Adaptive System Sensitive Partitioning of AMR Applications on Heterogeneous Clusters *

Shweta Sinha (shwetas@caip.rutgers.edu) and Manish Parashar (parashar@caip.rutgers.edu)
The Applied Software Systems Laboratory,
Department of Electrical and Computer Engineering,
Rutgers, The State University of New Jersey

Abstract.
This paper presents the design and evaluation of an adaptive system sensitive partitioning and load balancing framework for distributed adaptive mesh refinement applications on heterogeneous and dynamic cluster environments. The framework uses system capabilities and current system state to select and tune appropriate partitioning parameters (e.g. partitioning granularity, load per processor) to maximize overall application performance. Furthermore, it uses dynamic load sensing (using NWS) to adapt to the load dynamics in the cluster.

Keywords: System Sensitive adaptive partitioning, dynamic load-balancing, heterogeneous computing, structured adaptive mesh refinement

1. Introduction

Dynamically adaptive methods for the solution of partial differential equations that employ locally optimal approximations can yield highly advantageous ratios for cost/accuracy when compared to methods based upon static uniform approximations. These techniques seek to improve the accuracy of the solution by dynamically refining the computational grid in regions of high local solution error. Distributed implementations of these adaptive methods offer the potential for accurate solutions to realistic models of important physical phenomena. These implementations however, lead to interesting challenges in dynamic resource allocation, dynamic data-distribution and load balancing, communications and coordination, and resource management.

Moving these applications to dynamic and heterogeneous cluster computing environments introduces a new level of complexity. These environments require the runtime selection and configuration of application components and parameters based on the availability and state of the resources. However, the complexity and heterogeneity of

* The research presented in this paper was supported in part by the National Science Foundation under Grant Numbers ACI 9984357 (CAREERS) and WU-HT-99-19 P029786F (KDI) awarded to Manish Parashar

the environment make selection of a "best" match between system resources, mappings and load distributions, communication mechanisms, etc. non-trivial. System dynamics coupled with application adaptivity makes application and run-time management a significant challenge.

In this paper we present the design and evaluation of an adaptive system sensitive partitioning and load balancing framework for distributed adaptive mesh refinement (AMR) applications on heterogeneous and dynamic cluster environments. The framework uses system capabilities and current system state to select and tune appropriate distribution parameters. Current system state is obtained at runtime using the NWS (Network Weather Service) [3] resource monitoring tool. System state information along with system capabilities are then used to compute relative computational capacities of each of the computational nodes in the cluster. These relative capacities are used by a heterogeneous "system-sensitive" partitioner for dynamic distribution and load-balancing. The heterogeneous partitioner has been integrated into the GrACE (Grid Adaptive Computational Engine) infrastructures' [1, 2] adaptive runtime system. GrACE is a data-management framework for parallel/distributed AMR, and is being used to provide AMR support for varied applications including reservoir simulations, computational fluid dynamics, seismic modeling and numerical relativity.

The rest of this paper is organized as follows: Section 2 discusses related work. Section 3 describes parallel/distributed AMR methods and the adaptive grid structure defined by these hierarchical adaptive mesh-refinement techniques. Section 4 introduces the GrACE adaptive computational engine. Section 5 outlines the architecture of the system sensitive runtime management framework. Section 6 describes the framework implementation and presents an experimental evaluation. Section 7 discusses the results. Section 8 presents some concluding remarks and directions for future work.

2. Related Work

A taxonomy of related work in partitioning/load-balancing techniques is shown in Figure 1. Techniques are classified based on whether they address system and/or application dynamics. The four quadrants shown in Figure 1 are discussed below. Note that in this discussion we focus on techniques that address system heterogeneity.

Static Application and Static Environment: There exists a large body of research addressing load balancing techniques where the system as well as the application are static. Most of these techniques,

Static Application , Static System State [5]	Static Application , Dynamic System State [3, 6, 7, 9, 13, 14, 15]
Dynamic Application, Static System State [8, 16,17,18]	Dynamic Application, Dynamic System State [19]

Figure 1. Taxonomy of related work on load balancing based on system and application state. The references are shown in brackets above.

however, do not address system heterogeneity. One of the most recent works that addresses this issue is by Figueira et. al in [5]. They have defined an algorithm which determines the best allocation of tasks to resources in heterogeneous systems. They have discussed the mapping of parallel applications with multiple tasks to heterogeneous platforms. Their work however does not take into account the dynamism in the environment. Furthermore, it addresses static applications.

Static Application and Dynamic Environment: Techniques described in [6], [9], and [13] address dynamic heterogeneous environments. These techniques primarily differ in the way they quantify system capabilities in the heterogeneous system. Watts et. al in [6] have presented different techniques to measure the resource capacities. One method uses a benchmarking program to determine the relative speeds of various machines. These offline performance metrics, along with other specifications, such as the machines' memory capacities, can be placed in a file which is read at the beginning of the computation. Another method to measure the resource capacities' is the use of system-measured utilization numbers. M. Ripeanu in [9] has developed a measure of efficiency that considers both the relative computational power of each processor and the share of time used on each processor. The relative speed is measured using synthetic benchmarks or by using the real application itself as a benchmark. Cai et. al in [13] have simulated a heterogeneous environment based only on CPU processing power for their study. Gross et. al in [14] have also emphasized the need

for external measurements of network mechanisms to support effective adaptive execution of large distributed scientific applications.

Some authors have also proposed "matching and scheduling" algorithms for a dynamic heterogeneous environment. For example, Topcuoglu, et al in [20] have presented two algorithms, the Heterogeneous Earliest-Finish-time and the Critical-Path-on-a -processor for scheduling DAGs on a bounded number of heterogeneous processors. Maheswaran et. al in [7] have suggested a new dynamic algorithm called the "hybrid-remapper " which uses the run-time values that become available upon completion of subtasks as well as machine availabilities during application execution time. Leinberger et. al in [15] have addressed a workload distribution problem for a computational grid with multi-resource servers. Servers in the grid have multiple resource capacities and the applications submitted by the users have multiple resource requirements. To fully utilize all K resources at each site, they suggest a heuristic to *match* the job mix at each server with the capabilities of that server, in addition to balancing the load across servers. Andersen et. al in [12] have also studied dynamic load-balancing for a static application. However, their approach to redistribution is through initiation of requests to a target machine from either idle machines asking for work or from busy machines relinquishing excess work.

Dynamic Application and Static Environment: A lot of research work has been done that addresses dynamic applications and static environments, however, they do not take heterogeneity of the system into account. M. Parashar et. al in [2] have addressed the partitioning of dynamic applications on a static environment. They have not addressed the heterogeneity of the system. References [8], [16], and [17] have also looked at the partitioning of dynamic applications but they have also not taken system heterogeneity or dynamic system state into account. G. Karypis, et. al in [18] have implemented algorithms for partitioning unstructured graphs and for computing fill-reducing orderings of sparse matrices. B. Hendrickson, et. al in [21] have developed a library called Zoltan that includes a suite of dynamic load-balancing algorithms, including both geometric and graph-based algorithms.

Dynamic Application and Dynamic Environment: Most existing partitioning and load balancing research addresses either dynamic applications or dynamic system environments, and fall in one of the quadrants discussed above. The system-sensitive partitioning/load-balancing framework presented in this paper addresses both *dynamic heterogeneous* environments and *dynamic* adaptive applications. Furthermore, it introduces a cost model that defines the relative capacities

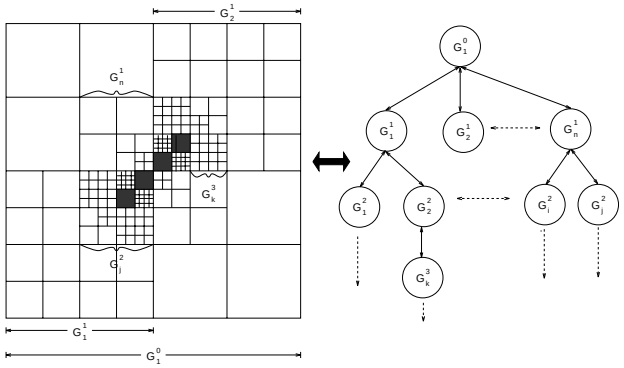

Figure 2. Adaptive Grid Hierarchy - 2D (Berger-Oliger AMR Scheme)

of system resources based on current system parameters such as CPU availabilities, memory usage and link bandwidth. System parameters are obtained using a resource monitoring tool.

3. Problem Description: Distributed AMR Applications

Dynamically adaptive numerical techniques for solving differential equations provide a means for concentrating computational effort to appropriate regions in the computational domain. In the case of hierarchical AMR methods, this is achieved by tracking regions in the domain that require additional resolution by dynamically overlaying finer grids over these regions. AMR-based techniques start with a base coarse grid with minimum acceptable resolution that covers the entire computational domain. As the solution progresses, regions in the domain requiring additional resolution are tagged and finer grids are overlaid on the tagged regions of the coarse grid. Refinement proceeds recursively so that regions on the finer grid requiring more resolution are similarly tagged and even finer grids are overlaid on these regions. The resulting grid structure is a dynamic adaptive grid hierarchy. The adaptive grid hierarchy corresponding to the AMR formulation by Berger & Oliger [4] is shown in Figure 2. Figure 3 shows a sequence of grid hierarchies for a 2-D Buckley-Leverette oil reservoir simulation application. The operations associated with the Berger-Oliger algorithm are outlined below.

Time Integration: Time integration is the update operation performed on each grid at each level of the adaptive grid hierarchy. Integration uses an application specific operator.

Figure 3. AMR Grid Structure (2D Example)

Inter-Grid Operations: Inter-grid operations are used to communicate solutions values along the adaptive grid hierarchy. Two primary inter-grid operations are *Prolongations*, defined from a coarser grid to a finer grid, and *Restrictions*, defined from a finer grid to a coarser grid.

Regriding: The regriding operation consists of three steps: (1) flagging regions needing refinement based on an application specific error criterion, (2) clustering flagged points, and (3) generating the refined grid(s). The regriding operation can result in the creation of a new level of refinement or additional grids at existing levels, and/or the deletion of existing grids.

3.1. Decomposing the Adaptive Grid Hierarchy

Key requirements while partitioning the adaptive grid hierarchy across processors are: (1) expose available data-parallelism; (2) minimize communication overheads by maintaining inter-level and intra-level locality; (3) balance overall load distribution; and (4) enable dynamic load re-distribution with minimum overheads. A balanced load distribution and efficient re-distribution is particularly critical for parallel AMR-

based applications as different levels of the grid hierarchy have different computational loads. In case of the Berger-Oliger AMR scheme for time-dependent applications, space-time refinement result in refined grids which not only have a larger number of grid elements but are also updated more frequently (i.e. take smaller time steps). The coarser grid are generally more extensive and hence its computational load cannot be ignored. Furthermore, the AMR grid hierarchy is a dynamic structure and changes as the solution progresses, thereby making efficient dynamic re-distribution critical.

4. Grid Adaptive Computational Engine (GrACE)

The adaptive "system sensitive" partitioning mechanisms presented in this paper have been integrated into the GrACE [1, 2] adaptive runtime system. GrACE is an object-oriented toolkit for the development of parallel and distributed applications based on a family of adaptive mesh-refinement and multigrid techniques. It provides a high-level programming abstraction and allows users to simply and directly express distributed computations on adaptive grids, and is built on a distributed dynamic data-management substrate.

4.1. GrACE Distributed Dynamic Data Management Substrate

The GrACE data management substrate implements a "semantically specialized" distributed shared memory and provides distributed and dynamic data-management support for large-scale parallel adaptive applications. The lowest layer of the infrastructure implements a Hierarchical Distributed Dynamic Array (HDDA). The HDDA provides array semantics to hierarchical and physically distributed data. HDDA objects encapsulate dynamic load-balancing, interactions and communications, and consistency management. The next layer adds application semantics to HDDA objects to implement application objects such as grids, meshes and trees. This layer provides an object-oriented programming interface for directly expressing multi-scale, multi-resolution AMR computations. The upper layers of the infrastructure provide abstractions, components and modules for method-specific computations.

4.1.1. *Hierarchical Distributed Dynamic Array*

The primitive data structure provided by the data-management infrastructure is an array which is hierarchical in that each element of the array can recursively be an array, and dynamic in that the array

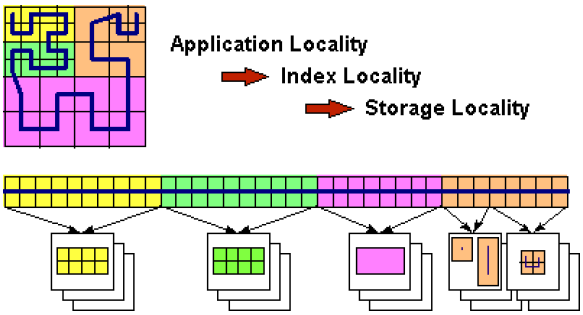

Figure 4. Distributed Storage of Dynamic Objects

can grow and shrink at run-time. The array of objects is partitioned and distributed across multiple address spaces with communication, synchronization and consistency transparently managed for the user. The lowest level of the array hierarchy is an object of arbitrary size and structure. The primary motivation for defining such a generalized array data-structure is that most application domain algorithms are formulated as operations on grids and their implementation is defined as operations on arrays. Such array based formulations have proven to be simple, intuitive and efficient, and are extensively optimized by current compilers. Providing an array interface to the dynamic data-structures allows implementations of new parallel and adaptive algorithms to reuse existing kernels at each level of the HDDA hierarchy. Like conventional arrays HDDA translates index locality (corresponding spatial application locality) to storage locality, and must maintain this locality despite its dynamics and distribution. The HDDA implementations is composed of a hierarchical index space and distributed dynamic storage and access mechanisms. The former is derived directly from the application domain using space-filling mappings [10], and the latter uses extensible hashing techniques [11]. Figure 4 shows the overall HDDA storage scheme.

5. System Sensitive Runtime Management Architecture

A block diagram of the adaptive, system sensitive runtime partitioning framework is shown in Figure 5. The framework consists of three key components - the system state monitoring tool, the capacity calculator and the system sensitive partitioner. In this framework, we first monitor the state of resources associated with the different computing nodes in the cluster, and use this information to compute their relative computational capacities. The relative capacities are then used by

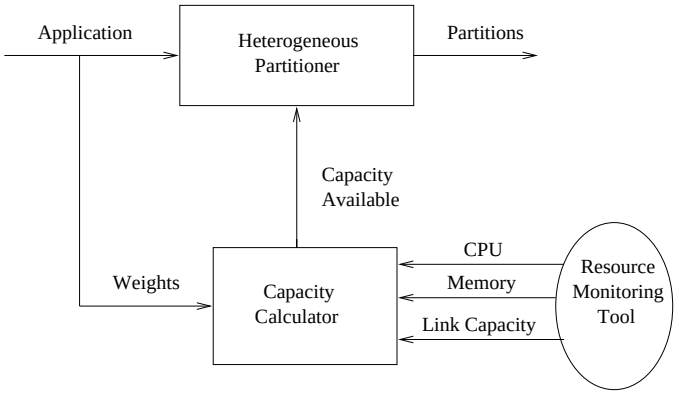

Figure 5. Block Diagram of the System Model

the heterogeneous system-sensitive partitioner for dynamic distribution and load-balancing. The different components are described below.

5.1. Resource Monitoring Tool

System characteristics and current state are determined at run-time using an external resource monitoring tool. The resource monitoring tool gathers information with minimum intrusion about the CPU availability, memory usage and link-capacity of each processor. This information is then passed to the Capacity Calculator as shown in Figure 5.

5.2. Capacity Metric

Using system information obtained from the resource monitoring tool, a relative capacity metric is computed for each processor as follows. Let us assume that there are K processors in the system among which the partitioner distributes the work load. For node k, let $\mathcal{P}_k$ be the percentage of CPU available , $\mathcal{M}_k$ the available memory, and $\mathcal{B}_k$ the link bandwidth, as provided by NWS. The available resource at k is first converted to a fraction of total available resources, i.e. $P_k = \mathcal{P}_k / \sum_{i=1}^{K} \mathcal{P}_i$, $M_k = \mathcal{M}_k / \sum_{i=1}^{K} \mathcal{M}_i$, and $B_k = \mathcal{B}_k / \sum_{i=1}^{K} \mathcal{B}_i$. The relative capacity C_k of a processor is then defined as the weighted sum of these normalized quantities

$$C_k = w_p P_k + w_m M_k + w_b B_k \tag{1}$$

where w_p, w_m, and w_b are the weights associated with the relative CPU, memory, and link bandwidth availabilities, respectively, such that $w_p + w_m + w_b = 1$. The weights are application dependent and reflect its computational, memory, and communication requirements. Note that

$\sum_{k=1}^{K} C_k = 1$. If the total work to be assigned to all the processors is denoted by L, then the work L_k that can be assigned to the kth processor can be computed as $L_k = C_k L$.

5.3. The System Sensitive Partitioner

The system sensitive partitioner, called *ACEHeterogeneous*, has been integrated into the GrACE runtime and provides adaptive partitioning and load-balancing support for AMR applications on heterogeneous systems. In GrACE, component grids in the adaptive grid hierarchy are maintained as a list of bounding boxes. A bounding box is a rectilinear region in the computational domain and is defined by a lower bound, upper bound and a stride (defined by the level of refinement) along each dimension. Every time the applications regrids, the bounding box list is updated and is passed to the partitioner for distribution and load balancing. A high level description of the partitioning process is presented below. The system interaction sequence diagram is presented in Figure 6.

- The relative capacities $C_k, k = 1, \ldots, K$ of the K processors over which the application is to be distributed and load balanced are obtained from the Capacity Calculator as shown in Figure 5.
- The total work L associated with the entire bounding box list is calculated.
- Using the capacity metric, the ideal work load L_k that can be assigned to the kth processor is computed.
- The bounding boxes are then assigned to the processors, with the kth processor receiving a total work load of W_k, which is approximately equal to L_k.
    - Both the list of bounding boxes as well as the relative capacities of the processors are sorted in an increasing order, with the smallest box being assigned to the processor with the smallest relative capacity. This eliminates unnecessary breaking of boxes (described below).
    - If the work associated with an available bounding box exceeds the work the processor can perform, a box is broken into two in a way that the work associated with at least one of the two boxes created is less than or equal to the work the processor can perform. While breaking a box the following constrains are enforced:

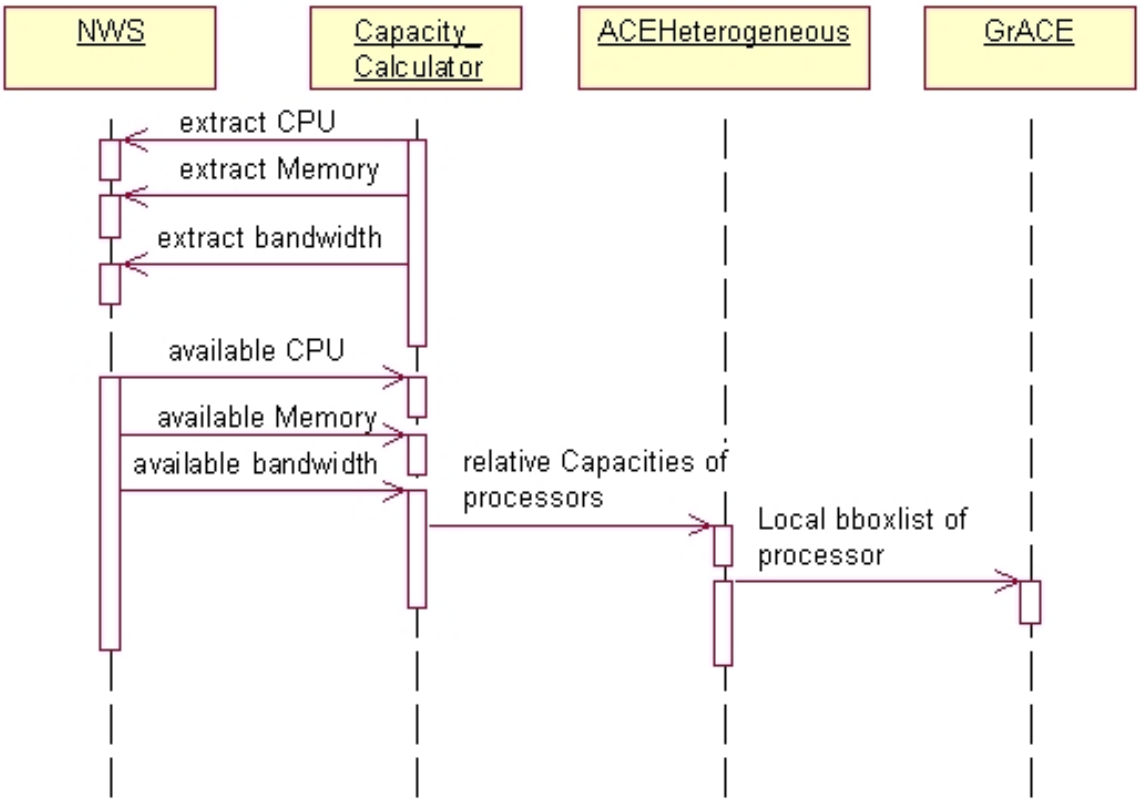

Figure 6. System Sequence Diagram.

  * Minimum box size: All boxes must be greater than or equal to this size. As a consequence of enforcing this constraint, the total work load W_k that is assigned to processor k may differ from L_k thus leading to a "slight" load imbalance. The amount of imbalance depends on the grid structure. We have found this to be less than 40% in our experiments.
  * Aspect ratio: The *aspect ratio* of a bounding box, defined as the ratio of the longest side to the shortest side. To maintain a good aspect ratio, a box is always broken along the longest dimension.

– The local output list of bounding boxes is returned to GrACE, which then allocates the grids to the particular processor.

6. Experimental Evaluation

A 3D compressible turbulence kernel executing on a linux based workstation cluster is used to experimentally evaluate the adaptive, system sensitive partitioning framework. The kernel solves the Richtmyer-Meshkov instability, and uses 3 levels of factor 2 refinement on a base mesh of size 128x32x32. The cluster consisted of 32 nodes interconnected by fast ethernet (100MB). The experimental setup consisted of a synthetic load generator and an external resource monitoring system. The evaluation consisted of comparing the runtimes and load-balance generated for the system sensitive partitioner with those for the default space-filling curve based partitioning scheme provided by GrACE. This latter scheme assumes homogeneous processors and performs an equal distribution of the workload on the processors. The adaptivity of the system sensitive partitioner to system dynamics, and the overheads of sensing system state were also evaluated.

6.1. Experimental Setup

6.1.1. *Synthetic Load Generation*

In order to compare the two partitioning schemes, it is important to have an identical experimental setup for both of them. Hence, the experimentation was be performed in a controlled environment so that the dynamics of the system state was the same in both cases. This was achieved using a synthetic load generator to load processors with artificial work. The load generator decreased the available memory and increased CPU load on a processor, thus lowering its capacity to do work. The load generated on the processor increased linearly at a specified rate until it reached the desired load level. Note that multiple load generators were run on a processor to create interesting load dynamics.

6.1.2. *Resource Monitoring*

We used the Network Weather Service (NWS) [3] resource monitoring tool to provide runtime information about system characteristics and current system state in our experiments. NWS is a distributed system that periodically monitors and dynamically forecasts the performance delivered by the various network and computational resources over a given time interval. In our experiments, we used NWS to monitor the fraction of CPU time available for new processes, the fraction of CPU available to a process that is already running, end-to-end TCP network bandwidth, and free memory. NWS has been engineered to be as non-intrusive as possible with typical CPU utilization less than 3% and a memory requirement of about 3300 KB [3].

6.1.3. *System Sensitive Load Distribution*
The following example illustrates the operation of the system sensitive partitioner. Consider a cluster with four nodes with the synthetic load generator used to load two of the machines. Using the current system state provided by NWS, the capacity calculator computes the relative capacities C_1, C_2, C_3 and C_4 as approximately 16%, 19%, 31% and 34% respectively. In these calculations, the three system characteristics, viz. CPU, memory, and link bandwidth, were assumed to be equally important to the application, i.e. $w_p = w_m = w_b = 1/3$. The relative capacities are then fed to the ACEHeterogeneous partitioner, which uses them to partition the overall work load L among the four processors. In this case the four processors are assigned work loads of $L_1 = 0.16L$, $L_2 = 0.19L$, $L_3 = 0.30L$ and $L_4 = 0.34L$, respectively. The partitioner appropriately assigns boxes (breaking large boxes if necessary) to processors to satisfy this distribution.

6.1.4. *Dynamic Load Sensing*
The system sensitive partitioner queries NWS at runtime to sense system load, computes the current relative capacities of the processors and distributes the workload based on these capacities. The sensing frequency depends on the dynamics of the cluster, and the overheads associated with querying NWS and computing relative capacities, and has to be chosen to balance the two factors. More frequent sensing (every few regrid steps) allows the system to adapt to rapid changes in system load but increases the sensing overheads. On the other hand, infrequent sensing (a few times during application execution) reduces these overheads but may increase overall execution time by preventing system from reacting to the cluster's load dynamics. Our experiments show that the overhead of probing NWS on a node, retrieving its system state, and computing its relative capacity is about 0.5 secs.

6.2. Experimental Results

6.2.1. *Application performance improvement*
The total application execution time using system sensitive partitioning and the default non-system sensitive partitioner is plotted in Figure 7. The percentage improvements are listed in Table 1. In this experiment, the application was run under the similar load conditions using the two partitioners. We calculate the relative capacities of the processors once before the start of the simulation. System sensitive partitioning reduced execution time by about 18% in the case of 32 nodes. We believe the improvement will be more significant in the case of larger cluster and in cluster with greater heterogeneity and load dynamics. Furthermore,

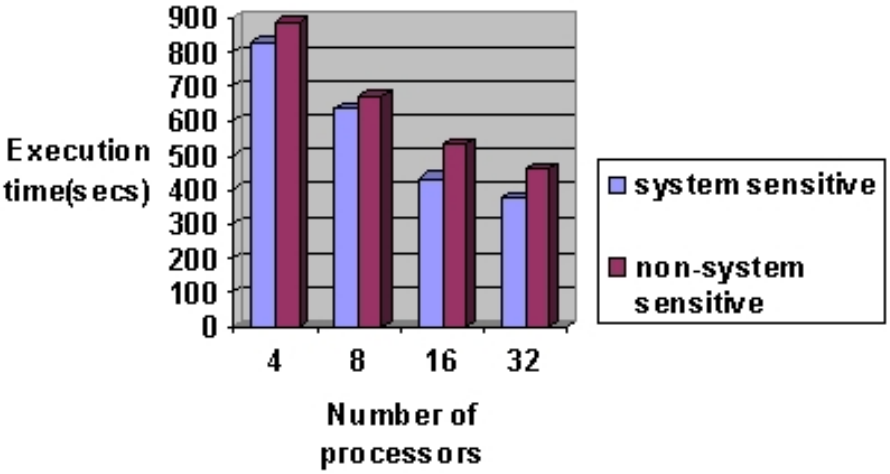

Figure 7. Application execution time for the system sensitive and default (non-system sensitive) partitioning schemes

Table I. *Percentage improvement that the system-sensitive partitioner yields over the default partitioner.*

Number of Processors	Percentage Improvement
4	7%
8	6%
16	18%
32	18%

increasing the sensing frequency also improves performance as shown in a later experiment.

6.2.2. *Load balance achieved*

This experiment investigates the load assignments and the effective load balance achieved using the two partitioners. In this experiment the relative capacities of the four processors were fixed at approximately 16%, 19%, 31% and 34%, and the application regrid every 5 iterations. The load assignment for the GrACE default and the system sensitive (ACEHeterogeneous) partitioners are plotted in Figures 8 and 9 respec-

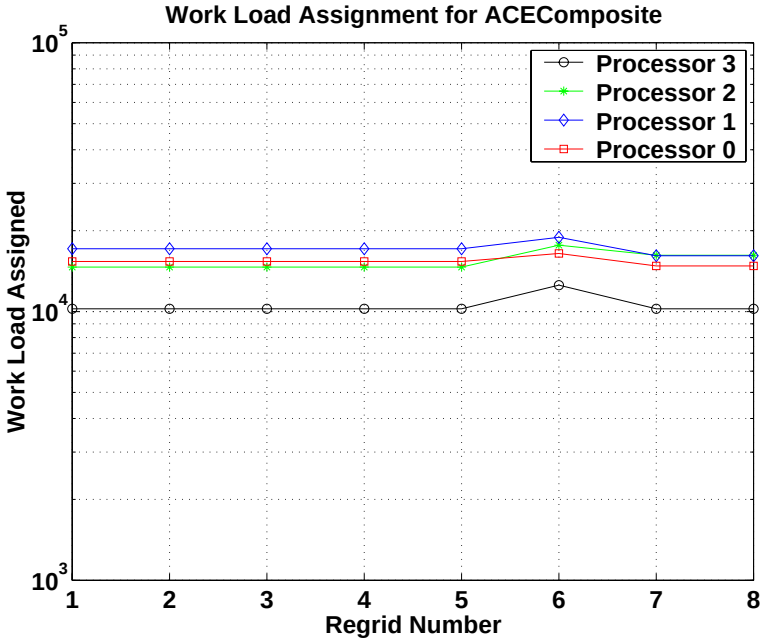

Figure 8. Work load assignments for the default partitioning scheme (Relative capacities of processors $0-3$ *equal to* $16\%, 19\%, 31\%,$ *and* 34% *respectively).*

tively. As expected, the GrACE default partitioner attempts to assign equal work to each processor irrespective of its capacity. The system sensitive partitioner however assigns work based on each processor's relative capacity.

Figure 10 shows the percentage of load imbalance for the system-sensitive scheme and the default scheme. For the kth processor, the load imbalance I_k is defined as

$$I_k = \frac{|W_k - L_k|}{L_k} \times 100 \quad \% \tag{2}$$

As expected, the GrACE default partitioner generates large load imbalances as it does not consider relative capacities. The system sensitive partitioner produces smaller imbalances. Note that the load imbalances in the case of the system sensitive partitioner are due to the constraints (minimum box size and aspect ratio) that have to be satisfied while breaking boxes.

6.2.3. *Adaptivity to Load Dynamics*

This experiment evaluates the ability of the system sensitive partitioner to adapt to the load dynamics in the cluster, and the overheads involved in sensing its current state. In this experiment, the synthetic

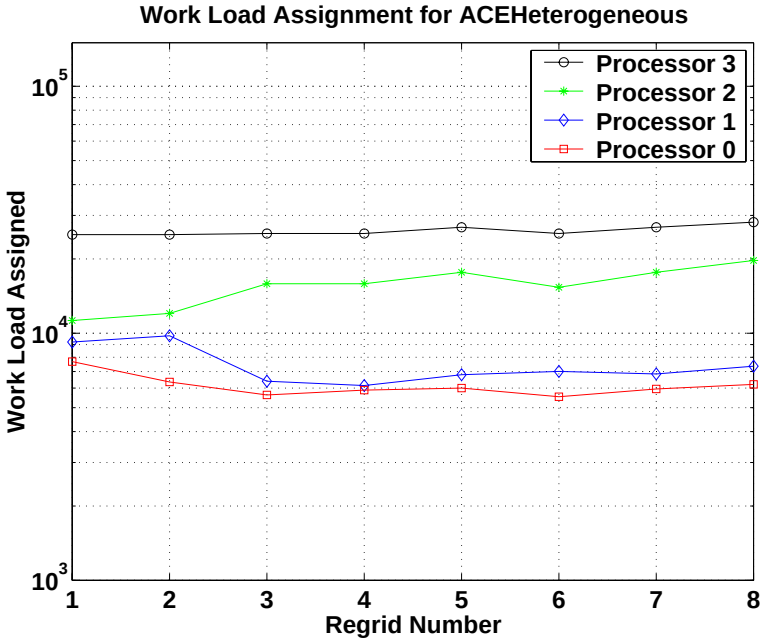

Figure 9. Work-load assignments for the ACEHeterogeneous partitioning scheme (Relative capacities of processors $0-3$ *equal to* $16\%, 19\%, 31\%,$ *and* 34% *respectively).*

load generator was used on two of the processors to dynamically vary the system load. The load assignments at each processor was computed for different sensing frequencies. Figure 11 shows the load assignment in the case where NWS was queried once before the start of the application and two times during the application run. The figure also shows the relative capacities of the processors at each sampling. It can be seen that as the load (and hence the relative capacities) of the processors change, the partitioning routine adapts to this change by distributing the work load accordingly. Also note that as the application adapts, the total work load to be distributed varies from one iteration to the next. As a result, the work load assigned to a particular processor is different in different iterations even though the relative capacity of the processor does not change.

Tables 2 and 3 illustrate the effect of sensing frequency on overall application performance. The synthetic load dynamics are the same in each case. Table 2 compares the application execution times for the cases where the system state is queried only once at the beginning and where it is queried every 40 iterations. It can be seen that dynamic runtime sensing significantly improves application performance. Table 3 presents application run time for different sensing frequencies - i.e. sensing every $10, 20, 30$ and 40 iterations. Figures 12, 13, 14 and 15 show

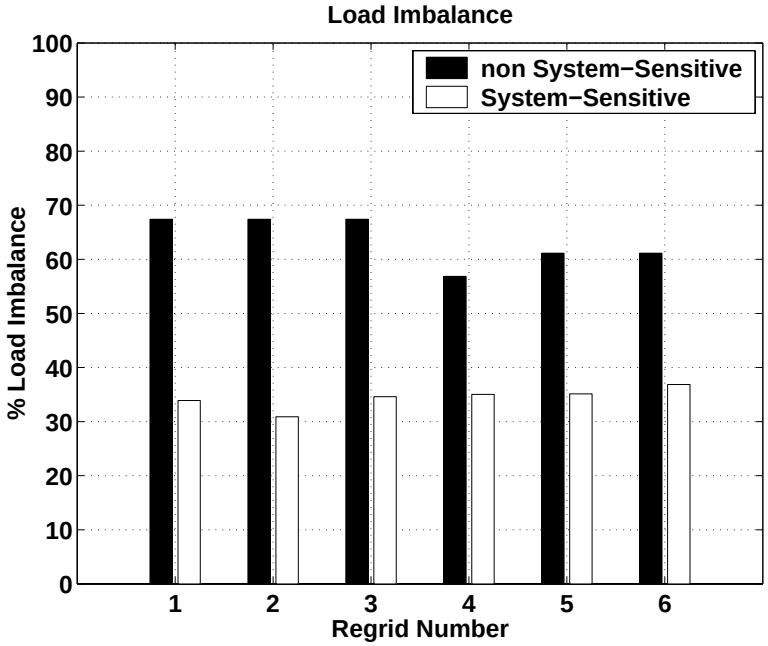

Figure 10. Load imbalance for the system sensitive and default partitioners

Table II. *Comparison of execution times using static and dynamic sensing*

Number of Processors	Execution time with Dynamic Sensing (secs)	Execution time with Sensing only once (secs)
2	423.7	805.5
4	292.0	450.0
6	272.0	442.0
8	225.0	430.0

the relative processor capacities and load assignments for each case. The best application performance is achieved for a sensing frequency of 20 iteration. This number largely depends on the load dynamics of the cluster and the sensing overheads.

7. Discussion of Results

It was shown through experiments that the system-sensitive partitioning reduced execution time by about 18% in the case of 32 nodes

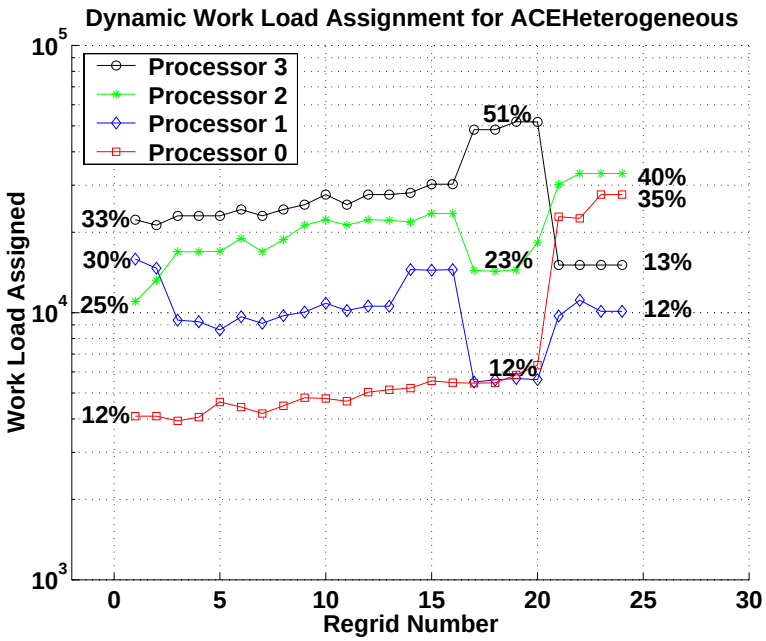

Figure 11. Dynamic load allocation using the system sensitive partitioner

Table III. *Execution time for a four processor run when NWS is probed at different iterations*

Frequency of calculating capacities	Execution time (secs)
10 iterations	316
20 iterations	277
30 iterations	286
40 iterations	293

as compared to the default partitioning scheme. We believe that the improvement will be more significant in the case of a larger cluster and in cluster with greater heterogeneity and load dynamics. The system-sensitive partitioning scheme also distributes work load in proportion to the relative capacities of the processors. Furthermore, the load imbalance associated is up to 45% lower than that of the default (non system-sensitive) partitioning scheme.

Through dynamic sensing, the system-sensitive partitioner adapts to the load dynamics of the cluster. As the load (and hence the relative capacities) of the processors change, the partitioning routine adapts

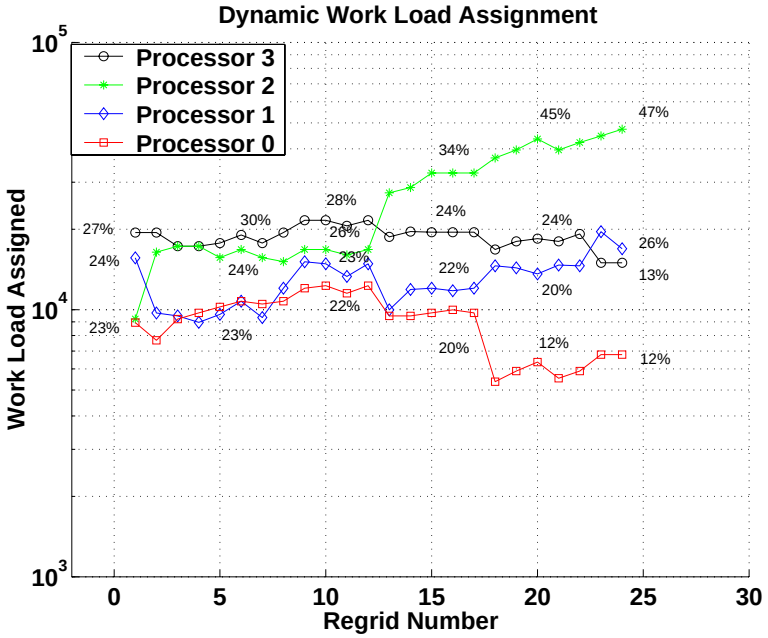

Figure 12. Dynamic load allocation for a system state sensing frequency of 10 iterations

to this change by distributing the work load accordingly. Dynamic runtime sensing also improved application performance by as much as 45% compared to sensing only once at the beginning of the simulation. The sensitivity of application performance to the frequency of sensing the system state was also studied. In our experimental setup sensing very frequently adversely effected the application performance. The best application performance was achieved for a sensing frequency of 20 iterations. The frequency of sensing largely depends upon the load dynamics of the cluster.

8. Conclusions and Future Work

This paper presented a system sensitive partition and load-balancing framework for distributed adaptive (AMR) applications in heterogeneous cluster environments. The framework uses system capabilities and current system state to appropriately distribute the application and balance load. System state is dynamically monitored using the Network Weather Service (NWS), and is used to compute current relative capacities of the computing nodes. These capacities are then used for partitioning and load balancing. An experimental evaluation of the

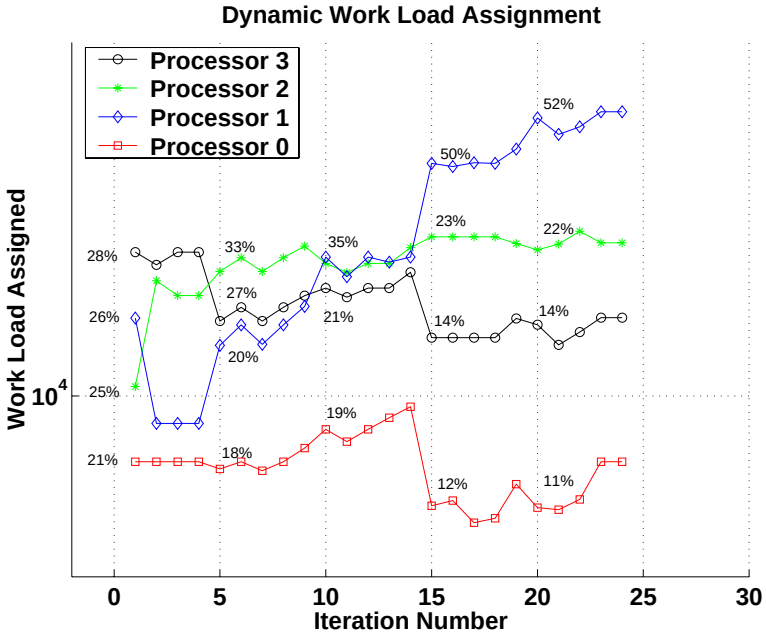

Figure 13. Dynamic load allocation for a system state sensing frequency of 20 iterations

framework was presented using a 3D AMR CFD kernel executing on a linux based cluster. A synthetic load generator was used to load the nodes in the cluster for the experiments. The experiments showed that dynamic system sensitive partitioning can significantly improve the load balance achieved and overall application performance. Furthermore, application performance is sensitive to the frequency of sensing of the system state. This frequency has to be chosen based on the sensing overheads and the cluster load dynamics.

We are currently working with a more careful choice of weights w_p, w_m, and w_b that will adequately reflect the computational needs of the application. Recall that in our scheme, w_p, w_m, and w_b are the weights associated with the relative CPU, memory, and link bandwidth availabilities, respectively, subject to the constraint that $w_p + w_m + w_b = 1$. In our experiments, although we have chosen the weights to be equal, they can in fact be chosen more carefully according to the computational needs of a particular application. For example, if the application is memory intensive, then a larger value can be assigned to w_m in comparison to w_p and w_b.

Although the system-sensitive partitioning scheme exhibits a lower load-imbalance compared to the default scheme (Figure 10), the load-imbalance of the former can be reduced even further. A primary cause

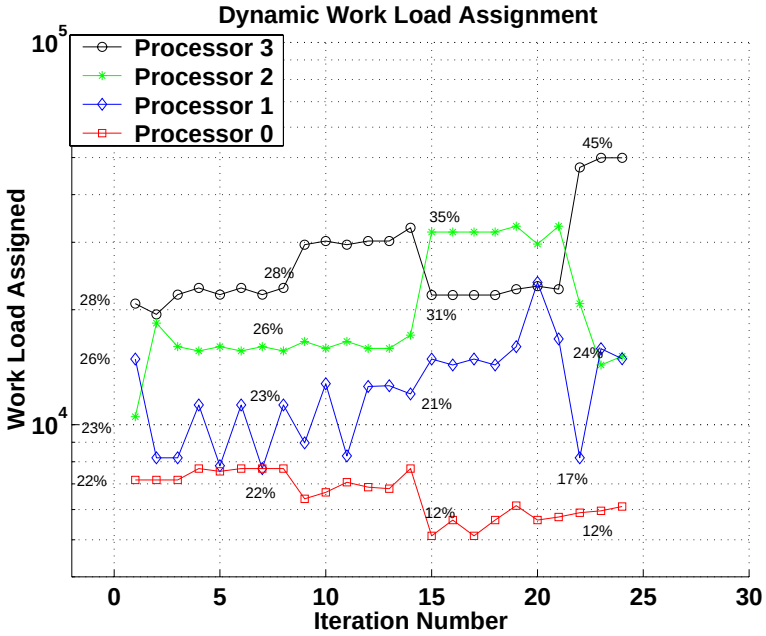

Figure 14. Dynamic load allocation for a system state sensing frequency of 30 iterations

of load-imbalance in the ACEHeterogeneous scheme can be attributed to the fact that the bounding box is cut only along the longest axis. If the box is instead cut along more axes, it could lead to finer partitioning granularity and hence better work assignments, which would in turn reduce the load-imbalance.

References

1. M. Parashar and J. C. Browne, "*System Engineering for High Performance Computing Software: The HDDA/DAGH Infrastructure for Implementation of Parallel Structured Adaptive Mesh Refinement*," IMA Volume 117: Structured Adaptive Mesh Refinement Grid Methods, IMA Volumes in Mathematics and its Applications. Springer-Verlag, pp. 1-18, 2000.
2. M. Parashar and J. C. Browne, "*On Partitioning Dynamic Adaptive Grid Hierarchies*," Proceedings of the 29th Annual Hawaii International Conference on System Sciences, January, 1996.
3. R. Wolski, N. T. Spring and J. Hayes, "*The Network Weather Service: A Distributed Resource Performance Forecasting Service for Metacomputing*," Future Generation Computing Systems, 1999.
4. M. J. Berger and J. Oliger "*Adaptive Mesh Refinement for Hyperbolic Partial Differential Equations*," Journal of Computational Physics, pp. 484-512, 1984.
5. S. M. Figueira and F. Berman "*Mapping Parallel Applications to Distributed Heterogeneous Systems*, " UCSD CS Tech Report # CS96-484, June 1996.

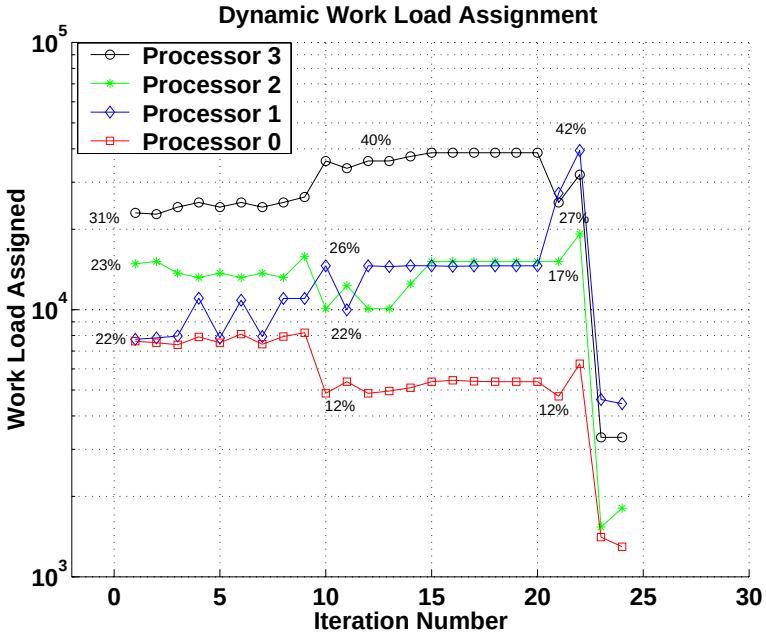

Figure 15. Dynamic load allocation for a system state sensing frequency of 40 iterations

6. J. Watts, M. Rieffel and S. Taylor "*Dynamic Management of Heterogeneous Resources*," High Performance Computing, 1998.
7. M. Maheswaran and H. J. Seigel "*A Dynamic Matching and Scheduling Algorithm for Heterogeneous Computing Systems*," 7th IEEE Heterogeneous Computing Workshop (HCW '98), Mar. 1998, pp. 57-69.
8. M. Shee "*Evaluation and optimization of load balancing/distribution techniques for dynamic adaptive grid hierarchies*," M.S. Thesis, Electrical and Computer Engineering Department, Rutgers University, May 2000.
9. M. Ripeanu "*Issues of Running Large Scientific Applications in a Distributed Environment*," M.S. Thesis, Computer Science Department, University of Chicago, November 2000.
10. H. Sagan, Space-filling curves, Springer-Verlag, 1994.
11. R. Fagin, Extendible Hashing - A Fast Access Mechanism for Dynamic Files, *ACM TODS*, **4**:315–344, 1979.
12. P. H. Andersen, J. K. Antonio "*Implementation and Utilization of a Heterogeneous Multicomputer Cluster for the Study of Load Balancing Strategies*," Proceedings of The Seventh Institute of Electrical and Electronics Engineers, Inc. (IEEE) International Symposium on High Performance Distributed Computing, 1998.
13. W. Cai, F L. Bu-Sung, and A. Heng "*A Simulation Study of Dynamic Load Balancing for Network-based Parallel Processing*," in Proceedings of 1997 International Symposium on Parallel Architectures, Algorithms and Networks (I-SPAN'97), pp. 383-389, IEEE Computer Society Press, Taipei, 14-16 Dec. 1997.

14. T. Gross, P. Steenkite, and J. Subhlok "*Adaptive Distributed Applications on Heterogeneous Networks*," in proceedings of 8th Heterogeneous Computing Workshop , April 1999.
15. W. Leinberg, G. Karypis, V. Kumar, and R. Biswas "*Load Balancing Across Near-Homogeneous Multi-Resource Servers*," proceedings of the 9th Heterogeneous Computing Workshop (HCW 2000).
16. S. Ramanathan "*Performance optimization for Dynamic Adaptive Grid Hierarchies*," M.S thesis, Electrical and Computer Engineering Department, Rutgers University, May 2001.
17. J. Steensland, S. Chandra, M. Thune, and M. Parashar "*Characterization of domain-based partitioners for parallel SAMR applications*," IASTED International Conference on Parallel and Distributed Computing and Systems, 2000.
18. G. Karypis, K. Schloegel, and V. Kumar "*ParMETIS: Parallel graph partitioning and sparse matrix ordering library*," Technical report, Dept. of Computer Science, University of Minnesota, Minneapolis, 1997.
19. S. Sinha, and M. Parashar "*System Sensitive Runtime Management of Adaptive Applications*," Proceedings of the 10th IEEE Heterogeneous Computing Workshop (HCW 2001), San Francisco, CA, April 2001.
20. H. Topcuoglu, S. Hariri, and M. Y. Wu "*Task Scheduling Algorithms for Heterogeneous Processors*," in proceedings of 8th Heterogeneous Computing Workshop , April 1999.
21. B. Hendrickson and K. Devine "*Design of Dynamic Load-Balancing Tools for Parallel Applications*," accepted to the International Conference on Supercomputing, Santa Fe, NM, May 2000.

Address for Offprints:
Rutgers, The State University of New Jersey
Piscataway, NJ 08855-8060
USA