System Sensitive Runtime Management of Adaptive Applications

Shweta Sinha Manish Parashar
The Applied Software Systems Laboratory, ECE/CAIP, Rutgers University
Piscataway, NJ 08855-8060
{*shwetas, parashar*}*@caip.rutgers.edu*

Abstract

This paper presents the design and evaluation of a system sensitive partitioning and load balancing framework for distributed adaptive grid hierarchies that underlie parallel adaptive mesh-refinement (AMR) techniques for the solution of partial-differential equations. The framework uses system capabilities and current system state to select and tune appropriate distribution parameters (e.g. partitioning granularity, load per processor) to maximize overall application performance.

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 the accurate solution of realistic models of important physical phenomena. These implementations however, lead to interesting challenges in dynamic resource allocation, data-distribution and load balancing, communications and coordination, and resource management.

Moving these applications to dynamic and heterogeneous networked computing environments introduces a new level of complexity. These environments require the selection and configuration of application components based on the availability and state of the resources. However, the complexity and heterogeneity of 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 a system sensitive partitioning and load balancing framework for distributed adaptive grid hierarchies that underlie parallel AMR (adaptive mesh-refinement) techniques for the solution of partial-differential equations. The framework uses system capabilities and current system state to select and tune appropriate distribution parameters. Current system state is obtained using a resource monitoring tool called NWS (Network Weather Service) [2]. The current system state of the computational nodes is then used to compute the relative computational capacities of each of the nodes. The relative capacities are then 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) infrastructure [1], and provides system sensitive partitioning/load-balancing support for AMR applications. It is known as the ACEHeterogeneous partitioning scheme.

The rest of this paper is organized as follows: Section 2 discusses related work, Section 3 describes the adaptive grid structure defined by hierarchical adaptive mesh-refinement techniques, Section 4 talks about GrACE, Section 5 describes the system architecture of the system sensitive framework, Section 6 describes the simulation setup and experimental results to evaluate how well the system-sensitive partitioner works, and Section 7 presents some concluding remarks.

2. Related Work

A lot of work has been done in the area of load balancing on a heterogeneous network. Some work has also been done to map a *static* application to a *heterogeneous* environment when the environment is dynamic. The authors in [4] have defined an algorithm which determines the best al-

Reference	Environment State	Application State
S. M. Figueira, et. al	static	static
J. Watts, et. al	dynamic	static
M. Maheswaran, et. al	dynamic	static
R. Wolski, et. al	dynamic	static
ACEHeterogeneous	static	dynamic

Table 1. *Summary of the simulation set-ups used by various systems.*

location of tasks to resources in heterogeneous systems. It however does not take into account the dynamism in the environment and uses a static application. Reference [5] has also presented techniques for dynamic load balancing in heterogeneous computing environments. They have also shown that performance improves when the computers capacities' are calculated dynamically. [6] has proposed a new dynamic algorithm, called the hybrid remapper, for improving the initial static matching and scheduling. The hybrid-remapper uses the run-time values that become available for subtask completion times and machine availabilities during application execution time. They have also used a static application. Reference [2] talks about finding a suitable match in a heterogeneous network when the network conditions are changing.

What is unique to our approach is the fact that we will take into account the heterogeneous network as well as the dynamism in the application.

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 and 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 [3] is shown in Figure [1].

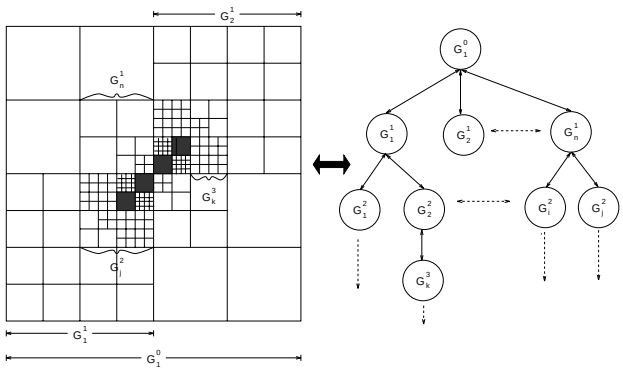

Figure 1. *Adaptive Grid Hierarchy - 2D (Berger-Oliger AMR Scheme)*

Operation on the hierarchy defined by this 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 difference operator.

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

The AMR grid hierarchy is a dynamic structure and changes as the solution progresses, thereby making efficient dynamic re-distribution critical.

The "system-sensitive" partitioner load-balances the computational load amongst the nodes based on the current system state of the computational nodes.

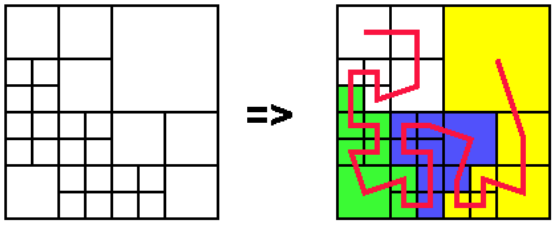

Figure 2. Hierarchical Space-filling Mappings

4. GrACE

The ACEHeterogeneous partitioning scheme has been integrated into GrACE. GrACE [1] is an approach to distributing AMR grid hierarchies. It is an object-oriented toolkit for the development of parallel and distributed applications based on a family of adaptive mesh-refinement and multigrid techniques. GrACE is built on a " semantically specialized" distributed shared memory substrate that implements a hierarchical distributed dynamic array.

4.1. Unified Distributed Dynamic Data Management for Interactive AMR

Its primary objective is to provide a distributed and dynamic data-management substrate to support development, implementation and interactive execution of 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 components and modules for visualization, interaction and method-specific computations.

4.2. Hierarchical Distributed Dynamic Array

The primitive data structure provided by the unified 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 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 must translate index locality (corresponding spatial application locality) to storage locality, and must maintain this locality despite its dynamics and distribution. Applying "separation of concerns" to the HDDA design , it is decomposed into two components; hierarchical index spaces, and distributed dynamic storage and access. These components are described below.

4.3. Hierarchical Index Spaces:

The hierarchical extendible index space component of the HDDA is derived directly from the application domain using space-filling mappings, which are computationally efficient, recursive mappings from N-dimensional space to 1-dimensional space (see Figure [2]). The solution space is first partitioned into segments. The space filling curve then passes through the midpoints of these segments. The mapping functions are computationally efficient and consist of logical bit-interleaving operation on the N-dimensional coordinates. Space filling mappings encode application domain locality and maintain this locality through expansion and contraction. The self-similar or recursive nature of these mappings can be exploited to represent a hierarchical structure and to maintain locality across different levels of the hierarchy. The hierarchical index-space is used by the HDDA as the basis for application domain partitioning, as a global address space for allocating storage to application objects, as a global name-space for name resolution, and for communication scheduling.

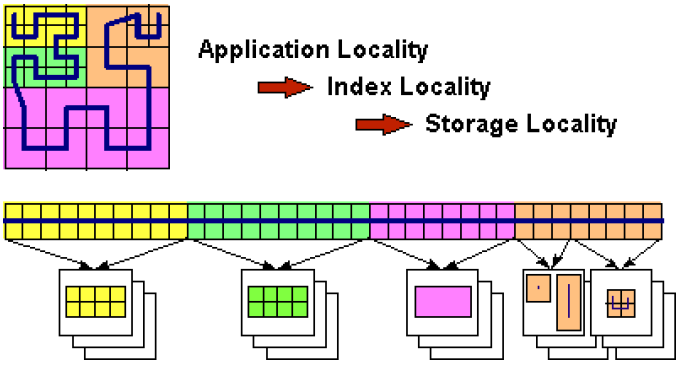

Figure 3. Distributed Storage of Dynamic Objects

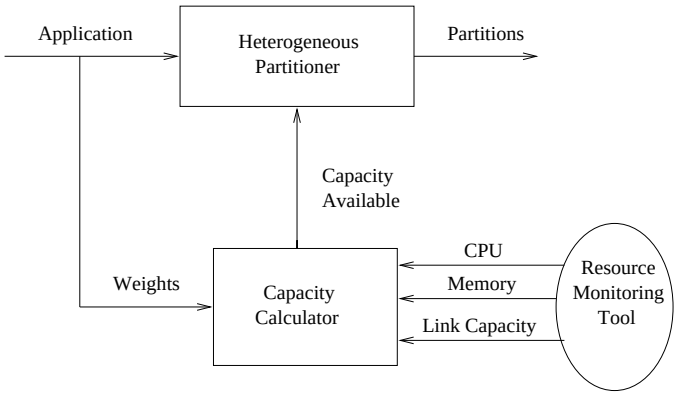

Figure 4. *Block Diagram of the System Model*

4.4. Distributed, Dynamic Storage and Access:

Data storage is implemented using extendible hashing techniques with contractions of the hierarchical index-space indices serving as hash keys. Extendible hashing provides efficient management mechanisms for dynamic data-bases. Spans of the hash keys space are mapped to units of storage called hash buckets and expansion and contraction of the key space are handled efficiently by splitting and merging these buckets. These operations are local involving at most two buckets. As spans of the index space are mapped to contiguous storage within a bucket by the hashing scheme, index locality (which encodes applications domain locality) is translated into storage locality. Data locality is preserved without copying. Partitioning of the applications domain and associated distribution of HDDA objects is achieved by partitioning the index space among processing elements and assigning ownership to HDDA buckets. Buckets are used as the units of communications and caching. The overall HDDA distributed dynamic storage scheme is shown in Figure [3].

5. System Architecture

We now describe the design of the system sensitive framework. In the system sensitive framework, we first monitor the state of resources associated with the different computing nodes and use this information to compute their relative computational capacities. The relative capacities are then used by the heterogeneous system-sensitive partitioner which has been integrated into GrACE for dynamic distribution and load-balancing as shown in Figure [4[.

5.1. Resource Monitoring Tool

System characteristics and state are determined at runtime using a resource monitoring tool. The resource monitoring tool gathers information 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 [4] and discussed below.

5.2. Capacity Metric

Using system information obtained with the resource monitoring tool, a relative capacity metric is computed for each processor using a linear model. 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

$$C_k = w_p P_k + w_m M_k + w_b B_k \qquad (1)$$

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 calculated as $L_k = C_k L$.

5.3. ACEHeterogeneous Partitioning Routine

As discussed earlier, the ACEHeterogeneous partitioning routine has been integrated into the GrACE infrastruc-

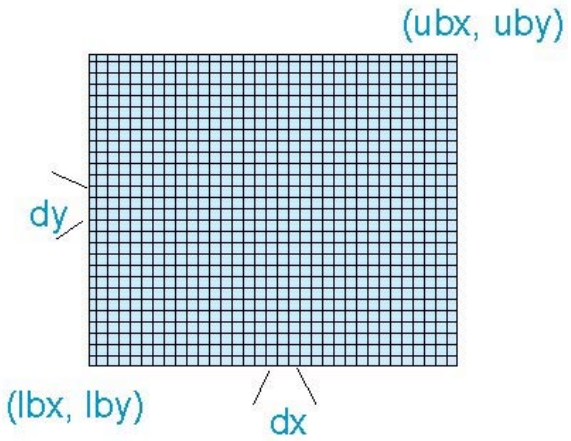

Figure 5. *Diagram of a 2-D Bounding Box*

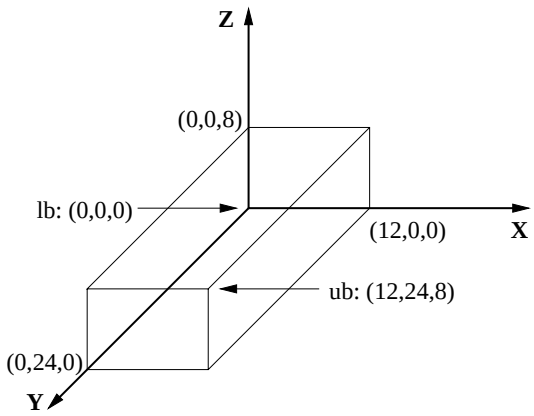

Figure 6. *Diagram of a 3-D Bounding Box. The lower and upper bounds are denoted by lb and ub, respectively.*

ture and provides load-balancing support for AMR applications. In GrACE, the grids in the adaptive grid hierarchy are maintained in the form of a bounding box list. A 2-D bounding box is shown in Figure [5]. The bounding box is an abstraction which addresses a region in the computational domain. At every timestep of the simulation, the partitioning routine gets a list of bounding boxes as input from GrACE. The partitioning routine then partitions this bounding box list.

A high level description of the partitioning process comprises of the following steps below:

- The relative capacities $C_k, k = 1, \ldots, K$ of the K processors on which the application will run are obtained from the Capacity Calculator as shown in Figure [4].
- The total work L associated with the bounding box list is calculated.
- Using the capacity metric, the ideal work load L_k that can be assigned to the kth processor is calculated.
- 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.
- At any point, if the work associated with a bounding box exceeds the work the processor can perform, then it is broken into two in a way that the work associated with at least one of the two broken boxes is less than or equal to the work the processor can perform.
- Since GrACE has a limit on the minimum size of a box, the bounding boxes have to be broken under this constraint. As a consequence, at times, 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 *aspect ratio* of a bounding box, defined as the ratio of the longest side to the shortest side, is also taken into account while breaking the box. We consider only the longest axis of the bounding box and break along that direction. For example, consider a box with the coordinates of the lower and upper bounds as (0,0,0) and (12,24,8) respectively, as shown in Figure [6]. In this Figure, since the length of the bounding box is longest in the y direction, we break the box along the xz plane. Breaking the box along any other axis would result in bounding boxes of very uneven sizes [7].
- 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 and hence reduces the associated overhead.
- The local output list of bounding boxes is returned to GrACE which then allocates the work to the particular processor.

6. Simulation Setup and Experimental Results

A 3D WAVEAMR application was used for our simulations. To evaluate the performance of this system sensitive approach, we used the overall execution time as a metric. We compared the execution time of the ACEHeterogeneous partitioning scheme with ACEComposite partitioning scheme [1]. This is one of the schemes that is used by GrACE for partitioning. This scheme performs an equal distribution of the workload on the processors and does not

take into account the heterogeneity that is associated with the processors on which the application is running.

6.1. Artificial Load Generation

In order to compare the two partitioning schemes, it is important to have the same simulation environment for both of them. Hence, the simulation has to be performed in a controlled environment so that the system state is the same in both the simulations. To achieve this, we created an artificial load generator which can load a processor with artificial work. The load generator was used to decrease the memory and CPU availabilities of a processor, thus lowering its capacity to do any additional work. This is how the capacities of the processors were artificially varied in our simulations.

6.2. Resource Monitoring

As described in the system architecture, the system state of the processors is obtained using a resource monitoring tool. In our experimental setup, system characteristics and state are determined at run-time using the NWS resource monitoring tool [2].

The 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. NWS currently monitors 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 latency, end-to-end TCP network bandwidth, free memory, and the amount of space unused on a disk.

6.3. Load Distribution

As an example, we now describe the load distribution among two processors named Aguada and Bambolim. For both the partitioning schemes, the artificial load generator was run only on Aguada. The Capacity Calculator extracted the system state from NWS and used it to calculate the relative capacities C_1 and C_2 for both the processors using (1) above. We assumed that all three system characteristics, namely CPU, memory, and link bandwidth, were equally important to the application and hence we chose $w_p = w_m = w_b = 1/3$. The relative capacities were then used as input to the Heterogeneous partitioner. The processor Aguada was artificially loaded so that its relative capacity was $C_1 = 0.3$ (or 30%) and that of Bambolim was $C_2 = 0.7$ (or 70%). Since the Heterogeneous partitioner divides the work load L proportionately among the two processors based on the above capacities, Aguada and Bambolim can be assigned work loads of $L_1 = 0.3L$ and $L_2 = 0.7L$, respectively.

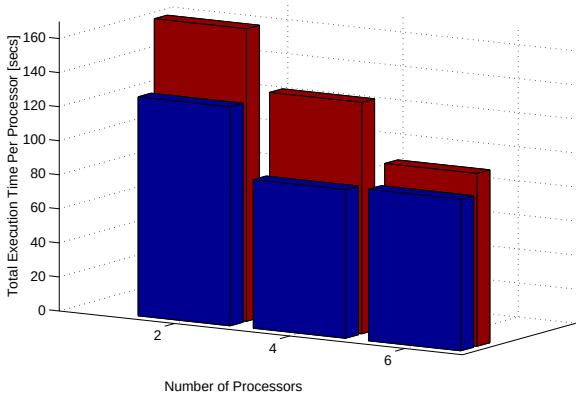

Figure 7. *Comparison of execution time of the two partitioning schemes*

6.4. Results

The total execution time under the ACEHeterogeneous partitioning scheme was obtained and plotted as the first row in the Figure [7]. The same application was then run under the same conditions using the ACEComposite partitioning scheme, and the total execution time under this scheme was plotted as the second row in Figure [7]. We can see that for the 2 processor run, the total execution time under the ACEHeterogeneous partitioning scheme was around 128 secs, whereas that under the ACEComposite partitioning scheme was around 171 secs. Therefore, a speedup was obtained by taking the heterogeneity of the processors into account before distributing the work load among them. Similar tests were also done on 4 and 6 processors shown in Figure [7]. The relative capacities for the 4 processor run were approximately $16\%, 19\%, 30\%$, and 34%, and that for the 6 processor run were approximately $11\%, 13\%, 14\%, 20\%, 20\%$, and 21%. The results from these runs (as shown in Figure [7]) also demonstrate the improvement the ACEHeterogeneous partitioning scheme yields over the ACEComposite partitioning scheme.

The work load assignment for the 4 processor run, under the ACEComposite partitioning scheme has been plotted in Figure [8]. The plot shows the work load assignment after every 5 iterations. The work assignments for the processors are similar to each other because the ACEComposite partitioning scheme ignores the relative capacities and distributes the work equally among the processors.

The work load assignment under the ACEHeterogeneous partitioning scheme has been plotted in Figure [9]. The relative capacities of the four processors were approximately 16%, 19%, 30% and 34%. In this case, we can see from the plot that the work load was assigned in proportion to the

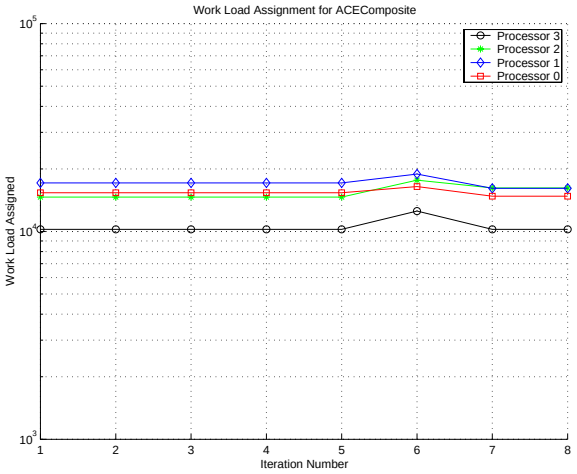

Figure 8. *Work-load assignments for the ACEComposite partitioning scheme. The relative capacities of the processors are* $16\%, 19\%, 30\%, 34\%$

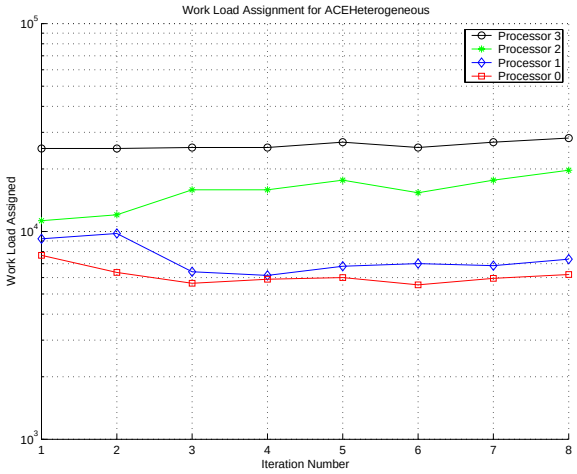

Figure 9. *Work-load assignments for the ACEHeterogeneous partitioning scheme. The relative capacities of the processors are* $16\%, 19\%, 30\%, 34\%$

relative capacities of the processors.

Figures [8] and [9] do not reflect the percentage load imbalance. For the kth processor, the load imbalance I_k is defined as

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

The percentage of load imbalance for ACEComposite and ACEHeterogeneous has been plotted in Figures [10] and [11] respectively. As expected, since the ACEComposite partitioning scheme distributes work equally regardless of the different relative capacities, the load imbalance in Figure [10] is higher than that in Figure [11]. Since a bounding box is cut only along a particular axis and also due to the minimum box size constraint, we do see some imbalance for the ACEHeterogeneous scheme as well, as shown in Figure [11]. In spite of this, the load imbalance associated with the ACEHeterogeneous partitioning scheme is significantly lower than that of the ACEComposite partitioning scheme.

7. Conclusions and Future Work

This paper presented a framework that uses the current system state of the computing nodes to distribute an application among the nodes. The framework monitors the availability of resources at the computing nodes, calculates the relative capacities of these nodes, and then assigns work in proportion to their capacities. It was shown through simulation that this scheme reduces the total execution time of the application and the load imbalance as compared to a scheme that does not take the relative capacities of the computing nodes into account. We are currently extending this work to accommodate a more careful choice of weights w_p, w_m, and w_b which will adequately reflect the computational needs of the application.

References

[1] Manish Parashar and James C. Browne, "*On Partitioning Dynamic Adaptive Grid Hierarchies*," Proceedings of the 29th Annual Hawaii International Conference on System Sciences, January, 1996.

[2] Rich Wolski, Neil T. Spring and Jim Hayes, "*The Network Weather Service: A Distributed Resource Performance Forecasting Service for Metacomputing*," Future Generation Computing Systems, 1999, www.cs.ucsd.edu/groups/hpcl/apples/hetpubs.html

[3] Marsha J. Berger and Joseph Oliger "*Adaptive Mesh Refinement for Hyperbolic Partial Differential Equations*," Journal of Computational Physics, pp. 484-512, 1984.

[4] Silvia M. Figueira and Francine Berman "*Mapping Parallel Applications to Distributed Heterogeneous Systems*, " UCSD CS Tech Report # CS96-484, June 1996.

[5] Jerrel Watts, Marc Rieffel and Stephen Taylor "*Dynamic Management of Heterogeneous Resources*," High Performance Computing, 1998.

[6] Muthucumaru Maheswaran and Howard Jay Seigel " *A Dynamic Matching and Scheduling Algorithm for*

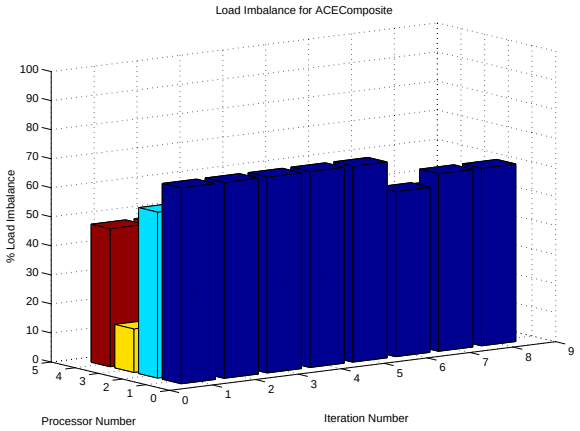

Figure 10. *Percentage imbalance for the ACEComposite partitioning scheme*

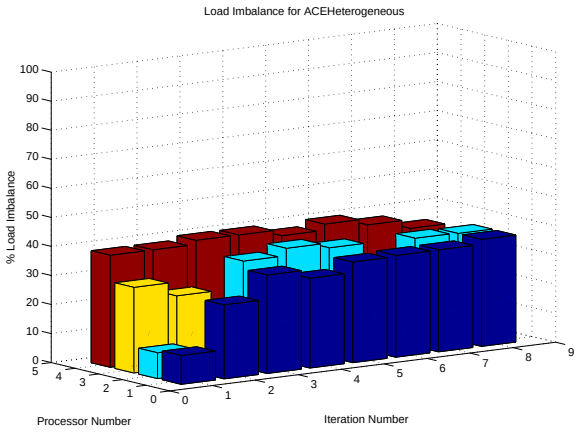

Figure 11. *Percentage imbalance for the ACE-Heterogeneous partitioning scheme*

Heterogeneous Computing Systems," 7th IEEE Heterogeneous Computing Workshop (HCW '98), Mar. 1998, pp. 57-69.

[7] Mausumi Shee "*Evaluation and optimization of load balancing/distribution techniques for dynamic adaptive grid hierarchies*," M.S. Thesis, Electrical and Computer Engineering Department, Rutgers University, 2000.

Biography

Shweta Sinha received her B.S. (high honors) degree from the Computer Science department at Rutgers University, and is currently pursuing an M.S. degree in the Electrical and Computer Engineering department at Rutgers University, New Jersey. Her areas of interest include Wired and Wireless Networking, Internet Security, and Distributed Computing.

Manish Parashar ...