Dynamic Load Partitioning Strategies for Managing Data of Space and Time Heterogeneity in Parallel SAMR Applications*

Xiaolin Li and Manish Parashar

The Applied Software Systems Laboratory
Department of Electrical & Computer Engineering
Rutgers University, Piscataway, NJ 08854, USA
Email: {xlli, parashar}@caip.rutgers.edu

Abstract. This paper presents the design and experimental evaluation of two dynamic load partitioning and balancing strategies for parallel Structured Adaptive Mesh Refinement (SAMR) applications: the Level-based Partitioning Algorithm (LPA) and the Hierarchical Partitioning Algorithm (HPA). These techniques specifically address the computational and communication heterogeneity across refinement levels of the adaptive grid hierarchy underlying these methods. An experimental evaluation of the partitioning schemes is also presented.
Keywords: Dynamic Load Balancing, Parallel and Distributed Computing, Structured Adaptive Mesh Refinement, Scientific Computing

1 Introduction

Large-scale parallel/distributed simulations are playing an increasingly important role in science and engineering and are rapidly becoming critical research modalities in academia and industry. While the past decade has witnessed a significant boost in CPU, memory and networking technologies, the size and resolution of these applications and their corresponding computational, communication and storage requirements have grown at an even faster rate. As a result, applications are constantly saturating available resources. Dynamically adaptive techniques, such as Structured Adaptive Mesh Refinement (SAMR) [1], can yield highly advantageous ratios for cost/accuracy when compared to methods based upon static uniform approximations. SAMR provides a means for concentrating computational effort to appropriate regions in the computational domain. These techniques can lead to more efficient and cost-effective solutions to time dependent problems exhibiting localized features. Parallel implementations of these methods offer the potential for accurate solutions of physically realistic models of complex physical phenomena. However, the dynamics and space and time heterogeneity of the adaptive grid hierarchy underlying SAMR algorithms makes their efficient parallel implementation a significant challenge.

* Support for this work was provided by the NSF via grants numbers ACI 9984357 (CAREERS), EIA 0103674 (NGS) and EIA-0120934 (ITR), DOE ASCI/ASAP (Caltech) via grant numbers PC295251 and 1052856.

Traditional parallel implementation of SAMR applications [3] [5] [7] have used dynamic partitioning/load-balancing algorithms that view the system as a flat pool of (usually homogeneous) processors. These approaches are based on global knowledge of the state of the adaptive grid hierarchy, and partition the grid hierarchy across the set of processors. Global synchronization and communication 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 synchronization in this structure.

This paper presents the design and experimental evaluation of dynamic load partitioning and balancing strategies for parallel SAMR applications that specifically address the computational and communication heterogeneity across refinement levels of the adaptive grid hierarchy underlying these methods. In this paper, we first analyze the space/time computational and communication behavior of parallel SAMR applications. We then present the design and experimental evaluation of two dynamic load partitioning and balancing strategies for parallel SAMR applications: Level-based Partitioning Algorithm (LPA) and Hierarchical Partitioning Algorithm (HPA). These algorithms are also presented in combination.

The rest of the paper is organized as follows. Section 2 gives an overview on SAMR technique and presents the computation and communication behavior of parallel SAMR applications. Section 3 presents the LPA and HPA dynamic partitioning and load balancing strategies. Section 4 presents the experimental evaluation of these partitioners. Section 5 presents conclusions.

2 Problem Description

SAMR techniques track regions in the domain that requires additional resolution and dynamically overlay 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 over laid on these 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 for the Structured Berger-Oliger AMR is a dynamic adaptive grid hierarchy [1].

2.1 Computation and communication behavior for parallel SAMR

In the targeted SAMR formulation, the grid hierarchy is refined both in space and in time. Refinements in space create finer level grids which have more grid points/cells than their parents. Refinements in time mean that finer grids take smaller time steps and hence have to be advanced more often. As a result, finer grids not only have greater computational loads but also have to be integrated and synchronized more often. This results in space and time heterogeneity in the SAMR adaptive grid hierarchy. Furthermore, regridding occurs at regular intervals at each level and result in refined regions

are created, moved and deleted. Together, these characteristics of SAMR applications makes their efficient parallel implementation a significant challenge.

Parallel implementations of hierarchical SAMR applications typically partition the adaptive heterogeneous grid hierarchy across available processors, and each processor operates on its local portions of this domain in parallel. Each processor starts at the coarsest level, integrates the patches at this level and performs intra-level or ghost communications to update the boundaries of the patches. It then recursively operates on the finer grids using the refined time steps - i.e. for each step on a parent grid, there are multiple steps (equal to the time refinement factor) on the child grid. When the parent and child grid are at the same physical time, inter-level communications are used to inject information from the child to its parent. Dynamic re-partitioning and re-distribution is typically required after this step.

The overall performance of parallel 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. 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.

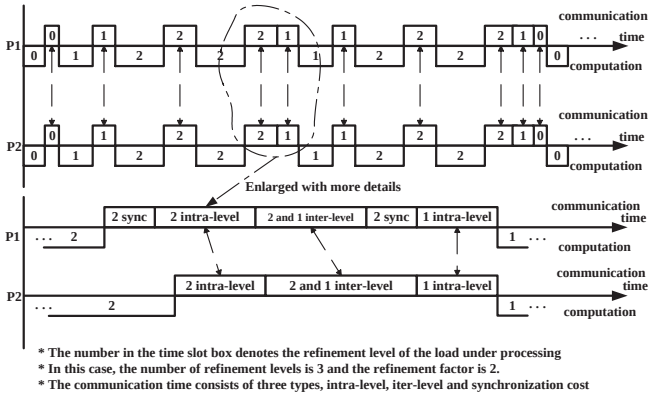

Fig. 1. Timing Diagram for Parallel SAMR Algorithm

The timing diagram (note that the timing is not drawn to scale) in Figure 1 illustrates the operation of the SAMR algorithms described above using a 3 level grid hierarchy. For simplification, only the computation and communication time behaviors of processors $P1$ and $P2$ are shown. The three components of communication overheads (listed in Section 2.2) are illustrated in the enlarged portion of the time line. This figure shows the exact computation and communication patterns for parallel SAMR implementations. Note that the timing diagram shows that there is one time step on the coarsest level (level 0) of the grid hierarchy followed by two time steps on the first refinement level and four time steps on the second level, before the second time step on level 0 can start. Also note that the computation and communication for each refinement levels are

interleaved. This behavior makes it quite challenging to partition the dynamic SAMR grid hierarchy to both balance load and minimize communication/synchronization overheads.

2.2 Communication overheads for parallel SAMR applications

As described above and shown in Figure 1, the communication overheads of parallel SAMR applications primarily consist of three components: (1) *Inter-level communications* defined between component grids at different levels of the grid hierarchy and consist of prolongations (coarse to fine transfer and interpolation) and restrictions (fine to coarse transfer and interpolation). (2) *Intra-level communications* required to update the grid-elements along the boundaries of local portions of a distributed grid, consists of near-neighbor exchanges. These communications can be scheduled so as to be overlapped with computations on the interior region; (3) *Synchronization cost*, which occurs when the load is not well balanced among all processors. These costs may occur at any time step and at any refinement level due to the hierarchical refinement of space and time in SAMR applications.

Clearly, an optimal partitioning of the SAMR grid hierarchy and scalable implementations of SAMR applications requires careful consideration of the timing pattern as shown in Figure 1 and the three communication overhead components. Critical observations from the timing diagram in Figure 1 are that, in addition to balancing the total load assigned to each processor and maintaining parent child locality, we also have to balance the load on each refinement level and address the communication and synchronization costs within a level. The load partitioning and balancing strategies presented in the following section address these issues.

3 Dynamic Load Partitioning and Balancing Strategies

The dynamic partitioning algorithms presented in this paper are based on a core Composite Grid Distribution Strategy (CGDS) [7]. This domain-based partitioning strategy performs a composite decomposition of the adaptive grid hierarchy using Space Filling Curve (SFC) [8]. Space filling curves are locality preserving recursive mappings from n-dimensional space to 1-dimensional space. CGDS uses SFCs, and partitions the entire SAMR domain into sub-domains such that each sub-domain keeps all refinement levels in the sub-domain as a single composite grid unit. Thus all inter-level communication are local to a sub-domain and the inter-level communication time is greatly reduced. The resulting composite grid unit list (GUL) for the overall domain must now be partitioned and balanced across processors.

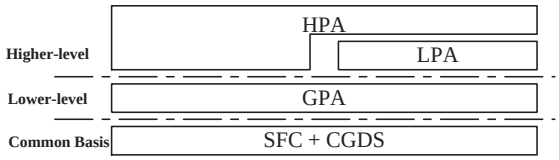

Fig. 2. Layers of Partitioning Algorithms

A Greedy Partitioning Algorithm (GPA) is used to partition the global GUL to produces a local GUL for each processor. The key motivation for using the GPA scheme is

that it is fast and efficient as it scans the global GUL only once to try to equally distribute load among all processors. This is important as the number of composite grid units can be large and regridding steps can be quite frequent. This scheme works very well for homogeneous computational domain. However, for heterogeneous computational domains, it may cause large intra-level synchronization cost due to load imbalance for each refinement level.

To improve the performance of GPA scheme, we propose two efficient partitioning schemes, Level-based Partitioning Algorithm (LPA) and Hierarchical Partitioning Algorithm (HPA). As shown in Figure 2, GPA is a lower-level partitioning scheme which can work independently. LPA and HPA are higher-level partitioning schemes that work on the top of the lower-level scheme. Specially, HPA can work either on the top of LPA or directly on the top of GPA. Detailed description and analysis of LPA and HPA are presented in the following two subsections.

3.1 Level-based partitioning algorithm

The level-based partitioning algorithm (LPA) essentially preprocesses a portion of the global GUL by disassembling it according to refinement levels, and feeds the resulting homogeneous composite GUL to GPA. The GPA then partitions this list to balance load. As a result of the preprocessing, the load on each refinement level is also balanced. This algorithm is as follows:

1. *Get the maximum refinement level MaxLev. Disassemble the global GUL into homogeneous GUL's according to grid unit's refinement depth, denoted by gul_array[lev]. The load assigned in the previous iteration is denoted by load_array[np].*
2. *Loop for refinement level lev = MaxLev to 0 reversely*
3. *Passing gul_array[lev] and load_array[np] to GPA to obtain local assignment.*
4. *In GPA, it will partition the load such that each processor get equal distribution on each refinement level.*

We observe that the LPA scheme partitions deep composite grid units before shallow grid units. The previous iteration has assigned some load on lower refinement level because we use the composite grid strategy. Since we cannot guarantee the perfect balance during the partition of each iteration, to compensate the possible imbalance introduced in higher level and equally partition the GUL on the lower level, we need to keep track of the load on lower level previously assigned. This is done using load _array[np]. LPA takes full advantages of CGDS by keeping parent-children relationships in the composite grid and localizing inter-level communications. Furthermore, it balances the load on each refinement level which reduces the synchronization cost as shown by the experimental evaluation and demonstrated by the following example.

Consider partitioning a one dimensional grid hierarchy with two refinement levels, as shown in Figure 3. For this 1-D example, GPA partitions the composite grid unit list into two subdomains. These two parts contain exactly same load: the load assigned to $P0$ is $2 + 2 \times 4$ while the load assigned to $P1$ is 10 units. From the viewpoint of GPA

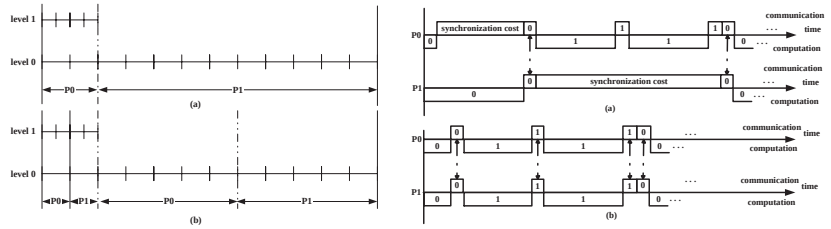

Fig. 3. Partitions of a 1-D Grid Hierarchy and Resulting Timing Diagrams (a) GPA (b) LPA

scheme, the partition result is perfectly balanced. However, due to the heterogeneity of SAMR algorithm, this distribution leads to large synchronization costs as shown in the timing diagram of Figure 3 (a). The LPA scheme takes these synchronization costs at each refinement level into consideration. For this simple example, LPA will produce a partition as shown in Figure 3 (b) which results in the computation and communication behavior as shown in Figure 3 (b). As a result, there is an improvement in overall execution time and a reduction in communication and synchronization time.

3.2 Hierarchical partitioning algorithm

In most parallel implementations of SAMR, such as ParaMesh [5], SAMRAI [3] and GrACE [6], load partitioning and balancing is collectively done by all processors 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 systems. In HPA, after initially obtaining the global GUL, the top processor group partitions it and assign portions to each processor subgroup in a hierarchical manner. In this way, HPA further localizes the communication to subgroups, enables concurrent communication and reduces the global communication and synchronization costs.

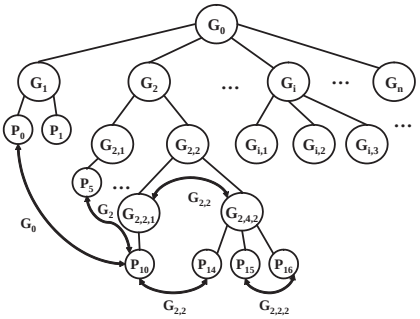

Fig. 4. A General Hierarchical Structure of Processor Groups

Figure 4 illustrates a general hierarchical tree structure of processor groups, where, G_0 is the root level group (grouplevel=0) which consists of all processors. G_i is the $i-th$ group at the group level 1. Note that only the leaves of the tree are processors. The communication between processors is accomplished through their closest common parent group. For example, for the processors P_{10} and P_{14}, they have common ancestor group G_0, G_2 and $G_{2,2}$, where $G_{2,2}$ is their closest common ancestor. Thus communication between P_{10} and P_{14} is via the group $G_{2,2}$. Similarly, the communication between

processors P_0 and P_{10} is via the group G_0. The detailed steps of HPA are as follows. More detailed description of HPA and its variants is presented in [4].

1. *Setup the processor group hierarchy according to group size and group levels.*
2. *Loop for group level lev=1 to num_group_level*
3. *Partition the global GUL into N_{lev} subdomains using GPA, 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 reaching the leaves of the group tree hierarchy. Partition the load portion L_i using GPA and assign the appropriate portion to the individual processor in the group G_i, for $i = 0, 1, ..., NP_j - 1$, where NP_j is the number of processors in the lowest group level.*

During the repartitioning phase, instead of performing global load balancing, HPA hierarchically checks for load imbalance. Starting from the lower level subgroups, if the imbalance is below some threshold, it only repartitions and redistributes load among the lowest subgroup, else if it is above the threshold, it checks for load imbalance in the next higher level group. It proceeds recursively up to the root processor group consisting of all processors in this way HPA thus reduces the amount of global synchronization required and enable incremental load balancing. Furthermore, HPA scheme exploits more communication parallelism through multiple concurrent communication channels among hierarchical groups.

4 Experimental Evaluation

The partitioning strategies are evaluated on the IBM SP2 cluster BlueHorizon at San Diego Supercomputer Center. The design of the adaptive runtime framework is driven by specific problems in enabling realistic simulations using AMR techniques. The 3-D Richtmyer-Meshkov instability application from computational fluid dynamics are used. RM3D has been developed by Ravi Samtaney as part of the virtual test facility at the Caltech ASCI/ASAP Center [2].

The input configurations are as follow: the base grid size is $128 \times 32 \times 32$, maximum 3 refinement levels, refinement factor is 2, granularity is 4, regrid every 4 time steps on each level, the total base level time steps are 100. Four partitioning schemes are used in the experiments, namely, GPA, LPA, HPA and HPA+LPA, running on a range of processors from 16 to 128. The comparison of the total execution time is shown in Figure 5. In this figure, the execution time for the different parameters are normalized against GPA on 16 processors (100%). We observe that execution time reduction using LPA scheme compared to GPA is about 48.8% on the average for these five system configurations. HPA scheme alone reduces the execution time by about 52.9% on the average. Applying HPA scheme on the top of LPA scheme, we gain further improvement reducing overall execution time by about 56% for 16 processors, 60% for 128

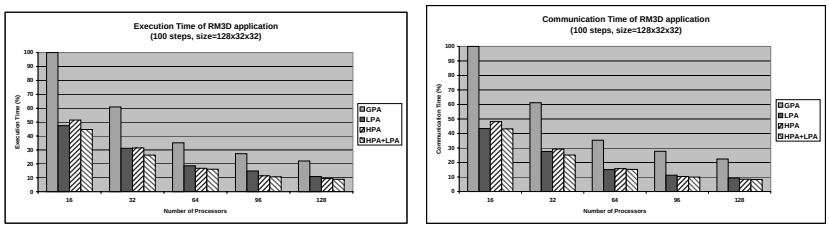

Fig. 5. Execution and Communication Time: GPA, LPA, HPA and HPA+LPA

processors, and 57.3% on the average. These reductions in the overall execution times are due to a reduction in communication times as shown in Figure 5. The figure also shows that, HPA greatly reduces the global communication time and exploits more concurrent communications. For all cases, HPA+LPA delivers the best performance since it takes full advantages of HPA and LPA.

5 Conclusions

In this paper we presented the design and experimental evaluation of two dynamic load partitioning and balancing strategies for parallel Structured Adaptive Mesh Refinement (SAMR) applications: the Level-based Partitioning Algorithm (LPA) and the Hierarchical Partitioning Algorithm (HPA). These techniques specifically address the computational and communication heterogeneity across refinement levels of the adaptive grid hierarchy underlying these methods. The efficiency of LPA and HPA is experimentally validated using a real SAMR application (RM3D). The improvement over the GPA scheme is significant. The overall reduction of execution time is about 48.8% for LPA, 52.9% for HPA and 57.3% for HPA+LPA on the average.

References

1. M. Berger and J. Oliger. Adaptive mesh refinement for hyperbolic partial differential equations. *Journal of Computational Physics*, 53:484–512, 1984.
2. J. Cummings, M. Aivazis, R. Samtaney, R. Radovitzky, S. Mauch, and D. Meiron. A virtual test facility for the simulation of dynamic response in materials. *Journal of Supercomputing*, 23:39–50, 2002.
3. R. D. Hornung and S. R. Kohn. Managing application complexity in the samrai object-oriented framework. *Concurrency and Computation - Practice & Experience*, 14(5):347–368, 2002.
4. X. Li and M. Parashar. Hierarchical partitioning techniques for structured adaptive mesh refinement applications. *(to appear) Journal of Supercomputing*, 2003.
5. P. MacNeice. Paramesh. `http://esdcd.gsfc.nasa.gov/ESS/macneice/paramesh/paramesh.html`.
6. M. Parashar. Grace. `http://www.caip.rutgers.edu/~parashar/TASSL/`.
7. M. Parashar and J. Browne. On partitioning dynamic adaptive grid hierarchies. In *29th Annual Hawaii Int. Conference on System Sciences*, pages 604–613, 1996.
8. H. Sagan. *Space Filling Curves*. Springer-Verlag, 1994.