Hierarchical Partitioning Techniques for Structured Adaptive Mesh Refinement (SAMR) Applications *

Xiaolin Li, Sivapriya Ramanathan, and Manish Parashar
The Applied Software Systems Laboratory
Department of Electrical and Computer Engineering
Rutgers, The State University of New Jersey
94 Brett Road, Piscataway, NJ 08854, USA
Email: {xlli, sivapriy, parashar}@caip.rutgers.edu

Abstract

This paper presents the design and preliminary evaluation of hierarchical partitioning and load-balancing techniques for distributed Structured Adaptive Mesh Refinement (SAMR) applications. The overall goal of these techniques is to enable the load distribution to reflect the state of the adaptive grid hierarchy and exploit it to reduce synchronization requirements, improve load-balance, and enable concurrent communications and incremental redistribution. The hierarchical partitioning algorithm (HPA) partitions the computational domain into subdomains and assigns them to hierarchical processor groups. Two variants of HPA are presented in this paper. The Static Hierarchical Partitioning Algorithm (SHPA) assigns portions of overall load to processor groups. In SHPA, the group size and the number of processors in each group is setup during initialization and remains unchanged during application execution. It is experimentally shown that SHPA reduces communication costs as compared to the Non-HPA scheme, and reduces overall application execution time by up to 41%. *The Adaptive Hierarchical Partitioning Algorithm (AHPA) dynamically partitions the processor pool into hierarchical groups that match the structure of the adaptive grid hierarchy. Initial evaluations of AHPA show that it can reduce communication costs by up to* 70%.

Keywords: *Dynamic Load Balancing, Hierarchical Partitioning Algorithm, Distributed Computing, Structured Adaptive Mesh Refinement*

*The work presented here was supported in part by the National Science Foundation via grants numbers ACI 9984357 (CAREERS), EIA 0103674 (NGS) and EIA-0120934 (ITR), and by DOE ASCI/ASAP (Caltech) via grant number PC295251.

1 Introduction

With the rapid growth in computing and communication technology, the past decade has witnessed a proliferation of powerful parallel and distributed systems and an ever-increasing demand for and practice of high performance computing [3, 6]. A key issue in parallel and distributed computing is the partitioning, balancing and scheduling of computational loads among processors to efficiently utilize the available computing, communication and storage resources, and to maximize overall performance and scalability. This is especially true in the case of dynamically adaptive applications such as those based on the adaptive mesh refinement methods, where the computational load of these applications changes as the application evolves.

In this paper, we present the design and preliminary evaluation of hierarchical partitioning and load-balancing techniques for distributed Structured Adaptive Mesh Refinement (SAMR) applications. Dynamically adaptive mesh refinement (AMR) [2] methods for the numerical solution of partial differential equations employ locally optimal approximations, and 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 methods offer the potential for accurate solutions of physically realistic models of complex physical phenomena. These implementations also lead to interesting challenges in dynamic resource allocation, data-distribution, and load balancing. Critical among these is the dynamic partitioning of the adaptive grid hierarchy at runtime to balance load, optimize communication and synchronization, minimize data migration costs, and maximize available parallelism.

Traditional distributed implementation of SAMR appli-

cations [1, 8, 9, 11] have used dynamic partitioning/load-balancing algorithms that view the system as a flat pool of (usually homogeneous) processors. These approaches are typically based on a global knowledge of the state of the adaptive grid hierarchy, and partition the grid hierarchy across the set of processors. Global synchronizations and communications is required to maintain this global knowledge and can lead to significant overheads on large systems. Furthermore, these approaches do not exploit the hierarchical nature of the grid structure and the distribution of communications and synchronizations in this structure.

The overall goal of the hierarchical partitioning algorithms (HPA) presented in this paper is to allow the distribution to reflect the state of the adaptive grid hierarchy and exploit it to reduce synchronization requirements, improve load-balance, and enable concurrent communications and incremental redistribution. These techniques partition the computational domain into subdomains and assigns these subdomains to dynamically configured hierarchical processor groups. Processor hierarchies and groups are formed to match natural hierarchies in the grid structure. In addition to providing good load-balance, this approach allows a large fraction of the communications required by the adaptive algorithms to be localized within a group. Furthermore, communications with different groups can proceed concurrently. Hierarchical partitioning also reduces the dynamic partitioning and data migration overheads by allowing these operations to be performed concurrently within different groups and incrementally across the domain.

Two variants of HPA are presented in this paper. The Static Hierarchical Partitioning Algorithm (SHPA) assigns portions of overall load to processor groups. In SHPA, the group size and the number of processors in each group is set in advance and remains unchanged during the execution. While SHPA is static in the sense that its group topology is unchanged during the execution, it does perform dynamic load balancing. To overcome the static nature of SHPA, we propose an Adaptive Hierarchical Partitioning Algorithm (AHPA) that dynamically partitions the processor pool into hierarchical groups that match the structure of the adaptive grid hierarchy. AHPA naturally adapts to the runtime behavior of SAMR applications. A preliminary evaluation of these two algorithms is presented. It is experimentally shown that SHPA reduces communication costs as compared to the Non-HPA scheme and results in a reduction in overall application execution time up to 41%. Furthermore, initial evaluations shows that AHPA reduces the communication cost up to 70%.

1.1 Related work

There exist a number of infrastructures that support parallel and distributed implementations of SAMR applications. Each such system represents a combination of design decisions in terms of algorithms, data structures, user interfaces, decomposition, mapping, and distribution and communication mechanisms. Table 1 summarizes a selection of the existing SAMR infrastructures and the partitioning approach used by them. Related work in hierarchical load-balancing is described below.

Table 1. Distributed SAMR Infrastructures

Infrastructure	Description
PARAMESH [9]	Extends serial code to parallel code based on partitioning tree representation of adaptive grid structure
SAMRAI [8]	Object oriented framework (based on LPARX) with patches mapped across processors at a level
BATSRUS [1]	Block-based approach with adaptation distributed over processors in computational pool in phases

Pollack [12] proposed a scalable hierarchical approach for dynamic load balancing in parallel and distributed systems and implemented a system, named PaLaBer (Parallel Load Balancer), on the Intel Paragon XP/S. It uses multi-level control for dynamic load balancing and for the communication manager. This hierarchical load balancer uses non-preemptive as well as preemptive process migration to balance load between the processors. However, the load balancing hierarchy is static in that once created the configuration remains fixed for the entire run. PaLaBer targets overall scheduling and load-balancing of tasks from multiple applications rather than dynamic load-balancing for adaptive applications such as SAMR. Compared to PalaBer, HPA strategy is more flexible and can be static or adaptive. Furthermore, HPA strategy addresses SAMR applications by taking into account the features of the computational domain and the adaptive nature of SAMR applications. Another related approach by Furuchi et al. [7] addressed load balancing for parallel OR-search programs. This load balancing system consists of a subtask generator that partitions a program into independent subtasks, and distributes them to the processing elements so as to balance workload. In this scheme, each subtask generator is in charge of a certain fixed number of processors, which form a processor group.

The rest of the paper is organized as follows. Section 2 provides a short introduction to SAMR and distributed SAMR implementations. Section 3 describes the hierarchical partitioning algorithm. The general HPA scheme is first presented and is followed by two variant, viz. Static HPA and Adaptive HPA. Experimental and simulation results are also presented and discussed. Conclusions are presented in Section 4.

2 Problem Description

Dynamically adaptive numerical techniques for solving differential equations provide a means for concentrating computational effort to appropriate regions in the computational domain. These techniques lead to more efficient and cost-effective solutions to time dependent problems exhibiting localized features. In the case of SAMR methods, this is achieved by tracking regions in the domain that require additional resolution and dynamically overlaying finer grids over these regions. These methods 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 of the AMR formulation by Berger and Oliger [2] is shown in Figure 1.

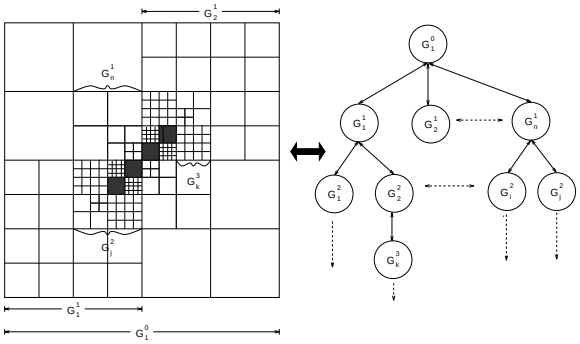

Figure 1. Adaptive Grid Hierarchy - 2D (Berger-Oliger AMR scheme)

Distributed implementations of SAMR applications partition the adaptive grid hierarchy across available processors, and operate on the local portions of this domain in parallel. The overall performance of these applications is thus limited by the ability to partition the underlying grid hierarchies at runtime to expose all inherent parallelism, minimize communication and synchronization overheads, and balance load. A critical requirement of the load partitioner is to maintain logical locality across partitions at different levels of the hierarchy and at the same level when they are decomposed and mapped across processors. The maintenance of locality minimizes the total communication and synchronization overheads. Distributed SAMR applications primarily require two types of communications:

Inter-level Communications: These communications are defined between component grids at different levels of the grid hierarchy and consist of prolongations (coarse to fine transfers) and restrictions (fine to coarse transfers). Inter-level communications require a gather/scatter type operation based on an interpolation or averaging stencil. These communications can lead to serialization bottlenecks for a naive decomposition of the grid hierarchy.

Intra-level Communication: Intra-level communications (also called ghost communications) are required to update the grid-elements along the boundaries of local portions of a distributed grid. These communications consist of near-neighbor exchanges based on the stencil defined by the difference operator. The communications are regular, and can be scheduled to overlap with computations on the interior region of the local portion of distributed grids.

The HPA strategy is based on the composite decomposition of the adaptive grid hierarchy that maintains domain locality [11]. This decomposition technique partitions the grid hierarchy such that all inter-level communication is local to a processor. This scheme uses space filling curves (SFC) [13], which are a class of locality preserving recursive mappings from n-dimensional space to 1-dimensional space. In HPA, after obtaining the composite representation of the adaptive grid hierarchy using SFC, we partition it and assign spans of the curve to processor groups in a hierarchical manner. This strategy takes advantages of the composite decomposition which reduces intra-level communications and localizes inter-level communication. Furthermore, it enables communications in different groups to proceed concurrently, localizes data-movement operations and can enable incremental redistribution.

3 Hierarchical Partitioning Algorithm

This section first presents the general HPA scheme and describes its operation. Two variants of the scheme, viz. Static and Adaptive HPA, are presented.

3.1 General HPA

The overall efficiency of parallel and distributed SAMR applications is limited by the ability to partition the underlying grid hierarchies at runtime to expose all inherent parallelism, minimize communication and synchronization overheads, and balance load. In most distributed implementations of SAMR [9, 10, 15], load scheduling and balancing is collectively done by all processors in the system and all processors maintain a global knowledge of the total workload. These schemes have the advantage of a better load balance. However these approaches require the collection and maintenance of global load information which makes them expensive, specially on large system. Partitioning in

these Non-HPA schemes [1] consists of the following steps:

- Global load information exchange and synchronization phase: All processors are engaged in this information exchange phase. After this phase, all the processors have a global view of the grid hierarchy.
- Load partitioning phase: All processors calculate the average load per processor and partition the grid hierarchy. This operation is done by each processor in the system.

The sequence of steps taking place in the Non-HPA scheme for partitioning and scheduling ghost communications is illustrated in the sequence diagram in Figure 2. Initially, all processors have the initial computational domain. Each processor partitions the domain into subdomains and assigns a subdomain to itself. During the load balancing phase, all processors synchronize and exchange their local domain information. At the end of this phase, every processor has a consistent global view of the domain. The partitioning algorithm then partitions the domain among the processors. After partitioning is complete, the processors migrate data that no longer belongs to their local subdomains. Each processor then schedules ghost communications based on its new local subdomain.

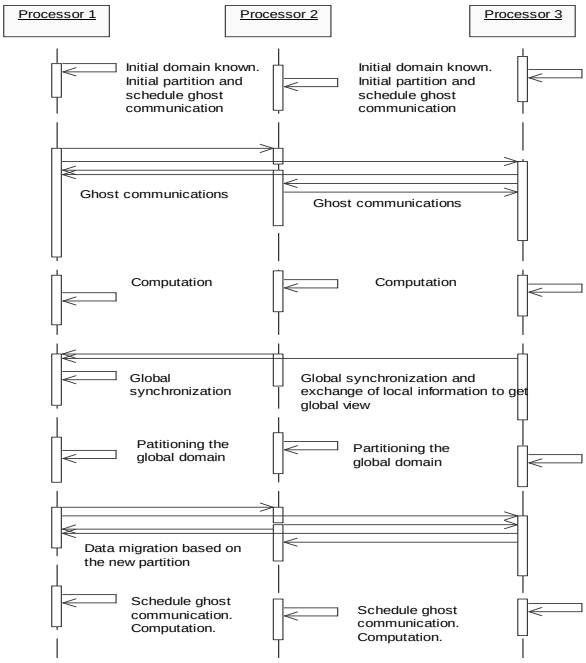

Figure 2. Sequence diagram of the Non-HPA scheme

[1] Note that a Non-HPA scheme can be viewed as a special case of HPA where there is only one group composed of all processors

In large parallel/distributed systems, the global information exchange and synchronization phase becomes a performance bottleneck. The HPA scheme presented in this paper does not propose a new partitioner, but a hierarchical partitioning strategy. The underlying partitioning scheme adopted is the composite decomposition method using space filling curve (SFC) technique [11, 13] as mentioned in Section 2. In this scheme, partitioning at different level is performed in parallel based on load information local to that level. Load is periodically propagated up the processor group hierarchy in an incremental manner. Furthermore communications are conducted in stages among the processors in a group hierarchically, rather than requiring communication and synchronization among all processors. This is achieved by dividing the processors into processor/compute groups as shown in Figure 3.

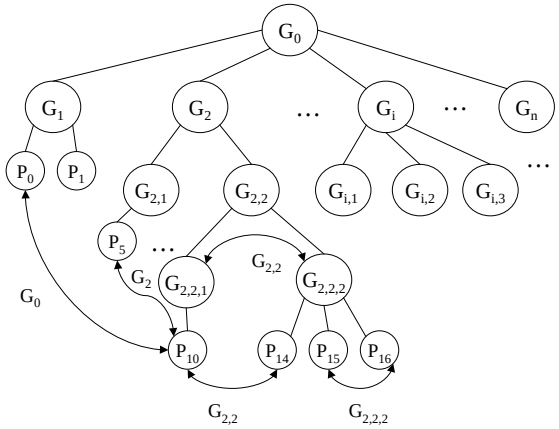

Figure 3. A general hierarchical structure of processor groups

Figure 3 illustrates a general hierarchical tree structure of processor groups, where, G_0 is the root level group (group level=0) and consists of all the processors. G_i is the $i-th$ group at the group level 1. Note that the processors form the leaves of the tree. The communication between processors is conducted through their closest common ancestor group which is their coordinater or master. For example, processors P_{10} and P_{14} have common ancestor groups G_0, G_2 and $G_{2,2}$. However their closest common ancestor group is $G_{2,2}$. Consequently their communication is via the group $G_{2,2}$ which is their coordinator or master. Similarly, communications between processors P_0 and P_{10} are via the group G_0.

In HPA, the partitioning phase is divided into sub-phases as follows.

- Local partitioning phase: The processors belonging to a processor group partition the group load based on a local load threshold and assign a portion to each processor within the group . Parent groups perform the partitioning among their children groups in a hierarchical manner.
- Global partitioning phase: The root group coordinator (group level 0) decides if a global repartitioning has to be performed among its children groups at the group level 1 according to the group threshold.

The pseudo-code for the new load balancing phase is given in Table 2.

The HPA scheme attempts to exploit the fact that given a group with adequate number of processors, and a carefully defined number of groups, the number of global partitioning phases can be minimized. The working step of the general HPA is illustrated by the sequence diagram in Figure 4.

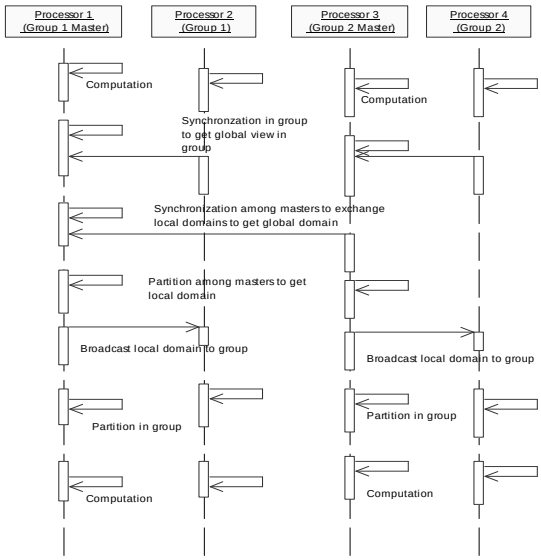

Figure 4. Sequence diagram of the HPA scheme

In this figure, we show a two level group hierarchy including the root group G_0. The hierarchical scheme first creates processor groups. After these groups are created and the initial grid hierarchy is setup, the group coordinators/masters partition the initial domain in the global partitioning phase. At the end of this phase the coordinators have a portion of the domain that is then partitioned among the processors in the group subtrees. Recursively, portions of the computational domain are partitioned further and finally assigned to individual processors at the leaves of the processor group hierarchy. This is the local partitioning phase.

3.2 Static HPA

In the Static HPA strategy, the group size and the group topology is defined at startup based on the available processors and the size of the problem domain. The static processor group hierarchy consists of two levels. It is static in the sense that once the group configuration is setup it will be fixed for the entire execution of the application. Even though it is static, SHPA does possess the basic advantages of the general HPA strategy. It localizes the load redistribution and balancing within processor groups and enables concurrent communication among processor groups. Note that, SHPA is still a dynamic load balancing algorithm [14], as load is dynamically redistributed within and across processor groups – only the processor group hierarchy remains static.

The load partitioning and assignment procedure is shown in Table 3. As described in the table, the number of groups, $N_{totalgroups}$, is defined at application startup. The load balancing phase in SHPA is similar to the steps in Table 2 with two group levels.

The Static HPA is implemented as part of the GrACE library [10]. The groups are created using communicators provided by the MPI library. Communication within groups is through intracommunicators while communication between processors belonging to different groups is through intercommunicators.

The Static HPA scheme is evaluated on the IBM SP2 cluster at University of California, San Diego. The application used in these experiments is a numerical relativity kernel (Wave3D) and belongs to the general class of SAMR applications. Wave3D solves a coupled set of partial differential equations. This kernel is a part of the Cactus numerical relativity toolkit [4].

The experiments measure the total execution time of the simulations using Static HPA and Non-HPA schemes. In Figure 5, we observe that, the maximum performance gain is obtained for 128 processors with 8 groups, a 41% overall execution time reduction as compared to the Non-HPA scheme. As shown by the evaluation, the benefits of SHPA depends on the appropriate selection of the number of processor groups, which in turn depends on the system and the application. The adaptive HPA scheme attempts to address the limitation by dynamically managing processor groups.

3.3 Adaptive HPA

In this section, we propose an Adaptive HPA strategy. In the Static HPA strategy, the total number of groups is predefined and keep unchanged throughout the execution of the application. In order to account for the application's runtime dynamics, the AHPA proposes an adaptive strategy. AHPA dynamically partitions the computational do-

Table 2. Load balancing phase in the general HPA

```
/* in the highest group composed of individual processors */
if(my_load > threshold) {
    do a local partition in a group;
}
grouplevel = highest level;
while(grouplevel > 0){
    if(group_load > group_threshold) {
        do a partition among children groups at grouplevel;
        broadcast new composite list through parent group;
    }
    grouplevel --;
}
begin computation; ...
```

Table 3. Load partitioning and assignment in Static HPA

Step 1. Use SFC to obtain the composite grid unit (CGU) list.

Step 2. Partition the CGU list into $N_{totalgroups}$ subdomains.

Step 3. Assign the load L_i on subdomain R_i to a group of processors G_i such that the number of processors NP_i in the group G_i is proportional to the load L_i, i.e., $NP_i = L_i/L_{sum} \times NP_{sum}$, where L_{sum} is the total size of load and NP_{sum} is the total number of processors.

Step 4. Partition the load portion L_i and assign the appropriate portion to the individual processor in the group G_i, for $i = 0, 1, ..., NP_{totalgroups} - 1$.

main into subdomains to match its current adaptations. The subdomains created may have unequal loads. The algorithm then assigns the subdomains to corresponding nonuniform hierarchical processor groups. The detailed steps are presented in Table 4. Note that the definition of processor groups may take into consideration the system architecture - for example, group size can be chosen to match the size of a SMP node in a SMP cluster.

As shown in Table 4, the AHPA scheme partitions the computational domain according to its refinement level. This partitioning scheme naturally matches the state of the grid hierarchy. The partitioning and assignment procedure presented in the table is repeated at each regrid as the SAMR applications progress. Note that, when the number of processors assigned to one group is too large, SHPA can be applied in this group. Load balancing phase in AHPA is similar to the steps in Table 2 with dynamic group sizes and a dynamic number of group levels.

The AHPA scheme is evaluated using trace-driven simulations. The simulations are conducted as follows. First, we obtain the refinement trace for an SAMR application by running the application for a single processor. We then feed the trace file to the HPA partitioners to partition and produce a new trace file for multiple processors. Finally, we input the new trace file into our SAMR simulator to obtain the runtime performance measurements on multiple processors. The simulation results for the 2D Transport Equation and the Wave3D applications are shown in Figure 6.

In Figure 6, we observe that the communication cost (measured as the total message size for intra-level and inter-level communication) is greatly reduced using HPA schemes as compared to the Non-HPA scheme. This is primarily due to reduced global communication and concurrent communications in hierarchical processor groups. Compared to the SHPA scheme, AHPA scheme further reduces communication cost. In the figure, the communication cost increases as the number of processors increases due to an increase inter-processor communication traffic. An important observation is that, the rate of increase for the Non-HPA and SHPA schemes are greater than that for the AHPA scheme. This indicates that the AHPA scheme has a better scalability. The reduction in communication cost is

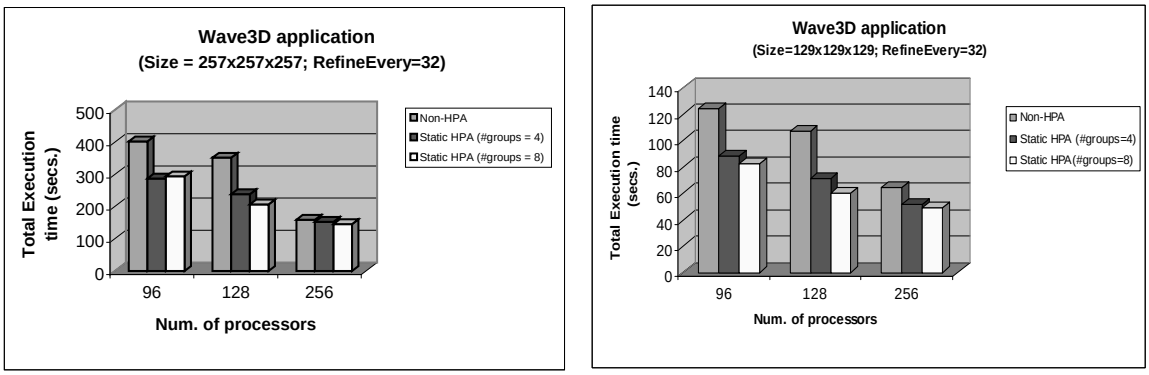

Figure 5. Execution time: Static HPA versus Non-HPA scheme

Table 4. Load partitioning and assignment in Adaptive HPA

Step 1. Use SFC to obtain the composite grid unit (CGU) list.

Step 2. Partition the CGU list into subdomains such that subdomains R_i (i is odd) consists of subdomains whose refinement level is not greater than i/2 and R_j (j is even) consists of subdomains whose refinement level is not less than j/2. R_0 consists of whole domain.

Step 3. Assign the load L_i on subdomain R_i to a group of processors G_i such that the number of processors NP_i in the group G_i is proportional to the load L_i, i.e., $NP_i = L_i/L_{sum} \times NP_{sum}$, where L_{sum} is the total size of load and NP_{sum} is the total number of processors.

Step 4. Partition the load portion L_i and assign the appropriate portion to the individual processor in the group G_i, for $i = 0, 1, ..., NP_{totalgroups} - 1$.

significant, up to 70%, for the AHPA scheme as compared to the Non-HPA scheme. These simulations validate that the Adaptive HPA scheme is an efficient solution to gain better system performance. The experimental evaluation of AHPA scheme is in progress and will be released soon.

4 Conclusions

In this paper we proposed a hierarchical partitioning and balancing strategy for the distributed implementations of SAMR applications. The HPA scheme takes advantage of hierarchical organization of the processor groups to restrict communications to smaller groups, thereby reducing the global communication and synchronization cost and exploiting concurrent communication. We presented two variant HPA scheme,viz. the Static HPA (SHPA) and the Adaptive HPA (AHPA). In the SHPA scheme, the total number of groups is defined a priori and the group topology is fixed or static during the execution of SAMR applications. In the AHPA scheme, the processor pool is adaptively partitioned into hierarchical groups at runtime to match the adaptive behavior of the SAMR applications. The HPA schemes are validated using experiments and simulations. It is experimentally shown that SHPA reduces communication costs as compared to the Non-HPA scheme, and reduces overall application execution time by up to 41%. AHPA dynamically partitions the processor pool into hierarchical groups that match the structure of the adaptive grid hierarchy. Initial evaluations of AHPA show that it can reduce communication costs by up to 70%. An experimental evaluation of the AHPA scheme is ongoing.

Other variants of HPA are also quite promising - for example an Adaptive HPA taking into consideration the runtime system state. The meta-partitioner method as proposed in [5] can be incorporated into the HPA scheme framework to apply different HPA schemes for different system and application runtime characteristics.

References

[1] BATSRUS. http://hpcc.engin.umich.edu/HPCC/codes /2/BATSRUSv2.html.

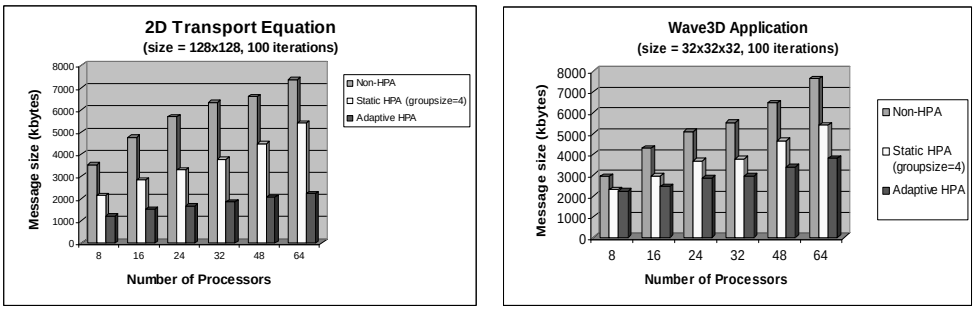

Figure 6. Communication cost: comparison of Non-HPA, Static HPA and Adaptive HPA schemes

[2] M. Berger and J. Oliger. Adaptive mesh refinement for hyperbolic partial differential equations. *Journal of Computational Physics*, 53:484–512, 1984.

[3] R. Buyya, editor. *High Performance Cluster Computing*, volume 1. Prentice Hall, 1999.

[4] CACTUS. Cactus computation toolkit, http://www.cactuscode.org/.

[5] S. Chandra, J. Steensland, M. Parashar, and J. Cummings. An experimental study of adaptive application sensitive partitioning strategies for samr applications. In *2nd Los Alamos Computer Science Institute Symposium*, Oct. 2001.

[6] I. Foster, C. Kesselman, and S. Tuecke. The anatomy of the grid: Enabling scalable virtual organizations. *International Journal of High Performance Computing Applications*, 15:200–222, 2001.

[7] M. Furuchi, K. Taki, and N. Ichiyoshi. A multi-level load balancing scheme for OR-parallel exhaustive search programs on the Multi-PSI. In *The Second ACM SIGPLAN Symposium on Principles and Practice of Parallel Programming*, pages 100–106, 1996.

[8] S. Kohn. SAMRAI: Structured adaptive mesh refinement applications infrastructure. Technical report, Lawrence Livermore National Laboratory, 1999.

[9] P. MacNeice. Paramesh, http://sdcd.gsfc.nasa.gov/rib /repositories/inhouse_gsfc/Users_manual/amr.html, 1999.

[10] M. Parashar. GrACE, http://www.caip.rutgers.edu /~parashar/TASSL/Projects/GrACE, 2001.

[11] M. Parashar and J. Browne. On partitioning dynamic adaptive grid hierarchies. In *29th Annual Hawaii International Conference on System Sciences*, pages 604–613, Jan.1996.

[12] R. Pollak. A hierarchical load balancing environment for parallel and distributed supercomputer. In *International Symposium on Parallel and Distributed Supercomputing*, Fukuoka, Japan, Sep. 1995.

[13] H. Sagan. *Space Filling Curves*. Springer-Verlag, 1994.

[14] B. A. Shirazi, A. R. Hurson, and K. M. Kavi. *Scheduling and load balancing in parallel and distributed systems*. IEEE Computer Society Press, Los Alamitos, 1995.

[15] J. Steensland. http://www.tdb.uu.se/~johans/research /vampire/vampire1.html, 2000.