. Group A consists of n_A processors and each processor has the performance of p_A; group B consists of n_B processors and each processor have the performance of p_B. Then the *global balancing* process will partition the workload into two portions: $(W \times \frac{n_A \times p_A}{n_A \times p_A + n_B \times p_B})$ for group A and $(W \times \frac{n_B \times p_B}{n_A \times p_A + n_B \times p_B})$ for group B. Basically, this step entails moving the groups' boundaries slightly from underloaded groups to overloaded groups so as to balance the system. Further, only the grids at level 0 are involved in this process and the finer grids do not need to be redistributed. The reason is that the finer grids would be reconstructed completely from the grids at level 0 during the following smaller time-steps.

Figure 6 shows an example of global redistribution. The top graph shows the overall grid hierarchy at time t. Group A is overloaded because more refinements occur in its local regions. After calculating the gain and the cost of a global redistribution by using the heuristic methods proposed in the previous subsections, the scheme determines that performing a global load

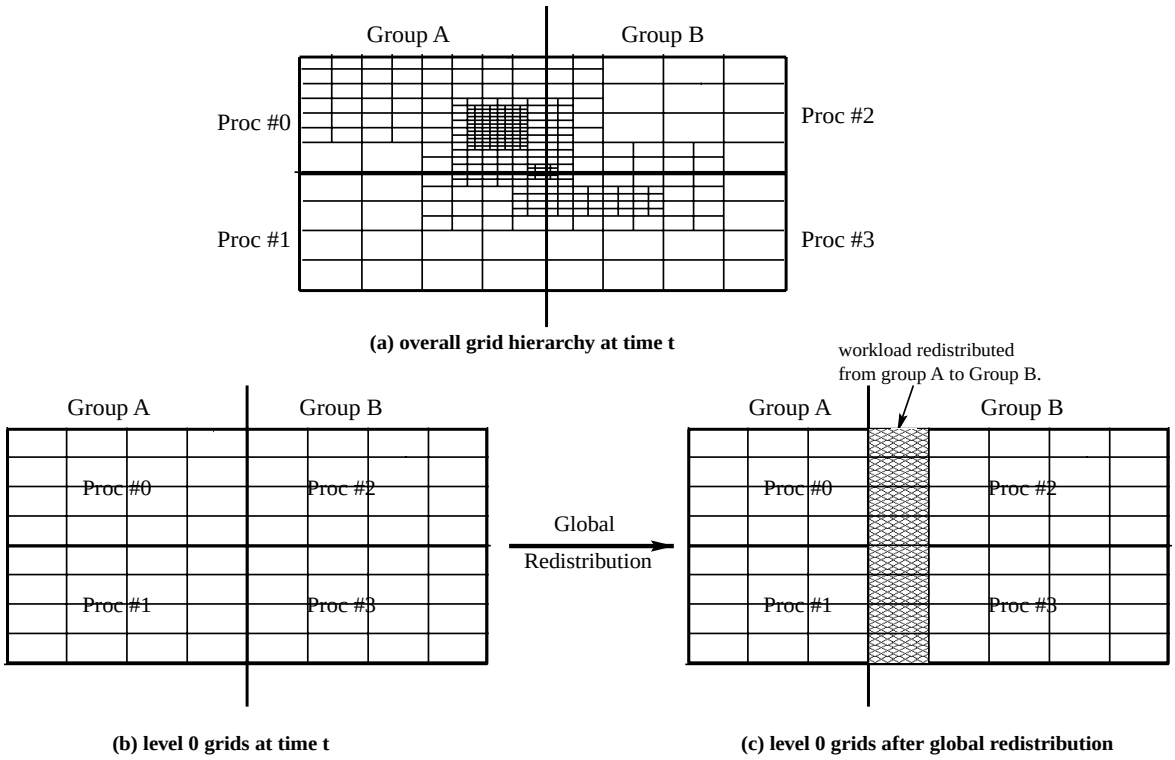

Fig. 6. An example for global redistribution.

balancing is advantageous; thus, a global redistribution is invoked. Figure 6(b) shows the level 0 grids at time t; and Fig. 6(c) represents the level 0 grids after global redistribution process. The boundary between two groups are moved slightly to the overloaded group. The amount of level 0 grids in the shade, about $\frac{W_A(t)-W_B(t)}{2\times W_A(t)} \times W_A^0(t)$, is redistributed from the overloaded Group A to the underloaded Group B.

5. Experimental results

In this section, we present the experimental results on two real SAMR datasets comparing the proposed distributed DLB with parallel DLB scheme on distributed systems. Both of the executables were compiled with MPICH-G2 library [18] and *Globus* toolkit was used to launched the experiments. Since the processors in the systems (described below) have the same performance, the difference in performance between two DLBs reflects the advantage of taking into consideration the heterogeneity and dynamic load of the networks.

One dataset is *AMR64* and the other is *ShockPool3D*. *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. The two datasets have different adaptive behaviors. *AMR64* is designed to simulate the formation of a cluster of galaxies, so many grids are randomly distributed across the whole computational domain; *ShockPool3D* is designed to simulate the movement of a shock wave (i.e., a plane) that is slightly tilted with respect to the edges of the computational domain, so more and more grids are created along the moving shock wave plane.

Two distributed systems are tested in our experiments: one consists of two machines at ANL connected by a local area network (fiber-based Gigabit Ethernet); the other is a system connecting two machines at ANL and NCSA by a wide area network, MREN that has ATM OC-3 networks. Note that both networks are shared networks and all the machines are 250MHz SGI Origin2000. *AMR64* is tested on the LAN-connected system, and *ShockPool3D* is tested on the WAN-connected system. For each configuration, the distributed scheme was executed immediately following the parallel scheme. This was done so that the two executions would have the similar network environments (e.g., periods of high traffic due to sharing of the networks or low traffic); thus, the performance

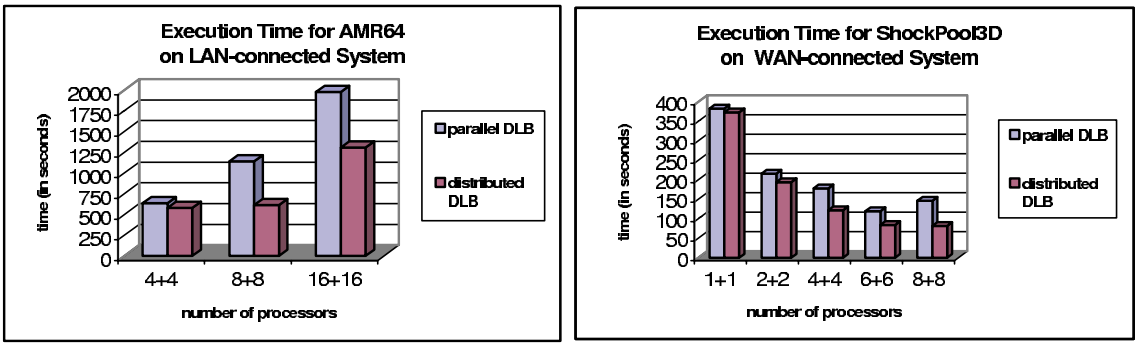

Fig. 7. Execution time for AMR64 and ShockPool3D.

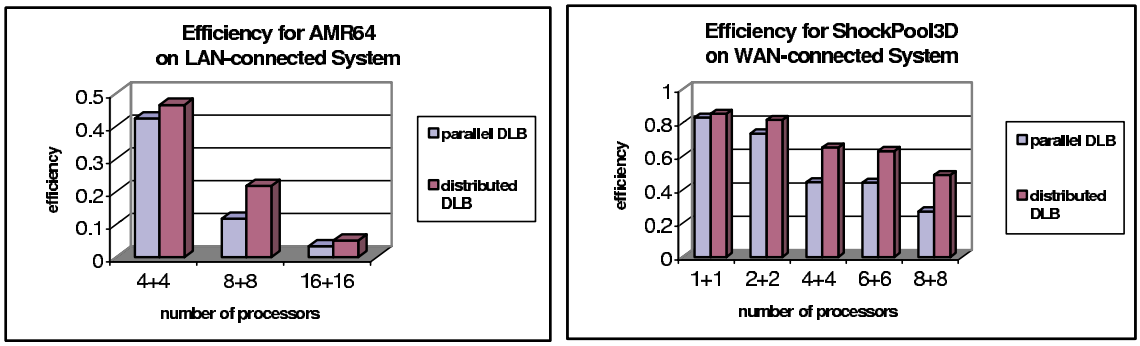

Fig. 8. Efficiency for AMR64 and ShockPool3D.

difference shown in the below is attributed to the difference in the DLB schemes.

Figure 7 compares the total execution times with varying configurations for both datasets. It is shown that the total execution time by using the proposed distributed DLB is reduced greatly as compared to using parallel DLB, especially as the number of processors is increased. The relative improvements are as follows: for *AMR64*, it is in the range of 9.0%–45.9% and the average improvement is 29.7%; for *ShockPool3D*, it is ranging from 2.6% to 44.2% and the average improvement is 23.7%.

Figure 8 gives the comparison of efficiency with varying configurations. Here, the efficiency is defined as: $efficiency = \frac{E(1)}{E \times P}$, where $E(1)$ is the sequential execution time on one processor, E is the execution time on the distributed system, and P is equal to the summation of each processor's performance relative to the performance used for sequential execution [4]. In this experiment, all the processors have the same performance, so this P is equal to the number of processors. As we can observe, the efficiency by using distributed DLB is improved significantly. For *AMR64*, the efficiency is improved by 9.9%–84.8%; for *ShockPool3D*, the efficiency is increased by 2.6%–79.4%.

6. Summary and future work

In this paper, we proposed a dynamic load balancing scheme for distributed systems. This scheme takes into consideration (1) the heterogeneity of processors and (2) the heterogeneity and dynamic load of networks. To address the heterogeneity of processors, each processor is assigned a relative performance weight. When distributing workload among processors, the load is distributed proportionally to these weights. To deal with the heterogeneity of network, the scheme divides the load balancing process into *global load balancing phase* and *local load balancing phase*. Further, the scheme addresses the dynamicity of networks by adaptively choosing an appropriate action based on the observation of the traffic on the networks. For global redistribution, a heuristic method was proposed to evaluate the computational gain and the redistribution cost. The experiments, however, illustrate the advantages of our DLB to handle the heterogeneity and dynamic load of the networks. The experiments show that by using this distributed DLB scheme, the total execution time can be reduced by 9%–46% and the average improvement is more than 26%, as compared to using parallel

DLB scheme which does not consider the heterogeneous and dynamic features of distributed systems.

Our future work will focus on including more heterogeneous machines and larger real datasets into our experiments. Further, we will connect this proposed DLB scheme with tools such as the NWS service [28] to get more accurate evaluation of underlying networks. Lastly, a detailed sensitivity analysis of parameters used in this distributed DLB scheme will also be completed.

References

[1] S. Baden, N. Chrisochoides, M. Norman and D. Gannon. Structured Adaptive Adaptive Mesh Refinement (SAMR) Grid Methods, *IMA Volumes in Mathematics and its Applications*, vol. 117 Springer-Verlag, NY, 1999.

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

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

[4] J. Chen, *Mesh Partitioning for Distributed Systems*, Ph.D. Thesis, Northwestern University, 1999.

[5] J. Chen and V. Taylor, ParaPART: Parallel Mesh Partitioning Tool for Distributed Systems, *Concurrency: Practice and Experience* **12** (2000), 111–123.

[6] J. Chen and V. Taylor, PART: Mesh Partitioning for Efficient Use of Distributed Systems, To appear in *IEEE Transactions on Parallel and Distributed Systems*.

[7] G. Cybenko, Dynamic load balancing for distributed memory multiprocessors, *IEEE Transactions on Parallel and Distributed computing* **7** (October, 1989), 279–301.

[8] K. Dragon and J. Gustafson, *A low-cost hypercube load balance algorithm*, Proc. Fourth Conf. Hypercubes, Concurrent Comput. and Appl., 1989, pp. 583–590.

[9] R. Elsasser, B. Monien and R. Preis, *Diffusive Load Balancing Schemes for Heterogeneous Networks*, Proc. SPAA'2000, Maine, 2000.

[10] I. Foster and C. Kesselman, *The Grid: Blueprint for a New Computing Infrastructure*, Morgan Kaufmann Publishers, San Francisco, California, 1999.

[11] Globus Project Team, *Globus Project*, World Wide Web, http://www.globus.org, 1996.

[12] A. Grimshaw and W. Wulf, The Legion Vision of a Worldwide Virtual Computer, *Communications of the ACM* **40**(1) (January, 1997).

[13] W. Johnston, D. Gannon and B. Nitzberg, *Grids as Production Computing Environments: The Engineering Aspects of NASA's Information Power Grid*, IEEE Computer Society Press, 1999.

[14] The KeLP Programming System, World Wide Web, http://www-cse.ucsd.edu/groups/hpcl/scg/kelp.html.

[15] Z. Lan, V. Taylor and G. Bryan, *Dynamic Load Balancing For Structured Adaptive Mesh Refinement Applications*, Proc. of ICPP'2001, Valencia, Spain, 2001.

[16] Z. Lan, V. Taylor and G. Bryan, *Dynamic Load Balancing For Adaptive Mesh Refinement Applications: Improvements and Sensitivity Analysis*, Proc. of IASTED PDCS'2001, Anaheim, CA, 2001.

[17] F. Lin and R. Keller, The gradient model load balancing methods, *IEEE Transactions on Software Engineering* **13**(1) (January, 1987), 8–12.

[18] MPICH Project Team, World Wide Web, http://www.niu.edu/mpi/.

[19] H. Neeman, *Autonomous Hierarchical Adaptive Mesh Refinement for Multiscale Simulations*, Ph.D. Thesis, UIUC, 1996.

[20] M. Norman and G. Bryan, *Cosmological adaptive mesh refinement*, Computational Astrophysics, 1998.

[21] L. Oliker and R. Biswas, PLUM: parallel load balancing for adaptive refined meshes, *Journal of Parallel and Distributed Computing* **47**(2) (1997), 109–124.

[22] K. Schloegel, G. Karypis and V. Kumar, Multilevel diffusion schemes for repartitioning of adaptive meshes, *Journal of Parallel and Distributed Computing* **47**(2) (1997), 109–124.

[23] S. Sinha and M. Parashar, *Adaptive Runtime Partitioning of AMR Applications on Heterogeneous clusters*, submitted to the 3rd IEEE International Conference on Cluster Computing, Newport Beach, CA, March, 2001.

[24] A. Sohn and H. Simon, *Jove: A dynamic load balancing framework for adaptive computations on an sp-2 distributed multiprocessor*, NJIT CIS Technical Report, New Jersey, 1994.

[25] R. Steven, P. Woodward, T. DeFanti and C. Catelett, From the I-WAY to the National Technology Grid, *Communications of the ACM* **40**(11) (1997), 50–61.

[26] C. Walshaw, *Jostle: partitioning of unstructured meshes for massively parallel machines*, Proc. Parallel CFD'94, 1994.

[27] M. Willebeek-LeMair and A. Reeves, Strategies for dynamic load balancing on highly parallel computers, *IEEE Transactions on Parallel and Distributed Systems* **4**(9) (September, 1993), 979–993.

[28] R. Wolski, *Dynamically Forcasting Network Performance using the Network Weather Service*, Technical Report CS-96-494, U.C. San Diego, 1996.