Hybrid Runtime Management of Space-Time Heterogeneity for Parallel Structured Adaptive Applications

Xiaolin Li, *Member, IEEE* and Manish Parashar, *Senior Member, IEEE*

***Abstract*— Structured adaptive mesh refinement (SAMR) techniques provide an effective means for dynamically concentrating computational effort and resources to appropriate regions in the application domain. However, due to their dynamism and space-time heterogeneity, scalable parallel implementation of SAMR applications remains a challenge. This paper investigates hybrid runtime management strategies and presents an adaptive hierarchical multi-partitioner (AHMP) framework. AHMP dynamically applies multiple partitioners to different regions of the domain, in a hierarchical manner, to match the local requirements of the regions. Key components of the AHMP framework include a segmentation-based clustering algorithm (SBC) that can efficiently identify regions in the domain with relatively homogeneous partitioning requirements, mechanisms for characterizing the partitioning requirements of these regions, and a runtime system for selecting, configuring and applying the most appropriate partitioner to each region. Further, to address dynamic resource situations for long running applications, AHMP provides a hybrid partitioning strategy (HPS), which involves application-level pipelining, trading space for time when resources are sufficiently large and under-utilized, and an application-level out-of-core strategy (ALOC), trading time for space when resources are scarce in order to enhance the survivability of applications. The AHMP framework has been implemented and experimentally evaluated on up to 1280 processors of the IBM SP4 cluster at San Diego Supercomputer Center.**

***Index Terms*— Parallel Computing, Structured Adaptive Mesh Refinement, Dynamic Load Balancing, Hierarchical Multi-Partitioner, High Performance Computing**

I. Introduction

Simulations of complex physical phenomena, modeled by systems of partial differential equations (PDE), are playing an increasingly important role in science and engineering. Furthermore, dynamically adaptive techniques, such as the dynamic structured adaptive mesh refinement (SAMR) technique [1], [2], are emerging as attractive formulations of these simulations. Compared to numerical techniques based on static uniform discretization, SAMR can yield highly advantageous ratios for cost/accuracy by concentrating computational effort and resources to appropriate regions adaptively at runtime. SAMR is based on block-structured refinements overlaid on a structured coarse grid, and provide an alternative to the general, unstructured AMR approach [3], [4]. SAMR techniques have been used to solve complex systems of PDEs that exhibit localized features in varied domains including computational fluid dynamics, numerical relativity, combustion simulations, subsurface modeling and oil reservoir simulation [5]–[7].

Large-scale parallel implementations of SAMR-based applications have the potential to accurately model complex physical phenomena and provide dramatic insights. However, while there have been some large-scale implementations [8]–[13], these implementations are typically based on application-specific customizations, and general scalable implementations of SAMR applications continue to present significant challenges. This is primarily due to the dynamism and space-time heterogeneity exhibited by these applications. SAMR-based applications are inherently dynamic because the physical phenomena being modeled and the corresponding adaptive computational domain change as the simulation evolves. Further, adaptation naturally leads to a computational domain that is spatially heterogeneous, i.e., different regions in the computational domain and different levels of refinements have different computational and communication requirements. Finally, the SAMR algorithm periodically regrids the computational domain causing regions of refinement to be created/deleted/moved to match the physics being modeled, i.e., it exhibits temporal heterogeneity.

The dynamism and heterogeneity of SAMR applications have been traditionally addressed using dynamic partitioning and load-balancing algorithms, e.g., the mechanisms presented in [10] and [13], which partition and load-balance the SAMR domain when it changes. More recently, it was observed in [14], that, for parallel SAMR applications, the appropriate choice and configuration of the partitioning/load-balancing algorithm depends on the application, its runtime state and its execution context. This lead to the development of meta-partitioners [14], [15], which select and configure partitioners (from a pool of partitioners) at runtime to match the application's current requirements. However, due to the spatial heterogeneity of the SAMR domain, computation and communication requirements can vary significantly across the domain, and as a result, using a single partitioner for the entire domain can lead to decompositions that are locally inefficient. This is especially true for large-scale simulations that run on over a thousand processors.

The objective of the research presented in this paper is to address this issue. Specifically, this paper builds on our earlier research on meta-partitioning [14], adaptive hierarchical partitioning [16], and adaptive clustering [17], and

X. Li is with the Scalable Software Systems Laboratory, Department of Computer Science, Oklahoma State University, Stillwater, OK 74078. Email: xiaolin@cs.okstate.edu.

M. Parashar is with The Applied Software Systems Laboratory, Department of Electrical & Computer Engineering, Rutgers University, Piscataway, NJ 08854. Email: parashar@caip.rutgers.edu.

investigates hybrid runtime management strategies and an adaptive hierarchical multi-partitioner (AHMP) framework. AHMP dynamically applies multiple partitioners to different regions of the domain, in a hierarchical manner, to match local requirements. This paper first presents a segmentation-based clustering algorithm (SBC) that can efficiently identify regions in the domain (called *clusters*) at runtime that have relatively homogeneous requirements. The partitioning requirements of these cluster regions are determined and the most appropriate partitioner from the set of available partitioners is selected, configured and applied to each cluster.

Further, this paper also presents two strategies to cope with different resource situations in the case of long running applications: (1) the hybrid partitioning algorithm, which involves application-level pipelining, trading space for time when resources are sufficiently large and under-utilized, and (2) the application-level out-of-core strategy (ALOC), which trades time for space when resources are scarce in order to improve the performance and enhance the survivability of applications. The AHMP framework and its components are implemented and experimentally evaluated using the RM3D application on up to 1280 processors of the IBM SP4 cluster at San Diego Supercomputer Center.

The rest of the paper is organized as follows. Section II presents an overview of the SAMR technique and describes the computation and communication behaviors of its parallel implementation. Section III reviews related work. Section IV describes the AHMP framework and hybrid runtime management strategies for parallel SAMR applications. Section V presents an experimental evaluation for the framework using SAMR application kernels. Section VI presents a conclusion.

II. Problem Description

A. Structured Adaptive Mesh Refinement

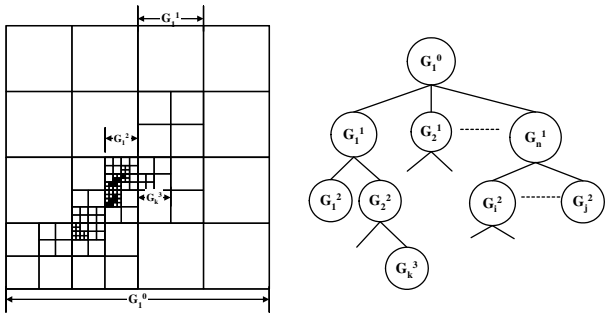


Fig. 1. Adaptive Grid Hierarchy for 2D Berger-Oliger SAMR [1]. The left figure shows a 2-D physical domain with localized multiple refinement levels. The right figure represents the refined domain as a grid hierarchy.

Structured Adaptive Mesh Refinement (SAMR) formulations for adaptive solutions to PDE systems track regions in the computational domain with high solution errors that require additional resolution and dynamically overlay finer grids over these regions. SAMR 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 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 SAMR grid structure is a dynamic adaptive grid hierarchy as shown in Figure 1.

B. Computation and Communication Requirements of Parallel SAMR

Parallel implementations of SAMR applications typically partition the adaptive grid hierarchy across available processors, and each processor operates on its local portions of this domain in parallel. The overall performance of parallel SAMR 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.

Communication overheads of parallel SAMR applications primarily consist of four 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 component grid, consisting of near-neighbor exchanges; (3) *Synchronization cost*, which occurs when the load is not balanced among processors; (4) *Data migration cost*, which occurs between two successive regridding and re-mapping steps.

Partitioning schemes for SAMR grid hierarchies can be classified as patch-based, domain-based, and hybrid [14]. A patch, associated with one refinement level, is a rectangular region that is created by clustering adjacent computational grids/cells at that level [18]. In patch-based schemes, partitioning decisions are made independently for each patch at each level. Domain-based schemes partition the physical domain and result in partitions or subdomains that contain computational grids/cells at multiple refinement levels. Hybrid schemes generally follow two steps. The first step uses domain-based schemes to create partitions, which are mapped to a group of processors. The second step uses patch-based or combined schemes to further distribute the partition within its processor group. Domain-based partitioning schemes have been studied in [13], [14], [19]. Three hybrid schemes are presented in [13]. In general, pure patch-based schemes outperform domain-based schemes when balancing the workload is the only consideration. However, patch-based schemes incur considerable overheads for communications between refinement levels, e.g., prolongation and restriction operations. Typically boundary information is much smaller than the information in the entire patch. Further, if the patch-based method completely ignores locality, it might cause severe communication bottleneck between refinement levels. In contrast, domain-based schemes distribute subdomains that contain all refinement levels to processors and hence eliminate the inter-level communications. However, for applications with strongly localized and deeply refined regions, domain-based

schemes are inadequate to well balance workloads. More detailed description and comparisons of these partitioning schemes can be found in [13], [14], [18]. This paper mainly focuses on domain-based schemes and also presents a new hybrid scheme.

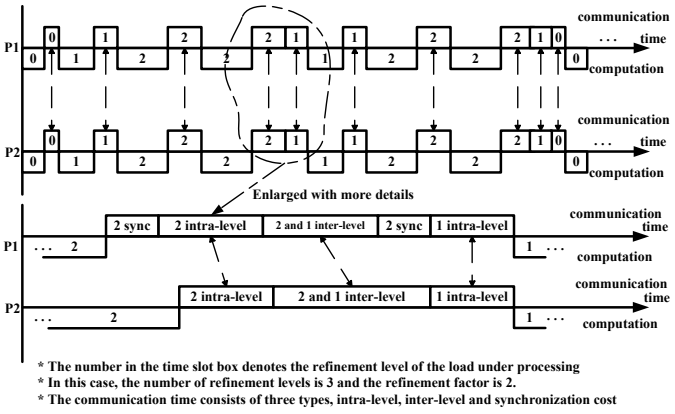


Fig. 2. Timing Diagram for Parallel SAMR

The timing diagram (note that the figure is not to scale) in Figure 2 illustrates the operation of the SAMR algorithm using a 3 level grid hierarchy. The process shown in the figure shows a typical parallel SAMR application using a domain-based partitioning scheme. For simplicity, the computation and communication behaviors of only two processors, $P1$ and $P2$, are shown. The communication overheads are illustrated in the magnified portion of the time line. This figure illustrates the exact computation and communication patterns for a parallel SAMR implementation. 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. Further, the computation and communication steps for each refinement level are interleaved. This behavior makes partitioning the dynamic SAMR grid hierarchy to both balance load and minimize communication overheads a significant challenge.

C. Spatial and Temporal Heterogeneity of SAMR Applications

The space-time heterogeneity of SAMR applications is illustrated using the 3-D compressible turbulence simulation kernel solving the Richtmyer-Meshkov (RM3D) instability [5] in Figure 3. The figure shows a selection of snapshots of the RM3D adaptive grid hierarchy as well as a plot of its load dynamics at different regrid steps. Since the adaptive grid hierarchy remains unchanged between two regrid steps, the workload dynamics and other features of SAMR applications are hence measured with respect to regrid steps. The workload in this figure represents the computational/storage requirement, which is computed based on the number of grid points in the grid hierarchy. Application variables are typically defined at these grid points and are updated at each iteration of the simulation, and consequently, the computational/storage requirements are proportional to the number of grid points. The snapshots in this figure clearly demonstrate the dynamics and space-time heterogeneity of SAMR applications - different subregions in the computational domain have different computational and communication requirements and regions of refinement are created, deleted, relocated, and grow/shrink at runtime.

III. Related Work

Parallel SAMR implementations presented in [10], [12], [13] use dynamic partitioning and load-balancing algorithms. These approaches view the system as a flat pool of processors. They are based on a 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.

Dynamic load balancing schemes proposed in [20] involve two phases: *moving-grid phase* and *splitting-grid phase*. The first phase is intended to move load from overloaded processors to underloaded processors. The second phase is triggered when the direct grid movement cannot balance the load among processors. These schemes improve the performance by focusing load balance and are suited for coarse-grained load balancing without considering the locality of patches. Dynamic load balancing schemes have been further extended to support distributed SAMR applications [21] using two phases: global load balancing and local load balancing. These load balancing in these schemes does not explicitly address the spatial and temporal heterogeneity exhibited by SAMR applications. In the SAMRAI library [10], [18], after the construction of computational patches, patches are assigned to processors using a greedy bin-packing procedure. SAMRAI uses the Morton space-filling curve technique to maintain spatial locality for patch distribution. To further enhance scalability of SAMR applications using the SAMRAI framework, Wissink et. al. proposed a *Recursive Binary Box Tree* algorithm to improve communication schedule construction [18]. A simplified point-clustering algorithm based on Berger-Regoutsos algorithm [22] has also been presented. The reduction of runtime complexity using these algorithms substantially improves the scalability of parallel SAMR applications on up to 1024 processors. As mentioned above, the SAMRAI framework uses patch-based partitioning schemes, which result in good load balance but might cause considerable inter-level communication and synchronization overheads. SAMR applications have been scaled on up to 6420 processors using FLASH and PARAMESH packages [8], [12]. The scalability is achieved using large domain dimensions ranging from 128x128x2560 to 1024x1024x20480. Further, a Morton space-filling curve technique has been applied to maintain spatial locality in PARAMESH. However, the space-time heterogeneity issues are not addressed explicitly.

A recent paper [23] presents a hierarchical partitioning and dynamic load balancing scheme using the Zoltan toolkit [3]. The proposed scheme first uses the multilevel graph partitioner

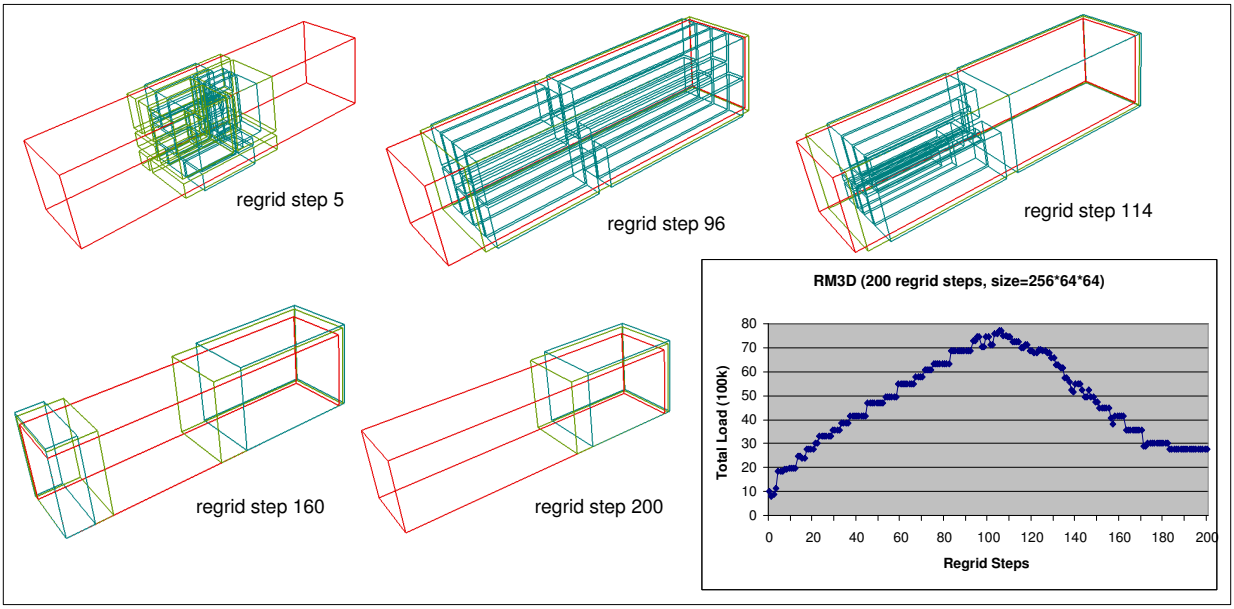


Fig. 3. Spatial and Temporal Heterogeneity and Load Dynamics for a 3D Richtmyer-Meshkov Simulation using SAMR.

in ParMetis [11] for across node partitioning in order to minimize communication across nodes. It then applies the recursive inertial bisection (RIB) method within each node. The approach was evaluated in small systems with eight processors in the paper. The characterization of SAMR applications presented in [14] is based on the entire physical domain. The research in this paper goes a step further by considering the characteristics of individual subregions. The concept of natural regions is presented in [24]. Two kinds of natural regions are defined: unrefined/homogeneous and refined/complex. The framework proposed then uses a bi-level domain-based partitioning scheme to partition the refined subregions. This approach is one of the first attempts to apply multiple partitioners concurrently to the SAMR domain. However, this approach restricts itself to applying only two partitioning schemes, one to the refined region and the other to the unrefined region.

IV. Hybrid Runtime Management Strategies

A. Adaptive Hierarchical Multi-Partitioner Framework

Figure 4 shows the basic operation of the AHMP framework. A critical task is to dynamically and efficiently identify regions that have similar requirements, called *clusters*. A cluster is a region of connected component grids that has relatively homogeneous computation and communication requirements. The input of AHMP is the structure of the current grid hierarchy (an example is illustrated in Figure 1), represented as a list of regions, which defines the runtime state of the SAMR application. AHMP operation consists of the following steps. First, a clustering algorithm is used to identify cluster hierarchies. Second, each cluster is characterized and its partitioning requirements identified. Available resources are also partitioned into corresponding resource groups based on the relative requirements of the clusters. A resource group is a set of possibly phyically proximal processors that are assigned

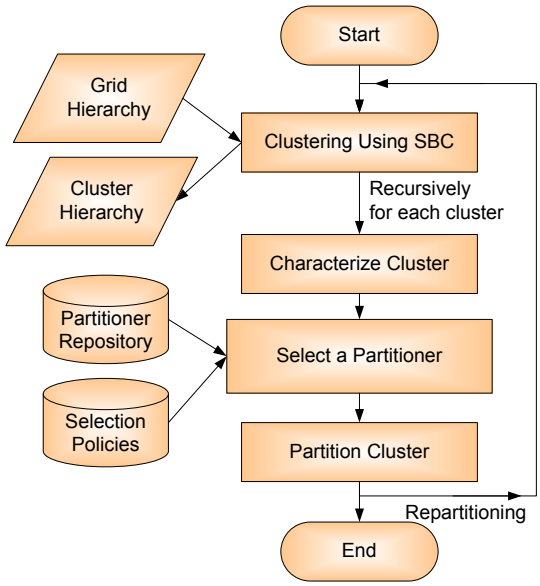


Fig. 4. A Flowchart for the AHMP Framework

a coarse-grained partition of the computational workload. A resource group is dynamically constructed based on the load assignment and distribution. Third, these requirements are used to select and configure an appropriate partitioner for each cluster. The partitioner is selected from a partitioner repository using selection policies. Finally, each cluster is partitioned and mapped to processors in its corresponding resource group. The strategy is triggered locally when the application state changes. State changes are determined using the load-imbalance metric described below. Partitioning proceeds hierarchically and incrementally. The identification and isolation of clusters uses a segmentation-based clustering (SBC) scheme. Partitioning schemes in the partitioner repository include Greedy Partitioning Algorithm (GPA), Level-

based Partitioning Algorithm (LPA), and others presented in Section IV-C. Partitioner selection policies consider cluster partitioning requirements, communication/computation requirements, scattered adaptation, and activity dynamics [14]. This paper specifically focuses on developing partitioning policies based on cluster requirements defined in terms of refinement homogeneity, which is defined in Section V-A.

AHMP extends our previous work on the Hierarchical Partitioning Algorithm (HPA) [16], which hierarchically applies a single partitioner, reducing global communication overheads and enabling incremental repartitioning and rescheduling. AHMP additionally addresses spatial heterogeneity by applying the most appropriate partitioner to each cluster based on its characteristics and requirements. As a result, multiple partitioners may concurrently operate on different subregions of the computational domain.

The load imbalance factor (LIF) metric is used as the criteria for triggering repartitioning and rescheduling within a local resource group, and is defined as follows,

$$LIF_A = \frac{\max_{i=1}^{A_n} T_i - \min_{i=1}^{A_n} T_i}{\sum_{i=1}^{A_n} T_i / A_n} \quad (1)$$

where A_n is the total number of processors in resource group A, and T_i is the estimated relative execution time between two consecutive regrid steps for the processor i, which is proportional to its load. In the numerator of the right-hand side of the above equation, we use the difference between maximum and minimum execution times to better reflect the impact of synchronization overheads. The local load imbalance threshold is γ_A. When $LIF_A > \gamma_A$, the repartitioning is triggered inside the local group. Note that the imbalance factor can be recursively calculated for larger groups as well.

B. Clustering Algorithms for Cluster Region Identification

The objective of clustering is to identify well-structured subregions in the SAMR grid hierarchy, called clusters. As defined above, a cluster is a region of connected component grids with relatively homogeneous partitioning requirements. This section describes the segmentation-based clustering (SBC) algorithm, which is based on space-filling curves (SFC) [25]. The algorithm is motivated by the locality-preserving property of SFCs and the localized nature of physical features in SAMR applications. Further, SFCs are widely used for domain-based partitioning for SAMR applications [13], [24]–[26]. Note that clusters are similar in concept to natural regions proposed in [24]. However, unlike natural regions, clusters are not restricted to strict geometric shapes, but are more flexible and take advantage of the locality-preserving property of SFCs.

Typical SAMR applications exhibit localized features, and thus result in localized refinements. Moveover, refinement levels and the resulting adaptive grid hierarchy reflect the application runtime state. Therefore, clustering subregions with similar refinement levels is desired.

The segmentation-based clustering algorithm is based on ideas in image segmentation [27]. The algorithm first defines load density factor (LDF) as follows:

$LDF(rlev)$ = (associated workload of patches with $levels >= rlev$ on the subregion) / (volume of the subregion at $rlev$)

where $rlev$ denotes the refinement level and the volume is for the subregion of interest in a 3-D domain (it will be area and length in case of 2-D and 1-D domains, respectively).

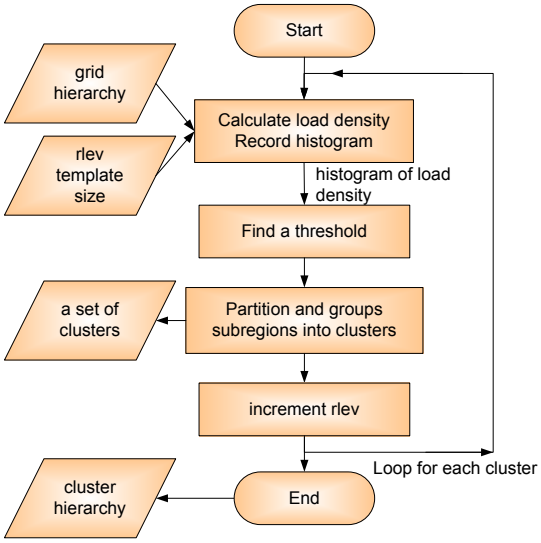


Fig. 5. Segmentation-based Clustering Algorithm

The SBC algorithm is illustrated in Figure 5. SBC aims to cluster domains with similar load density together to form cluster regions. The algorithm first smoothes out subregions that are smaller than a predefined threshold, which is referred to as the template size. Template size is determined by the stencil size of the finite difference method and the granularity constraint, maintaining appropriate computation/communication ratios to maximize performance and minimize communication overheads. A subregion is defined by a bounding box with lower-bound and upper-bound coordinates and the strides/steps along each dimension. The subregion list input to the SBC algorithm is created by applying the SFC indexing mechanism on the entire domain that consists of patches of different refinement levels. SBC follows the SFC index to extract subregions (defined by rectangular bounding boxes) from the subregion list until the size of the accumulated subregion set is over the template size. It then calculates the load density for this set of subregions and computes a histogram of its load density. SBC continues to scan through the entire subregion list, and repeats the above process, calculating the load density and computing histograms. Based on the histogram of the load density obtained, it then finds a clustering threshold θ. A simplified intermeans thresholding algorithm by iterative selection [27], [28] is used as shown below.

The goal of the thresholding algorithm is to partition the SFC-indexed subregion list into two classes C_0 and C_1 (which may not necessarily be two clusters as shown in Figure 7) using an "optimal" threshold T with respect to LDF, so that the LDF of all subregions in $C_0 \leq T$ and the LDF of all subregions in $C_1 > T$. Let μ_0 and μ_1 be the mean LDF of

C_0 and C_1, respectively. Initially, a threshold T is selected, for example, the mean of the entire list as a starting point. Then, for the two classes generated based on T, μ_0' and μ_1' are calculated, and a new threshold is computed as $T' = (\mu_0' + \mu_1')/2$. This process is repeated until the value of T converges.

Using the threshold obtained, subregions are further partitioned into several cluster regions. As a result, a hierarchical structure of cluster regions is created by recursively calling the SBC algorithm for finer refinement levels. The maximum number of clusters created can be adjusted to the number of processors available. Note that this algorithm has similarities to the point clustering algorithms proposed by Berger and Regoutsos in [22]. However, the SBC scheme differs from this scheme in two aspects. Unlike the Berger-Regoutsos scheme, which creates fine-grained clusters, the SBC scheme targets coarser granularity clusters. SBC also takes advantage of the locality-preserving properties of SFCs to potentially reduce data movement costs between consecutive repartitioning phases.

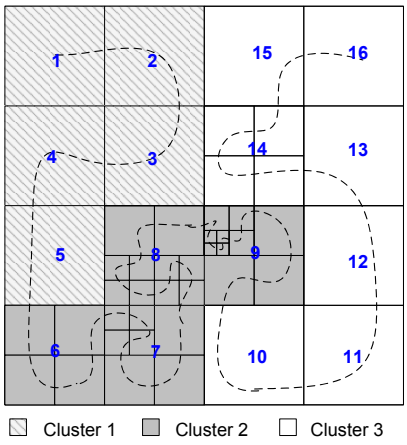


Fig. 6. Clustering Results for the SBC Scheme

The SBC algorithm is illustrated using a simple 2-D example in Figure 6. In this figure, SBC results in three clusters, which are shaded in the figure. Figure 7 shows the load density distribution and histogram for an SFC-indexed subdomain list. For this example, the SBC algorithm creates three clusters defined by the regions separated by the vertical lines in the figure on the left. The template size in this example is two boxes on the base level. The right figure shows a histogram of the load density. For efficiency and simplicity, this histogram is used to identify the appropriate threshold. For this example, the threshold is identified in between 1 and 9 using the intermeans thresholding algorithm. While there are many more sophisticated approaches for identifying good thresholds for segmentation and edge detection in image processing [27], [29], this approach is sufficient for our purpose. Note that a predefined minimum size for a cluster region is assumed. In this example, the subregion with index 14 in Figure 6 does not form a cluster as its size is less than the template size. It is instead clustered with another subregion in its proximity.

C. Partitioning Schemes and Partitioner Selection

For completeness, several partitioning algorithms within the GrACE package [30] are briefly described. The greedy partitioning algorithm (GPA) [13] is the default partitioning algorithm in GrACE. First, GPA partitions the entire domain into sub-domains such that each sub-domain keeps all refinement levels as a single composite grid unit. Thus all inter-level communications are local to a sub-domain and the inter-level communication time is eliminated. Second, GPA rapidly scans this list only once attempting to equally distribute load among all processors. It helps in reducing partitioning costs and works quite well for a relatively homogeneous computational domain.

For applications with localized features and deep grid hierarchies, GPA can result in load imbalances and hence lead to synchronization overheads at higher levels of refinement. To overcome this problem, the level-based partitioning algorithm (LPA) [16] attempts to simultaneously balance load and minimize synchronization cost. LPA essentially aims to balance workload at each refinement level among all processors in addition to balancing overall load. To further improve the runtime performance, the hierarchical partitioning algorithm (HPA) enables 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 [31]. HPA partitions the computational domain into subdomains and assigns them to hierarchical processor groups. Other partitioners in the partitioner repository include the bin-packing partitioner (BPA), the geometric multilevel + sequence partitioner (G-MISP+SP), and p-way binary dissection partitioner (pBD-ISP) [14], [16].

The characterization of clusters is based on their computation and communication requirements, runtime states, and the refinement homogeneity defined in Section V-A. An *octant approach* is proposed in [14] to classify the runtime states of a SAMR application with respect to (a) the adaptation pattern (scattered or localized); (b) whether runtime is dominated by computations or communications; and (c) the activity dynamics in the solution. A meta-partitioner is then proposed to enable the selection/mapping of partitioners according to the current state of an application in the octant. The mapping is based on an experimental characterization of partitioning techniques and application states using 5 partitioning schemes and 7 applications. The evaluation of partitioner quality is based on a five-component metric, including load imbalance, communication requirements, data migration, partitioning-introduced overheads, and partitioning time. In addition to these characterization and selection policies, we also consider refinement homogeneity. The overall goal of these new policies is to obtain better load balance for less refined clusters, and to reduce communication and synchronization costs for highly refined clusters. For example, the policy dictates that the GPA and G-MISP+SP partitioning algorithms be used for clusters with refinement homogeneity greater than some threshold and partitioning algorithms LPA and pBD-ISP be used for clusters with refinement homogeneity greater below the threshold.

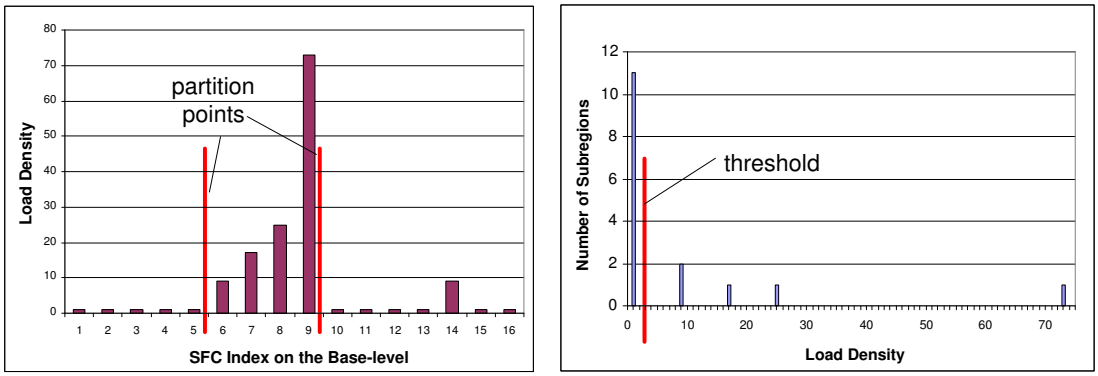


Fig. 7. Load Density Distribution and Histogram for SBC

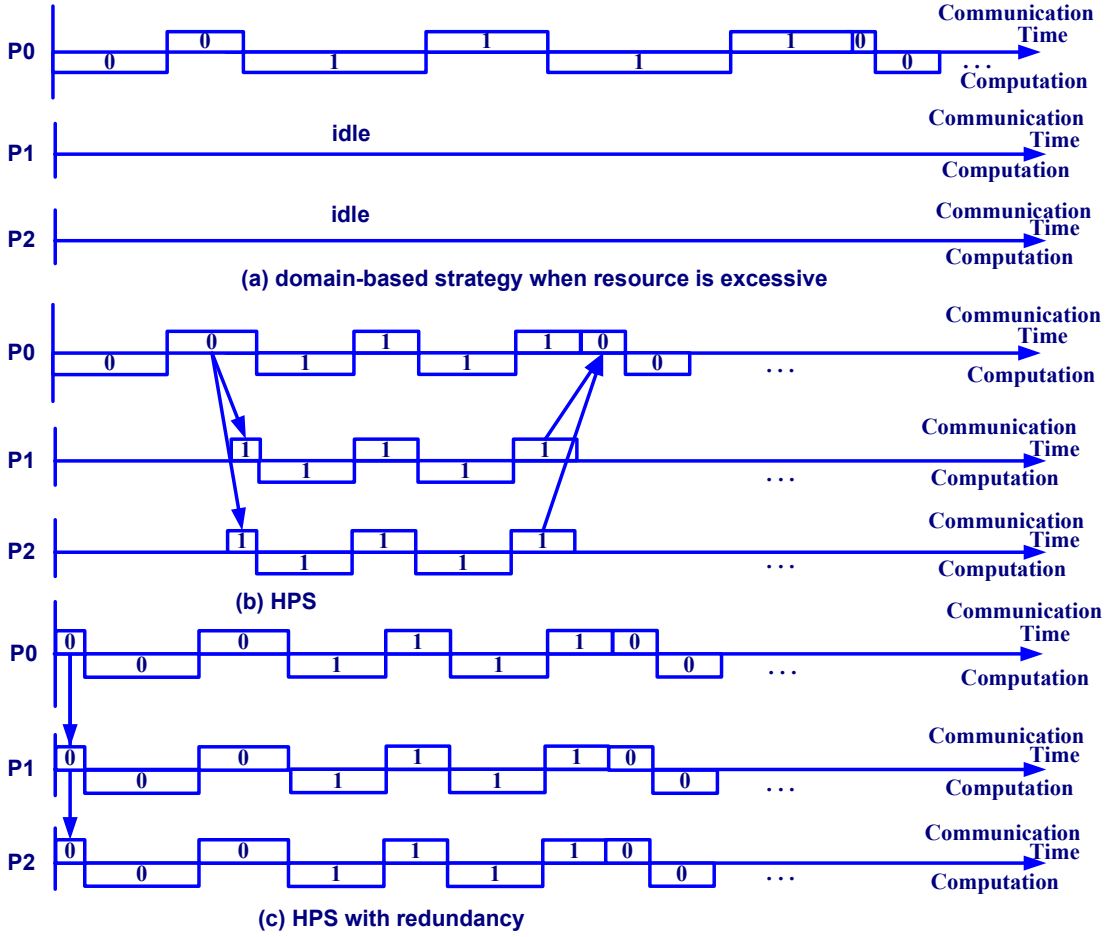


Fig. 8. Hybrid Partitioning Strategy

D. Hybrid Partitioning Strategy (HPS)

The parallelism that can be exploited by domain-based partitioning schemes is typically limited by granularity constraints. In cases where a very narrow region has deep refinements, domain-based partitioning schemes will inevitably result in significant load imbalances, especially when using a large-scale system. For example, assume that the predefined minimum dimension of a 3D block/grid on the base level is 4 grid points. In this case, the minimum workload (minimum partition granularity δ) of a composite grid unit with 3 refinement levels is $4^3+2\times8^3+2\times2\times16^3 = 17472$, i.e., the granularity constraint is $\delta \geq 17472$ units. Such a composite block can result in significant load imbalance if domain-based partitioning is used exclusively. To expose more parallelism in these cases, a patch-based partitioning approach must be used. HPS combines domain-based and patch-based schemes. To reduce communication overheads, the application of HPS is restricted to a cluster region that is allocated to a single resource group. Further, HPS is only applied when certain conditions are met. These conditions include: (1) resources are sufficient, (2) resources are under-utilized, and (3) the gain by

using HPS outweighs the extra communication cost incurred. For simplicity, the communication and computation process of HPS is illustrated using three processors in a resource group in Figure 8. The cluster has two refinement levels in this example.

HPS has two variants: one is pure hybrid partitioning with pipelining (pure HPS) and the other is hybrid partitioning with redundant computation and pipelining (HPS with redundancy). HPS splits the smallest domain-based partitions into patches at different refinement levels, partitions the finer patches into n partitions and assigns each partition to a different processor in the group of n processors. When this process extends to many refinement levels, it is analogous to the pipelining process, where the operation at each refinement level represents each pipelining stage. Since the smallest load unit on the base-level (level 0) can not be further partitioned, the pure HPS scheme maps the level 0 patch to a single processor, while the HPS with redundancy scheme redundantly computes the level 0 patch at all processors in the group. Although pure HPS saves redundant computation, it needs inter-level communication from the level 0 patch to the other patches, which can be expensive. In contrast, HPS with redundancy trades computing resource for less inter-level communication overheads. To avoid significant overheads, HPS schemes are applied only in a small resource group, e.g., an SMP node with 8 processors.

To specify the criteria for choosing HPS, the resource sufficiency factor (RSF) is defined by,

$$RSF = \frac{N_{rg}}{L_q / L_\delta} \tag{2}$$

where L_q denotes the total load for a cluster region, N_{rg} denotes the total number of processors in a resource group, and L_δ, the granularity, denotes the load on the smallest base-level subregion with the maximum refinement level. In the case of deep refinements, L_δ can be quite large. When $RSF > \rho$, where ρ is the threshold, and resources are under-utilized, HPS can be applied to explore additional parallelism. The threshold is determined statistically through sensitivities analysis for a set of applications.

The basic operations of HPS consist of pairing two refinement levels, duplicating the computation on the coarser patch and partitioning the finer patch across the resource group. The operation of HPS is as follows: (1) AHMP generates a set of clusters, and assigns each cluster to a resource group; (2) Within each resource group, the algorithm checks whether or not to trigger HPS, and if the criteria defined above are met, AHMP selects HPS for the cluster and the corresponding resource group; (3) HPS splits the cluster into patches at different refinement levels, assigns the patches to individual processors within the resource group, and coordinates the communication and computation as illustrated in Figure 8. Note that HPS can be recursively applied to patches of deeper refinement hierarchies.

E. Application-Level Out-of-Core (ALOC) Strategy

When available physical memory is not sufficient for the application, one option is to rely on the virtual memory mechanism of the operating system (OS). OS handles page faults and replaces less frequently used pages with the required pages from disks. OS however has little knowledge of the characteristics of an application and its memory access pattern. Consequently, it will result in many unnecessary swap-in and swap-out operations, which are very expensive. Data rates from disks are approximately two orders of magnitude lower than those from memory [32]. In many systems, OS sets a default maximum limit on physical and virtual memory allocation. When an application uses up this quota, it cannot proceed and crashes. Experiments show that system performance degrades during excessive memory allocation due to high page fault rates causing memory thrashing [33]–[35]. As a result, the amount of allocated memory and the memory usage pattern play a critical role in overall system performance.

To address these issues, an application-level out-of-core scheme (ALOC) is designed that exploits the application memory access patterns and explicitly keeps the working-set of application patches while swapping out other patches.

The ALOC mechanism proactively manages application-level *pages*, i.e., the computational domain patches. It attempts to not only improve performance, but also enhance survivability when available memory is insufficient. For instance, as shown in Figure 3, the RM3D application requires 4 times more memory during the peak requirements than the average requirements, while the peak time lasts for less than 10% of the total execution time.

As illustrated in Figure 9, the ALOC scheme incrementally partitions the local grid hierarchy into temporal virtual computational units (*T-VCU*) according to refinement levels and runtime iterations. In the figure, the notation *T-VCU*$^a_{b,c}$ denotes the temporal VCU, where a denotes the time step at the base level, b denotes the current refinement level, and c is the time step at the current level. To avoid undesired page-swapping, ALOC releases the memory allocated by lower-level patches and explicitly swaps them out to the disk. The ALOC mechanism is triggered when the ratio between the amount of memory allocated and the size of the physical memory is above a predefined threshold or the number of page faults increases above a threshold. The appropriate thresholds are selected based on experiments. Specifically, each process periodically monitors its memory usage and the page faults incurred. Memory usage information is obtained by tracking memory allocations and de-allocations, while page fault information is obtained using the *getrusage()* system call.

V. Experimental Evaluation

A. Evaluating the Effectiveness of the SBC Clustering Algorithm

To aid the evaluation of the effectiveness of the SBC clustering scheme, a clustering quality metric is defined. The metric consists of two components: (1) the static quality and (2) the dynamic quality of the cluster regions generated. The static quality of a cluster is measured in terms of its refinement homogeneity and the efficiency of the clustering algorithm. The dynamic quality of a cluster is measured in terms of its communication costs (intra-level, inter-level, and data migration). These criteria are defined as follows.

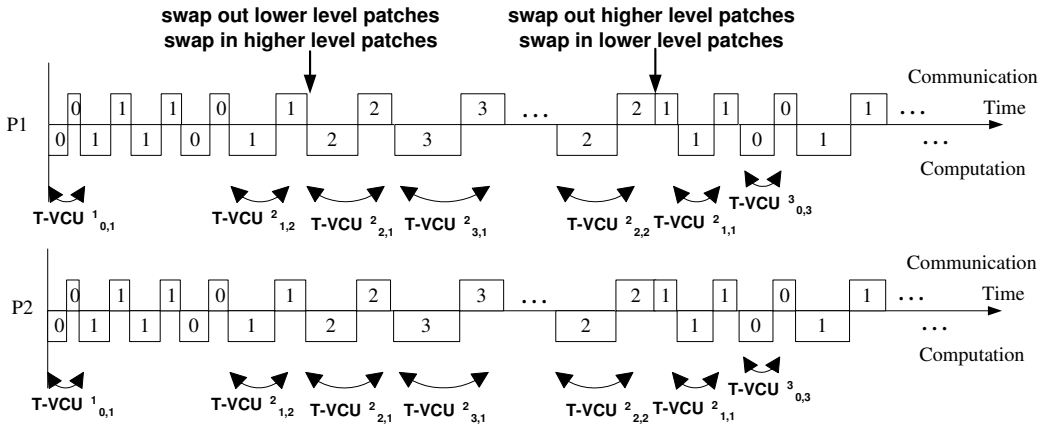


Fig. 9. Application-Level Out-of-core Strategy

(1) **Refinement Homogeneity**: This measures the quality of the structure of a cluster. Let $|R_i^{total}(l)|$ denote the total workload of a subregion or a cluster at refinement level l, which is composed of $|R_i^{ref}(l)|$, the workload of refined regions, and $|R_i^{unref}(l)|$, the workload of un-refined regions at refinement level l. Refinement homogeneity is recursively defined between two refinement levels as follows:

$$H_i(l) = \frac{|R_i^{ref}(l)|}{|R_i^{total}(l)|} \quad (3)$$

$$H_{all}(l) = \frac{1}{n}\sum_{i=1}^{n} H_i(l), \text{if } |R_i^{ref}(l)| \neq 0 \quad (4)$$

where n is the total number of subregions that have refinement level $l+1$. A goal of AHMP is to maximize the refinement homogeneity of a cluster as partitioners work well on relatively homogeneous regions.

(2) **Communication Cost**: This measures the communication overheads of a cluster and includes inter-level communication, intra-level communication, synchronization cost, and data migration cost as described in Section II-B. A goal of AHMP is to minimize the communication overheads of a cluster.

(3) **Clustering Cost**: This measures the cost of the clustering algorithm itself. As mentioned above, SAMR applications require regular re-partitioning and re-balancing, and as a result clustering cost becomes important. A goal of AHMP is to minimize the clustering cost.

Partitioning algorithms typically work well on highly homogeneous grid structures and can generate scalable partitions with desired load balance. Hence, it is important to have a quantitative measurement to specify the homogeneity. Intuitively, the refinement homogeneity metric attempts to isolate refined clusters that are potentially heterogeneous and are difficult to partition. In contrast, unrefined or completely refined clusters are homogeneous at that refinement level.

The effectiveness of SBC-based clustering is evaluated using the metrics defined above. The evaluation compares the refinement homogeneity of the 6 SAMR application kernels with and without clustering. These application kernels span multiple domains, including computational fluid dynamics (compressible turbulence: RM2D and RM3D, supersonic flows: ENO2D), oil reservoir simulations (oil-water flow: BL2D and BL3D), and the transport equation (TP2D). The applications are summarized in Table I.

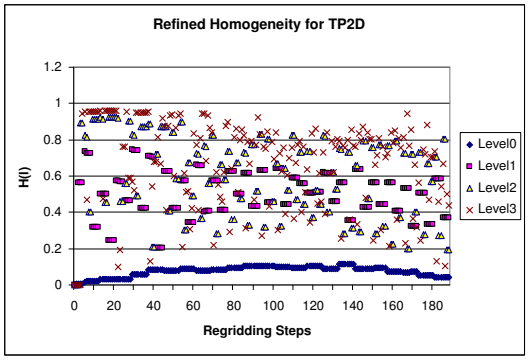


Fig. 10. Refinement Homogeneity for the Transport2D Application Kernel (4 levels of refinement)

Figure 10 shows the refinement homogeneity for the TP2D application with 4 refinement levels without any clustering. The refinement homogeneity is smooth for level 0 and very dynamic and irregular for levels 1, 2 and 3.

TABLE II
AVERAGE REFINEMENT HOMOGENEITY $H(l)$ FOR 6 SAMR APPLICATIONS

Application	Level0	Level1	Level2	Level3
TP2D	0.067	0.498	0.598	0.6680
RM2D	0.220	0.680	0.830	0.901
RM3D	0.427	0.618		
ENO2D	0.137	0.597	0.649	0.761
BL3D	0.044	0.267		
BL2D	0.020	0.438	0.406	0.316

The average refinement homogeneity of the 6 SAMR applications without clustering is presented in Table II. The refinement homogeneity is calculated for the entire domain and averaged among 100 regridding steps. The table shows that the refinement homogeneity $H(l)$ increases as the refinement

TABLE I
SAMR APPLICATION KERNELS

Apps	Dim	Description	Characteristics
TP	2D	A benchmark kernel for solving transport equation, included in the GrACE toolkit [30].	Intense activity in very narrowly concentrated regions. Key partitioning requirement: minimize partitioning overheads.
RM	2D/3D	A compressible turbulence application solving the Richtmyer-Meshkov (RM) instability. RM is a fingering instability which occurs at a material interface accelerated by a shock wave. This instability plays an important role in studies of supernova and inertial confinement fusion. It is a part of the virtual shock physics test facility (VTF) developed by the ASCI/ASAP Center at Caltech [5].	Intense activity in relatively larger and scattered regions. Key partitioning requirement: minimize communication and balance load at each refinement level.
ENO	2D	A computational fluid dynamics application for studying supersonic flows. The application has several features including bow shock, Mach stem, contact discontinuity, and a numerical boundary. ENO2D is also a part of the VTF, a suite of computational applications [5].	Intense activity in larger regions. Key partitioning requirement: minimize load imbalance.
BL	2D/3D	An application for studying oil-water flow simulation (OWFS) following the Buckley-Leverette model. It is used for simulation of hydrocarbon pollution in aquifers. This kernel is a part of the IPARS reservoir simulation toolkit (Integrated Parallel Accurate Reservoir Simulator) developed by the University of Texas at Austin [36].	Intense activity in very narrow and sparse regions, which are highly scattered. Key partitioning requirement: minimize communication and data migration overheads.

level l increases. This observation well reflects the physical properties of SAMR applications, i.e., refined regions tend to be further refined. Moreover, these applications typically exhibit intensive activities in narrow regions. Typical ranges of $H(l)$ are: $H(0) \in [0.02, 0.22]$, $H(1) \in [0.26, 0.68]$, $H(2) \in [0.59, 0.83]$ and $H(3) \in [0.66, 0.9]$. Several outliers require some explanation. In case of the BL2D application, average $H(2) = 0.4$. However, the individual values of $H(2)$ are in the range $[0.6, 0.9]$ with many scattered zeros. Since the refinement homogeneity on level 3 and above is typically over 0.6 and refined subregions at higher refinement levels tend to be more scattered, the clustering scheme focuses efforts on clustering level 0, 1 and 2. Furthermore, based on these statistics, we select the thresholds θ for switching between different lower-level partitioners as follows: $\theta_0 = 0.4$, $\theta_1 = 0.6$, and $\theta_2 = 0.8$, where the subscripts denote the refinement level.

Figure 11 and Table III demonstrate the improvements in *refinement homogeneity* using the SBC algorithm. Figure 11 shows the effects of using SBC on level 0 for the Transport2D application. The original homogeneity $H(0)$ is in the range [0, 0.15], while the improved homogeneity using SBC is in the range [0.5, 0.8].

The effects of clustering using SBC for the 6 SAMR applications are presented in Table III. In the table, the gain is defined as the ratio of the improved homogeneity over the original homogeneity at each level. The gains for TP2D, ENO2D, BL3D, and BL2D on level 0 are quite large. The gains for RM3D and RM2D applications are smaller because these applications already exhibit high refinement

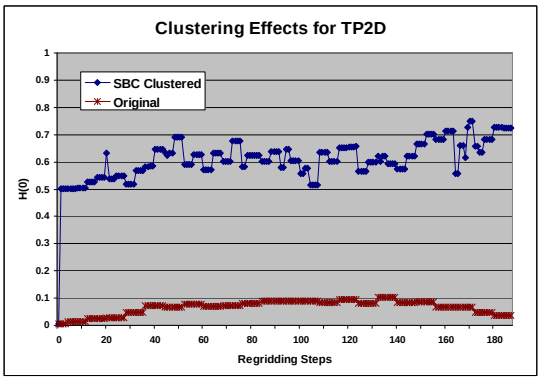


Fig. 11. Homogeneity Improvements using SBC for TP2D

homogeneity starting from level 0 as shown in Table II. These results demonstrate the effectiveness of the clustering scheme. Moreover, clustering increases the effectiveness of partitioners and improves overall performance as shown in the next section.

Communication Costs: The evaluation of communication cost uses a trace-driven simulation. The simulations are conducted as follows. First, a trace of the refinement behavior of the application at each regrid step was obtained by running the application on a single processor. Second, this trace is fed into the partitioners to partition and produce a new partitioned trace for multiple processors. Third, the partitioned trace is then fed into the SAMR simulator, which was developed at Rutgers University [37], to obtain the runtime performance measurements on multiple processors. Figure 12 shows the

TABLE III
HOMOGENEITY IMPROVEMENTS USING SBC FOR 6 SAMR APPLICATIONS

Application	Level0	Level1	Gain on Level0	Gain on Level1
TP2D	0.565	0.989	8.433	1.986
RM2D	0.671	0.996	3.050	1.465
RM3D	0.802	0.980	1.878	1.586
ENO2D	0.851	0.995	6.212	1.667
BL3D	0.450	0.583	10.227	2.184
BL2D	0.563	0.794	28.150	1.813

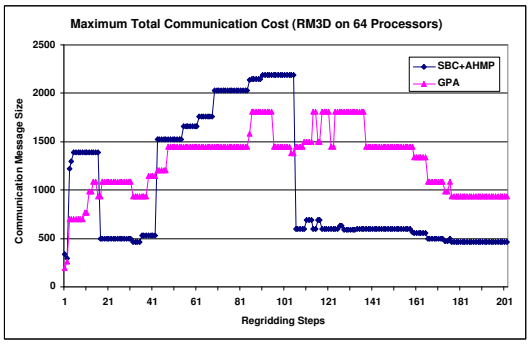


Fig. 12. Maximum Total Communication for RM3D on 64 Processors

total communication cost for the RM3D application on 64 processors for GPA and AHMP (using SBC) schemes. The figure shows that the overall communication cost is lower for SBC+AHMP. However, in the interval between regrid steps 60 and 100, SBC+AHMP exhibits higher communication costs. This is because the application is highly dynamic with scattered refinements during this period. The snapshot at the regrid step 96 in Figure 3 illustrates the scattered refinements. This in turn causes significant cluster movement during re-clustering. Note that the exiting simulator does not measure synchronization costs and thus does not reflect full performance benefits that can be achieved using AHMP. The actual performance gains due to AHMP are seen in Figure 15 based on actual runs.

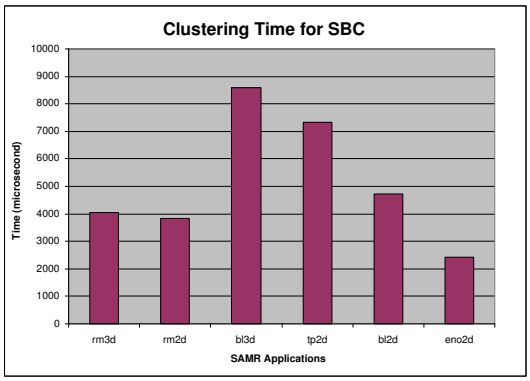


Fig. 13. Clustering Costs for the 6 SAMR Application Kernels

Clustering Costs: The cost of the SBC clustering algorithm is experimentally evaluated using the 6 different SAMR application kernels on a Beowulf cluster (Frea) at Rutgers University. The cluster consists of 64 processors and each processor has a 1.7 GHz Pentium IV CPU. The costs are plotted in Figure 13. As seen in this figure, the overall clustering time on average is less than 0.01 second. Note that the computational time between successive repartitioning/rescheduling phases is typically in the order of 10's of seconds, and as a result, the clustering costs are not significant.

B. Performance Evaluation

This section presents the experimental evaluation of AHMP. The overall performance benefit is evaluated on DataStar, the IBM SP4 cluster at San Diego Supercomputer Center. DataStar has 176 (8-way) P655+ nodes (SP4). Each node has 8 (1.5 GHz) processors, 16 GB memory, and CPU peak performance is 6.0 GFlops. The evaluation uses the RM3D application kernel with a base grid of size 256x64x64, up to 3 refinement levels, and 1000 base level time steps. The number of processors used was between 64 and 1280.

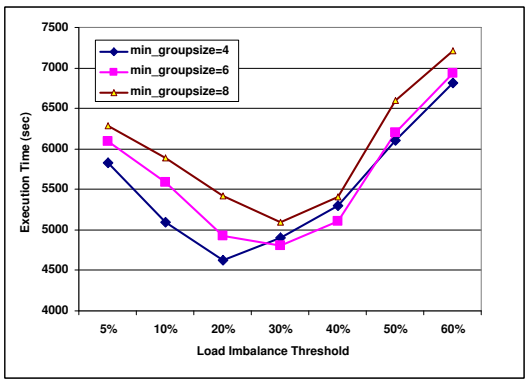


Fig. 14. Impact of Load Imbalance Threshold for RM3D on 128 Processors

Impact of Load Imbalance Threshold and Resource Group Size: As mentioned in Section 3, the load imbalance threshold γ is used to trigger repartitioning and redistribution within a resource group, where *min_group_size* is the minimum number of processors allowed in a resource group when the resource is partitioned hierarchically. This threshold plays an important role because it affects the frequency of redistribution and hence the overall performance. The impact of this threshold for different sizes of resource groups for the RM3D application on 128 processors is plotted in Figure 14. When γ increases from 5% to around 20% to 30%, the execution time

decreases. On the other hand, when γ increases from 30% to 60%, the execution time increases significantly. Smaller values of γ result in more frequent repartitioning within a resource group, while larger thresholds may lead to increased load imbalance. The best performance is obtained for $\gamma = 20\%$ and $min\_group\_size = 4$. Due to the increased load imbalance, larger group sizes do not enhance performance. The overall performance evaluation below uses $\gamma = 20\%$ and $min\_group\_size = 4$.

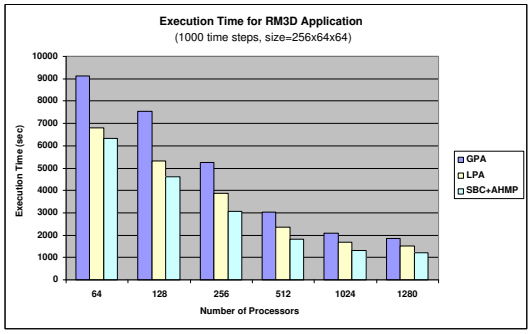


Fig. 15. Overall Performance for RM3D

Overall Performance: The overall execution time is plotted in Figure 15. The figure plots execution times for GPA, LPA and AHMP. The plot shows that SBC+AHMP delivers the best performance. Compared to GPA, the performance improvement is between 30% to 42%. These improvements can be attributed to the following factors: (1) AHMP takes advantage of the strength of different partitioning schemes matching them to the requirements of each cluster; (2) the SBC scheme creates well-structured clusters that reduce the communication traffic between clusters; (3) AHMP enables incremental repartitioning/redistribution and concurrent communication between resource groups, which extends the advantages of HPA [16].

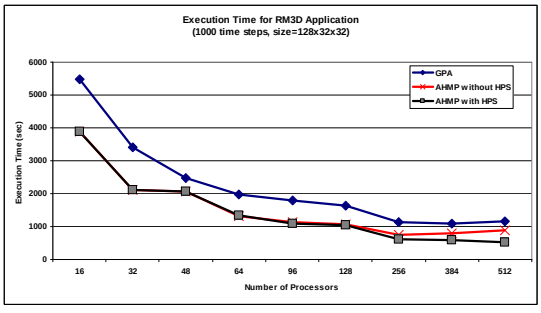


Fig. 16. Experimental Results: AHMP with and without HPS

Impact of Hybrid Partitioning: To show the impact of the hybrid partitioning strategy, we conduct the experiment using RM3D with a smaller domain, 128x32x32. All the other parameters are same as in the previous experiment. Due to the smaller computational domain, without HPS, the overall performance degrades when the application is deployed on a cluster with over 256 processors. The main reason is that, without HPS, the granularity constraint and the increasing communication overheads overshadow the increased computing resources. In contrast, with HPS, AHMP can further scale up to 512 processors with performance gains up to 40% compared to the scheme without HPS. Note that the maximum performance gain (40%) is achieved when using 512 processors, where the scheme without HPS results in the degraded performance.

Impact of Out-of-Core: The ALOC scheme has been implemented using the HDF5 library [38], which is widely used to store scientific data. The effect of the out-of-core scheme is evaluated using RM3D on the Frea Beowulf cluster. The configuration of RM3D consists of a base grid of size $128 \times 32 \times 32$, 4 refinement levels, and 100 base-level time steps (totally 99 regridding steps). The number of processors used is 64. Without ALOC, it took about 13507 seconds to complete 63 regridding steps at which point the application crashed. With ALOC, the application successfully completed the execution of 99 regridding steps. The execution time for the same 63 regridding steps was 9573 seconds, which includes 938 seconds for explicit out-of-core I/O operations. Figure 17 shows the page faults distribution and the execution time for experiments using NonALOC and ALOC schemes. As seen in the figure, without ALOC, the application incurs significant page faults. With ALOC, the number of page faults is reduced. As a result, the ALOC scheme improves the performance and enhances the survivability.

VI. Conclusion

This paper presented hybrid runtime management strategies and the adaptive hierarchical multi-partitioner (AHMP) framework to address space-time heterogeneity in dynamic SAMR applications. A segmentation-based clustering algorithm (SBC) was used to identify cluster regions with relatively homogeneous partitioning requirements in the adaptive computational domain. The partitioning requirements of each cluster were identified and used to select the most appropriate partitioning algorithm for the cluster. Further, this paper presented hybrid partitioning strategy (HPS), which involves pipelining process and improves the system scalability by trading space for time, and the application-level out-of-core scheme (ALOC), which addresses insufficient memory resources and improves overall performance and survivability of SAMR applications. Overall, the AHMP framework and its components have been implemented and experimentally evaluated on up to 1280 processors. The experimental evaluation demonstrated both, the effectiveness of the clustering as well as the performance improvements using AHMP. Future work will consider adaptive tuning of control parameters and will extend the proposed strategies to support workload heterogeneity in multi-physics applications.

References

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

[2] M. Berger and P. Colella, "Local adaptive mesh refinement for shock hydrodynamics," *Journal of Computational Physics*, vol. 82, pp. 64–84, 1989.

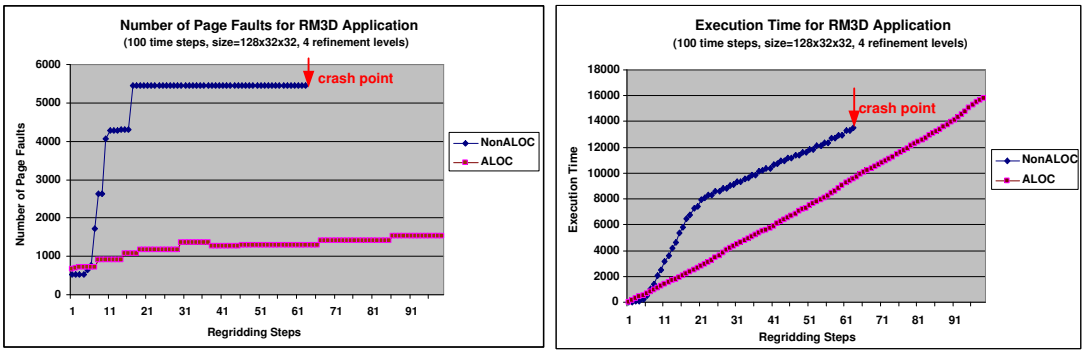


Fig. 17. Number of Page Faults: NonALOC versus ALOC

[3] K. Devine, E. Boman, R. Heaphy, B. Hendrickson, and C. Vaughan, "Zoltan data management services for parallel dynamic applications," *Computing in Science and Engineering*, vol. 4, no. 2, pp. 90–97, 2002.

[4] S. Das, D. Harvey, and R. Biswas, "Parallel processing of adaptive meshes with load balancing," *IEEE Transactions on Parallel and Distributed Systems*, vol. 12, no. 12, pp. 1269–1280, 2001.

[5] 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*, vol. 23, pp. 39–50, 2002.

[6] S. Hawley and M. Choptuik, "Boson stars driven to the brink of black hole formation," *Physical Review D*, vol. 62:10, no. 104024, 2000.

[7] J. Ray, H. N. Najm, R. B. Milne, K. D. Devine, and S. Kempka, "Triple flame structure and dynamics at the stabilization point of an unsteady lifted jet diffusion flame," *Proceedings of Combust. Inst. 2000*, vol. 25, no. 1, pp. 219–226, 2000.

[8] A. Calder, H. Tufo, J. Turan, M. Zingale, G. Henry, B. Curtis, L. Dursi, B. Fryxell, P. MacNeice, K. Olson, P. Ricker, R. Rosner, and F. Timmes, "High performance reactive fluid flow simulations using adaptive mesh refinement on thousands of processors," in *Proceedings of the 2000 ACM/IEEE conference on Supercomputing (CDROM)*, 2000.

[9] L. V. Kale, "Charm," uRL: http://charm.cs.uiuc.edu/research/charm/.

[10] R. D. Hornung and S. R. Kohn, "Managing application complexity in the SAMRAI object-oriented framework," *Concurrency and Computation - Practice & Experience*, vol. 14, no. 5, pp. 347–368, 2002.

[11] G. Karypis and V. Kumar, "Parmetis," 2003, http://www-users.cs.umn.edu/~karypis/metis/parmetis/index.html.

[12] P. MacNeice, "Paramesh," http://esdcd.gsfc.nasa.gov/ESS/macneice/paramesh/paramesh.html.

[13] M. Parashar and J. Browne, "On partitioning dynamic adaptive grid hierarchies," in *29th Annual Hawaii International Conference on System Sciences*, 1996, pp. 604–613.

[14] J. Steensland, S. Chandra, and M. Parashar, "An application-centric characterization of domain-based SFC partitioners for parallel SAMR," *Ieee Transactions on Parallel and Distributed Systems*, vol. 13, no. 12, pp. 1275–1289, 2002.

[15] P. E. Crandall and M. J. Quinn, "A partitioning advisory system for networked data-parallel programming," *Concurrency: Practice and Experience*, vol. 7, no. 5, pp. 479–495, 1995.

[16] X. Li and M. Parashar, "Dynamic load partitioning strategies for managing data of space and time heterogeneity in parallel SAMR applications," in *The 9th International Euro-Par Conference (Euro-Par 2003)*, *Lecture Notes in Computer Science*, vol. 2790. Springer-Verlag, 2003, pp. 181–188.

[17] ——, "Using clustering to address the heterogeneity and dynamism in parallel SAMR application," in *The 12th Annual IEEE International Conference on High Performance Computing (HiPC05)*, 2005.

[18] A. Wissink, D. Hysom, and R. Hornung, "Enhancing scalability of parallel structured AMR calculations," in *The 17th ACM International Conference on Supercomputing (ICS03)*, 2003, pp. 336–347.

[19] M. Thune, "Partitioning strategies for composite grids," *Parallel Algorithms and Applications*, vol. 11, pp. 325–348, 1997.

[20] Z. Lan, V. Taylor, and G. Bryan, "Dynamic load balancing for adaptive mesh refinement applications: Improvements and sensitivity analysis," in *The 13th IASTED International Conference on Parallel and Distributed Computing and Systems (PDCS2001)*, 2001.

[21] ——, "Dynamic load balancing of SAMR applications on distributed systems," *Journal of Scientic Programming*, vol. 10:4, pp. 319–328, 2002.

[22] M. Berger and I. Regoutsos, "An algorithm for point clustering and grid generation," *IEEE Transactions on Systems, Man and Cybernetics*, vol. 21, no. 5, pp. 1278–1286, 1991.

[23] J. D. Teresco, J. Faik, and J. E. Flaherty, "Hierarchical partitioning and dynamic load balancing for scientific computation," Williams College Department of Computer Science, Tech. Rep. CS-04-04, 2004, (also in Proceedings of PARA '04).

[24] J. Steensland, "Efficient partitioning of structured dynamic grid hierarchies," Ph.D. dissertation, Uppsala University, 2002.

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

[26] J. Pilkington and S. Baden, "Dynamic partitioning of non-uniform structured workloads with spacefilling curves," *IEEE Transactions on Parallel and Distributed Systems*, vol. 7, no. 3, 1996.

[27] R. C. Gonzalez and R. E. Woods, *Digital Image Processing*, 2nd ed. Upper Saddle River, NJ: Prentice Hall, 2002.

[28] T. Ridler and S. Calvard, "Picture thresholding using an iterative selection method," *IEEE Transactions on Systems, Man and Cybernetics*, vol. 8, no. 630-632, 1978.

[29] N. Otsu, "A threshold selection method from gray-level histogram," *IEEE Transactions on Systems, Man and Cybernetics*, vol. 6, no. 1, pp. 62–66, 1979.

[30] M. Parashar, "Grace," http://www.caip.rutgers.edu/~parashar/TASSL/.

[31] X. Li and M. Parashar, "Hierarchical partitioning techniques for structured adaptive mesh refinement applications," *Journal of Supercomputing*, vol. 28, no. 3, pp. 265 – 278, 2004.

[32] J. L. Hennessy, D. A. Patterson, and D. Goldberg, *Computer Architecture: A Quantitative Approach*. Morgan Kaufmann, 2002.

[33] M. S. Potnuru, "Automatic out-of-core execution support for CHARM++," Master thesis, University of Illinois at Urbana-Champaign, Tech. Rep., 2003.

[34] N. Saboo and L. V. Kale, "Improving paging performance with object prefetching," in *International Conference on High Performance Computing (HiPC01)*, 2001.

[35] J. Tang, B. Fang, M. Hu, and H. Zhang, "Developing a user-level middleware for out-of-core computation on grids," in *IEEE International Symposium on Cluster Computing and the Grid*, 2004, pp. 686–690.

[36] "IPARS," http://www.cpge.utexas.edu/new_generation/.

[37] S. Chandra and M. Parashar, "A simulation framework for evaluating the runtime characteristics of structured adaptive mesh refinement applications," Center for Advanced Information Processing, Rutgers University, Tech. Rep. TR-275, Sep. 2004.

[38] "Hdf5," http://hdf.ncsa.uiuc.edu/HDF5/.

APPENDIX
GLOSSARY

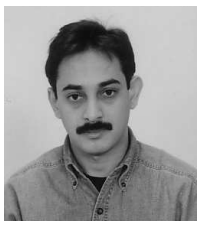
- AHMP: Adaptive Hierarchical Multi-Partitioner Strategy
- ALOC: Application-Level Out-of-core Strategy
- BPA: Bin-packing Partitioning Algorithm
- G-MISP+SP: Geometric Multilevel Inverse SFC Partitioning + Sequence Partitioning
- GPA: Greedy Partitioning Algorithm
- LDF: Load Density Factor

- HPA: Hierarchical Partitioning Algorithm
- HPS: Hybrid Partitioning Strategy
- LIF: Load Imbalance Factor
- LPA: Level-based Partitioning Algorithm
- pBD-ISP: p-way Binary Dissection-Inverse SFC Partitioning
- RSF: Resource Sufficiency Factor
- SAMR: Structured Adaptive Mesh Refinement Technique
- SBC: Segmentation-Based Clustering Algorithm
- SFC: Space-Filling Curve Technique

ACKNOWLEDGMENT

The authors would like to thank Sumir Chandra and Johan Steensland for many insightful research discussions. The authors would also like to thank the editors and referees for their suggestions, which have helped improve the quality and presentation of this paper. The research presented in this paper is supported in part by National Science Foundation via grants numbers ACI 9984357, EIA 0103674, EIA 0120934, ANI 0335244, CNS 0305495, CNS 0426354 and IIS 0430826, and by Department of Energy via the grant number DE-FG02-06ER54857.

Xiaolin Li is Assistant Professor of Computer Science at Oklahoma State University. He received the BE degree from Qingdao University, China, the ME degree from Zhejiang University, China, and the PhD degrees from National University of Singapore and Rutgers University, USA. His research interests include distributed systems, sensor networks, and software engineering. He is directing the Scalable Software Systems Laboratory (`http://www.cs.okstate.edu/~xiaolin/S3Lab`). He is a member of IEEE.



Manish Parashar is Professor of Electrical and Computer Engineering at Rutgers University, where he also is director of the Applied Software Systems Laboratory. He received a BE degree in Electronics and Telecommunications from Bombay University, India and MS and Ph.D. degrees in Computer Engineering from Syracuse University. He has received the Rutgers Board of Trustees Award for Excellence in Research (2004-2005), NSF CAREER Award (1999) and the Enrico Fermi Scholarship from Argonne National Laboratory (1996). His research interests include autonomic computing, parallel & distributed computing (including peer-to-peer and Grid computing), scientific computing, and software engineering.

Manish is a senior member of IEEE, member of the executive committee of the IEEE Computer Society Technical Committee on Parallel Processing (TCPP), part of the IEEE Computer Society Distinguished Visitor Program (2004-2006), and a member of ACM. He is the co-founder of the IEEE International Conference on Autonomic Computing (ICAC), serves on the editorial boards of several journals, and on the steering and program committees of several international workshops and conferences. For more information please visit `http://www.caip.rutgers.edu/~parashar/`.