

Hierarchical Partitioning Techniques for Structured Adaptive Mesh Refinement Applications *

Xiaolin Li (xlli@caip.rutgers.edu) and Manish Parashar (parashar@caip.rutgers.edu)

Department of Electrical and Computer Engineering, Rutgers University, 94 Brett Road, Piscataway, NJ 08854

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 59%. 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

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, 7]. 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

* 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 and 1052856.



mesh refinement methods, where the computational load 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 applications [5, 8, 10, 12] 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 assign 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 59%. Furthermore, initial evaluations show that AHPA reduces the communication cost by 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, decompositions, mappings, distribution and communication mechanisms. Table I summarizes a selection of the existing SAMR infrastructures and the partitioning approach used by them. Related work in hierarchical load-balancing is described below.

Pollack [13] 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 multilevel 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. In [9], the performance of hierarchical load sharing in heterogeneous

Table I. Distributed SAMR Infrastructures

| Infrastructure | Description |
|----------------|---|
| Chombo [5] | Consisted of four core modules: BoxTools, AMRTools, AMRTimeDependent, AMRElliptic. The load balance strategy follows Kernighan-Lin multilevel partitioning algorithm. |
| GrACE [11] | Object-oriented adaptive computational engine with pluggable domain-based partitioners |
| ParaMesh [10] | Extends serial code to parallel code based on partitioning octree representation of adaptive grid structure with predefined block sizes |
| SAMRAI [8] | Object oriented framework (based on LPARX) with patches mapped across processors at each level |

distributed systems was analyzed. Like PaLaBer, they targeted a general distributed systems and only addressed the design of distributed policies for scheduling independent task queues on systems. Teresco et al. [17] proposed a hierarchical partition model for parallel adaptive finite element applications. Their hierarchical model addresses the heterogeneous processor and network speeds. However, this model mainly focused on the computer systems hierarchy rather than on partitioning the computational domain hierarchy. Baden et al. [1] proposed a two-tier communication model, intra-node and inter-node communication on SMP clusters. They use a spare SMP processor to handle node-level communications.

The rest of this 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, 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 for the AMR formulation by Berger and Olinger [2] is shown in Figure 1.

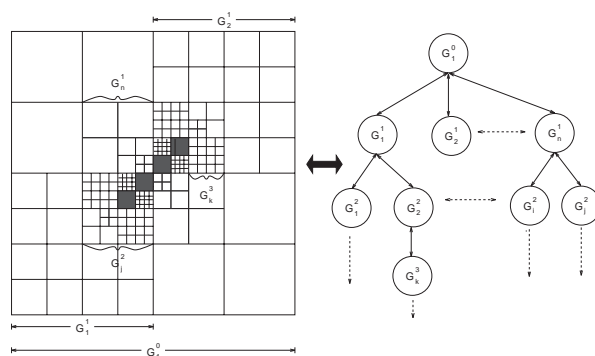


Figure 1. Adaptive Grid Hierarchy - 2D (Berger-Olinger AMR scheme)

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 SAMR partitioners is maintaining logical locality across partitions at different levels of the hierarchy and at the same level when they are decomposed and mapped across processors. This 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 naive decompositions 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 [12]. 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) [14], 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 to reduce intra-level communications and localize 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 [10, 11, 16], load scheduling and balancing is done collectively by all the processors in the system and all the processors maintain a global knowledge of the state of the system and the total workload. These schemes have the advantage of achieving a better load balance. However these approaches require the collection and maintenance of global load information which makes them expensive, specially on large systems. Note that a Non-HPA scheme can be viewed as a special case of HPA scheme where there is only one group containing all the processors. Partitioning in such a Non-HPA scheme consists of the following steps:

- Global load information exchange and synchronization phase: All the 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 the processors calculate the average load per processor and partition the grid hierarchy. This operation is replicated on 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. At the startup, all the processors have

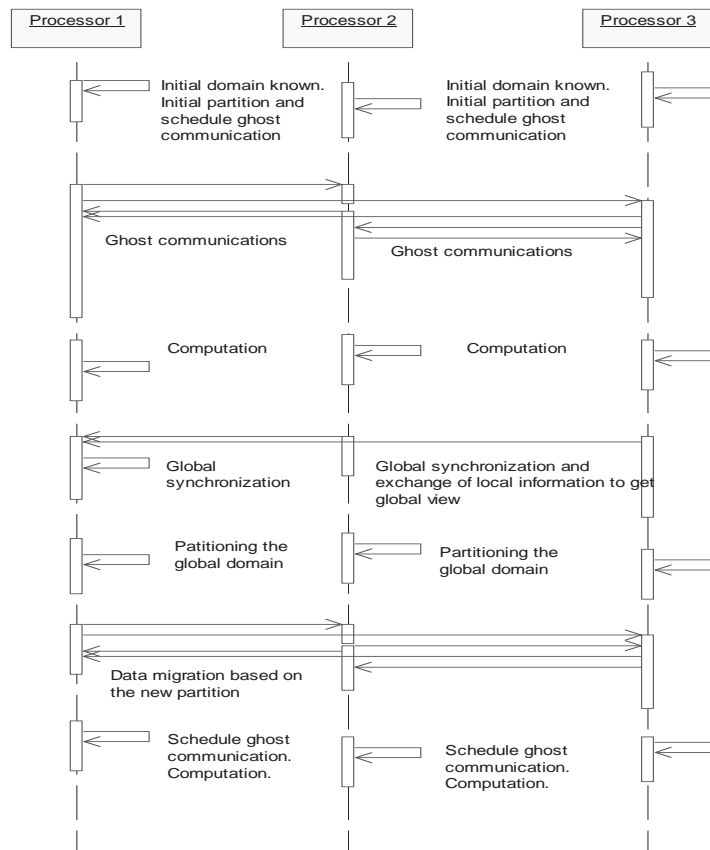


Figure 2. Sequence diagram of the Non-HPA scheme

the initial computational domain. Each processor partitions the domain into subdomains and assigns a subdomain to itself. During the load balancing phase, all the 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.

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 the space filling curve (SFC) technique [12, 14] as mentioned in Section 2. In this scheme, partitioning at different levels 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 hierarchically conducted among the processors in each group concurrently rather than requiring communication and synchronization among all the processors. This is achieved by dividing the processors into processor/compute groups as shown in Figure 3.

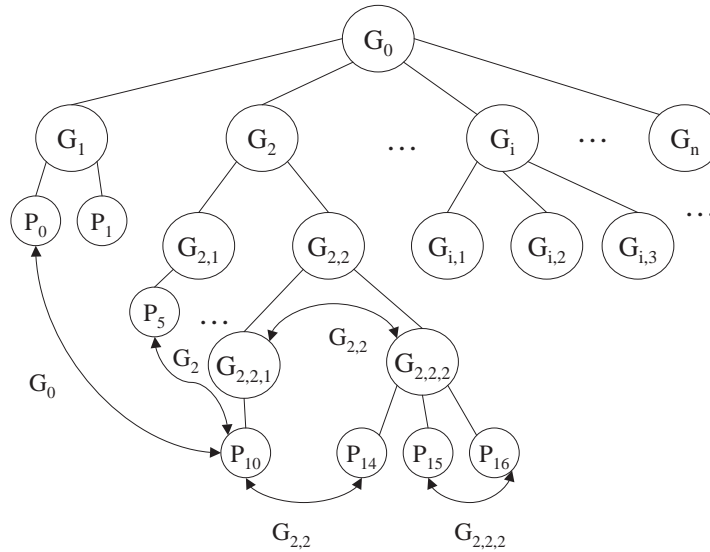


Figure 3. A general hierarchical structure of processor groups

Figure 3 illustrates a general hierarchical tree structure of processor groups, where, G_0 is the root level group (group level=0) containing all the processors, G_i is the i -th group at 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 coordinator 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 load balancing phase in the general HPA is given in Table II.

Table II. Load balancing phase in the general HPA

-
1. In the highest level group, if(my_load greater than local_threshold), perform the local partition in each group.
 2. Loop from group level lev=num_group_level to 1
 3. If(group_load greater than group_threshold), perform the group partition among children groups at lev, broadcast new composite list through parent group. If(lev equals 1) it is a global partition among groups at level 1.
 4. End of the loop
 5. Begin computation...
-

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

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

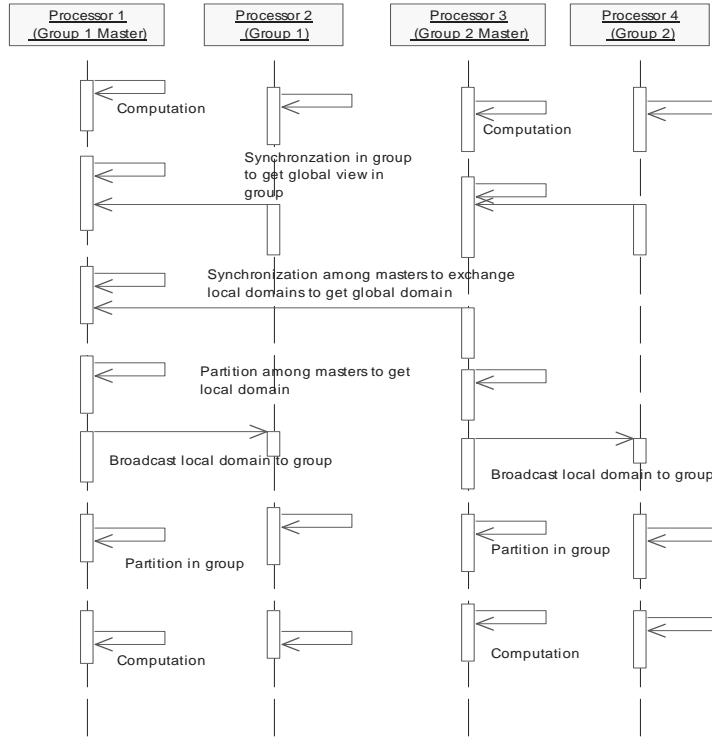


Figure 4. Sequence diagram of the HPA scheme

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. 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 [15], as load is dynamically redistributed within and across processor groups – only the processor group hierarchy remains static.

The load partitioning and assignment procedure is presented in Table III. 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 II.

Table III. Hierarchical Partitioning Algorithm

-
1. Setup the processor group hierarchy according to group size and group levels. Apply SFC to obtain the composite grid unit list (GUL).
 2. Loop from group level $lev=1$ to num_group_level
 3. Partition the global GUL into N_{lev} subdomains, where N_{lev} is the number of processor groups at this level.
 4. 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 in the parent group level.
 5. Loop until the leaves of the group tree hierarchy are reached. Partition the load portion L_i and assign the appropriate portion to the individual processor in the group G_i , for $i = 0, 1, \dots, NP_j - 1$, where NP_j is the number of processors in the lowest group level.
-

The Static HPA is implemented as part of the GrACE toolkit [11]. 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.

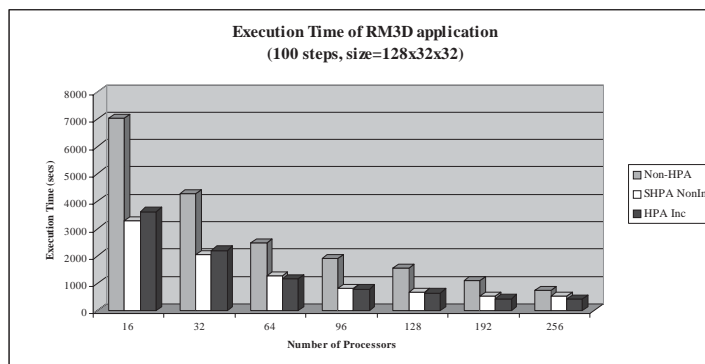


Figure 5. Execution time: Static HPA versus Non-HPA scheme

The Static HPA scheme is evaluated on the IBM SP2 cluster at San Diego Supercomputer Center. The application used in these experiments is the 3-D Richtmyer-Meshkov instability solver (RM3D) encountered in compressible fluid dynamics. RM3D has been developed by Ravi Samtaney as part of the virtual test facility at the Caltech

ASCI/ASAP Center [6]. The experiments measure the total execution time of RM3D using Static HPA and Non-HPA schemes. To evaluate the benefits of incremental load balancing, we performed two experiments for Static HPA scheme: SHPA without incremental balancing (labeled as SHPA NonInc in the figure), and SHPA with incremental load balancing (labeled as SHPA Inc in the figure). In Figure 5, we observe that SHPA schemes significantly improve the overall execution time. The maximum performance gain is obtained for 192 processors using SHPA Inc scheme, about 59% reduction in the overall execution time as compared to Non-HPA scheme. We also observe that, for relatively small number of processors, the SHPA NonInc scheme outperforms the SHPA Inc scheme. The reason is that SHPA NonInc scheme has the advantage of better load balancing than the SHPA Inc scheme since it re-distributes the load globally more frequently. However, for larger number of processors, due to significant reduction of the synchronization and global communication overheads with incremental load balancing, the SHPA Inc scheme outperforms the SHPA NonInc scheme in the long run. As shown by the evaluation, the benefits of SHPA depends on the appropriate setup of processor group hierarchies, which in turn depends on the system and the application. The adaptive HPA scheme attempts to address this 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 remains 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 domain into subdomains to match 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 IV. 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 IV, 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 large, SHPA can be

Table IV. Load partitioning and assignment in Adaptive HPA

-
1. Use SFC to obtain the composite grid unit list (GUL).
 2. Partition the GUL 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.
 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.
 4. Partition the load portion L_i and assign the appropriate portion to the individual processor in the group G_i , for $i = 0, 1, \dots, NP_{totalgroups} - 1$.
-

applied in this group. Load balancing phase in AHPA is similar to the steps in Table II 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 is obtained by running the application for a single processor. Then the trace file is fed into HPA partitioners to partition and produce a new trace file for multiple processors. Finally, the new trace file is input into the SAMR simulator¹ 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.

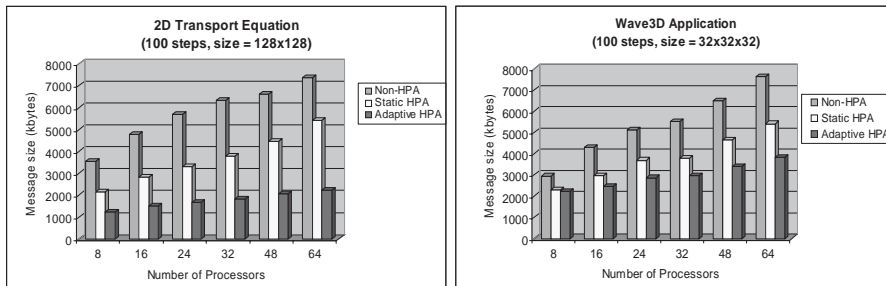


Figure 6. Communication cost: comparison of Non-HPA, Static HPA and Adaptive HPA schemes

¹ SAMR simulator was developed by Manish Parashar at Rutgers University as a part of ARMaDA project (http://www.caip.rutgers.edu/TASSL/Projects/ARMaDA/performance_simulator.html)

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 costs. In the figure, the communication cost increases as the number of processors increases due to an increase of 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 significant, up to 70%, for the AHPA scheme as compared to the Non-HPA scheme. These simulations validate that the Adaptive HPA scheme is potentially an efficient solution to gain better system performance. The experimental evaluation of the AHPA scheme is in progress and will be released soon.

4. Conclusions

In this paper, hierarchical partitioning and balancing strategies for distributed implementations of SAMR applications were proposed. HPA schemes take advantage of the hierarchical organization of the processor groups to enable the load distribution to reflect the state of the adaptive grid hierarchy and thereby reducing the global communication and synchronization costs, improving load balance, exploiting concurrent communication, and enabling the incremental redistribution. Two variant HPA scheme were presented, namely, 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 59%. 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 proposed in [4] can be combined with the HPA scheme framework to apply different HPA schemes for different system and application runtime characteristics or for different parts at each level of the grid hierarchy.

References

1. Baden, S. B. and S. J. Fink: 1998, ‘Communication overlap in multi-tier parallel algorithms’. In: *Conf. Proc. Supercomputing 98*. Orlando, FL.
2. Berger, M. and J. Olinger: 1984, ‘Adaptive Mesh Refinement for Hyperbolic Partial Differential Equations’. *Journal of Computational Physics* **53**, 484–512.
3. Buyya, R. (ed.): 1999, *High Performance Cluster Computing*, Vol. 1. Prentice Hall.
4. Chandra, S., J. Steensland, M. Parashar, and J. Cummings: Oct. 2001, ‘An Experimental Study of Adaptive Application Sensitive Partitioning Strategies for SAMR Applications’. In: *2nd Los Alamos Computer Science Institute Symposium*.
5. Colella, P. and et. al.: 2003, ‘Chombo’. URL: <http://seesar.lbl.gov/anag/chombo/>.
6. Cummings, J., M. Aivazis, R. Samtaney, R. Radovitzky, S. Mauch, and D. Meiron: 2002, ‘A virtual test facility for the simulation of dynamic response in materials’. *Journal of Supercomputing* **23**, 39–50.
7. Foster, I., C. Kesselman, and S. Tuecke: 2001, ‘The anatomy of the grid: Enabling scalable virtual organizations’. *International Journal of High Performance Computing Applications* **15**, 200–222.
8. Kohn, S.: 1999, ‘SAMRAI: Structured Adaptive Mesh Refinement Applications Infrastructure’. Technical report, Lawrence Livermore National Laboratory.
9. Lo, M. and S. Dandamudi: 1996, ‘Performance of Hierarchical Load Sharing in Heterogeneous Distributed Systems’. In: *Int. Conf. on Parallel and Distributed Computing Systems*. Dijon, France, pp. pp. 370–377.
10. MacNeice, P.: 1999, ‘Paramesh’. URL: <http://esdcd.gsfc.nasa.gov/ESS/macneice/paramesh/paramesh.html>.
11. Parashar, M.: 2003, ‘GrACE’. URL: <http://www.caip.rutgers.edu/~parashar/TASSL/Projects/GrACE>.
12. Parashar, M. and J. Browne: Jan.1996, ‘On Partitioning Dynamic Adaptive Grid Hierarchies’. In: *29th Annual Hawaii International Conference on System Sciences*. pp. 604–613.
13. Pollak, R.: Sep. 1995, ‘A hierarchical load balancing environment for parallel and distributed supercomputer’. In: *International Symposium on Parallel and Distributed Supercomputing*. Fukuoka, Japan.
14. Sagan, H.: 1994, *Space Filling Curves*. Springer-Verlag.
15. Shirazi, B. A., A. R. Hurson, and K. M. Kavi: 1995, *Scheduling and load balancing in parallel and distributed systems*. Los Alamitos: IEEE Computer Society Press.
16. Steensland, J.: 2000, ‘VAMPIRE’. URL: <http://www.tdb.uu.se/~johans/research/vampire/vampire1.html>.
17. Teresco, J. D., M. W. Beall, J. E. Flaherty, and M. S. Shephard: 2000, ‘A hierarchical partition model for adaptive finite element computation’. *Computer Methods in Applied Mechanics and Engineering* **184**, 269–285.

