# ADAPTIVE RUNTIME MANAGEMENT OF SPATIAL AND TEMPORAL HETEROGENEITY OF DYNAMIC SAMR APPLICATIONS

BY XIAOLIN LI

A dissertation submitted to the

Graduate School—New Brunswick

Rutgers, The State University of New Jersey

in partial fulfillment of the requirements

for the degree of

Doctor of Philosophy

Graduate Program in Electrical and Computer Engineering

Written under the direction of

Professor Manish Parashar

and approved by

_____

_____

_____

_____

_____

New Brunswick, New Jersey

October, 2005

## ABSTRACT OF THE DISSERTATION

# Adaptive Runtime Management of Spatial and Temporal Heterogeneity of Dynamic SAMR Applications

**by Xiaolin Li**

**Dissertation Director: Professor Manish Parashar**

Structured adaptive mesh refinement (SAMR) techniques provide an effective means for dynamically concentrating computational effort and resources to appropriate regions in the application domain and have the potential for enabling highly accurate solutions to simulations of complex systems. However, due to the dynamism and space-time heterogeneity exhibited by these techniques, their scalable parallel implementation continues to present significant challenges.

This thesis aims at designing and evaluating an adaptive runtime management system for parallel SAMR applications by explicitly considering their spatial and temporal heterogeneity on large systems. The key idea is to identify relatively homogenous regions in the computational domain at runtime and apply the most appropriate algorithms that address local requirements of these regions. A hybrid space-time runtime management strategy based on this idea has been developed,

which consists of three components. First, adaptive hierarchical strategies dynamically apply multiple partitioners to different regions of the application domain, in a hierarchical manner, to match the local requirements. Novel clustering and partitioning algorithms are developed. The strategy allows incremental repartitioning and rescheduling and concurrent operations. The second component is an application-level pipelining strategy, which trades space for time when resources are sufficiently large and under-utilized. The third component is an application-level out-of-core strategy, which trades time for space when resources are scarce in order to improve the performance and enhance the survivability of applications.

The proposed solutions have been implemented and experimentally evaluated on large-scale systems including the IBM SP4 cluster at San Diego Supercomputer Center with up to 1280 processors. These experiments demonstrate the performance benefits of the developed strategies. Finally, the GridMate simulator is developed to investigate applicability of theses strategies in Grid environments.

# Acknowledgements

I would like to express my sincere gratitude to my advisor Prof. Manish Parashar for his invaluable guidance, insightful ideas, and continuous encouragement during the course of this work and throughout my study at Rutgers. I am very thankful to Prof. Ivan Marsic, Prof. Hoang Pham, Prof. Deborah Silver, and Prof. Yanyong Zhang for being on my thesis committee and for their advice and suggestions regarding the thesis and beyond. I would like to thank Sumir Chandra and Johan Steensland for valuable research discussions and collaboration. I would like to thank Julian Cummings, Jaideep Ray, Ravi Samtaney, and Mary Wheeler for collaboration on the real-world SAMR applications, and Victors Berstis and Luba Cherbakov for collaboration on the Grid simulator. Moreover, I would like to thank my colleagues at The Applied Software Systems Laboratory (TASSL) and other friends at Rutgers for their friendship and help, which makes my study at Rutgers enjoyable and fruitful. I am also thankful to staff at the Center for Advanced Information Processing (CAIP) and Department of Electrical & Computer Engineering for their assistance and support.

I am grateful to Paul & Diana, Rick & Lorie, Jerry & Lucia, Zhengyi & Yingxiu, Zhengzhong & Xiuchun for their warmhearted help and guidance. My special thanks go to my family for their endless love and spiritual support.

The work presented in this thesis was supported in part by NSF via grants ACI 9984357, EIA 0103674, EIA 0120934, ANI 0335244, CNS 0305495, CNS 0426354 and IIS 0430826, and by DOE ASCI/ASAP (Caltech) via grant PC295251. I thank these funding agencies for their generous financial support.

# Table of Contents

# List of Tables

# List of Figures

# Chapter 1
# Introduction

## 1.1 Motivation

The last decade has witnessed a dramatic technology boost in computing, networking and storage. A personal computer today is as fast as a supercomputer in 1990 and can be installed with more than 100 GB storage, which is as much as an entire supercomputer center in 1990 [28]. The rapid advance of technologies, driven by Moore's law [66] for CPUs, storage systems, as well as Gilder's law [49] for networks, has led to parallel and distributed systems of unprecedented scales. Harnessing resources of such scale requires efficient runtime management strategies.

At the same time, a new generation of large-scale scientific and engineering simulations that require an increasing amount of computing and storage resources to provide new insights into complex systems, such as interacting black holes and neutron stars, formations of galaxies, combustion simulation, subsurface modeling and oil reservoir simulation [41, 51, 70]. Dynamically adaptive techniques, such as the dynamic structured adaptive mesh refinement (SAMR) technique [26], 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 adaptively concentrating computational effort and resources to appropriate regions of the domain at runtime. Large-scale parallel implementations of SAMR-based applications have the potential for accurately modeling complex physical phenomena and providing dramatic insights.

However, due to the dynamism and space-time heterogeneity, the scalable parallel implementation remains a significant challenge. Specifically, 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 exponential growth in computing, networking and storage technologies has also ushered in a new distributed computing paradigm, Grid computing, for solving grand challenge problems in varied domains of science, engineering and business [46, 47, 45]. A number of major Grid infrastructures are being developed and deployed [6, 11, 18] and many grand challenge problems are being tackled by exploiting the power of the Grid [1, 5, 9, 12, 15, 16]. Migrating dynamic and heterogeneous SAMR applications to Grid environments introduces another level of complexity and challenges.

### 1.1.1 Problem Statement

As mentioned above, SAMR applications exhibit dynamism and space-time heterogeneity, making their scalable implementation a significant challenge. To address the scalability and performance issues, this thesis aims at designing, implementing and evaluating an adaptive runtime management system for parallel SAMR applications by explicitly considering their dynamism and space-time heterogeneity on large-scale systems. The specific objectives are as follows.

- To explore the characteristics of parallel SAMR applications: to investigate

their communication/computation behavior and to analyze the localized features of the computational domain.

- To explicitly address the synchronization costs caused by the dynamism of parallel SAMR applications, by identifying the sources of these costs and designing efficient strategies to reduce their impact on performance.

- To explicitly address the spatial and temporal heterogeneity exhibited in SAMR applications. Specifically, to characterize the heterogeneity, identify and isolate subproblems, and design a divide-and-conquer strategy to address it. Further, to design strategies to enhance survivability of the application when the available resource is not sufficient.

- To investigate the applicability of the proposed strategies in Grid environments. In particular, to consider the coordination of application partitioning and resource scheduling strategies.

## 1.2 Research Overview

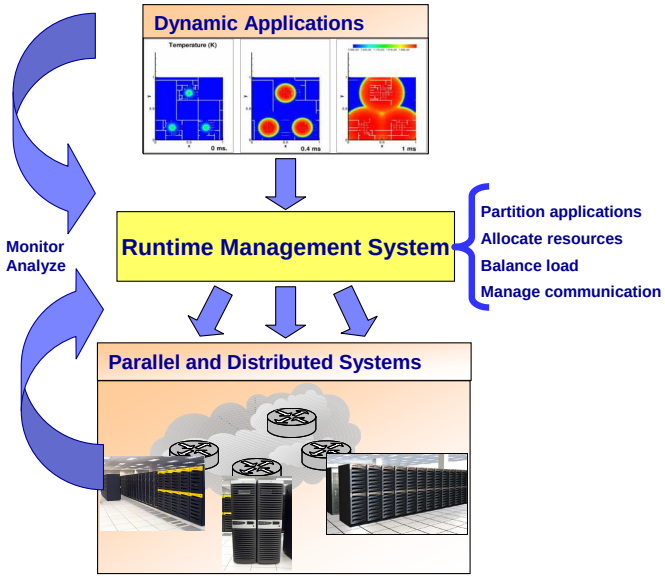


Figure 1.1: An Illustrative Runtime Management System

A runtime management system (RMS) is in charge of partitioning applications at runtime and distributing each partition to available resources in order to efficiently utilize resources and minimize the execution time of applications. Figure 1.1 shows an RMS as a middleware system that bridges the gap between applications and operating systems and provides efficient runtime support, primarily including dynamic partitioning, resource allocation, coordination, and load balancing. In the figure, the input application is a combustion simulation of hydrogen-air mixture with three initial ignition spots [70]. As shown in the figure, a RMS monitors the runtime states of applications and resources, analyzes requirements, and delivers a plan for partitioning, scheduling and executing the application on parallel and distributed systems to maximize their performance, efficiency and utilization. In case of SAMR applications, the RMS must address their dynamism and space-time heterogeneity.

The overall goal of this thesis is to design, implement and evaluate an adaptive runtime management system for parallel SAMR applications on parallel and distributed systems. To address the spatial and temporal heterogeneity, the key idea is to identify relatively homogenous regions in the computational domain at runtime and apply the most appropriate algorithms that address local requirements of these regions. A hybrid space-time runtime management strategy based on this idea has been developed, which consists of three components. First, adaptive hierarchical strategies dynamically apply multiple partitioners to different regions of the application domain, in a hierarchical manner, to match the local requirements. Novel clustering and partitioning algorithms are developed. The strategy allows incremental repartitioning and rescheduling and concurrent operations. The second component is an application-level pipelining strategy, which trades space for time when resources are sufficiently large and under-utilized. The third component is an application-level out-of-core strategy, which trades time for space when resources are scarce in order to improve the performance

and enhance the survivability of applications. To investigate the applicability of theses strategies in Grid environments, GridMate, a Grid simulator, has been developed.

## 1.3 Contributions

The main contributions of this thesis are as follows.

- A hybrid space-time runtime management strategy and system (HRMS). HRMS has been implemented and experimentally evaluated on large-scale systems with up to 1280 processors. These experiments demonstrate that HRMS successfully improves the overall performance and scalability by explicitly addressing the dynamism and heterogeneity of SAMR applications. The contributions of HRMS are elaborated as follows:
  - HRMS addresses the dynamism of SAMR applications by identifying and minimizing two kinds of synchronization costs: one is due to the flat organization of processors, and the other is due to the load imbalance at each refinement level. Hierarchical partitioning strategies organize processors in a hierarchical manner to match the hierarchy of the computational domain and enable incremental redistribution and concurrent communication to reduce the first kind of synchronization costs. Moreover, the level-based partitioning strategy explicitly addresses the second kind of synchronization costs by balancing both the overall workload and the workload at each refinement level.
  - HRMS addresses the space-time heterogeneity of SAMR applications by identifying and characterizing the localized requirements of the computational domain, and further applying multiple partitioners to match these localized requirements. In particular, the segmentation-based clustering scheme is developed to formulate well-structured subregions

and characterize their local requirements. These subregions are exploited by the adaptive hierarchical multi-partitioner to enable multiple partitioners to concurrently operate on different subregions.

  - HRMS is also able to handle different resource situations. When the resources are under-utilized, the application-level pipelining scheme leverages domain-based schemes to reduce communication overheads and patch-based schemes to reduce load imbalance. In contrast, when the available physical memory capacity is not sufficient to support the application, an application-level out-of-core scheme has been developed to exploit the memory access pattern of SAMR applications and enable incremental operation on patches at different refinement levels. As a result, it enhances the survivability.

- The design and evaluation of GridMate, a Grid simulator for distributed scientific applications on multi-site clusters. GridMate is able to model a multi-site heterogeneous Grid computing environment. It adopts a super-scheduler and local-scheduler scheduling paradigm and integrates partitioning and scheduling schemes in HRMS. As a result, it enables the performance evaluation of HRMS schemes on multi-site Grid environments.

## 1.4 Outline of the Thesis

The rest of the thesis is organized as follows. Chapter 2 presents the problem description and related work. Chapter 3 presents the conceptual framework of HRMS. Chapter 4 presents the hierarchical partitioning algorithm. Chapter 5 presents the adaptive hierarchical multi-partitioner approach. GridMate, a Grid simulator for distributed scientific applications, is presented in Chapter 5. Chapter 6 concludes the thesis and presents some future research directions.

# Chapter 2
# Problem Description and Related Work

As mentioned in the previous chapter, SAMR applications exhibit dynamism and spatial and temporal heterogeneity. As a result, it is a challenging problem to dynamically partition applications and distribute them on dynamic resources at runtime to maximize performance, efficiency and resource utilization. This chapter first describes the target applications, which are based on the SAMR technique. It then investigates the spatial and temporal heterogeneity of SAMR applications and the computation and communication patterns for parallel SAMR implementations. Further, it analyzes the computation and communication workload. After that, this chapter presents a taxonomy for runtime management for SAMR applications and describes related research efforts and systems.

## 2.1 Structured Adaptive Mesh Refinement

The applications targeted in this thesis are large-scale dynamic scientific and engineering applications, such as scientific simulations that solve sophisticated partial differential equations (PDEs) using Adaptive Mesh Refinement (AMR) [26] techniques. Partial differential equations are widely used for mathematically modeling and studying physical phenomena in science and engineering, such as heat transfer in solids, interacting black holes and neutron stars, electrostatics of conductive media, formations of galaxies, combustion simulation, wave propagation, and subsurface flows in oil reservoirs [41, 51, 70]. One numerical method to solve PDEs is to compute approximate solutions for discrete points by uniformly discretizing

the physical domain. This approach results in a homogeneous grid system with a uniform resolution over the entire physical domain. However, many physical phenomena exhibit shocks, discontinuities or steep gradients in localized regions of the physical domain. In these cases, to achieve acceptable resolution in these small regions, uniform resolution methods will result in a very fine discretization of the entire domain. Consequently, it requires a significant amount of unnecessary computation and storage.

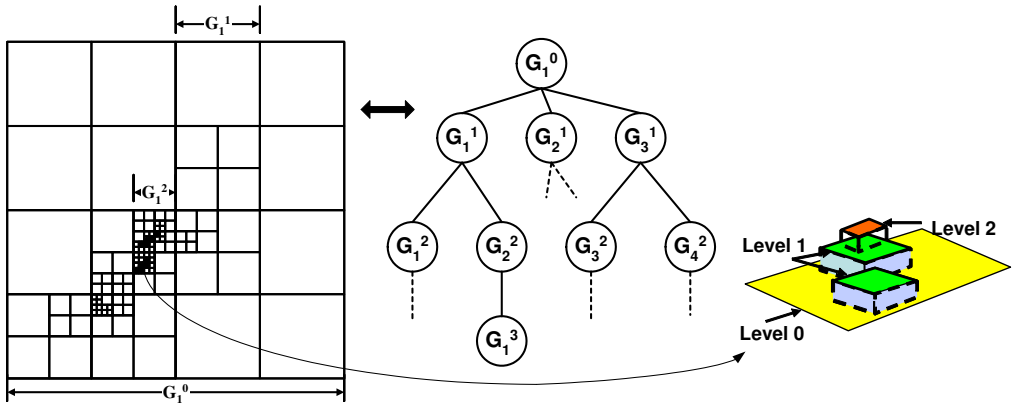


Figure 2.1: Adaptive Grid Hierarchy - 2D (Berger-Oliger AMR scheme), Courtesy: M. Parashar

Instead of using uniform discretization or refinement to obtain finer resolution, SAMR techniques dynamically apply non-uniform refinement on different regions according to their local resolution requirements. SAMR techniques track regions in the domain that require additional resolution and dynamically overlay finer grids over these regions. This technique concentrates computational efforts (finer grids with finer integration time steps) on regions that exhibit higher numerical errors. 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 [26] as shown in Figure 2.1.

### 2.1.1 SAMR Algorithm

The primary steps involving in the SAMR algorithm are presented in Table 2.1.

Table 2.1: The Berger-Oliger AMR Algorithm

1. Recursive Procedure Integrate(*level*)
2. If (isRegridTime)
3. Regrid()
4. EndIf
5. Boundary update (including ghost region communication)
6. Evolve one time step ($\Delta t$) on all patches on level *level*
7. If (*level* + 1 exists)
8. Recursive Integrate(*level* + 1)
9. Update(*level*, *level* + 1)
10. EndIf
11. EndIntegrate

To better understand the SAMR algorithm, we show a simple example. Consider a 1-dimensional wave equation,

$$u_t + a u_x = 0 \tag{2.1}$$

with

$$u(x,0) = u_0(x) \quad \text{(initial condition)} \tag{2.2}$$

$$u(0,t) = u_L(t) \quad \text{(boundary condition)} \tag{2.3}$$

where $a$ is a constant and $a > 0$.

Using the first-order difference approximation, we obtain the numerical solution as follows,

$$u_i^{n+1} = (1 - c)u_i^n + cu_{i-1}^n \tag{2.4}$$

where $c$ is the courant number, $c = a\Delta t/\Delta x$.

To achieve higher accuracy using conventional uniform discretization numerical techniques, we have two options: one is to apply higher order numerical solution, and the other is to apply finer space and time resolution. Note that these approaches are applied uniformly on the entire space and time domain.

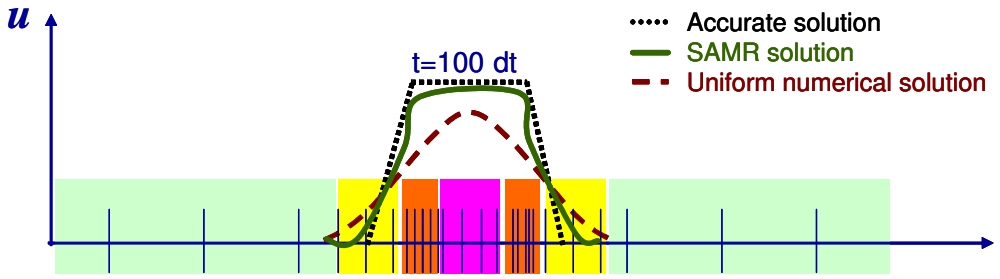


Figure 2.2: SAMR Example: 1-D Wave Equation

The better solution is to apply adaptive mesh refinement technique as illustrated in Figure 2.2. From this figure, we observe that the SAMR technique applies finer resolution merely in the small subdomains that require finer resolution, i.e., regions with high truncation errors. Compared to the static uniform discretization approach, SAMR method can achieve one to two orders of magnitude of savings in computation and storage costs for many applications with localized features.

Figure 2.3 shows a real-world application, the Richtmyer-Meshkov instability flows in a 2D slice of a 3D application (RM3D) using the SAMR technique [72]. We observe that, to accurately capture the intensive activities in those narrow regions around the shock-wave front, much finer resolution is applied, while coarser refinement is applied in the relatively homogenous and inactive regions.

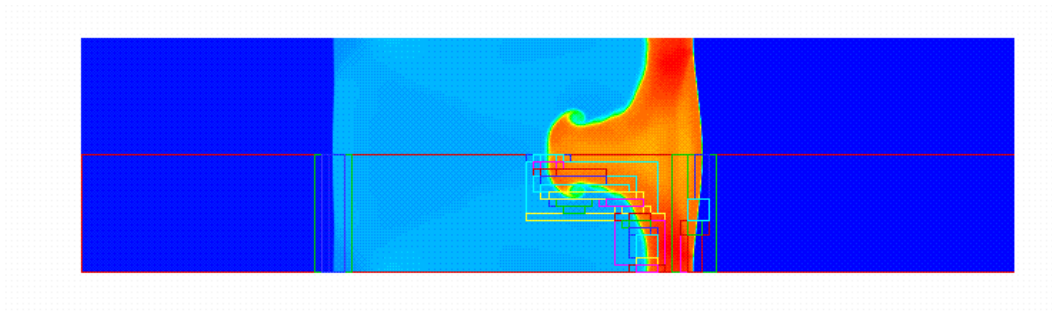

Figure 2.3: SAMR Example: Richtmyer-Meshkov Instability Flows in a 2D Slice of RM3D, Courtesy: R. Samtaney

## 2.2 Spatial and Temporal Heterogeneity of SAMR Applications

SAMR techniques can yield highly advantageous ratios for cost/accuracy when compared to methods based upon static uniform approximations [67]. Parallel implementations of these methods offer the potential for accurate solutions of physically realistic models of complex physical phenomena. However, the dynamism and spatial and temporal heterogeneity of the adaptive grid hierarchy underlying SAMR algorithms makes their efficient distributed implementation a significant challenge. To illustrate the irregular and complex behaviors of SAMR applications, two examples are presented below.

Figure 2.4 shows a series of snapshots of a 2D SAMR-based combustion simulation of hydrogen-air mixture [69]. The simulation starts with the initial ignition at 3 spots. As seen in the figure, the application exhibits high dynamics and spatial and temporal heterogeneity. Specifically, the triple flame structure requires higher resolution and thus more computational resources, while the rest of the domain does not. Further, as the simulation proceeds, the triple hot-spots expand and relocate, and cause the regions of refinement created, deleted and relocated. This behavior of the SAMR application reveals the temporal heterogeneity.

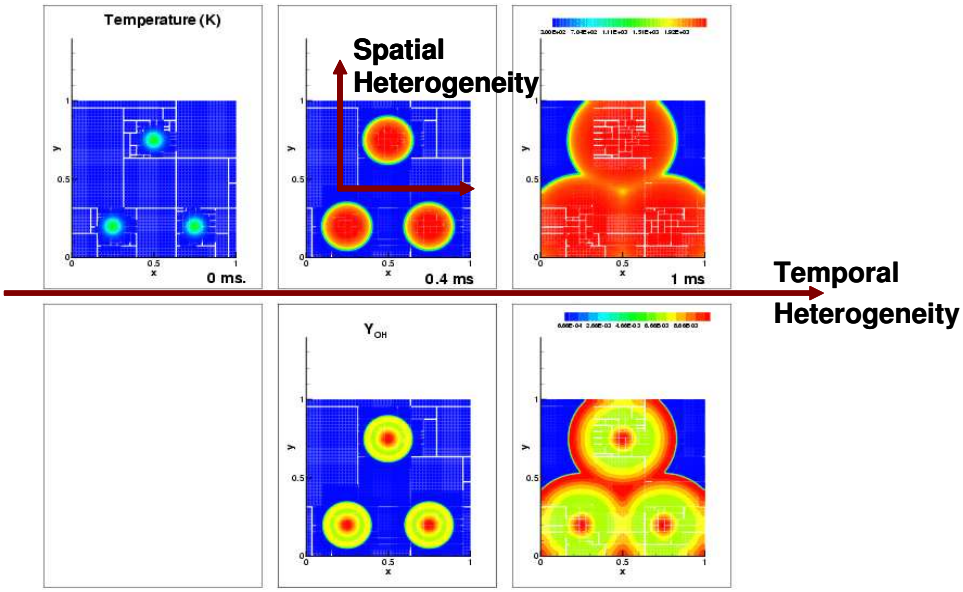


Figure 2.4: Flames simulation: ignition of $H_2$-Air mixture in a non-uniform temperature field. Courtesy: J. Ray, et al.

The 3-D compressible turbulence simulation kernel solving the Richtmyer-Meshkov (RM3D) instability [41] also demonstrates the space-time heterogeneity of SAMR applications. Figure 2.5 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, we plot workload dynamics in terms of 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 spatial and temporal 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.

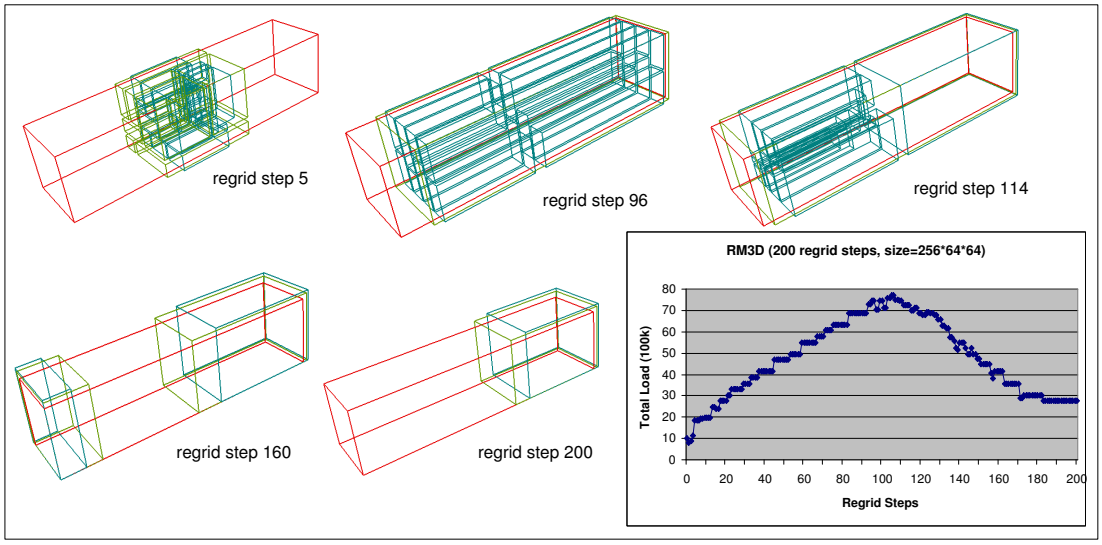


Figure 2.5: Spatial and Temporal Heterogeneity and Workload Dynamism of a 3D Richtmyer-Meshkov Simulation using SAMR

## 2.3 Computation and Communication Patterns for Parallel SAMR Applications

The overall performance of parallel SAMR implementations 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 for 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. Preserving locality minimizes the total communication and synchronization overheads.

The timing diagram (note that the timing is not drawn to scale) in Figure 2.6 illustrates the operation of the SAMR algorithm with a 3 level grid hierarchy. For simplicity, the computation and communication behaviors of only two processors, $P1$ and $P2$, are shown. The three components of SAMR communication overheads (listed in Section 2.3.1) are illustrated in the enlarged portion of the time line. This figure shows the exact computation and communication patterns for parallel SAMR implementations. The timing diagram shows that there is one time step

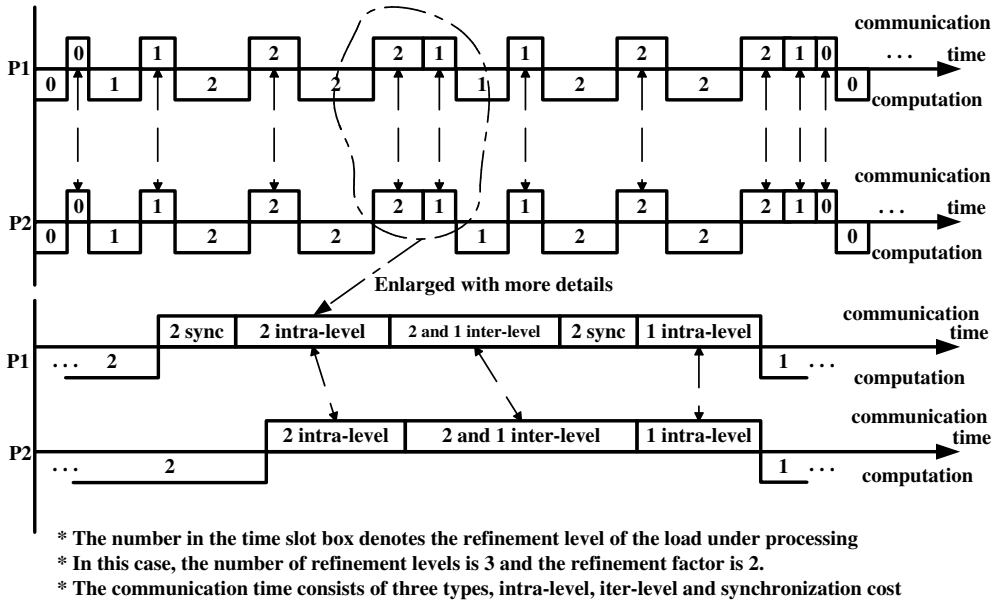


Figure 2.6: Timing Diagram for Distributed SAMR Algorithm

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.

### 2.3.1 Communication Overheads for Distributed SAMR Applications

As shown in Figure 2.6, the 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 grid, consisting of near-neighbor exchanges; (3) *Synchronization cost*,

which occurs when the workload is not well balanced among all processors; (4) *Data migration*, which occurs between two successive regridding and re-mapping steps. These costs occur at all time steps and at all refinement levels due to the hierarchical refinement of space and time in the SAMR formulation.

## 2.4 Computation and Communication Workload Analysis

The previous section illustrates the computation and communication patterns in distributed SAMR applications. This section further analyzes the computation and communication requirements. In this discussion, we use the following notations.

$CP^{(0)}$: denotes the computation cost for a region with bounding box $(lbx, lby, lbz, ubx, uby, ubz)$ on the coarsest level (level 0), where, $(lbx, lby, lbz)$ denote the coordinates of the lower bound and $(ubx, uby, ubz)$ denote the coordinates of the upper bound.

$CP^{(l)}$: denotes the computation cost for a patch at refinement level $l$.

$CM^{(0)}$: denotes the communication cost for a region with the bounding box $(lbx, lby, lbz, ubx, uby, ubz)$ on the coarsest level (level 0), where, $(lbx, lby, lbz)$ denote the coordinates of the lower bound and $(ubx, uby, ubz)$ denote the coordinates of the upper bound.

$CM^{(l)}$: denotes the communication cost for a patch at refinement level $l$.

$\alpha$: denotes the computation cost on a single grid point. It is assumed that all grid points at all levels carry the same workload.

$\beta$: denotes the communication cost for exchanging a unit size of message.

$\theta$: denotes the communication overhead such as the synchronization and startup cost.

$r$: denotes the refinement factor.

$d$: denotes the dimension of the application domain.

$\Delta x, \Delta y, \Delta z$: denote the step sizes in each of the 3 dimensions for the region on the coarsest level. The coarsest level without refinements is also referred to as the base-level.

$nx, ny, nz$: denote the number of grid points in $x, y, z$ dimensions respectively.

### 2.4.1 Computation Workload

Using notations defined in the previous section, we calculate the computation cost on the base level of grid hierarchy as follows,

$$CP^{(0)} = \alpha \times nx \times ny \times nz \tag{2.5}$$

$$= \alpha \times \left(\frac{ubx - lbx}{\Delta x} + 1\right) \times \left(\frac{uby - lby}{\Delta y} + 1\right) \times \left(\frac{ubz - lbz}{\Delta z} + 1\right) \tag{2.6}$$

The Berger-Oliger's SAMR algorithm refines in time as well as in space. We assume the refinement factor $r$ remains same for all levels of refinement and is used to refine in both time and space. Thus, the grids on the fine level $l + 1$ will be advanced $r$ time steps for every coarse time step on the level $l$. Thus, the computation cost for the same region as the base domain on the level $l$, $l = 1, 2, ..., n$, is given by,

$$CP^{(l)} = r^{d+1} \times CP^{(l-1)} \tag{2.7}$$

$$= (r^{d+1})^l \times CP^{(0)} \tag{2.8}$$

We define a composite grid as a sub-domain with multiple refinement levels starting with the base level. Thus, the computation cost for a composite grid can

be computed as, by,

$$CP_{composite} = \sum_{l=0}^{maxlev} CP^{(l)} \tag{2.9}$$

$$= \sum_{l=0}^{maxlev} (r^{d+1})^l \times CP^{(0)} \tag{2.10}$$

To obtain the relationship between patches with equivalent computation cost, we assume two cubic regions $R_i$ and $R_j$. The base level patch in the region $R_i$ is of size $nx_i \times ny_i \times nz_i$ ($nx_i = ny_i = ny_i$), and the base level patch in the region $R_j$ is of size $nx_j \times ny_j \times nz_j$ ($nx_j = ny_j = ny_j$), respectively. From the equation (2.5), the patch at refinement level $m$ in the region $R_i$ and the patch at refinement level $n$ in the region $R_j$ are of equivalent computation cost if the following equations are satisfied.

$$CP^{(i)} = CP^{(j)} \tag{2.11}$$

$$(r^{d+1})^i \times CP_i^{(0)} = (r^{d+1})^j \times CP_j^{(0)} \tag{2.12}$$

$$r^{i(d+1)} \times \alpha \times nx_i^3 = r^{j(d+1)} \times \alpha \times nx_j^3 \tag{2.13}$$

$$nx_i = nx_j \times \sqrt[3]{r^{(d+1)(j-i)}} \tag{2.14}$$

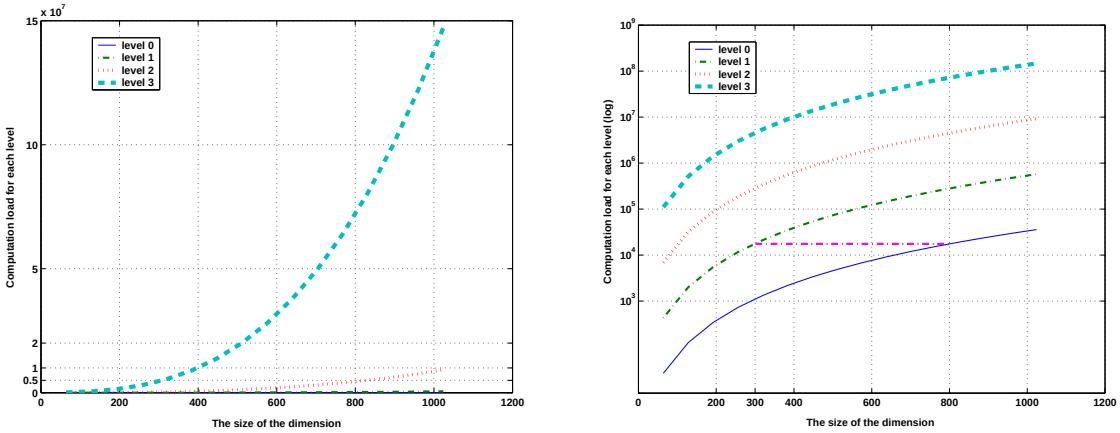


Figure 2.7: Computation Load Versus Refinement Levels

Figure 2.7 shows the workload distribution for different refinement levels in terms of the size of the domain. Note that the right figure uses the log of the

workload size. An important observation is that the highest refinement level dominates the computation cost. The right figure shows that the patch of $size = 800 \times 800 \times 800$ at the base level (level 0) requires the same computation workload as the patch of $size = 300 \times 300 \times 300$ at the refinement level 1 .

### 2.4.2 Communication Workload

The communication in SAMR applications mainly includes intra-level and inter-level communication.

Consider a 3 dimension domain assuming uniform refinement, the inter-level communication cost from the base level to the refinement level 1 of grid hierarchy is given by,

$$\begin{aligned} CM_{inter}^{(0\rightarrow 1)} &= \beta \times nx \times ny \times nz \\ &= \beta \times \left(\frac{ubx - lbx}{\Delta x} + 1\right) \times \left(\frac{uby - lby}{\Delta y} + 1\right) \times \left(\frac{ubz - lbz}{\Delta z} + 1\right) \end{aligned}$$

The inter-level communication for the same region from a level $l$ patch to the corresponding level $l + 1$ patch is given by,

$$CM_{inter}^{(l\rightarrow l+1)} = r^{d+1} \times CM_{inter}^{(l-1\rightarrow l)} \tag{2.15}$$

The intra-level communication (also called ghost communication) occurs during the exchange of the solution at the boundary based on the stencil used by the forward difference operation, including the edge, face and corner stencil. Similarly, assuming uniform refinement, the intra-level communication on the base level of the grid hierarchy is given by,

$$CM_{intra}^{(0)} = \beta \times (edge + face + corner) \tag{2.16}$$

The intra-level communication for the same region on the level $l + 1$ and the level $l$ has the following relationship,

$$CM_{intra}^{(l+1)} = r^{d+1} \times CM_{intra}^{(l)} \tag{2.17}$$

Thus the total communication cost is given by,

$$CM = CM_{inter} + CM_{intra} + \theta \tag{2.18}$$

where $CM_{inter}$ and $CM_{intra}$ include all communication costs at all refinement levels.

## 2.5 Requirement Analysis for Partitioning and Scheduling Strategies

In order to achieve the desired performance, an optimal partitioning of the SAMR grid hierarchy and scalable implementations of SAMR applications require careful consideration of the timing pattern shown in Figure 2.6. From above analysis and observations, the desired properties of an SAMR partitioner include:

1. *Balance load.* The objective of load balancing is to ensure that the same amount of load is assigned to each processor in homogeneous systems, or the amount of load that is assigned to a processor is proportional to the processor's capacity in heterogeneous systems.

2. *Preserve locality.* Preserving locality requires that communication are localized and possibly on the same processor when the workload is distributed. In the case of SAMR applications, this implies (a) maintaining the parent-child locality to minimize the inter-level communication overhead; (b) assigning the proximate application subregions to closely-networked resources or resource groups to minimize the intra-level and inter-level communication overheads; (c) maintaining locality between two repartition steps to minimize the data migration overheads.

3. *Be fast and efficient.* Parallel SAMR requires regular regridding and repartitioning. A good partitioner should be fast and efficient.

Note that these three properties may contradict each other. For example, to achieve perfect load balancing, fine granularity is preferred. However, fine granularity may affect the locality property and incur more partitioning and communication overheads. A good strategy strives to achieve the best overall trade-off.

## 2.6 Taxonomy for Runtime Management of SAMR Applications

In this section, we present a taxonomy for dynamic application management, particularly for partitioning SAMR applications. The taxonomy is based on the characteristics of applications and application-level partitioners as shown in tables 2.2 and 2.3. We decouple the classification of applications and their partitioners. In this way, we can obtain a better understanding of both aspects. The decoupling also implicitly indicates that we can potentially apply different partitioners for different application characteristics in a flexible manner.

Table 2.2: Classification of Application Characteristics

| **Application Characteristics** | **Categories** |
|---|---|
| Execution Mode | Computation-Intensive, Communication-Intensive, IO-Intensive |
| Activity | Dynamic (Localized Adaptivity, Scattered Adaptivity), Static |
| Granularity | Fine-Grained, Coarse-Grained, Indivisible |
| Dependency | Independent, Workflow, Hybrid |

In Table 2.2, applications are characterized based on their execution mode, activity, granularity and dependency. The execution mode can be computation-intensive, communication-intensive or IO-intensive [39, 80]. Most SAMR applications are computation-intensive, belonging to high performance scientific computing category. Due to deep levels of adaptive refinements, SAMR applications can also be communication-intensive. In some cases, when dealing with large amounts of data, SAMR applications can fall into IO-intensive category. Experiments show

that, during the entire course of execution, SAMR applications may run in different execution modes as the simulated physical phenomena evolve [39, 80]. The activities of applications are classified as dynamic or static. Many embarrassingly parallel applications belong to the static application category, for example, parallel geometrical transformations of images and Monte Carlo simulations [84]. SAMR applications are dynamic in nature because of their high adaptivity. Dynamic behaviors of SAMR applications may demonstrate localized adaptivity pattern or scattered adaptivity pattern in different execution phases. From the perspective of divisibility, some applications are fine-grained [83], some are coarse-grained [20], while others are not divisible at all [31, 75]. Workloads in the divisible load scheduling paradigm are assumed to be homogeneous and arbitrarily divisible in the sense that, each portion of the load can be independently processed on any processor on the network. Coarse-grained divisible applications typically involve dependencies among subtasks. Indivisible tasks are atomic and cannot be further divided into smaller sub-tasks, and have to be completely processed on a single processor. SAMR applications fall into the fine-grained or coarse-grained divisible categories. When the underlying physical domain exhibits more homogeneity, the load associated with this domain belongs to the fine-grained divisible category. When the physical domain exhibits scattered heterogeneity with deep refinements, however, the load may be classified as coarse-grained divisible. Note that SAMR applications involve iterative operations and frequent communications. Therefore, they do not belong to the embarrassingly parallel category. The last criterion is the dependency. Independent applications are common in the divisible load scheduling category, such as parallel low level image processing and distributed database query processing [83]. Workflow applications are composed of several modules or subtasks that must run in order, for example, data-driven parallel applications, scientific simulations, and visualization applications. For SAMR applications, although load partitions can be processed independently,

they need communications iteratively. If dynamic load balancing strategies are adopted, repartitioning may result in load movement or process migration, thus exhibiting a hybrid dependency.

Table 2.3: Classification of Application Partitioners

| **Partitioner Characteristics** | **Categories** | |
|---|---|---|
| Organization | static single-partitioner, adaptive single-partitioner (meta-partitioner), adaptive hierarchical multiple-partitioner | |
| Decomposition | Data Decomposition (Domain-based, Patch-based, Hybrid), Functional Decomposition | |
| Partitioning Method | Binary dissection, Multilevel, SFC-based, etc. | |
| Operations | Dynamic | Repartitioning policy (Periodic, Event Driven), System Sensitive |
| | Static | |

As shown in Table 2.3, application-level partitioners/schedulers are classified with respect to their organization, decomposition method and operations. The organization of partitioners falls into three categories: static single partitioner, adaptive single partitioner, adaptive multiple partitioner. For static single partitioner approaches, one predefined partitioning and repartitioning strategy is adopted throughout the entire lifecycle of SAMR applications. For adaptive single partitioner approaches, also termed meta-partitioner in the literature [80], the most appropriate partitioning routine is selected based on the runtime behavior of SAMR applications. Adaptive multiple partitioner approaches not only select appropriate partitioning strategies based on the runtime state of the SAMR application but also apply multiple partitioners simultaneously to local relatively homogeneous regions of the domain based on the local requirements. In this thesis, we propose such an adaptive hierarchical multiple partitioning strategy. Decomposition methods can be classified as data decomposition and functional decomposition. Functional decomposition exploits functional parallelism by dividing problems into a series of subtasks. Pipelining techniques can be employed

to speedup applications with functional parallelism.

Data decomposition is commonly applied to achieve data parallelism for applications that require the same operations to be performed on different data elements, for example, SPMD (Single Program Multiple Data) applications. In the case of SAMR applications, data decomposition methods can be further classified as patch-based and domain-based.

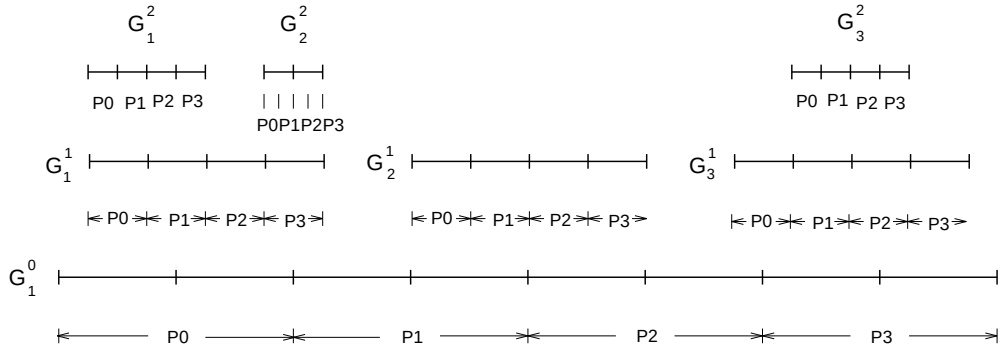


Figure 2.8: Patch-based Decomposition of 1D Application, Courtesy: M. Parashar

Patch-based decomposition methods make partitioning decisions for each patch at different refinement levels independently [36, 57, 67, 77]. An example of patch-based decomposition for a 1-dimensional domain is illustrated in Figure 2.8. Patch-based technique results in well-balanced load distribution as long as each patch is sufficiently large. Since workload is balanced for each patch, an implicit feature of this scheme is that it does not need redistribution when a patch is deleted at runtime. However, it does not preserve locality and result in considerable inter-level communication overheads. Moreover, these communications may lead to serializing of the computation and severe performance bottleneck. Inter-level communications occur during restriction and prolongation data transfer operations between parent children (coarser-finer) patches. For instance, in Figure 2.8, a restriction operation from the patch $G_1^1$ to the patch $G_1^0$ requires all other processors to communicate with the processor $P_0$.

In contrast, domain-based approaches partition the physical domain rather than the individual patches at each refinement level [36, 67, 77]. A subdomain

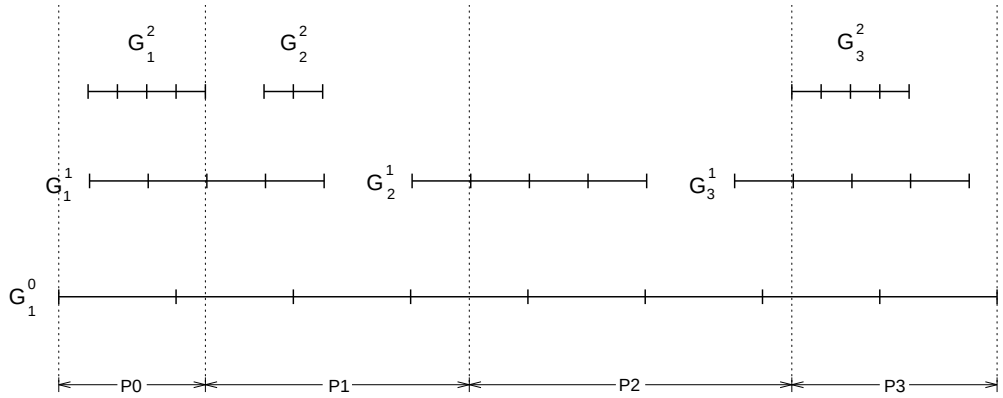


Figure 2.9: Domain-based Decomposition of 1D Application, Courtesy: M. Parashar

includes all patches on all refinement levels in that region. An example for a 1-dimensional application is illustrated in Figure 2.9. From this figure, we observe that domain-based schemes maintain the parent-child locality while striving to balance overall workload distribution. As a result, these techniques substantially eliminate inter-level communication overheads and the associated communication bottleneck. However, due to their strict domain-based partitioning, they may incur considerable load imbalance when the application has deep refinements on very narrow regions with strongly localized features. In these cases, hybrid schemes that combine patch-based and domain-based techniques are required.

Partitioning methods for SAMR applications include dissection-based methods, multilevel methods, and space filling curve (SFC) based methods [71]. These methods are described in detail in the next section. Further, partitioning can be static or dynamic. Static schemes use one or possibly multiple predefined partitioning methods and remain same partitions of the domain for the entire execution process. Dynamic schemes apply partitioning and repartitioning methods at runtime and dynamically change the partitions of the domain to match the requirements of the application. Specifically, dynamic schemes can be further classified as periodic , event-driven or system-sensitive according to their repartitioning policies. Since SAMR applications typically execute in an iterative

manner, the natural repartitioning policies are periodic. For example, repartitioning is invoked every $n$ iterations. Event-driven repartitioning policies is more flexible in the sense that repartitioning is triggered only when it is needed. Furthermore, system-sensitive dynamic schemes adapt to system availability, capacity and changing patterns.

### 2.6.1 Related Work on Managing Distributed SAMR Applications

Efficiently managing distributed SAMR applications and minimizing their overall execution time requires load balance and minimumal communication overheads. To achieve these two objectives, most partitioners take advantages of the geometry of the application, its locality and load dynamics, to partition and balance the workload among processors. The underlying partitioning algorithms include binary dissection algorithms [32], multilevel algorithms [52, 74] and space-filling curve (SFC) based algorithms [67, 68]. The binary dissection algorithm proceeds by recursively partitioning the physical domain into two parts such that it equalizes the computational load in each part, naturally generating a binary tree structure. Its variants, such as parametric binary dissection, extend the basic binary dissection method to minimize communication cost between the two partitions. Multilevel algorithms usually consist of the following steps: coarsening the graph that represents the domain, partitioning the coarse graph, and refining the partitions step by step. Their primary objectives are to minimize the cut edge weight, which approximates the total communication cost in the underlying solver. Due to the complexity of multilevel algorithms, however, they are more suitable for static partitioning schemes.

SFC has been widely used in domain-based partitioning schemes [67, 68,

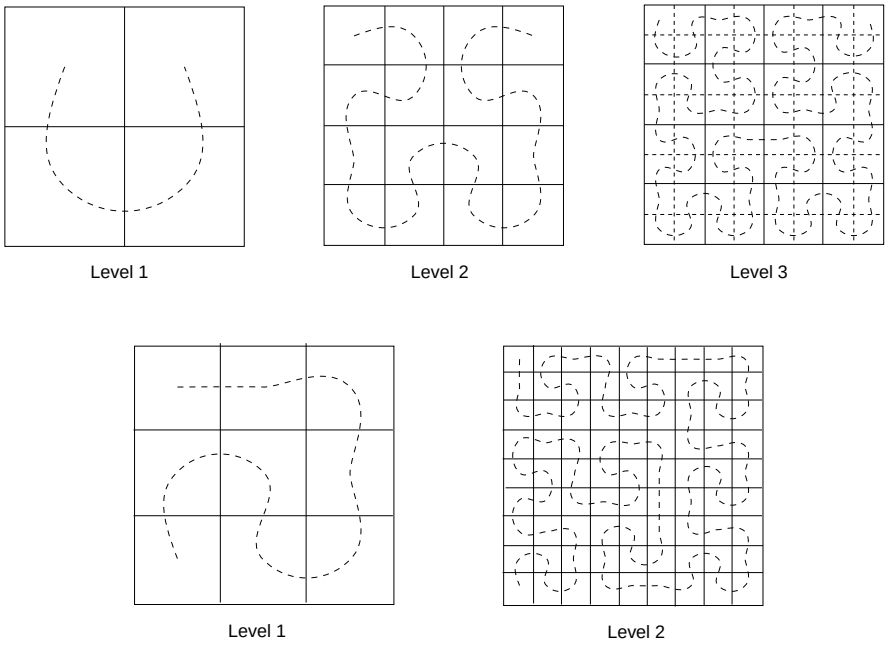


Figure 2.10: Space Filling Curves - Self-Similarity Property. Courtesy: M. Parashar

71, 77]. SFCs are locality-preserving recursive self-similar mappings from $n$-dimensional space to 1-dimensional space. Specifically, locality preserving implies that points that are close together in the 1-dimensional space (the curve) are mapped from points that are close together in the $n$-dimensional space. Self-similarity implies that, as a $d$-dimensional region is refined, the refined subregions can be recursively filled by curves having the same structure as the curve filling the original (unrefined) region, but possibly a different orientation. Figure 2.10 illustrates this property for a 2-dimensional region with refinements by factors of 2 and 3. In the case of SAMR grid hierarchies, the locality-preserving and self-similar properties of SFC mappings are exploited to represent the hierarchical structure and to maintain locality across different hierarchy levels. Furthermore, these techniques are computationally efficient.

There exist a number of infrastructures that support parallel and distributed implementations of structured and unstructured AMR applications. Each of these systems represents a combination of design choices. Table 2.4 summarizes these

Table 2.4: Parallel and Distributed AMR Infrastructures

| System | Execution Mode | Granularity | Partitioner Organization | Decomposition | Institute |
|---|---|---|---|---|---|
| **CHARM** | Compu - intensive | Coarse - grained | Static single - partitioner | Domain -based | UIUC |
| **Chombo** | Comp - intensive | Fine -grained, coarse - grained | Static single - parti tioner | Domain -based | LBNL |
| **HRMS/ GrACE** | Comp - intensive | Fine -grained, coarse - grained | Adaptive multi - parti tioner, Hybrid strategies | Domain -based, hybrid | Rutgers |
| **Nature+ Fable** | Comp - intensive | Coarse - grained | Single meta - parti tioner | Domain -based, hybrid | Sandia |
| **ParaMesh** | Comp - intensive | Fine -grained, coarse - grained | Static single - part itioner | Domain -based | NASA |
| **ParMetis** | Comp - intensive, comm - intensive | Fine -grained | Static single - part itioner | Graph -based | Minnesota |
| **PART** | Comp - intensive | Coarse - grained | Static single - partitioner | Domain -based | Northwestern |
| **SAMRAI** | Comp - intensive, comm - intensive | Fine -grained, coarse - grained | Static single - parti tioner | Patch -based | LLNL |

systems and their features according to our taxonomy.

Charm++ [55] is a comprehensive parallel C++ library that supports processor virtualization and provides an intelligent runtime system. Its AMR module offers flexible representation and communication for implementing the AMR algorithm. It uses the domain-based decomposition method and a quad-tree structure to partition and index the computational domain and supports coarse-grained partitioning. Chombo [3] consists of four core modules: BoxTools, AMRTools, AMRTimeDependent, AMRElliptic. Its load balance strategy follows Kernighan-Lin multilevel partitioning algorithm. GrACE [8] is an object-oriented adaptive computational engine with pluggable domain-based partitioners. HRMS built on GrACE consists of an adaptive hierarchical multi-partitioner scheme and other hybrid schemes proposed in this thesis. Nature+Fable formulates a meta-partitioner approach by characterizing both partitioners and application domains [77]. ParaMesh [65] uses octree representation of the adaptive grid structure

with predefined blocksizes and uses this representation to partition the SAMR domain. ParMetis [56] applies multilevel hypergraph partitioning and repartitioning techniques to balance the load. PART [40] considers heterogeneities in the application and the distributed system. It uses simulated annealing to perform the backtracking search for desired partitions. Nevertheless, simplistic partitioning schemes are used in the PART system. SAMRAI [57] is an object oriented framework (based on LPARX) It uses a patch-based decomposition scheme.

## 2.7 Concluding Remarks

This chapter presented the background and related work for parallel SAMR implementations and gave an overview of the proposed strategies. First, the dynamism and space-time heterogeneity of parallel SAMR applications was described and the computation/communication patterns were identified and analyzed. To have a common context for comparison and evaluation, the taxonomy for runtime management of SAMR applications was then presented. Related work on managing SAMR applications was surveyed and discussed using our taxonomy.

# Chapter 3

# Hybrid Space-Time Runtime Management Strategy

The overarching goal of this thesis is to design adaptive runtime management strategies for complex SAMR-based scientific applications to minimize their overall execution time on large-scale parallel and distributed systems (PDS). The key challenge is how to efficiently manage the dynamism and space-time heterogeneity exhibited in SAMR applications. Confronted with these multi-dimensional challenges, we propose a solution framework, hybrid space-time runtime management strategy and system (HRMS), which adapts to the requirements of applications at runtime. It also takes advantage of a number of schemes by combining and integrating their strengths for varied scenarios at runtime.

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 [43, 55, 54, 56, 65, 67], these implementations are typically based on application-specific customizations and the general scalable implementation of SAMR applications continues to present significant challenges. This is primarily due to the dynamism and space-time heterogeneity exhibited by these applications as described in previous sections. 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.

To address the dynamism and space-time heterogeneity in SAMR applications, this chapter presents an overview of the proposed solution - hybrid space-time runtime management strategy and system (HRMS). In addition, a set of SAMR application kernels used in this thesis is summarized.

## 3.1 Conceptual Overview of HRMS

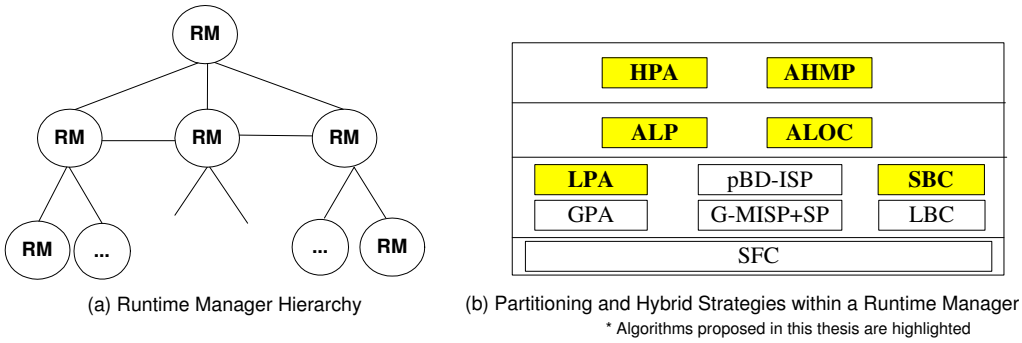


Figure 3.1: Conceptual Architecture of HRMS

To address the dynamism and space-time heterogeneity in SAMR applications, this thesis proposes the hybrid space-time runtime management system (HRMS) framework. Its conceptual architecture is presented in Figure 3.1. In the figure, runtime managers are organized in a hierarchical manner. Runtime managers reside in each resource group and communicate with each other using messages in a hierarchical manner. Further, they make management decisions concurrently based on the runtime states of the resource and the application.

Each runtime manager is equipped with a set of partitioning, clustering, scheduling and hybrid strategies organized in three layers as shown in Figure 3.1 (b). These strategies are all based on the space-filling curve technique (SFC) [71].

A set of domain-based partitioning and clustering schemes have been proposed, namely, LPA (level-based partitioning algorithm) [60], SBC (segmentation-based clustering algorithm), GPA (greedy partitioning algorithm, LBC (level-based clustering algorithm), G-MISP+SP (geometric multilevel + sequence partitioning), and pBD-ISP ($p$-way binary dissection algorithm) [39, 38, 67, 77] . Note that the algorithms proposed in this thesis are highlighted. These strategies decompose the application domain hierarchy using SFC. Based on these schemes, we have developed ALP (application-level pipelining) and ALOC (application-level out-of-core) algorithms. In the top layer, we have HPA (hierarchical partitioning algorithm) [61] and AHMP (adaptive hierarchical multiple partitioner) schemes. Strategies proposed in this thesis are summarized as follows.

- Hierarchical Partitioning Algorithm (HPA): HPA seeks to match the adaptive grid hierarchy to the hierarchically organized processor groups. Specifically, HPA divides resources into hierarchical processor groups and then partitions the domain accordingly in a hierarchical manner. Note that HPA uses one partitioning scheme over the entire domain for initial partitioning and for later repartitionings.

- Level-based Partitioning Algorithm (LPA): LPA is a simple but elegant partitioning method. It significantly reduces synchronization cost by balanacing the overall load as well as the load at every refinement level.

- Adaptive Hierarchical Multi-Partitioner Scheme (AHMP): AHMP extends HPA. It first hierarchically clusters the domain into several coarse subregions with similar properties, called cliques. It then recursively applies the most appropriate partitioning methods to each clique recursively. In this way, it enables multiple partitioners to concurrently operate on different subregions.

- Segmentation-based Clustering Algorithm (SBC): The SBC scheme generates the clique hierarchy to support the AHMP scheme by clustering together subregions exhibiting similar properties. SBC extends the level-based clustering scheme (LBC) that clusters the subregions simply by refinement levels.

- Application-level Pipelining Algorithm (ALP): When the resource is underutilized, the domain-based partitioning schemes can no longer speed up execution. ALP comes to rescue in this situation by integrating patch-based and domain-based schemes and applying them within a local resource group.

- Application-level Out-of-Core Algorithm (ALOC): When the available resources are insufficient, the ALOC scheme leverages the out-of-core scheme to enhance the survivability of SAMR applications.

## 3.2 Operations of HRMS

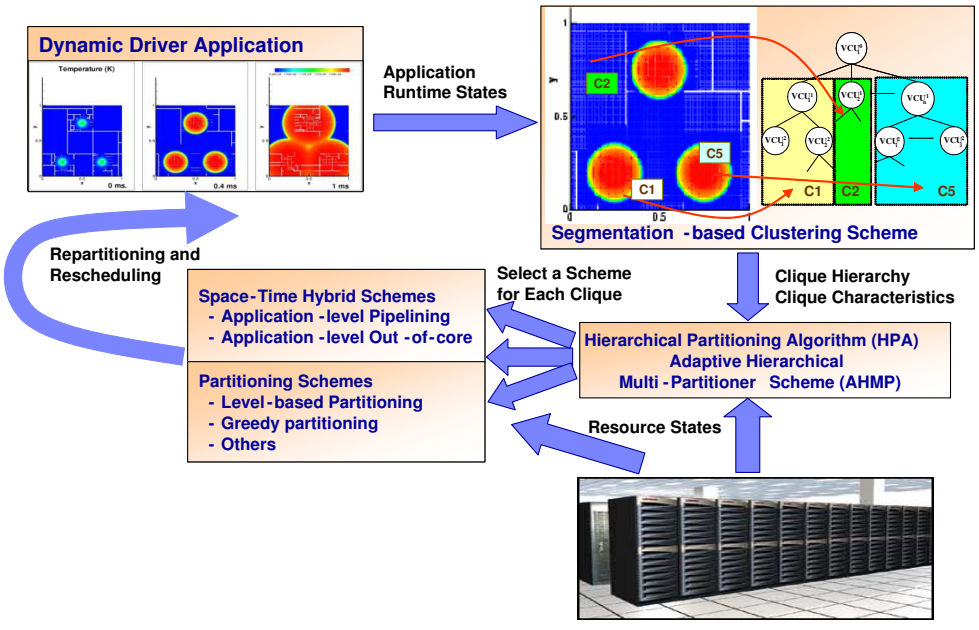


Figure 3.2: Workflow of HRMS

The basic operation of HRMS is illustrated in Figure 3.2. At runtime, HRMS monitors the application status and represents it as the adaptive grid hierarchy as illustrated in Figure 2.1. Clustering algorithms are then used to identify clique hierarchies from the structure of the current grid hierarchy. A clique region is a cluster of subregions that have relatively homogeneous requirements. We have two options to work on the clique hierarchy: HPA and AHMP. The HPA approach selects a single partitioner for the entire domain for the whole execution period. Alternatively, based on the characteristic of a clique and the current resource states, AHMP selects an appropriate partitioner for each clique from the partitioner repository. Essentially, AHMP extends HPA by enabling multiple partitioners concurrently on different subregions of the application domain. Finally, the clique is partitioned and mapped to processors in a processor group. These steps are recursively applied to each clique. When the application states change significantly, the repartitioning process is then triggered among local processor group hierarchically and incrementally. HRMS has two clustering algorithms: level-based clustering (LBC) and segmentation-based clustering (SBC) schemes. The available partitioners in the GrACE, an object-oriented infrastructure for enabling parallel SAMR applications [8], include GPA, LPA, BPA, and others [79]. The selection policies are defined according to the characteristics of each clique, including refinement homogeneity, communication/computation requirements, scattered adaptation, activity dynamics [79]. This thesis specifically focuses on developing policies based on refinement homogeneity, which will be defined in the next chapter.

Overall, the operations of HRMS follow two main workflows: the hierarchical partitioning scheme (HPA) and the adaptive hierarchical multi-partitioner scheme (AHMP), which are presented in the next two chapters.

## 3.3 SAMR Application Kernels for Experimental Evaluation

Table 3.1 summarizes the SAMR application kernels used for experimental evaluation in this thesis. 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), numerical relativity (Wave2D and Wave3D), and the transport equation (TP2D). We characterize the partitioning requirements of these applications in terms of load balancing, communication, data migration, and partitioning overheads [36, 79]. In this thesis, we use RM3D as the representative application for most experiments because it is highly dynamic and its partitioning requirements are challenging.

## 3.4 Concluding Remarks

The dynamism and space-time heterogeneity exhibited in SAMR applications make it challenging to efficiently partition and manage these applications on large systems. This chapter presented a hybrid space-time runtime management strategy and system (HRMS) that seeks to explicitly address these issues. The conceptual overview and operations of HRMS have been presented. In addition, a set of SAMR application kernels used in this thesis for the experimental evaluation were summarized.

Table 3.1: SAMR Application Kernels

| Apps | Dim | Description | Characteristics |
|---|---|---|---|
| TP | 2D | A benchmark kernel for solving transport equation, included in the GrACE toolkit [8]. | Intense activity in very narrowly concentrated regions. Key partition requirement: minimize partitioning overheads. |
| RM | 2D/3D | A compressible turbulence application solving the Richtmyer-Meshkov instability. It 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 [41]. | Intense activity in relatively larger regions, which are scattered. Key partition requirement: minimize communication and balance load on 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 [41]. | Intense activity in relatively larger regions. Key partition 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 [13]. | Intense activity in very narrow regions sparsely, which are highly scattered. Key partition requirement: minimize communication and data migration |
| Wave | 2D/3D | A numerical relativity kernel for solving Enstein's and gravitational equations used in Cactus toolkit [2, 20]. | Intense activity in narrow regions. Key partition requirement: minimize communication. |

# Chapter 4
# Hierarchical Partitioning Algorithms

Traditional distributed implementation of SAMR applications [3, 57, 65, 67] have used dynamic partitioning/load-balancing algorithms that view the system as a flat pool of 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 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 communication and synchronization in this structure.

The overall goal of the hierarchical partitioning algorithms (HPA) presented in this chapter 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 within 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 chapter: static hierarchical partitioning algorithm (SHPA) and adaptive hierarchical partitioning algorithm (AHPA). The SHPA scheme 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 AHPA scheme 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. To further reduce the synchronization cost, a level-based partitioning algorithm (LPA) has been developed by explicitly balancing the workload at each refinement level. Experimental evaluation of HPA, LPA and the combined schemes shows performance improvement over the existing solutions.

The HPA and LPA schemes are based on the composite grid distribution strategy (CGDS) and the greedy partitioning algorithm (GPA) [37, 67]. CGDS aims at partitioning the grid hierarchy such that all inter-level communication is local to a processor. Using the SFC technique, which is described in the last chapter, the $n$-dimensional computational domain is mapped to a 1-dimensional list of blocks and the partitioning task is thus reduced to partition this 1-dimensional representation. Based on the SFC technique, CGDS partitions the computational domain and results in a global grid unit list (GUL). Each grid unit represents a composite subdomain that includes all refinement levels, also termed the composite grid. Further, based on the concept of CGDS, the GPA scheme scans the global GUL only once to attempt to equally distribute workload among all processors. The key motivation for using the GPA scheme is that it is fast and efficient. This is important as the number of composite grid units can be large

and regrid steps can be quite frequent. Essentially, HPA and LPA schemes pre-process the global GUL and apply GPA on the re-organized global GUL. Hence, they take advantages of the composite grid decomposition technique to reduce intra-level communications and localize inter-level communication. Note that the GPA scheme is also referred to as the Non-HPA scheme when compared to HPA schemes.

## 4.1 Hierarchical Partitioning Algorithm

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

### 4.1.1 General HPA

In most parallel implementations of SAMR [8, 19, 54, 65], load distribution 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, referred to as Non-HPA schemes, have the advantage of achieving a better load balance. However, they require the collection and maintenance of global load information, which makes them expensive, specially on large systems. Partitioning in such Non-HPA schemes 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.

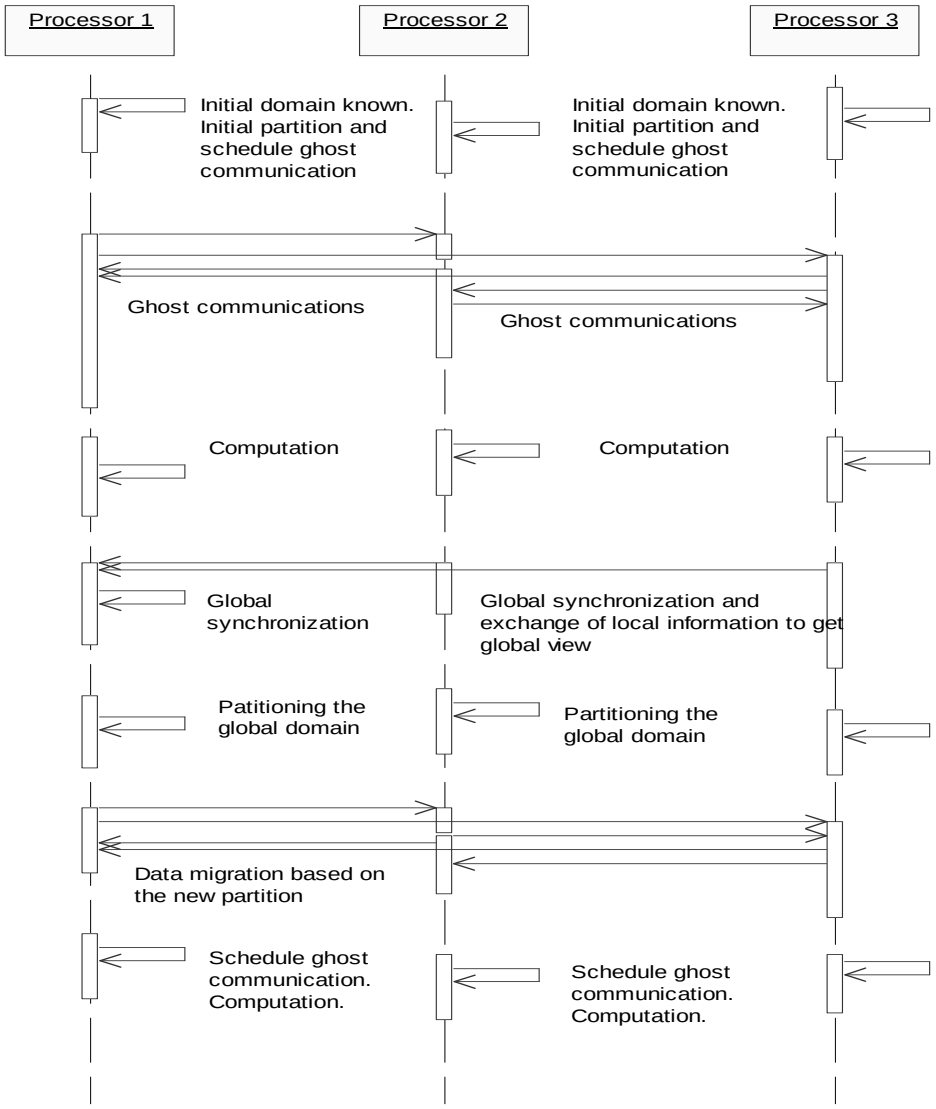


Figure 4.1: Sequence Diagram for the Non-HPA Scheme

The sequence diagram in Figure 4.1 shows the operations of the Non-HPA scheme for partitioning the domain and scheduling ghost communications is illustrated in the sequence diagram in Figure 4.1. At the startup, all processors have the initial computational domain. Each processor partitions the domain into subregions and assigns a subregions 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 belong to their local subregions. Each processor then schedules intra/inter level communications based on its new local subregions.

In large parallel/distributed systems, the global information exchange and synchronization phase becomes a performance bottleneck. The HPA scheme does not propose a new partitioner, but a hierarchical partitioning strategy. The underlying partitioning schemes can be GPA, BPA, or LPA [37, 60, 67].

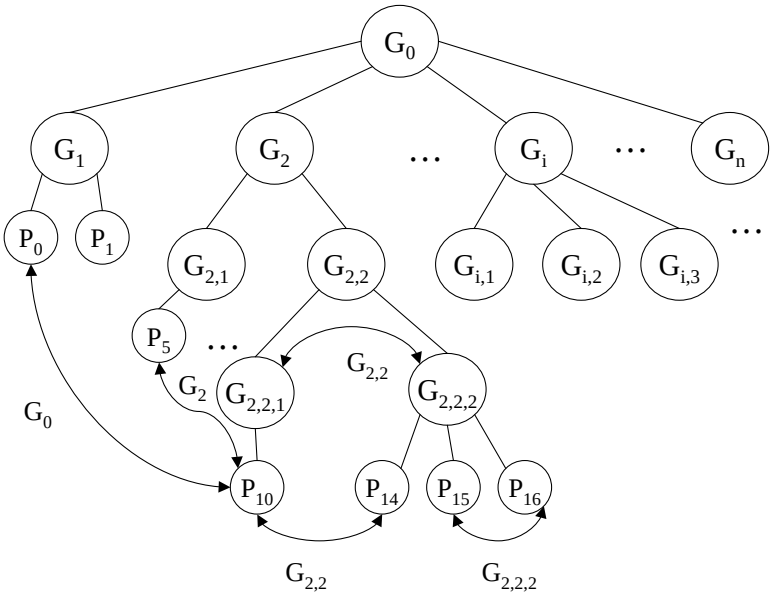


Figure 4.2: A General Hierarchical Structure of Processor Groups

Figure 4.2 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 individual 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 $G_0$. Furthermore, Partitioning within different processor groups is performed in parallel based on load information local to the processor groups. Workload is periodically propagated up the processor group hierarchy in an incremental manner. As a result, these localized operations in parallel reduce the global communication and synchronization overheads.

In HPA, the partitioning phase is divided into two 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 a portion of the workload is assigned 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 4.1.

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 4.3.

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

Table 4.1: 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==1) it is a global partition among groups at level 1.
4. End of the loop
5. Begin computation ...

processors in the group subtrees. Recursively, portions of the computational domain are partitioned further and finally assigned to individual processors at the leaves of the processor group hierarchy. This is the local partitioning phase.

### 4.1.2 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 [75], 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 4.2. 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 4.1.

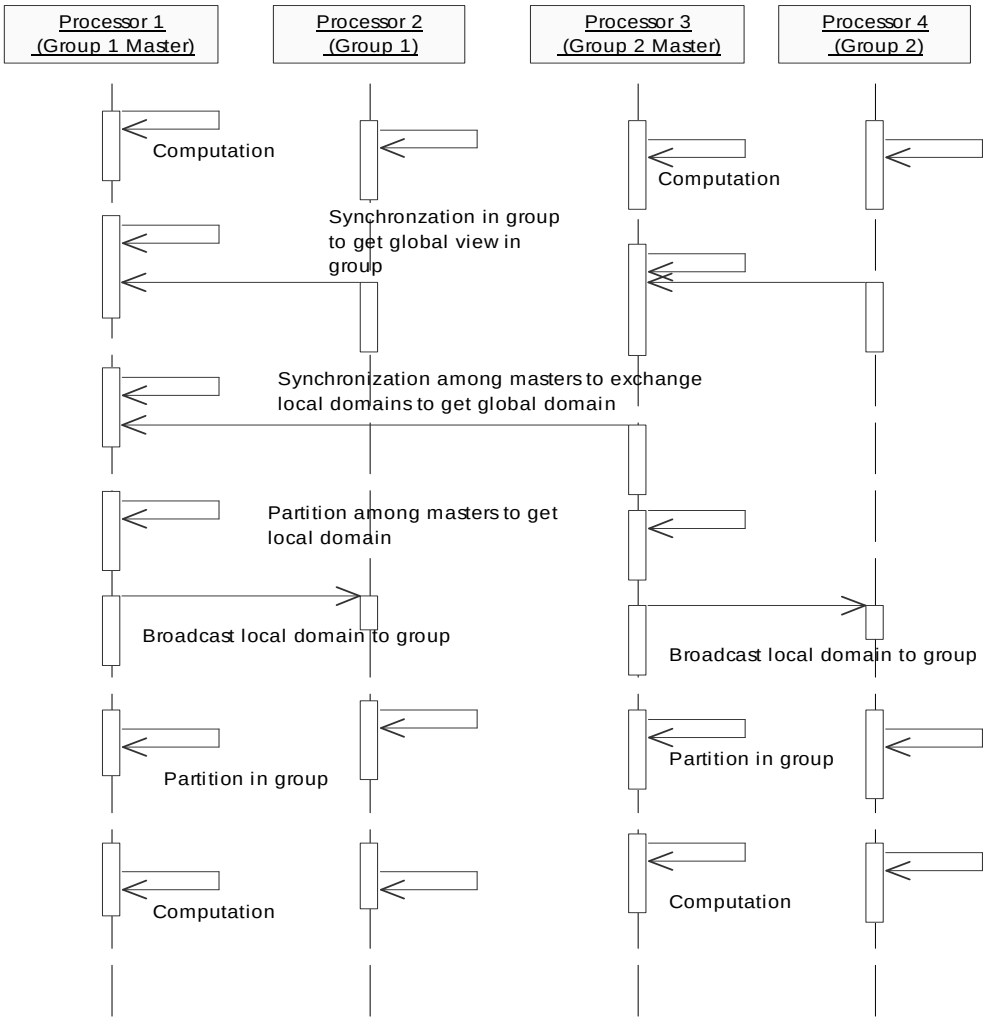


Figure 4.3: Sequence Diagram for the HPA Scheme

The Static HPA is implemented as part of the GrACE toolkit [8]. The groups are created using communicators provided by the MPI library. Communication within groups is through intracommunicators while communication between processors belonging to different groups is through intercommunicators.

The Static HPA scheme is evaluated on BlueHorizon, the IBM SP3 cluster at San Diego Supercomputer Center, which consists of 1152 processors. Each SP3 node has 8 processors running at 375 MHz and 4GB RAM, and CPU peak performance is 1.5 GFlops. Overall, BlueHorizon delivers a peak performance of 1.7 TeraFlops. The application used in these experiments is the RM3D as described

Table 4.2: 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 $i = 0, 1, ..., NP_j - 1$, where $NP_j$ is the number of processors in the lowest group level.

in Chapter 3. RM3D is a representative application that exhibits substantial spatial and temporal heterogeneity. 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 4.4, we observe that the SHPA scheme improves the overall execution time. The maximum performance gain is obtained for 192 processors using SHPA Inc scheme - about 59% reduction in the overal 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 balance than the SHPA Inc scheme since it redistributes the load more frequently. However, for larger number of processors, due to significant reduction of the synchronization

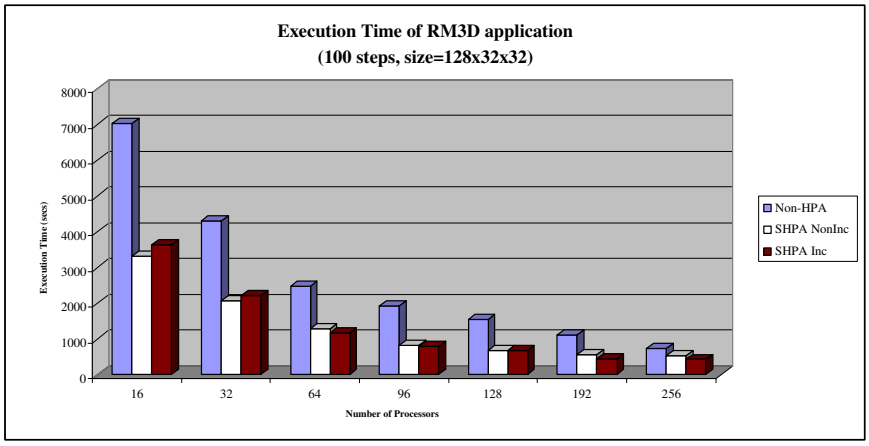


Figure 4.4: Execution time: Static HPA versus Non-HPA Schemes

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.

### 4.1.3 Adaptive HPA

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 4.3. 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.

Table 4.3: 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 $i = 0, 1, ..., NP_{totalgroups} - 1$.

As shown in Table 4.3, 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 applied in this group. The load balancing phase in AHPA is similar to the steps in Table 4.1 with dynamic group sizes and a dynamic number of group levels.

The communication cost of AHPA scheme is evaluated using trace-driven simulations. The simulations are conducted as follows. First, we obtain the refinement trace for an SAMR application by running the application for a single processor. 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 simulator[1] to obtain the runtime performance measurements on multiple

[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`)

processors. The simulation results for the 2D Transport Equation and Wave3D applications are shown in Figure 4.5.

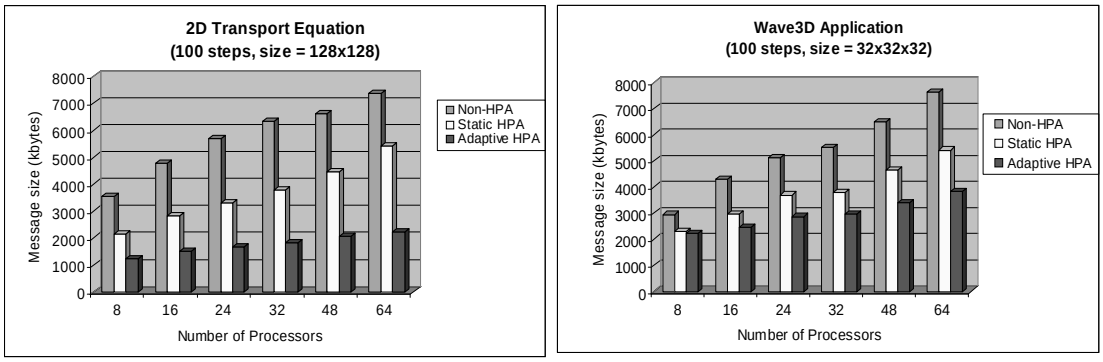


Figure 4.5: Communication Cost: Comparison of Non-HPA, Static HPA and Adaptive HPA Schemes

In Figure 4.5, 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. Compared to the SHPA scheme, AHPA scheme further reduces communication costs. As shown in the figure, the reduction in communication cost is up to 70%, for the AHPA scheme over the Non-HPA scheme. These simulation results validate that the Adaptive HPA scheme is potentially an efficient solution to gain better system performance.

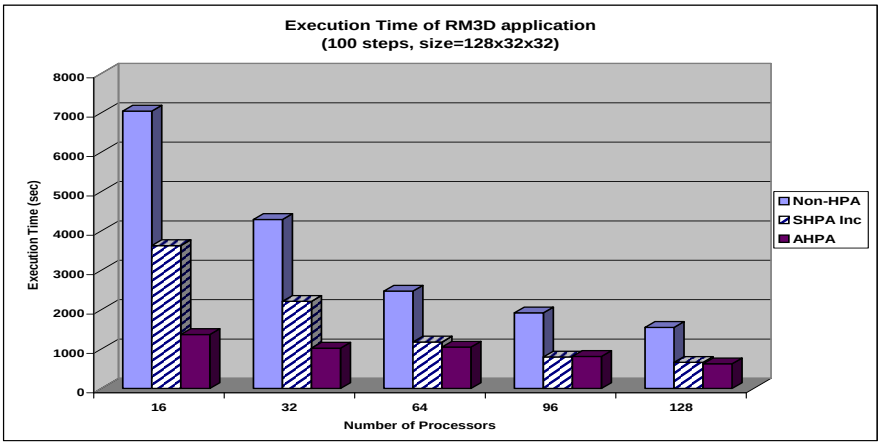


Figure 4.6: Execution Time: Static HPA versus Adaptive HPA Schemes

The experimental evaluation of AHPA is further conducted on BlueHorizon

using the RM3D application with the same configuration as in the previous section. Figure 4.6 shows the execution time for Non-HPA, SHPA Inc, and AHPA schemes. From the figure, we observe that the AHPA scheme further improves the overall performance compared to the SHPA scheme. An important observation is that AHPA marginally outperforms SHPA in larger systems. This motivates us to seek new strategies, which results in the adaptive hierarchical multi-partitioner strategy to be presented in the next chapter.

## 4.2 Level-based Partitioning Algorithm

As described in the last chapter, the computation and communication pattern of parallel SAMR applications requires partitioning schemes to balance load and preserve locality. One critical observation from the timing diagram in Figure 2.6 is that, in addition to balancing the total load assigned to each processor and maintaining parent child locality, we also need to balance the load on each refinement level and address the communication and synchronization costs within a level. The LPA scheme works towards this goal. GPA 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 further improve the performance, essentially, LPA preprocesses a portion of the global GUL by disassembling it according to refinement levels, and feeds the resulting homogeneous 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.

The procedure of LPA scheme is presented in Table 4.4. We observe that the LPA scheme partitions deep composite grid units before shallow grid units. Since we cannot guarantee perfect load balance during the partitioning at each iteration, to compensate the possible imbalance introduced in higher level and

Table 4.4: Level-based Partitioning Algorithm (LPA)

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.

equally partition the GUL on the lower level, we need to keep track of the load on lower levels that is 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.

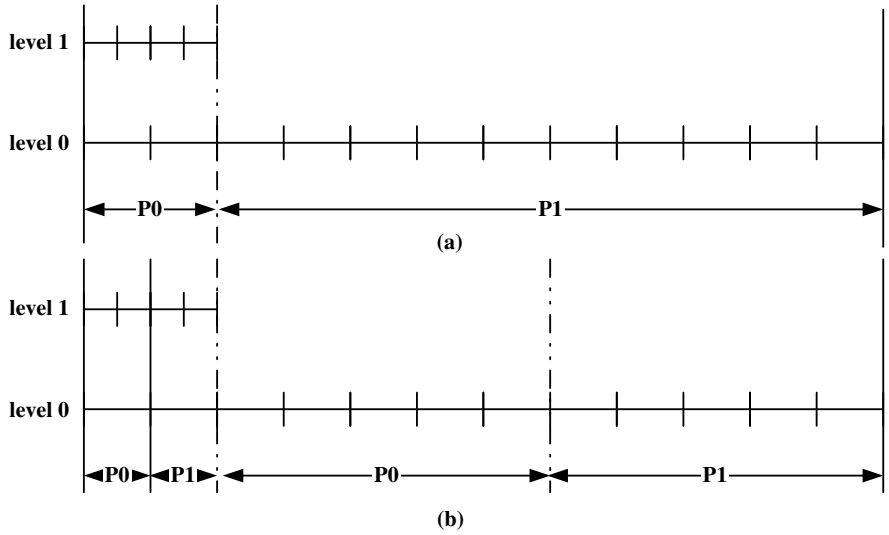


Figure 4.7: Partitions of a 1-D Grid Hierarchy (a) GPA (b) LPA

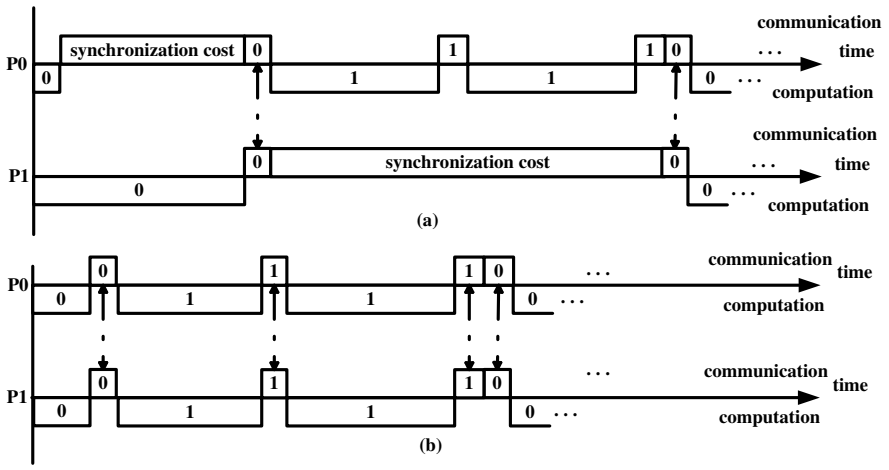


Figure 4.8: Timing Diagrams of the Example (a) GPA (b) LPA

To compare the partitioning effect of GPA and LPA, we show a simple example. Consider partitioning a one dimensional grid hierarchy with two refinement levels, as shown in Figure 4.7. For this 1-D example, GPA partitions the composite grid unit list into two subdomains. These two parts contain exactly same load: the workload assigned to $P0$ is $2 + 2 \times 4 = 10$ units while the workload assigned to $P1$ is also 10 units. From the viewpoint of GPA 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 4.8 (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 4.7 (b) which results in the computation and communication behavior as shown in Figure 4.8 (b). As a result, LPA improves the overall performance and reduces communication and synchronization time.

The experimental evaluation of LPA is conducted on BlueHorizon, the IBM SP3 cluster at San Diego Supercomputer Center. The RM3D application [41] is also used for the experiments. 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

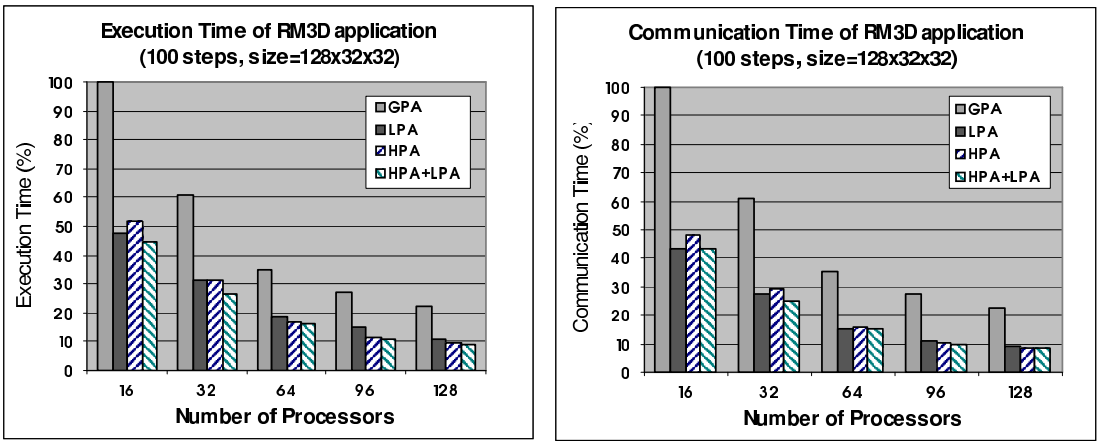


Figure 4.9: Execution and Communication Time

steps are 100. Four partitioning schemes are used in the experiments, namely, GPA, LPA, HPA and HPA+LPA. The number of processors used is between 16 and 128.

Figure 4.9 shows the execution times for GPA, LPA, HPA and HPA+LPA schemes. 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 along with the LPA scheme, we gain further improvement reducing overall execution time by about 56% for 16 processors, 60% for 128 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 4.9. The figure also shows that, HPA greatly reduces the global communication time. For all cases, HPA+LPA delivers the best performance since it takes full advantages of HPA and LPA.

## 4.3 Concluding Remarks

This chapter presented the hierarchical partitioning algorithms (HPA) and the level-based partitioning algorithm (LPA). HPA schemes allow the distribution to reflect the state of the adaptive grid hierarchy and dynamically match the computational domain hierarchy with the processor group hierarchy. By enabling incremental redistribution and concurrent communication, HPA schemes reduce the global communication and synchronization costs due to the dynamics of parallel SAMR applications. Further, the LPA scheme was presented to explicitly address the load balance issues for each refinement level, which is largely ignored in existing solutions. Combining HPA and LPA was experimentally evaluated and demonstrated performance improvement over the existing scheme.

# Chapter 5

# Adaptive Hierarchical Multi-Partitioner Strategy

SAMR dynamism/heterogeneity has been traditionally addressed using a dynamic partitioning and load-balancing algorithm that partitions and load-balances the domain when it changes, for example, the mechanism presented in [54, 67]. More recently, it was observed in [79], 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 leads to development of meta-partitioners [81], which select and configure partitioners at runtime, from a pool of partitioners, to match the application's current requirements. However, due to the spatial heterogeneity of the SAMR domain, the computation/communication requirements can vary significantly across the domain, and as a result, using 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 systems with many 1000's of processors.

The objective of the research presented in this chapter is to address this issue. Specifically, we investigate an adaptive multi-partitioner approach that dynamically applies multiple partitioners to different regions of the domain, in a hierarchical manner, to match the local requirements of the regions. In this chapter, we first present a segmentation-based clustering algorithm (SBC) that can efficiently identify regions in the domain at runtime, called *cliques*, that have relatively homogeneous requirements. Note that cliques are similar in concept to natural regions proposed by Steensland [77]. However, unlike natural regions, cliques are

not restricted to strict geometric shapes but are more flexible and take advantage of the locality-preserving property of SFCs. We then characterize the partitioning requirements of these clique regions and select the most appropriate partitioner for each clique. This research builds on the work on meta-partitioning [81] and adaptive hierarchical partitioning [60] to define an adaptive hierarchical multi-partitioner approach (AHMP). The experimental evaluation demonstrates the performance gains using AHMP. Further, to handle different resource situations, application-level pipelining (ALP) scheme has been developed to improve the performance when resources are under-utilized, and application-level out-of-core (ALOC) scheme has been developed to handle the case of inadequate resources.

## 5.1 Adaptive Hierarchical Multi-Partitioner Strategy

AHMP extends the hierarchical partitioning algorithm (HPA) presented in the last chapter and also enables incremental repartitioning and rescheduling to reduce global communication and synchronization costs. Furthermore, AHMP addresses spatial heterogeneity by applying the most appropriate partitioner to each clique based on its characteristics and requirements. As a result, multiple partitioners can concurrently operate on different subregions of the computational domain.

The basic operation of the AHMP strategy is presented in Figure 5.1. The input is the structure of the current grid hierarchy (an example is illustrated in Figure 2.1), which represents the runtime state of the SAMR application. The strategy consists of the following steps (see Table 5.1). First, the clustering algorithm is used to identify clique hierarchies. Second, each clique is characterized and its partitioning requirements identified and the available resources are partitioned into resource groups based on the relative requirements of the cliques. Third, these requirements are used by the adaptive hierarchical multi-partitioner

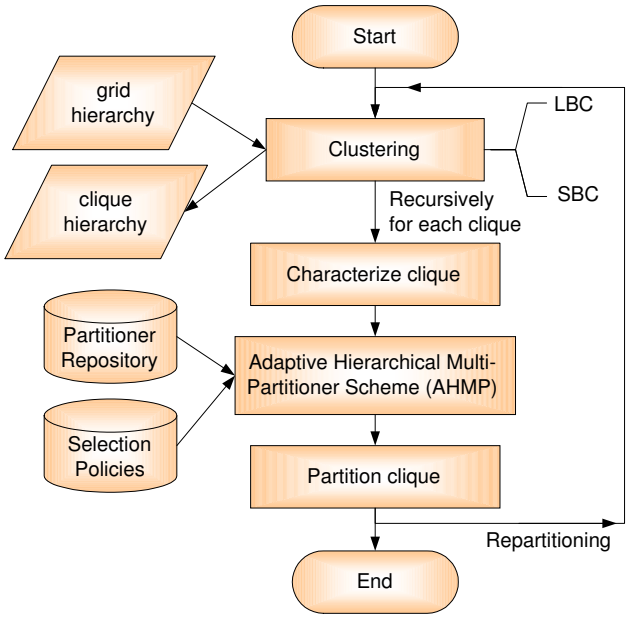


Figure 5.1: A Flowchart for the Adaptive Clustering and Partitioning Strategy

to select and configure an appropriate partitioner for each clique. The partitioner is selected from a partitioner repository using selection policies. Finally, each clique is partitioned and mapped to processors in a resource group.

Table 5.1: Adaptive Hierarchical Multi-Partitioner

1. Identify clique regions and characterize their states and requirements.
2. Characterize properties of partitioners.
3. Select the appropriate partitioner for each clique.
4. Repartition and reschedule incrementally and hierarchically within local resource group.

The strategy is triggered locally when the application states change significantly (determined using the load-imbalance metric described below), and partitioning proceeds hierarchically and incrementally. Two clustering algorithms, level-based clustering (LBC) and segmentation-based clustering (SBC) schemes, are developed. Partitioning schemes in the partitioner repository include GPA,

LPA, BPA, GMISP+SP, and pBD+ISP [37, 60, 79]. Partitioner selection policies are based on clique partitioning requirements defined in terms of refinement homogeneity, communication/computation requirements, scattered adaptation, activity dynamics [79]. This thesis specifically focuses on developing partitioning policies based on refinement homogeneity, which is defined in Section 5.6.1.

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:

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

where $A_n$ is the total number of processors in resource group A, and $T_i$ is the estimated execution time for the processor $i$, which is proportional to its load. The local load imbalance threshold is $\gamma_A$. 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.

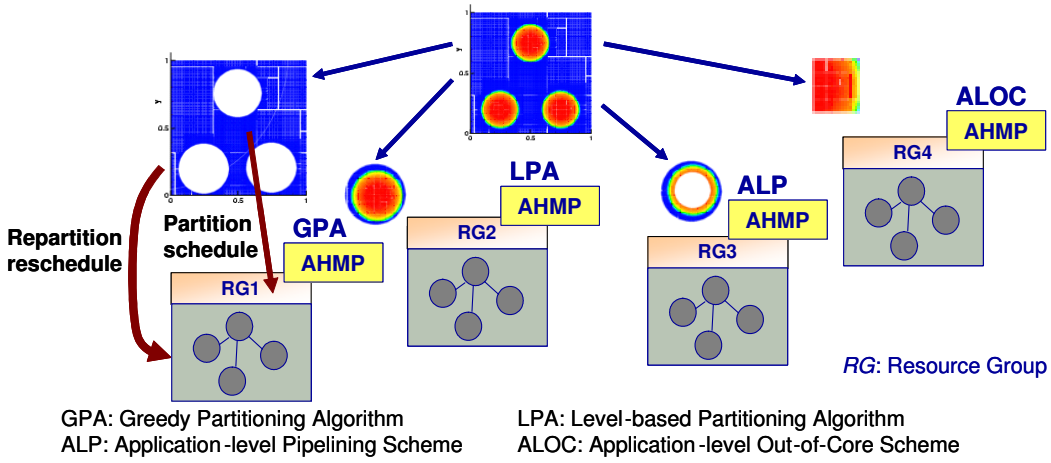


Figure 5.2: AHMP Operations - An Illustrative Example

The AHMP concept is illustrated in Figure 5.2 using a combustion simulation of hydrogen-air mixture with three initial ignition spots [70]. As shown in the figure, AHMP first applies SBC to obtain partitions of cliques and maps a resource group to each clique using the appropriate partitioning algorithm. When

the application requirements change significantly within a resource group, repartitioning is triggered and only affects the load assignment among processors in the resource group. As a result, AHMP facilitates localized data movement and communication, and enables concurrent operations across different cliques and resource groups.

## 5.2 Requirement Analysis of Clustering Schemes

The objective of clustering is to identify well-structured subregions in the SAMR grid hierarchy, called cliques. A clique is a quasi-homogeneous computational subdomain with relatively homogeneous partitioning requirements. By formulating well-structured cliques, clustering schemes attempt to ease the partitioning task since most partitioners are good at partitioning uniform/homogeneous regions. Towards this end, we identify several desired properties of an efficient and effective clustering scheme.

Subregions in a clique exhibit similar properties, such as similar refinement level structure, similar load density or similar dynamics. To minimize communication overheads presented in Chapter 2, we list the following guidelines for formulating these clique regions. (1) Clique structures should naturally reflect the state and characteristics of the current computational domain and provide an abstraction of the current computation, communication and storage requirements. (2) A clique should be connected and well structured. Well-structured cliques have simple interfaces or boundaries between them so that inter-clique communication is minimized. (3) Cliques defined in consecutive time steps should preserve locality. A clique abstracts and clusters some localized activity features in a sub-region. It strives to track the dynamics of computational domains. Locality-preserving feature is critical at reducing the data migration cost between two successive repartitioning and re-distribution steps. (4) Cliques should be of coarse granularity.

A clique is a high-level abstraction and defines a quasi-homogeneous sub-region. A fine-granularity clique will diminish the effect of clustering. (5) A clustering algorithm itself should be efficient. As mentioned before, parallel SAMR applications require regular and frequent re-partitioning and re-balancing. As a result, a clustering algorithm should be effective and efficient to minimize its overheads.

## 5.3 Segmentation-based Clustering (SBC)

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, we attempt to cluster subregions with similar refinement levels. An immediate solution is to cluster subregions purely based on refinement levels. We name this scheme as the level-based clustering algorithm (LBC). SFCs feature self-similarity, locality-preserving and efficiency, which provides a basis for formulating clustering algorithms for creating clique regions. Basically, LBC operates on the SFC-indexed subregion list. Following the SFC sequence, LBC simply groups subregions of similar refinements together.

A simple example of LBC operations on a 2-D space is illustrated in Figure 5.3. This figure shows a grid hierarchy with 0 to 3 refinement levels. The SFC traverses through the entire domain and transform the 2-D space into a 1-D sequence. Note that the SFC index in the figure is based on the base level only. The figure shows the resulting cliques, totally 11 subregions.

Based on the intuitive interpretation of refinement levels, LBC scheme is simple and efficient. However, it results in many small and irregularly-structured cliques as shown in Figure 5.3. A better solution is to employ a scheme that smoothes out small subregions and maintains spatial proximity. To this end,

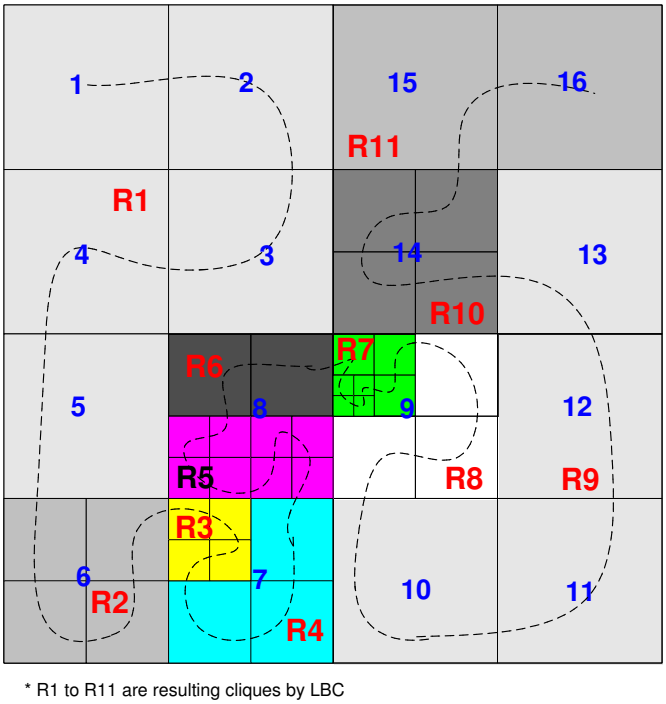


Figure 5.3: Clustering Results of LBC Algorithm

we borrow some ideas from image segmentation [50] to formulate a segmentation-based clustering algorithm (SBC), which is based on space-filling curves (SFC) [71]. The algorithm is motivated by the locality-preserving property of SFCs and the localized nature of the physical features in SAMR applications.

The segmentation-based clustering algorithm is based on ideas in image segmentation [50]. The algorithm first defines load density factor (LDF) as follows:

$$LDF(rlev) = \frac{\text{associated load on the subdomain}}{\text{volume of the subdomain at } rlev} \tag{5.1}$$

where $rlev$ denotes the refinement level and the volume (area and length in case of 2 and 1 D domains) for the subregion of interest. Note that LDF can be computed for the entire domain or for an individual clique.

The SBC algorithm is listed in Table 5.2. SBC strives to cluster domains with similar load density together to form clique regions. The algorithm first smooths out subregions that are smaller than a predefined threshold (the template size). To achieve this, SBC follows the SFC indices and extracts subregions (defined

Table 5.2: Segmentation-Based Clustering Algorithm

1. SBC(refinement-level rlev, SFC-indexed-subregion-list sfc_list)
2. Calculate the load density of the sfc_list based on the rlev and the template size, and record its frequency.
3. Find a threshold $\theta$ using the histogram of load density.
4. Partition and group subregions into several clusters based on the threshold $\theta$.
5. For each cluster, if it contains higher refinements than rlev, recursively call SBC(rlev+1, SFC-indexed-list of this cluster including higher refined subregions).

by rectangular bounding boxes) from the list until the size of the accumulated subregion set reaches the template size. SBC then calculates the load density for this set of subregions and records the frequency according to the load density. It continues to scan through the entire subregion list, and repeat the above process, calculating the load density and recording its frequency. At the third step, it finds a threshold $\theta$ based on the histogram of the load density obtained. The histogram represents the frequency/number of subregions with certain load density values. A simple intermeans thresholding algorithm [50] is used to find an appropriate threshold. Basically, the intermeans algorithm selects an initial threshold, partitions the histogram into two parts using the threshold, calculates the mean values in two parts, then uses the average of these two values as the new threshold to repeat the process until the threshold does not change significantly in successive iterations. While there are many more sophisticated approaches for identifying good thresholds for segmentation and edge detection in image processing [50], this method is sufficient for our purpose. Using the threshold obtained, SBC then partitions the domain into several clique regions. Finally, a hierarchical structure of clique regions is created by recursively calling the SBC algorithm for finer refinements. The maximum number of cliques

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 [27]. However, the SBC scheme differs from this scheme in two aspects. Unlike the Berger-Regoutsos scheme, which creates fine grained cluster, the SBS scheme targets coarser granularity cliques. SBC also takes advantage of the locality-preserving property of SFCs to potentially reduce data movement costs between consecutive repartitioning phases.

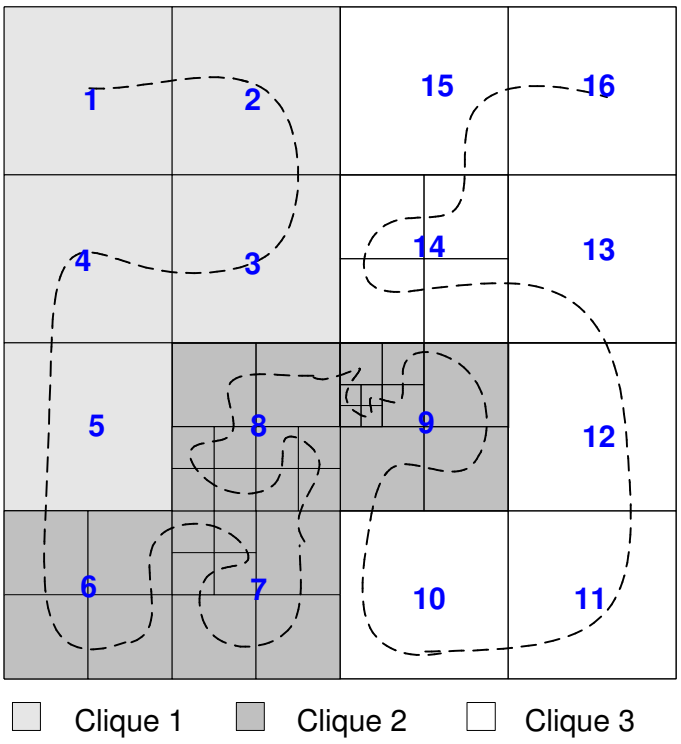


Figure 5.4: Clustering Results for the SBC Scheme

The SBC algorithm is illustrated using a simple 2-D example in Figure 5.4. In this figure, SBC results in three cliques, which are shaded in the figure.

Figure 5.5 shows the load density distribution and histogram for an SFC-indexed subdomain list. For this example, the SBC algorithm creates three cliques 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 figure on the right shows a histogram of the load density. For this example, based on the

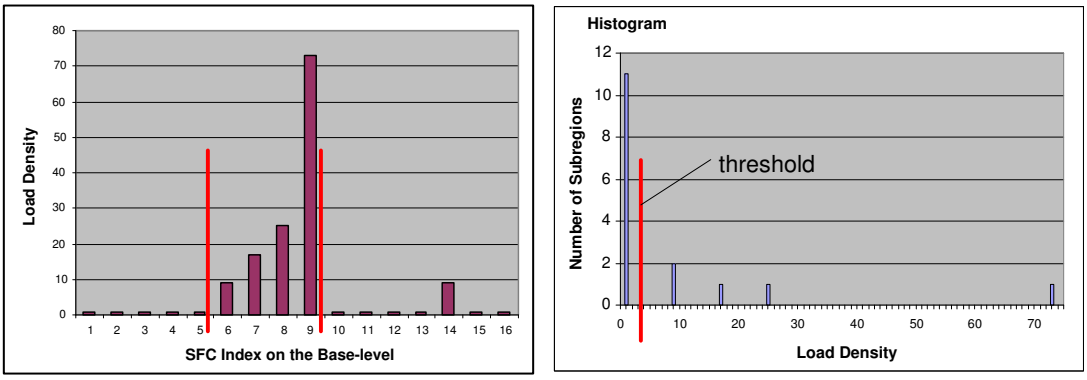


Figure 5.5: Load Density Distribution and Histogram for SBC

histogram, the threshold is identified in between 1 and 9 using the intermeans thresholding algorithm. Note that we assume a predefined minimum size for a clique region. In this example, the subregion with index 14 in Figure 5.4 does not form a clique as its size is less then the template size. It is instead clustered with another subregion in its proximity.

Also note that SBC automatically smoothes out the boundary between cliques. As mentioned in a recent paper [78], using buffer zones between partitions can eliminate significant synchronization cost. Buffer zones are coarser subregions that surround finer subregions in order to reduce communication and synchronization costs. Using smoothing, SBC implicitly creates buffer zones between cliques and hence reduces synchronization costs.

Comparing Figure 5.4 and 5.3, we observe that SBC creates better structured cliques than LBC. Experimental evaluation also confirms that SBC considerably outperforms LBC. Thus, we will focus on SBC in the rest of this chapter.

## 5.4 Application-level Pipelining Strategy

Due to the granularity constraint, domain-based partitioning schemes can explore only limited parallelism. In some cases when the deep refinements are located

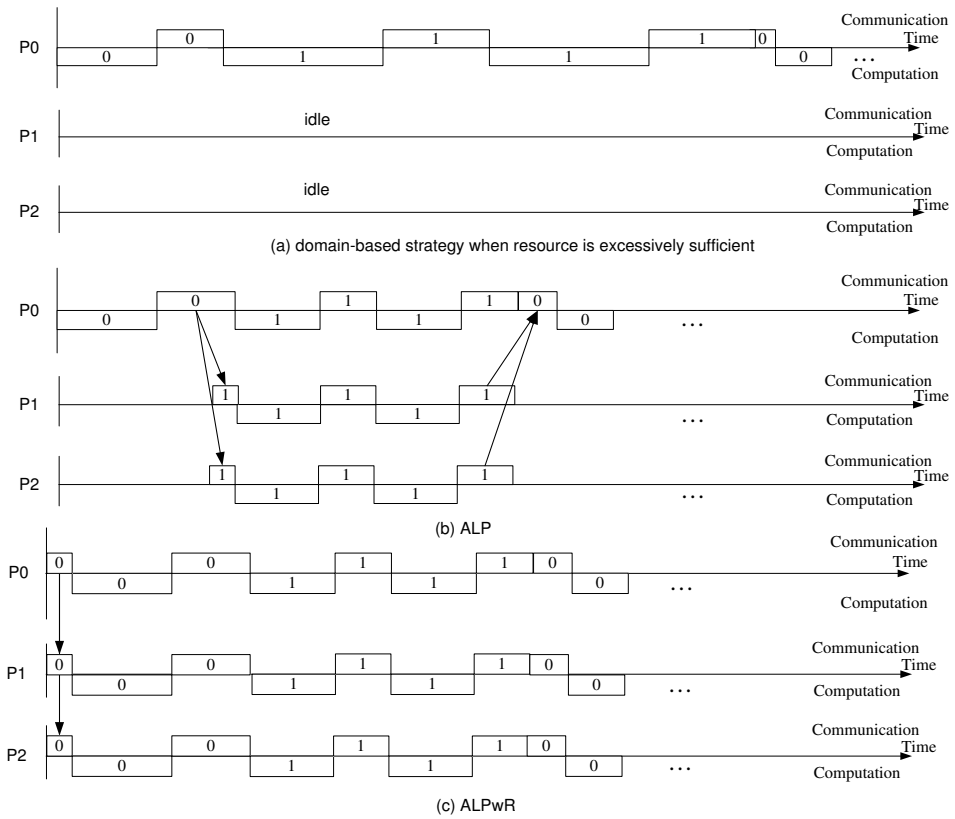


Figure 5.6: Application-level Pipelining Strategy

in a very narrow region, domain-based partitioning schemes will inevitably result in significant load imbalance when partitioning and scheduling an SAMR application onto a large-scale system. For example, assume the predefined minimum dimension of a 3D block/grid on the base level is 4 grid points. In this case, the minimum workload 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 only domain-based partitioning schemes are used. To expose more parallelism in these cases, a patch-based partitioning approach must be used. The ALP strategy combines domain-based and patch-based schemes. To reduce communication overheads, we restrict the application of the ALP scheme to a clique region that is allocated to a resource group when certain conditions are met. These conditions include: (1) resources are sufficient; (2) resources are under-utilized; (3) the gain by using ALP

outweighs the extra communication cost incurred. For simplicity, we illustrate a situation with three processors in a resource group in Figure 5.6. The clique has only two refinement levels.

ALP has two options: one is pure pipelining (ALP); the other is the pipelining with redundant computation (ALPwR). ALP splits the smallest domain-based partitions into patches of different refinement levels, partitions the finer patch into $n$ portions and schedules each portion to each processor. Since the smallest load unit on the base-level can not be further partitioned, the pure pipelining scheme processes the level 0 patch at $P_0$ while ALPwR scheme redundantly processes the level 0 patch at all participating processors. Although ALP saves redundant computation, it needs the inter-level communication between $P_0$ and other processors, which can be expensive. In contrast, ALPwR trades computing resource for less inter-level communication overheads. To avoid significant overheads, ALP schemes are applied only in a small and closely-networked resource group. The basic operations of ALP consist of pairing two refinement levels, duplicating the computation on the coarser patch and partitioning the finer patch among a small resource group. Thus, it creates a complicated hybrid partition hierarchy.

To specify the criteria for choosing ALP, we define the resource sufficiency factor (RSF) as follows.

$$RSF = \frac{N_{rg}}{L_q/L_\delta} \tag{5.2}$$

where $L_q$ denotes the total load for a clique region, $N_{rg}$ denotes the total number of processors in a resource group, and $L_\delta$, the granularity, denotes the load on the smallest base-level subregion with the maximum refinement level. In the case of deep refinements, $L_\delta$ is significantly large. When $RSF > \rho$ and resources are under-utilized, where $\rho$ is the threshold, we can apply ALP to explore more parallelism.

## 5.5 Application-level Out-of-Core Strategy

When the available physical memory of resources is not sufficient to execute the application, one option is to rely on the virtual memory mechanism of the operating system (OS). The OS will then handle page faults and replace the less frequently used pages by loading data from disks. Using this approach, OS has little knowledge of the application characteristics and its memory access pattern. Consequently, it will result in many unnecessary swap-in and swap-out operations which are very expensive. The data rates from the disk are approximately two orders of magnitude less than from the memory [53]. In many systems, OS sets a default maximum physical and virtual memory allocation. When the application uses up the preset quota of memory, it cannot proceed but crash. To show the influence of memory availability on the performance, we show two simple experiments.

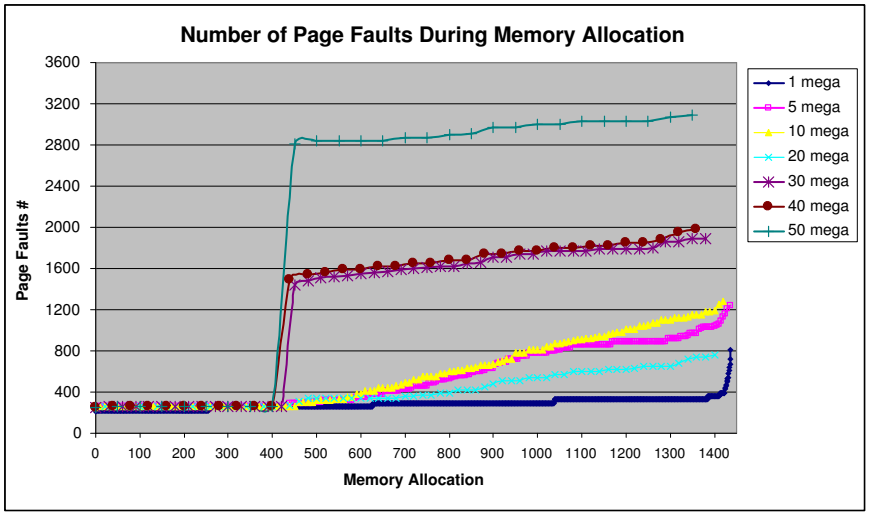


Figure 5.7: Number of Page Faults versus Allocated Memory

The experiments on the effects of allocated memory are performed on an Intel Pentium 4 machine with the following specifications: 1.70GHz CPU, 256 KB cache, 512 MB physical memory, 1 GB swap space, Linux 2.4 kernel. The first

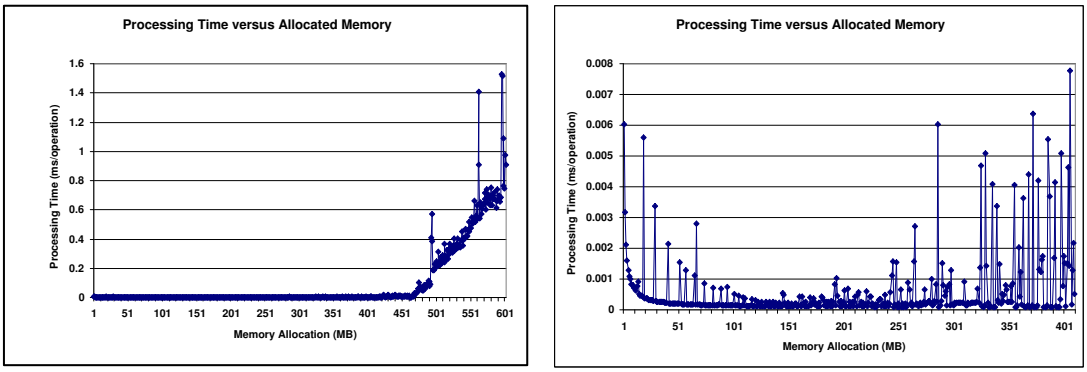


Figure 5.8: Processing Time versus Allocated Memory

experiment measures the number of page faults when we allocate memory incrementally. Seven cases with different incremental steps, 1 MB to 50 MB every loop, are shown in Figure 5.7. We clearly observe the dramatic increase of the number of page faults when we allocate memory up to about 400-500 MB, which is close to the physical memory capacity. To measure the performance degradation due to excessive memory allocation, we conduct another experiment to measure the computing speed. The operations are one floating-point multiplication and division on randomly selected memory addresses. Figure 5.8 shows serious performance degradation when we use up to about 400-500 MB memory. From these experiments, we observe that, as expected, the amount of the allocated memory plays a critical role at affecting the overall system performance. Thus we need a strategy that judiciously avoids this performance degradation.

Instead of blindly resorting to the OS, our solution is to design an application-level out-of-core scheme by exploiting the application memory access patterns to guide the paging process and explicitly keep the working-set of application patches while swapping out the unused patches.

We approach the solution through the application-level out-of-core (ALOC) mechanism to execute the application incrementally by proactively managing application-level *pages*. Using the ALOC strategy, we aim to not only improve the performance but also enhance the survivability under the severe shortage of

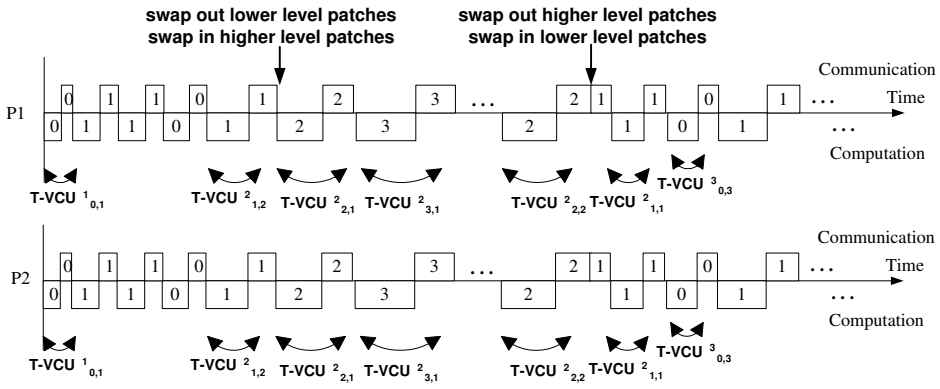


Figure 5.9: Application-level Out-of-core Strategy

resources. For instance, as shown in Figure 2.5, the RM3D application presented requires 4 times more memory during the peak time than the average while the peak time lasts for less than 10% of the total execution time. In this case, if the pre-allocated resources are sufficient to meet only the requirement of a very long running SAMR application for 90% of its execution time but not sufficient to accommodate the peak requirements of the application, without using ALOC strategy, it will result in a significant slow down or a crash (being killed by OS).

As illustrated in Figure 5.9, ALOC scheme incrementally partitions the local grid hierarchy into *T-VCU* (temporal virtual computational unit) according to refinement levels and runtime iterations. In the figure, the notation $T\text{-}VCU^{a}_{b,c}$ denotes the temporal VCU, where $a$ denotes the time step at the base level, $b$ for the current refinement level and $c$ for the time step at the current level. To avoid undesired page-swapping, ALOC automatically releases the memory held by lower-level patches and swaps them out to the disk. The condition to trigger the ALOC mechanism is that the ratio between the amount of the allocated memory and the amount of the physical memory is over a predefined threshold $\phi$.

## 5.6 Experimental Evaluation

### 5.6.1 Clustering Quality Metric

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

(1) **Refinement Homogeneity**: This measures the quality of the structure of a clique. Partitioning algorithms typically work very well on highly homogeneous grid structures and can generate scalable partitions with desired load balance. Let $|R_i^{total}(l)|$ denote the total size of a subregion or a clique at refinement level $l$, which is composed of $R_i^{ref}(l)$, the size of refined regions, and $R_i^{unref}(l)$, the size of un-refined regions at refinement level $l$. Refinement homogeneity is recursively defined between two refinement levels as follows:

$$\begin{aligned} H_i(l) &= \frac{|R_i^{ref}(l)|}{|R_i^{total}(l)|} \\ H_{all}(l) &= \frac{1}{n}\sum_{i=1}^{n} H_i(l), \text{if } |R_i^{ref}(l)| \neq 0 \end{aligned}$$

where $n$ is the total number of subregions that have refinement level $l+1$.

(2) **Communication Cost**: This measures the communication overheads of a clique and includes inter-level communication, intra-level communication, synchronization cost, and data migration cost as described in the previous chapter.

(3) **Clustering Cost**: This measures the efficiency of the clustering algorithm itself. As mentioned above, SAMR applications require regular re-partitioning and re-balancing, and as a result clustering cost become important.

### 5.6.2 Evaluating the Effectiveness of SBC Scheme

This section evaluates the effectiveness of SBC-based clustering using the metrics defined above. First, it compares the refinement homogeneity of 6 SAMR application kernels with and without clustering. These applications are summarized in Table 3.1.

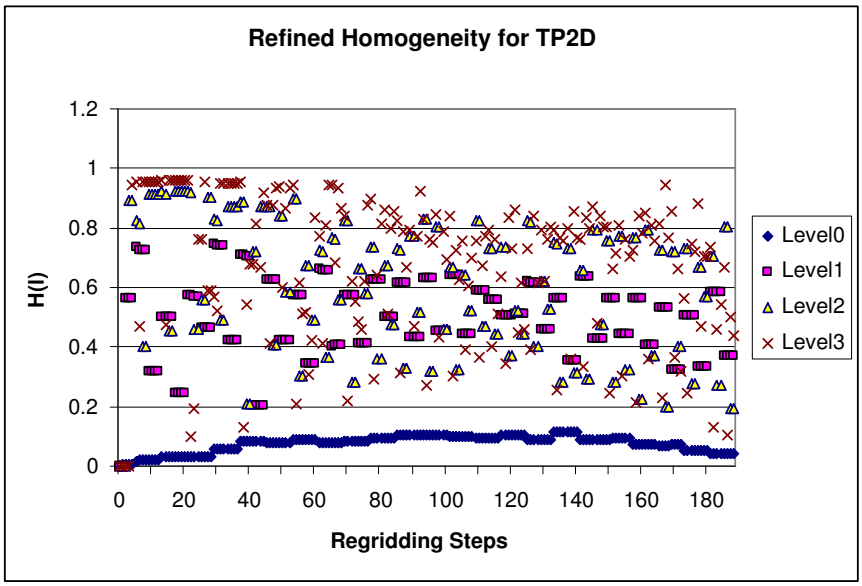


Figure 5.10: Refinement Homogeneity for the Transport2D Application Kernel (4 levels of refinement)

Figure 5.10 shows the refinement homogeneity at different regrid steps for the TP2D application with 4 refinement levels and without clustering. The refinement homogeneity is smooth for level 0 and very dynamic and irregular for levels 1, 2 and 3. The refinement homogeneity for RM3D and RM2D is illustrated in Figure 5.11. We observe similar irregular and dynamic refinement behavior.

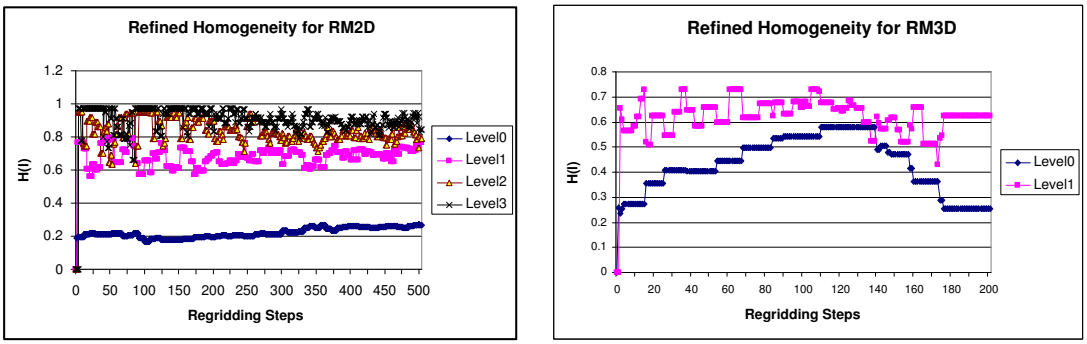


Figure 5.11: Refinement Homogeneity for RM2D (4 levels) and RM3D Applications (2 levels)

Table 5.3: 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 for 6 SAMR applications without clustering is presented in Table 5.3. The table shows that the refinement homogeneity $H(l)$ increases as the refinement level $l$ increases. 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 $H(3) \in [0.66, 0.9]$. Since the refinement homogeneity on level 3 and above is typically over 0.6 and refined subregions on deeper refinement levels tend to be more scattered, the clustering schemes will focus efforts on clustering level 0, 1 and 2. Furthermore, based on these statistics, we set the threshold $\theta$ 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. Based on the experiments presented in [79], policies are defined to select the partitioners GPA and G-MISP+SP for cliques with refinement homogeneity less than the threshold $\theta$, and the partitioners LPA

and pBD-ISP for cliques with refinement homogeneity greater than the threshold. The objective of this policy is to obtain better load balance for less refined cliques, and to reduce communication and synchronization costs for highly refined cliques.

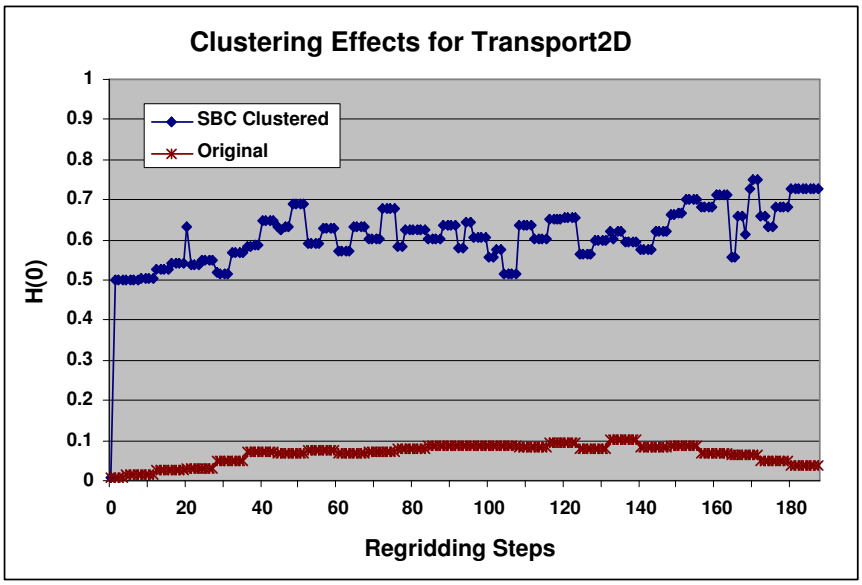


Figure 5.12: Homogeneity Improvements using SBC for TP2D

Figure 5.12 and Table 5.4 demonstrate the improvements in *refinement homogeneity* by using the SBC algorithm. Figure 5.12 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].

Table 5.4: 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 |

The effects of clustering using SBC for the 6 SAMR applications are presented

in Table 5.4. In this table, 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 homogeneity starting from level 0 as shown in Table 5.3. These results demonstrate the effectiveness of the clustering scheme. Moreover, clustering significantly increases the effectiveness of partitioners and improves overall performance as shown in the next chapter.

### 5.6.3 Performance Evaluation

This section presents an evaluation of the AHMP scheme using the clustering quality metrics defined above.

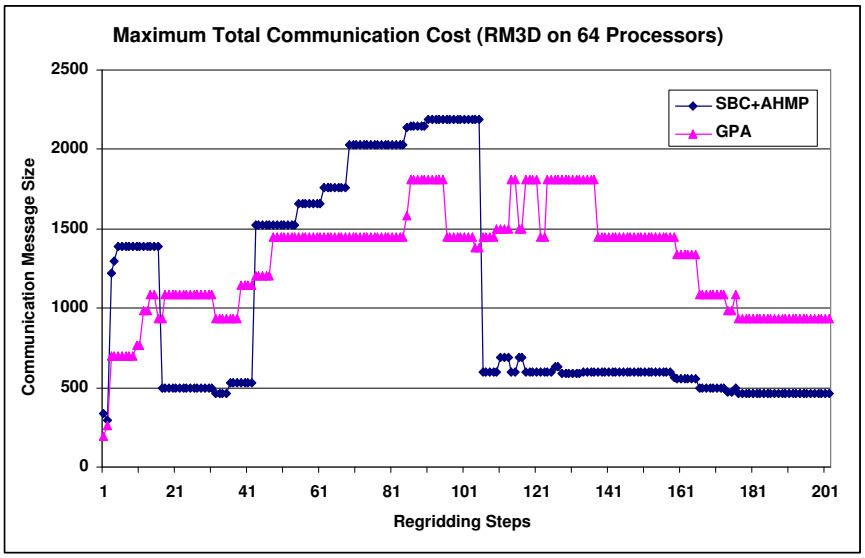


Figure 5.13: Maximum Total Communication for RM3D on 64 Processors

**Communication Costs:** The evaluation of communication cost uses a trace-driven simulation. Figure 5.13 shows the 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 in this period. The snapshot at the regrid step 96 in Figure 2.5 demonstrates the scattered refinements. This in turn causes significant clique movement during re-clustering. Note that the simulator does not measure synchronization costs. Since LPA has been shown to significantly reduce synchronization costs [60], selecting LPA within AHMP should further reduce these costs and improve performance.

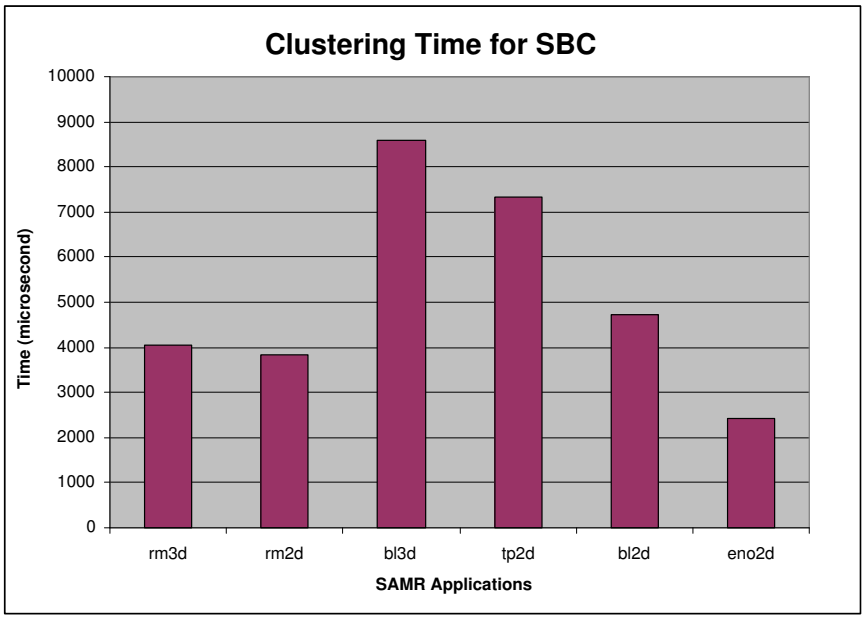


Figure 5.14: Clustering Costs for the 6 SAMR Application Kernels

**Clustering Costs:** The cost of the SBC clustering algorithm is experimentally evaluated using the 6 different SAMR application kernels on Frea, a Beowulf cluster at Rutgers University. The cluster consists of 64 processors and each processor has a 1.7 GHz Pentium IV CPU, 512 MB physical memory, 1 GB swap space, and a Linux 2.4 kernel. The costs are plotted in Figure 5.14. 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.

The overall performance benefit of the AHMP scheme 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. As described in Chapter 3, RM3D is highly dynamic, and exhibits scattered refinement activity and space-time heterogeneity. These characteristics make approaches that use single partitioner inadequate. As a result, RM3D is an appropriate application that can demonstrate the benefits using the AHMP scheme. In the experiment, the number of processors used was between 64 and 1280.

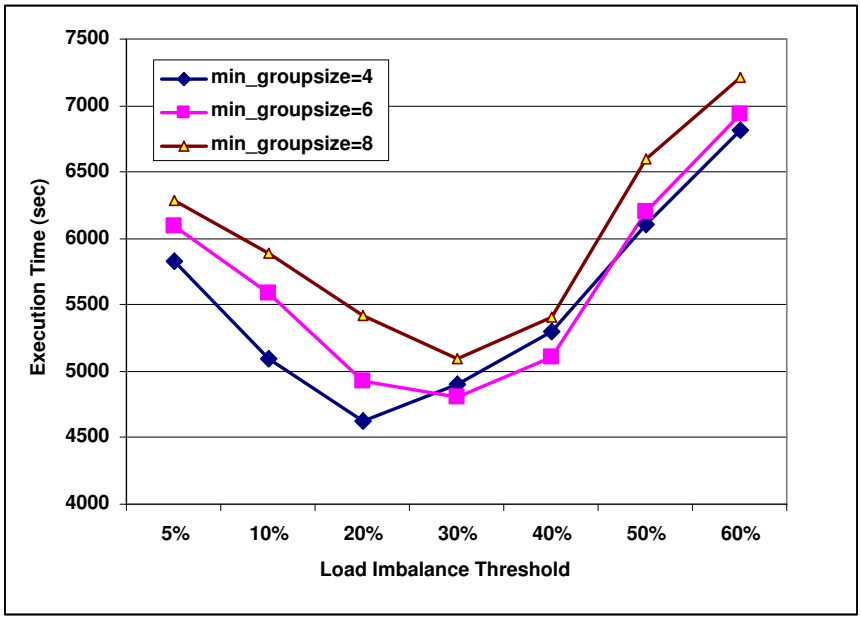


Figure 5.15: Impact of Load Imbalance Threshold for RM3D on 128 Processors

**Impact of Load Imbalance Threshold and Resource Group Size:** As mentioned in Section 3, the load imbalance threshold $\gamma$ is used to trigger repartitioning and redistribution within a resource group. 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 is plotted in Figure 5.15. When $\gamma$ increases from 5% to around 20% to 30%, the execution time decreases. On the other hand, when $\gamma$ increases from 30% to 60%, the execution time increases significantly. Smaller values of $\gamma$ 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 $min\_group\_size = 4$.

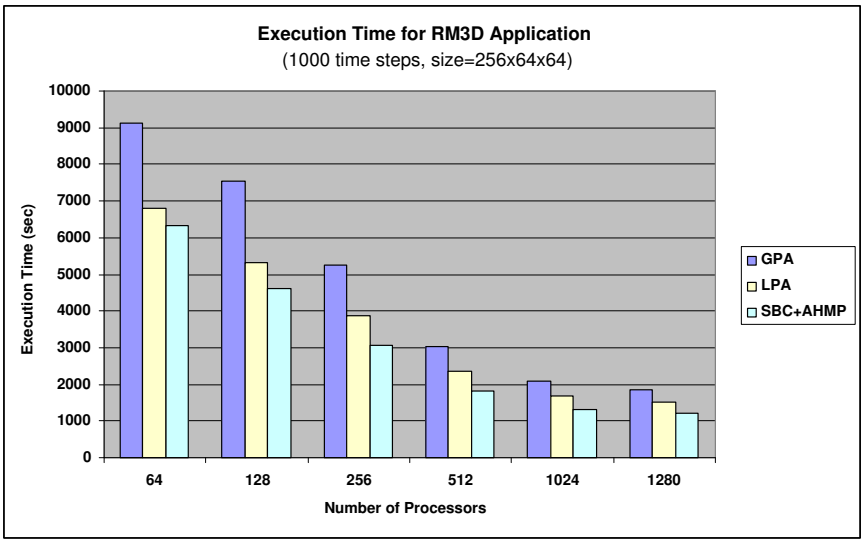


Figure 5.16: Overall Performance for RM3D

**Overall Performance:** The overall execution time is plotted in Figure 5.16. The figure plots execution times for static GPA, static LPA and the AHMP scheme with clustering, i.e., SBC+AHMP in the plot. 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) the AHMP scheme takes advantage of the strength of different partitioning schemes matching them to the requirements of each clique; (2) the SBC scheme creates well-structured cliques that reduce the communication

traffic between cliques; (3) the AHMP scheme enables incremental repartitioning/redistribution and concurrent communication between resource groups.

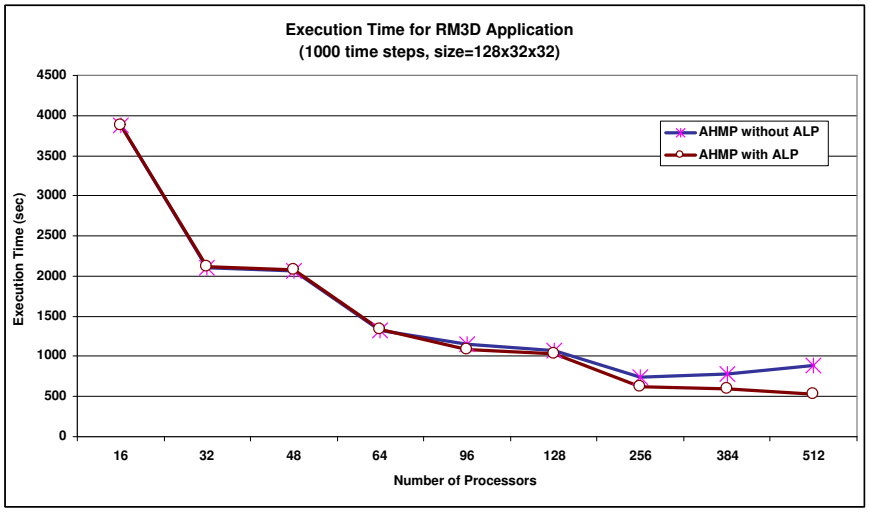


Figure 5.17: Experimental Results: AHMP with and without ALP

**Impact of Pipelining:** To show the impact of application-level pipelining scheme (ALP), 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 ALP, the overall performance degrades when we deploy it on a cluster with over 256 processors. The main reason is that, without ALP, the granularity constraint and the increasing communication overheads overshadow the increased computing resources. However, with ALP, AHMP can further scale up to 512 processors with performance gains up to 40% compared to the scheme without ALP. Note that the maximum performance gain (40%) is achieved when using 512 processors, wherein the scheme without ALP results in the degraded performance.

**Impact of Out-of-Core:** The ALOC scheme has been implemented using the HDF5 library [10], which is particularly suited 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

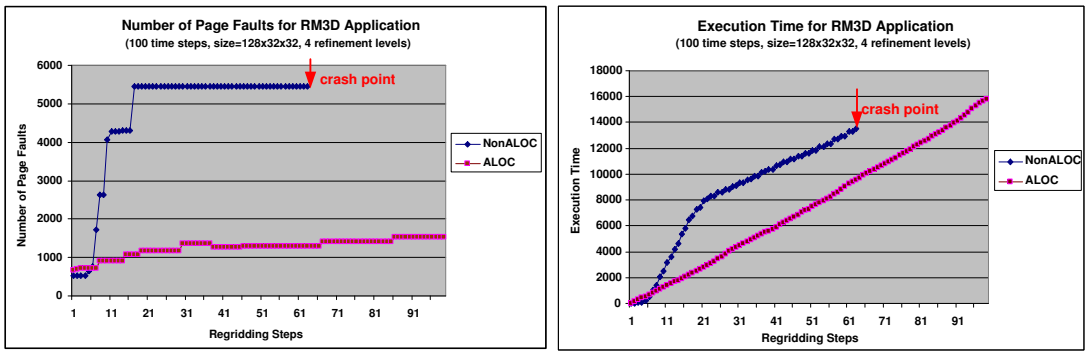


Figure 5.18: Number of Page Faults: NonALOC versus ALOC

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 5.18 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 due to memory thrashing. With ALOC, the number of page faults is reduced. As a result, the ALOC scheme improves the performance and enhances the survivability.

## 5.7 Concluding Remarks

This chapter presented the adaptive hierarchical multi-partitioner (AHMP) scheme to address the space-time heterogeneity in dynamic SAMR applications. The AHMP scheme applies multiple partitioners to different regions of the domain, in a hierarchical manner, to match the local requirements of the regions. The chapter first presented a segmentation-based clustering algorithm (SBC) that can efficiently identify clique regions in the domain at runtime, which have relatively homogeneous requirements. The partitioning requirements of these clique regions are then characterized, and the most appropriate partitioner for each

clique is selected. To handle different resource situations, two hybrid schemes have been developed. The application-level pipelining scheme (ALP) combines the domain-based and patch-based decomposition techniques when resources are under-utilized. In contrast, when resources are inadequate, the application-level out-of-core scheme (ALOC) has been developed to operate on the computational domain incrementally and enhance the survivability. This AHMP approach and its components have been implemented and experimentally evaluated using 6 SAMR application kernels. The evaluations demonstrated the effectiveness of the clustering and the performance improvements using AHMP strategies.

# Chapter 6

# GridMate: Simulation of Dynamic Applications on Multi-Site Grid Systems

The exponential growth in computing, networking and storage technologies has also ushered in a new computing era for harnessing the potential of heterogeneous and distributed resources on an unprecedented scale. Inspired by the pervasiveness, convenience, economics and open standards of the electrical power grid, Grid computing is rapidly emerging as the new computing paradigm of the $21^{st}$ century for solving grand challenge problems in varied domains of science, engineering and business [46, 47, 45]. Its goal is to provide a service-oriented infrastructure that leverages open standard protocols and services to enable coordinated resource sharing and problem solving in dynamic multi-institutional virtual organizations [45]. A number of major Grid infrastructures are being developed and deployed [6, 11, 18] and many grand challenge problems are being tackled by exploiting the power of the Grid [1, 5, 9, 12, 15]. Furthermore, resources on the Grid, including geographically distributed computers and storage systems, are also inherently heterogeneous and dynamic. The coupled heterogeneity and dynamism of resources and applications make runtime management of SAMR-based dynamic Grid applications a significant challenge.

The previous chapters experimentally demonstrate the performance of the proposed schemes. However, these experiments are restricted to a single-site supercomputer cluster. To aid the evaluation of the viability of the proposed strategies in Grid environments, we build a simulation environment that is suited to our needs. Grid-based SAMR applications exhibit three key distinguishing

characteristics: (1) They are inherently large and require large amount of computational resources, typically spanning multiple sites on the Grid. Furthermore, the exact resource requirements are often not known a priori and depend on the application runtime behavior. (2) They may execute for days, weeks or months and often the exact execution time is not known a priori. For instance, it is not always known how long a scientific and engineering simulation will have to run before it provides meaningful insights into the phenomenon being modelled. (3) They are highly dynamic and heterogeneous in space and time. In addition, their dynamics and heterogeneity patterns are not known a priori.

Thus, the desired simulator needs to model the systems and applications such that these realistic characteristics are well reflected. This chapter presents the design, operations and evaluation of the GridMate simulator.

## 6.1 Motivation

As described in the previous chapters, SAMR applications are highly dynamic and exhibit space-time heterogeneity. In Grid environments, we are confronted with a new dimension of complexity. Particularly, Grid systems consists of largely different software and hardware resources with changing capacity and availability and are inherently dynamic and heterogeneous. To demonstrate these characteristics, we show a typical scenario with two resource sites in Figure 6.1. The temporal heterogeneity is represented by the variation of available capacity (number of available processors) of a single resource site over time. The spatial heterogeneity is represented by the variation in the available resources across sites. In this chapter, we consider the heterogeneity at a coarse-granularity. Specifically, we focus on space-sharing scenarios and leave the time-sharing cases for future work. The resource usage patterns presented are derived from synthesized traces based on the real traces from supercomputer centers [64]. More details will be presented

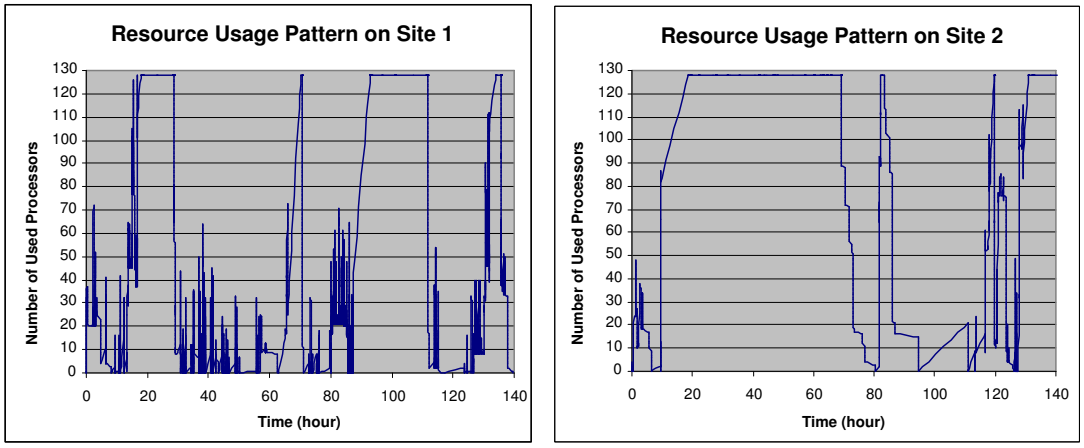


Figure 6.1: Spatial and Temporal Heterogeneity of Resources on Two Sites

in the experimental evaluation section.

## 6.2 Related Work

Grid computing is emerging as an important new distributed computing paradigm. Grid environments are inherently heterogeneous and highly dynamic. Further, the ever-increasing system complexity, scale and diversity of software and hardware make system management a significant challenge. To attack this challenge, a number of resource management systems have been developed, such as Globus [7, 42, 48], Condor [4, 62], AppLeS [29, 85] and Legion [14].

Performance evaluation plays a critical role in studying, calibrating, and comparing various resource management techniques and systems for Grids. However, in Grid environments, the capacity and availability of resources change with time, along with a wide spectrum of dynamic applications. Due to the inherent dynamism and heterogeneity in Grid environments, it is quite difficult to obtain repeatable and comparable performance evaluation under identical system setups. Hence, simulation techniques are adopted. Simulation has been widely used for modeling and studying real-world systems and phenomena. To enable simulation, researchers have proposed some general simulation languages (Parsec [21])

and specifications (DEVS and HLA [86]). Moreover, a large number of simulation tool kits and libraries have been developed, including NS2 [33], OpenNet, Ptolemy [63], SimJava [17]. However, because Grid computing involves complex interacting components, there are only a few simulators that can model Grid environments. These include MicroGrid [76], SimGrid [58], and GridSim [34].

MicroGrid, a prominent Grid emulator developed in UCSD, is based on the Globus Toolkit [7]. It offers a virtual Grid environment for simulating the execution of real applications. As an emulator, MicroGrid produces quite accurate simulation results. However, the simulation modelling and configuration process is quite demanding. In addition, due to its emulation nature, simulation based on MicroGrid is quite time-consuming. The SimGrid toolkit, developed in UCSD, features flexible application scheduling mechanisms. It supports modeling of time-shared resources and applications from realistic traces. The GridSim toolkit, developed in University of Melbourne, is a Java-based simulation tool. It supports modeling of space-shared and time-shared large-scale resources in Grid environments. It also supports the simulation of economy-based resource scheduling policies in the Grid.

The Grid simulators described above enable simulating a wide spectrum of scenarios in Grid environments. However, they consider only resource heterogeneity and do not address the coupled space-time heterogeneity of both resources and SAMR applications. As a result, using these simulation toolkits, one needs to manually create the graph representing the SAMR domain, manually partition it, and assign the partitions to the heterogeneous resources.

## 6.3 Conceptual Architecture

To explicitly address the coupled heterogeneity of both applications and resources, we design the simulator to leverage both application partitioning techniques and

resource scheduling techniques. Thus, our main tasks are to model systems, applications, application partitioning and resource scheduling heuristics. Following this rationale, we build a Grid simulator called GridMate. The multi-layer system architecture of GridMate is illustrated in Figure 6.2. The bomottom layer is the

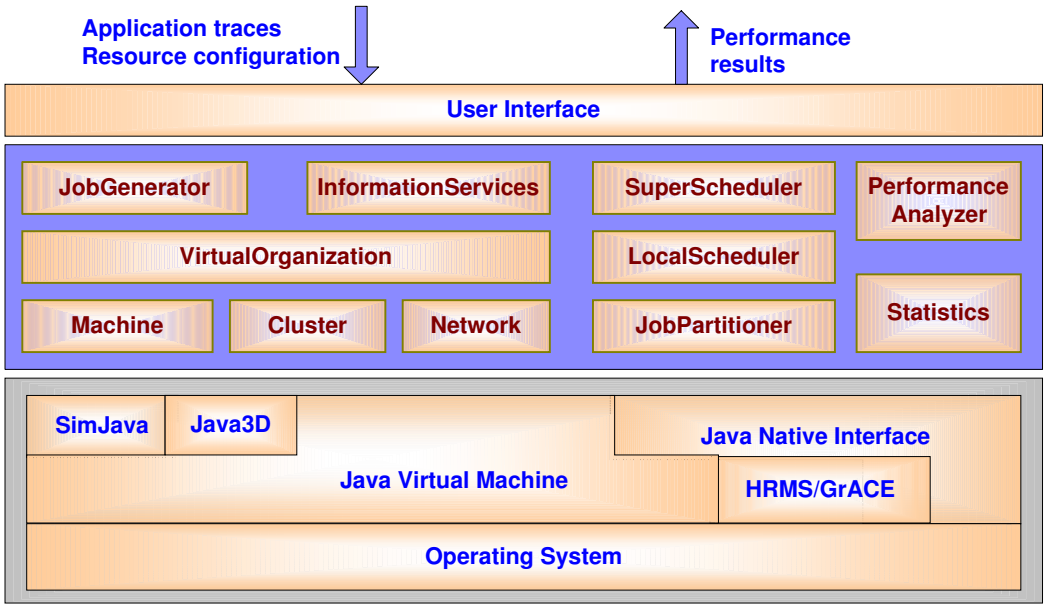


Figure 6.2: System Architecture of GridMate

operating system. On the top of the operating system, we have three components, Java Virtual Machine (JVM), GrACE and Java Native Interface (JNI). JVM provides the portable runtime support for Java bytecoded files. Because our specific application/job partitioners are implemented using the GrACE toolkit which is implemented in C++, we add a JNI wrapper layer to expose partitioning and other services to the simulator. On top of the JVM, we use the discrete-event simulation tool SimJava [17]. GridMate is built based on SimJava and GrACE (via JNI) [8]. It consists of the following major components: job generators, information services, virtual organization (machines, clusters, networks), local scheduler, super scheduler and performance monitor and analyzer. The input to the GridMate are a set of application traces including local jobs and SAMR jobs, resource configuration and scheduling policies. The output from the GridMate

are various performance results based on the performance metrics defined in the next section.

## 6.4 Scheduling Architecture and Operations

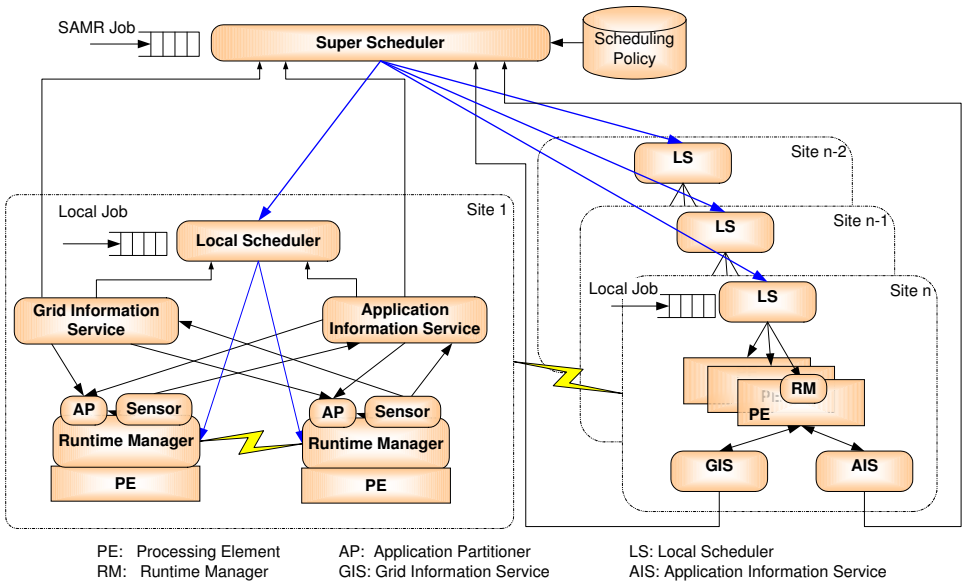


Figure 6.3: Conceptual Architecture of HRMS on a Multi-Site Grid

Handling the coupled heterogeneity of SAMR applications in complex grid computing environments is challenging. Figure 6.3 shows the scheduling architecture of GridMate. It attempts to incorporate mechanisms to enable hierarchical runtime management and self-adaptivity. In particular, the conceptual architecture consists of three different levels: the overall system level (Grid), local site level (VO) and individual machine level. Key components are:

- Super Scheduler (SS): The target SAMR job is submitted to the super scheduler. SS makes scheduling decisions according to scheduling policies and current runtime states of SAMR jobs and resources. SS is in charge of dynamic co-allocation of the SAMR job to different resource sites through their local schedulers. In the Grid, there can be a variety of SSs for different

classes of big jobs. In this thesis, we only consider the case when there is only one SS for the SAMR job.

- Local Scheduler (LS): There is a local job queue associated with each local scheduler. These local jobs could be batch or interactive with various job specifications, such as number of processors required, execution time, deadline etc. We differentiate these local jobs with SAMR jobs. LS makes scheduling decisions according to its local scheduling policies for local jobs and SAMR jobs.

- Runtime Manager (RM): Runtime managers are organized in a hierarchical fashion as shown in Figure 3.1. An RM is composed of application partitioners and sensors.

- Application Partitioner (AP): These are adaptive application-centric partitioners specialized for SAMR jobs. An AP resides in each processor in order to monitor the runtime requirements of applications and strives to improve performance by repartitioning and balancing for the target dynamic applications. APs take into account the application runtime characteristics to make partitioning or repartitioning decisions on behalf of SAMR jobs. Several partitioning strategies have been presented in the previous chapters.

- Sensors: These sensors monitor both resource and application runtime status. Grid information service (GIS) pulls the resource information from these sensors. Application information service (AIS) gathers the application runtime information from these sensors also. Resource sensors can be implemented using NWS [85], while application sensors are embedded into each application sub-task.

- Application Information Service (AIS): AIS collects the updated information of application runtime states from application sensors. AIS resides in

each local site and thus enables the aggregation of application states in a hierarchical fashion.

- Grid Information Service (GIS): GIS collects the updated information of resource states from resource sensors. It provides resource information to super scheduler and application partitioners so that they can make scheduling and partitioning decisions according to policies and current resource status.

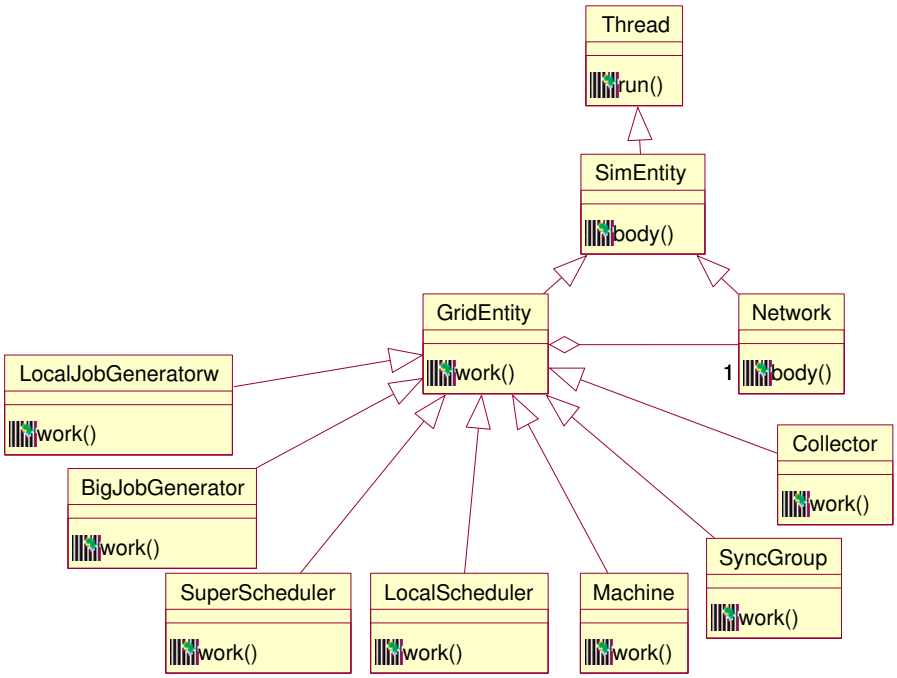


Figure 6.4: Class Diagram of GridMate Entities

As required by SimJava, all the simulation entities are derived from the "Sim_entity" class. Figure 6.4 plots a class diagram describing the inheritance hierarchy of major simulation entities. In the figure, GridEntity defines an abstract method *work()* to wrap the *body()* method as required by SimJava. Further, all these entities are derived from *Thread* of Java runtime library. When the simulation starts, all these entities will be instantiated and will execute in

parallel as independent threads. This multi-thread feature is crucial to simulate parallel applications. Every GridEntity object and its derived object is equipped with a *Network* communication channel, which enables flexible and transparent communication modeling. In the simulation model, the communication cost is associated with the sender's messaging overhead, which is proportional to the message size plus a small constant overhead. The non-blocked send approach is adopted such that a sender will not be blocked even though the corresponding receiver is not ready receiving. On the receiver's side, if the message is available, there is no communication delay for the receiver; if the message is not available, the receiver has options to process other tasks or to be blocked until the message arrives. This communication model simulates the non-blocking message passing paradigm (such as *MPI_Isend()* and *MPI_Irecv()* ), which is a common practice in Sparallel applications.

The sequence diagram in Figure 6.5 illustrates the primary interactions among GridMate entities. Note that the local job generators and global job generator execute in parallel. Their job arrivals can be overlapped and result in resource contention and jobs queued at local schedulers or super scheduler. For local jobs, we assume that the parallel execution times obtained from application traces include all processing time, communication time and other overheads. Since we intend to model the exact computation and communication patterns in parallel SAMR applications, for SAMR jobs, we explicitly consider the communication cost incurred during synchronization after each iteration due to resource heterogeneity and load imbalance.

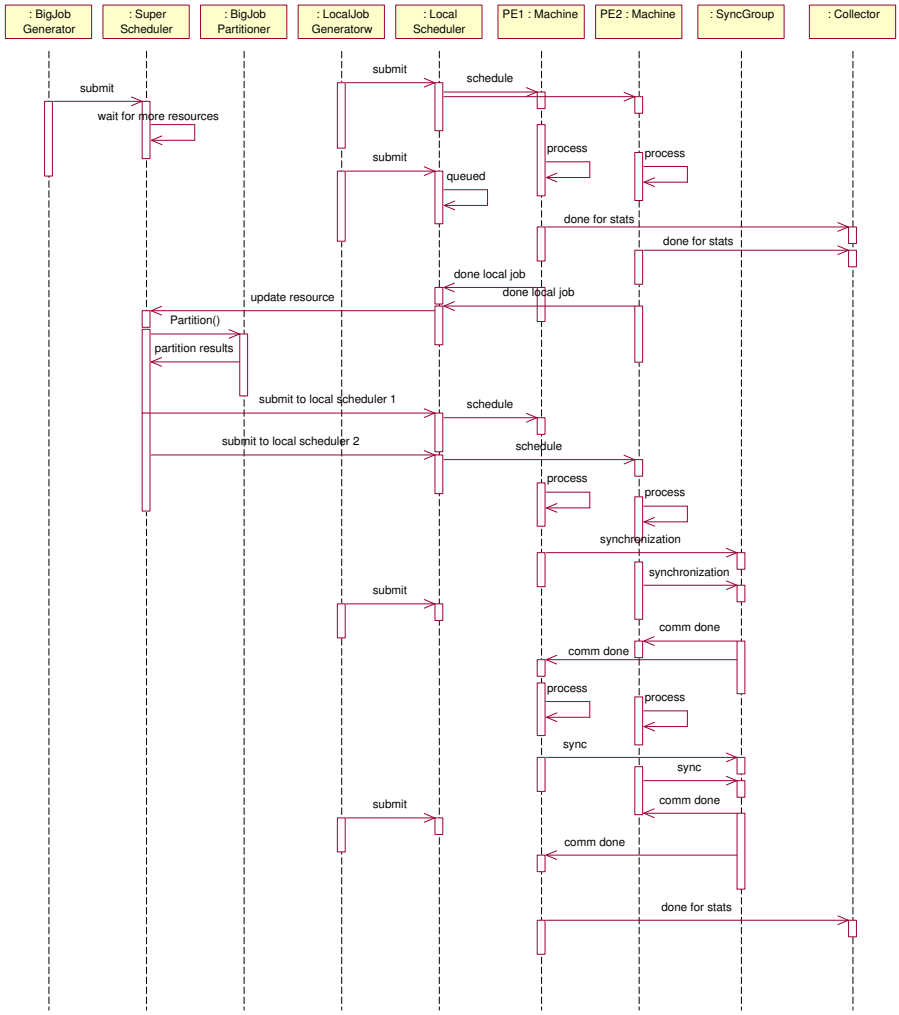


Figure 6.5: Sequence Diagram for Interaction among GridMate Entities

## 6.5 Experimental Evaluation

### 6.5.1 System Setup

In GridMate, the Grid system is composed of several computer/resource sites. Totally, we set up 4 resource sites. On each site, there are 128 homogeneous processors. On different sites, computers are heterogeneous in computing speed, communication bandwidth and memory capacity. Each site has its local scheduler and its local job arrivals follow the workload model presented in [64]. This

workload model is based on workload logs from three sites, San Diego Supercomputer Center (SDSC), Los Alamos National Lab (LANL) and Swedish Royal Institute of Technology. In this model, the job sizes follow a two-stage uniform distribution, job execution times follow the hyper-Gamma distribution and job arrivals follow two Gamma distributions. In this thesis, we will focus our study mainly on partitioning and scheduling SAMR jobs. For local job scheduling, a substantial research effort already exists [44].

The target application is RM3D [41]. Its execution trace is submitted to the super-scheduler and executed across sites. The performance evaluation of HRMS strategies is compared with the baseline scheme. The baseline scheme statically allocates resources such that they meet the peak requirement of the SAMR application.

### 6.5.2 Evaluation Metric

The performance evaluation metrics used are waiting time, execution time and response time for the SAMR job. Additionally, to compare with the baseline scheme, we define a processor efficiency factor $\eta$ and processor-time factor $\varsigma$ as follows.

$$\varsigma = \sum_{i=1}^{n} \sum_{j=1}^{s} (NC_j \times N_{i,j} \times \tau_{i,j}) \tag{6.1}$$

where, $n$ is the total number of application iterations/phases, $s$ is the total number of sites, $NC_j$ is the normalized capacity of one processor on site $j$, $N_{i,j}$ is the number of allocated processors, $\tau_{i,j}$ is the length of the $i$-th time interval and the subscript $(i, j)$ denotes in the $i$-th time interval on the site $j$. $\varsigma^h$ is denoted for the HRMS scheme and $\varsigma^b$ for the baseline scheme. This equation represents the normalized total computational resource consumption. The physical meaning of $\varsigma$ could be interpreted as the total execution time if the application is assigned to

a single standard processor (its $NC = 1$).

Thus the mean number of processors used is defined by,

$$\overline{N} = \frac{\varsigma}{T_{exe}} \tag{6.2}$$

where, $T_{exe}$ is the total execution time.

Using the processor-time factor $\varsigma$, we define the processor efficiency factor $\eta$ by the following equation.

$$\eta = \frac{\varsigma^b}{\varsigma^h} = \frac{\overline{N}^b \times T^b_{exe}}{\overline{N}^h \times T^h_{exe}} \tag{6.3}$$

### 6.5.3 Simulation Results

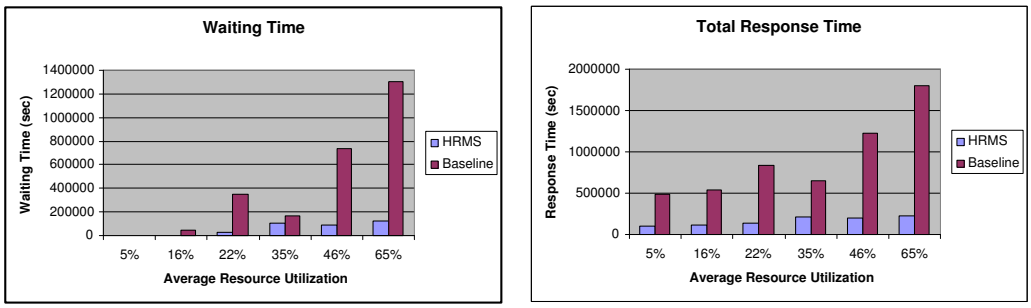


Figure 6.6: Waiting Time and Response Time: HRMS and Baseline Schemes

Figure 6.6 shows the waiting time and response time of the SAMR job with respect to the resource utilization using HRMS and baseline schemes respectively. The average resource utilization is measured for all resource sites with local job arrivals only. The simulation results show that the simple baseline scheme results in large waiting time due to its high requirement for large number of processors. The waiting time increases significantly as the resource utilization increases. While using HRMS scheme, we observe a significant performance boost for the SAMR job due to its adaptive policies taking full advantages of resource-centric and application-centric approaches. Compared to the baseline scheme, HRMS scheme achieves significant speedups.

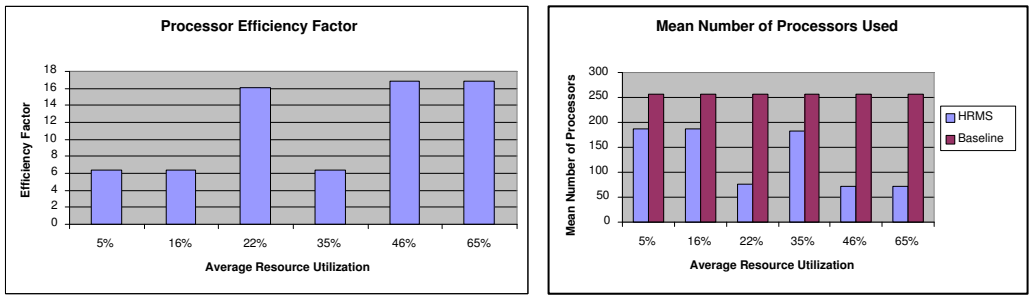


Figure 6.7: Processor Efficiency Factor and Mean Number of Processors Used: HRMS and Baseline Schemes

To demonstrate the resource usage of HRMS and baseline schemes, Figure 6.7 shows the processor efficiency factor and mean number of processors used, which are defined in equations (6.3) and (6.2) respectively. For the baseline scheme, its mean number of processors used is constant, 256 processors, due to its static resource allocation. Compared to the baseline scheme, the mean number of processors used for HRMS scheme is in the range from 70 to 190. One interesting observation is that the mean number of processors used for HRMS does not monotonically increase or decrease with respect to the resource utilization. This is because of the definition of $\overline{N}$ in the equation (6.2). Compared to the baseline scheme, HRMS scheme results in reduction on both the numerator and the denominator of the equation (6.2). As a comparison of these two schemes, the processor efficiency factor ranges from 6 to 17. These simulation results demonstrate the benefits of using HRMS strategies compared to the baseline scheme.

## 6.6 Concluding Remarks

This chapter complements the previous chapters by presenting a performance evaluation on multiple-site supercomputer clusters using the GridMate simulator. The conceptual architecture, scheduling architecture, and detailed operations of GridMate are described. Simulation results confirm our observations from the

real experiments on a single supercomputer cluster: HRMS strategies outperform the baseline scheme by judiciously taking into consideration the space-time heterogeneity.

# Chapter 7
# Summary, Conclusions and Future Work

## 7.1 Summary and Conclusions

This thesis presented the design and evaluation of an adaptive runtime management system and strategy for structured adaptive mesh refinement (SAMR) applications. Because of its ability to reduce computation and storage requirements, the SAMR technique is playing an increasingly important role in modeling and studying complex scientific and engineering phenomena. However, emerging applications keep saturating the available large-scale systems in order to gain insights on complex systems. Parallel and distributed implementations of SAMR have the potential to keep up with the increasing requirements. However, the space-time heterogeneity and dynamism due to adaptation make efficient runtime management of parallel SAMR applications a significant challenge.

To address challenges of managing these dynamic applications, a hybrid space time runtime management strategy (HRMS) framework has been developed. HRMS consists of a number of components: clustering, partitioning, scheduling and hybrid partitiioning strategies. These strategies work synergistically to address SAMR dynamics and heterogeneity. Specifically, the synchronization issues due to the dynamics in parallel SAMR applications have been addressed by the hierarchical partitioning algorithm (HPA) and level-based partitioning algorithm (LPA). Further, the adaptive hierarchical multi-partitioner (AHMP) strategy addresses space-time heterogeneity by identifying and characterizing a hierarchy of clique regions for SAMR applications, selecting the most appropriate partitioner

to partition each clique region, and mapping the clique hierarchy to resources in a hierarchical manner.

The hierarchical partitioning algorithm (HPA) enables the load distribution to reflect the state of the adaptive grid hierarchy. Its goal is to reduce synchronization costs, and enable incremental redistribution and concurrent communication. HPA partitions the computational domain into subdomains and assigns them to hierarchically organized processor groups. To further reduce the synchronization cost, a novel level-based partitioning algorithm (LPA) has been proposed. Most partitioning heuristics merely consider balancing the overall workload among all processors, which can incur significant synchronization cost due to load imbalance at each refinement level. Instead, LPA strives to balance both the overall workload and the workload on each refinement level. As a result, LPA outperforms other partitioning heuristics by reducing the synchronization cost. The combined scheme of HPA and LPA offers performance gains as demonstrated by the experiments presented.

A segmentation-based clustering scheme has been developed to identify and characterize regions (clique regions) with similar requirements. Segmentation-based clustering scheme (SBC) applies segmentation techniques to create well-structured cliques. Since SBC follows the space-filling curve approach, it preserves the locality while maintaining the structure of clique regions.

To exploit the identified clique hierarchy, hierarchical strategies have been developed to partition the clique hierarchy and map partitions to resource groups in a hierarchical manner. It has been observed that there is no single partitioner works well for all cases [79]. Motivated by this observation, an adaptive hierarchical multi-partitioner scheme (AHMP) has been developed to exploit the identified clique hierarchy and dynamically select the most appropriate partitioning algorithm for each clique region. The AHMP scheme extends the HPA scheme and thus also enables incremental redistribution and concurrent communication. As

a result, the overall performance has been improved. It has been experimentally demonstrated that AHMP improves the overall performance on large systems with up to 1280 processors in a supercomputer cluster.

To handle different resource situations, two hybrid strategies have been developed: one is the application-level pipelining scheme (ALP) and the other the application-level out-of-core scheme (ALOC). ALP is applied when resources are sufficient and under-utilized. Basically, ALP scheme combines the domain-based and patch-based partitioning schemes and attempts to overlap the operation on patches of different refinement levels. Experiments show that ALP improves the scalability of SAMR applications. When the available resource (particularly memory) capacity is not sufficient to support the application runtime, an ALOC scheme is used to enhance the survivability. To avoid the immature crash due to peak memory requirement for a short period, ALOC scheme enables incremental operation by keeping only active data patches in the memory and swapping out inactive patches into the disk.

To investigate the applicability of the proposed strategies in Grid environments, GridMate, a Grid simulator for distributed SAMR applications on multi-site clusters, has been designed and implemented following the discrete-event simulation technique. GridMate adopts a super-scheduler and local-scheduler scheduling paradigm and integrates partitioning and scheduling schemes in HRMS. It hence enables the performance evaluation of HRMS strategies in multi-site Grid environments. Simulation results showed promising performance gains using HRMS strategies in Grid environments.

## 7.2 Contributions

Due to the dynamism and space-time heterogeneity of SAMR applications and their complicated communication behavior, it remains a challenging problem to

improve their performance on large systems. This thesis presented a novel strategy, HRMS, that explicitly and successfully addresses the dynamism and heterogeneity.

### 7.2.1 Addressing the Synchronization Costs

By examining the irregular multiple level adaptation and dynamism of SAMR applications, this thesis identified two major sources of synchronization costs that can be the performance bottleneck on large systems. One source of synchronization costs is caused by the organization of the runtime management system. If processors are organized as a flat pool, the required global synchronization will cause significant overheads. To tackle this issue, hierarchical partitioning strategies were presented to organize these processors in a hierarchical manner to match the runtime requirements of SAMR applications. The other source of synchronization costs is due to the irregular locations of refinements at different levels. Most existing solutions seek to balance overall workload assignments among processors, which can cause load imbalance at each refinement level. This thesis proposed LPA seeking to balance both overall workload and workload at each refinement level. It has been experimentally demonstrated that the proposed solutions effectively improve the overall performance.

### 7.2.2 Addressing the Space-Time Heterogeneity

By examining the heterogeneity of SAMR applications, this thesis presented a strategy that explores the localized structures of the domain and matches the most appropriate partitioners to the localized requirements. Due to the heterogeneity, the computation/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. Extending the basic hierarchical

scheme, the proposed AHMP scheme dynamically employs multiple partitioners to different regions of the domain in a hierarchical manner. As a result, AHMP not only enables the incremental redistribution and concurrent communication, but also exploits the best strategy for each local region. In addition, SBC has been proposed to formulate well-structured cliques.

### 7.2.3 Handling Different Resource Situations

The dynamism of SAMR applications can also lead to under-utilized or inadequate resource situations. This thesis presented ALP to handle under-utilized resources and ALOC to handle the situation when resources are insufficient. It experimentally demonstrated ALP improves the performance and scalability. By exploring the memory access pattern of SAMR applications, ALOC meets the requirement to enhance the survivability.

### 7.2.4 Investigating the Applicability of HRMS in Grid Environments

The applicability of the proposed strategies in Grid environments was demonstrated by simulation using the GridMate simulator. Different from other Grid simulators, GridMate considers the coupled requirements of application partitioning and resource scheduling. The simulation demonstrated the performance gain using the proposed strategies.

### 7.2.5 Impact of the Research

The proposed strategies have experimentally demonstrated that adaptive strategies that match the appropriate partitioners with the adaptive grid hierarchy of the computational domain in a hierarchical manner successfully improved the performance and scalability of parallel SAMR applications. The methodology

used in the process of designing HRMS has broad impacts on the design of run-time management strategies for general dynamics applications. Essentially, the methodology first investigates the sources of performance bottleneck, characterizes the computational domain in a finer level, and employs appropriate strategies to attack identified subproblems. This methodology can be readily extended to design run time management strategies for other dynamic parallel applications, such as scientific applications based on adaptive finite-element methods, desktop Grid applications, and parallel cellular automata [35, 59, 73, 82]. Furthermore, since the SAMR technique has been adopted by a large class of scientific and engineering simulations, the improved performance using HRMS ushers in new opportunities for scientists to explore more challenging physical phenomena and reduce the turn-around time of their simulations. The enhanced survivability also enable scientists to zoom in finer details of the underlying physical simulation using limited resources. In addition, the design of GridMate by coupling the application partitioning and resource scheduling introduced a new approach for runtime management. The extension to GridMate supporting wide spectrum of applications is also promising.

## 7.3 Future Work

Based on the demonstrated performance improvement using HRMS strategies, there are enormous opportunities to explore based on and beyond HRMS. We envision three key potential research directions to extend the research presented in this thesis:

- Extension to the proposed partitioning and clustering schemes.
- Extension to the proposed adaptive hierarchical schemes.

- Extension to SAMR techniques by relaxing the synchronization requirements.

### 7.3.1 Extension to Partitioning and Clustering Schemes

This thesis has explored some critical characteristics of SAMR applications. It is intresting to discover other novel partitioning and clustering strategies to further pinpoint the application runtime requirements and enhance the performance. A possible extension to the work on LPA is to design an adaptive LPA scheme that adjusts partitioning policies according to the current requirements of SAMR applications. Specifically, for a SAMR application with the deeply refined domain, the adaptive LPA scheme can apply purely level-based partitioning, bi-level partitioning, or hybrid partitioning that uses LPA for the finest refinement levels and bi-level or triple-level schemes for lower levels. Another potential research direction is to devise judicious prediction strategies that not only predict the resource states but also application requirements based on the fundamental physical interpretation of applications. To support prediction, an interesting research is to explore the feasibility and efficiency of some machine learning techniques, such as artificial neural network and genetic algorithm to enable the guided adaptation through self-learning. Combining the prediction techniques and the proposed clustering schemes can potentially reduce the data migration cost and other communication overheads.

### 7.3.2 Extension to Adaptive Hierarchical Schemes

This thesis presented hierarchical partitioning algorithm (HPA) and adaptive hierarchical multi-partitioner scheme (AHMP). HPA and AHMP improve the overall performance by enabling incremental repartitioning and redistribution and concurrent communication. However, experimental evaluation in this thesis has been

conducted only on a single supercomputer cluster with homogeneous resources. To meet the insatiable computing requirements of the emerging large-scale applications, the Grid systems provide a sustained computing resource. Moreover, Grid systems are naturally organized in a hierarchical manner, which makes adaptive hierarchical schemes very suited to be extended to Grid environments. A potential strategy is to add a group of processors dedicated to handle communication and coordination. As a result, it can reduce the significant overheads involved in the across-site communication by caching, prefetching, and replication techniques, or by enlarging the ghost regions. The research on the application-level out-of-core (ALOC) has been implemented for a single cluster environment using the HDF5 library [10], which supports efficient data compression. To support migration in Grid environments, it is interesting to design an adaptive strategy to select to compress or not compress the data before migration based on the tradeoffs between the communication and computation costs. By relaxing the strict synchronization requirements in SAMR techniques as presented in the next section, the proposed adaptive hierarchical strategies can further improve the performance for the emerging applications.

### 7.3.3 Extension to SAMR Techniques

Parallel SAMR implementations have the potential to accurately model complex physical phenomena [67]. As shown in previous chapters, however, they involve complicated computation and communication patterns and exhibit space-time heterogeneity. Particularly, the requirement of strict synchronization at each refinement level for every iteration causes the performance bottleneck. Although existing research efforts, including the research presented in this thesis, have been committed to improve the performance and scalability of parallel SAMR applications, we believe that a fundamental extension of the original SAMR algorithms is necessary to achieve further substantial improvement. One potential research

direction is to combine SAMR and AIAC (asynchronous iterations-asynchronous communications) [23, 24, 25, 30] algorithms.

Parallel iterative algorithms has been classified into three categories: synchronous iterations-synchronous communications (SISC), synchronous iterations-asynchronous communications (SIAC), and asynchronous iterations-asynchronous communication (AIAC) [22]. In terms of this classification, the SAMR algorithms used in this thesis belong to the SIAC category. Using AIAC algorithms, all processors perform their iterations without considering the progress of other processors. Specifically, at the $i$-th iteration, a processor does not need to wait for the exact solution of the $(i-1)$-th iteration in the ghost regions from its neighbors but proceeds using whatever data available. As a result, the expensive synchronization costs exhibited in SISC and SIAC algorithms are completely eliminated. This unique feature of the AIAC technique makes it very suited to large-scale parallel and distributed implementations. It has been shown that AIAC techniques demonstrate great potential to improve the overall performance of parallel iterative applications in Grid and P2P environments [22, 73]. However, current research using AIAC algorithms focuses on applications based on conventional uniform-discretization numerical methods.

Combining adaptive numerical techniques (e.x. SAMR) and AIAC algorithms offers tremendous potential to further improve the overall performance. In this combined formulation, two challenging issues need to be addressed. One is the theoretical verification or extensive experimental verification of the feasibility of the combined algorithm. And the other is to build an infrastructure to efficiently support such new algorithms in terms of performance and usability. The key issues for the implementation include the following: (1) To support a decentralized convergence detection; (2) To handle decentralized repartitioning and dynamic load balancing; (3) To consider the dynamism, heterogeneity, and fault-tolerance in the large-scale Grid systems. In this new context, the proposed schemes in this

thesis need to be substantially modified to match the new requirements. However, we believe this new research direction provides many opportunities to explore the innovative extension of the proposed schemes and other novel heuristics.

# References


[1] BIRN, Biomedical Informatics Research Network project. URL: http://www.nbirn.net.

[2] Cactus computation toolkit. URL: http://www.cactuscode.org/.

[3] Chombo. URL: http://seesar.lbl.gov/anag/chombo/.

[4] Condor. URL: http://www.cs.wisc.edu/condor.

[5] Earth System Grid. URL: https://www.earthsystemgrid.org/.

[6] European DataGrid. URL: http://eu-datagrid.web.cern.ch/eu-datagrid/.

[7] Globus. URL: http://www.globus.org.

[8] Grace. URL: http://www.caip.rutgers.edu/TASSL/Projects/GrACE/.

[9] GriPhyN, Grid Physics Network project. URL: http://www.griphyn.org/.

[10] Hdf5. URL: http://hdf.ncsa.uiuc.edu/HDF5/.

[11] Information Power Grid. URL: http://www.ipg.nasa.gov/.

[12] International Virtual Observatory Alliance. URL: http://www.ivoa.net/.

[13] IPARS. URL: http://www.cpge.utexas.edu/new_generation/.

[14] Legion. URL: http://legion.virginia.edu/.

[15] PPDG, Particle Physics Data Grid. URL: http://www.ppdg.net.

[16] Seti@home. URL: http://setiathome.ssl.berkeley.edu.

[17] Simjava. URL: http://www.dcs.ed.ac.uk/home/hase/simjava/.

[18] TeraGrid. URL: http://www.teragrid.org/.

[19] Vampire. URL: http://www.tdb.uu.se/~johans/research/vampire/vampire1.html.

[20] G. Allen, T. Dramlitsch, I. Foster, N. T. Karonis, M. Ripeanu, E. Seidel, and B. Toonen. Supporting efficient execution in heterogeneous distributed computing environments with cactus and globus. In *The 2001 ACM/IEEE Conference on Supercomputing (CDROM)*, pages 52 – 52, Denver, Colorado, 2001.

[21] R. Bagrodia, R. Meyer, M. Takai, Y. Chen, X. Zeng, J. Martin, B. Park, and H. Song. Parsec: A parallel simulation environment for complex systems. *IEEE Computer*, 31(10):77–85, 1998. URL: `http://pcl.cs.ucla.edu/projects/parsec/`.

[22] J. M. Bahi, S. Contassot-Vivier, and R. Couturier. Coupling dynamic load balancing with asynchronism in iterative algorithms on the computational grid. In *Parallel and Distributed Processing Symposium, 2003. Proceedings. International*, pages 4– 13, 2003.

[23] J. M. Bahi, S. Contassot-Vivier, and R. Couturier. Dynamic load balancing and efficient load estimators for asynchronous iterative algorithms. *Parallel and Distributed Systems, IEEE Transactions on*, 16(4):289–299, 2005.

[24] J. M. Bahi, S. Contassot-Vivier, R. Couturier, and F. Vernier. A decentralized convergence detection algorithm for asynchronous parallel iterative algorithms. *Parallel and Distributed Systems, IEEE Transactions on*, 16(1):4–13, 2005.

[25] G. M. Baudet. Asynchronous iterative methods for multiprocessors. *Journal of ACM*, 25:226–244, 1978.

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

[27] M. Berger and I. Regoutsos. An algorithm for point clustering and grid generation. , 21(5):, 1991. *IEEE Transactions on Systems, Man and Cybernetics*, 21(5):1278–1286, 1991.

[28] F. Berman, G. Fox, and A. J. G. Hey, editors. *Grid Computing: Making the Global Infrastructure a Reality*. Wiley Publisher, April 2003.

[29] F. Berman, R. Wolski, H. Casanova, W. Cirne, H. Dail, M. Faerman, S. Figueira, J. Hayes, G. Obertelli, J. Schopf, G. Shao, S. Smallen, S. Spring, A. Su, and D. Zagorodnov. Adaptive Computing on the Grid Using AppLeS. *IEEE Transactions on Parallel and Distributed Systems*, 14(5):369–382, 2003.

[30] D. P. Bertsekas and J. N. Tsitsiklis. *Parallel and Distributed Computation: Numerical Methods*. Prentice Hall, 1989.

[31] S. Bokhari. *Assignment Problems in Parallel and Distributed Computing*. Kluwer Academic Publishers, Boston, Massachusetts, 1987.

[32] S. Bokhari, T. W. Crockett, and D. M. Nicol. Binary dissection: Variants and applications. Technical Report TR-97-29, 1997. URL: "citeseer.nj.nec.com/bokhari97binary.html".

[33] L. Breslau, D. Estrin, K Fall, S Floyd, J Heidermann, A Helmy, P Huang, S McCanne, K Varadhan, Y Xu, and H. Yu. Advances in network simulation. *IEEE Computer*, 33(5):5967, 2000.

[34] R. Buyya and M. Murshed. Gridsim: A toolkit for the modeling and simulation of distributed resource management and scheduling for grid computing. *Concurrency and Computation: Practice and Experience*, 14(13-15):11751220, 2002.

[35] C. R. Calidonna, C. D. Napoli, M. Giordano, M. M. Furnari, and S. D. Gregorio. A network of cellular automata for a landslide simulation. In *15th International Conference on Supercomputing*, pages 419 – 426, Sorrento, Italy, 2001.

[36] S. Chandra. Armada: A framework for adaptive application-sensitive runtime management of dynamic applications. Technical report, Rutgers University, 2002.

[37] S. Chandra, X. Li, and M. Parashar. Engineering an autonomic partitioning framework for grid-based samr applications. In L. T. Yang, editor, *Hardware/Software Support for Parallel and Distributed Scientific and Engineering Computing*. Kluwer Academic Publishers, September 2003.

[38] S. Chandra and M. Parashar. Armada: An adaptive application-sensitive partitioning framework for structured adaptive mesh refinement applications. In *IASTED International Conference on Parallel and Distributed Computing Systems (PDCS 02)*, pages 446 – 451, Cambridge, MA, 2002. ACTA Press.

[39] S. Chandra, J. Steensland, M. Parashar, and J. Cummings. An experimental study of adaptive application sensitive partitioning strategies for samr applications. In *2nd Los Alamos Computer Science Institute Symposium (also Best Research Poster at Supercomputing Conference 2001)*, 2001.

[40] J. Chen and V. Taylor. Mesh partitioning for efficient use of distributed systems. *IEEE Transactions on Parallel and Distributed Systems*, 13(1):67–79, 2002.

[41] 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.

[42] K. Czajkowski, I. Foster, N. Karonis, C. Kesselman, S. Martin, W. Smith, and S. Tuecke. A resource management architecture for metacomputing systems. In *Lecture Notes in Computer Science*, volume 1459, 1998.

[43] K. Devine, E. Boman, R. Heaphy, B. Hendrickson, and C. Vaughan. Zoltan data management services for parallel dynamic applications. *Computing in Science and Engineering*, 4(2):90–97, 2002.

[44] D. G. Feitelson. A survey of scheduling in multiprogrammed parallel systems. Technical report, IBM Research Report RC19790(87657), 1995.

[45] I. Foster and C. Kesselman. *The Grid: Blueprint for a New Computing Infrastructure*. Morgan Kaufmann, 2nd edition, 2004.

[46] I. Foster, C. Kesselman, J. Nick, and S. Tuecke. The physiology of the grid: An open grid services architecture for distributed systems integration, open grid service infrastructure wg, global grid forum, June 2002.

[47] I. Foster, C. Kesselman, and S. Tuecke. The anatomy of the grid: Enabling scalable virtual organizations. *International Journal of High Performance Computing Applications*, 15(3):200–222, 2001.

[48] I. Foster, Z. Kesselman, C. Lee, B. Lindqll, K. Nahrstedt, and A. Roy. A distributed resource management architecture that supports advance reservations and co-allocation. In *Seventh International Workshop on Quality of Service (IWQoS '99)*, 1999.

[49] G. Gilder, editor. *Gilders law on network performance. Telecosm: The World After Bandwidth Abundance*. Touchstone Books, 2002.

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

[51] S. Hawley and M. Choptuik. Boson stars driven to the brink of black hole formation. *Physical Review D*, 62:10(104024), 2000.

[52] B. Hendrickson and R. Leland. A multilevel algorithm for partitioning graphs. In *Supercomputing*, San Diego, 1995.

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

[54] 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.

[55] L. V. Kale. Charm. URL: `http://charm.cs.uiuc.edu/research/charm/`.

[56] G. Karypis. Parmetis, 2003. URL: `http://www-users.cs.umn.edu/~karypis/metis/parmetis/index.html`.

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

[58] A. Legrand, L. Marchal, and H. Casanova. Scheduling distributed applications: The simgrid simulation framework. In *the 3rd IEEE/ACM International Symposium on Cluster Computing and the Grid (CCGrid2003)*, Tokyo, Japan, May, 2003. IEEE Computer Society Press.

[59] T. J. Lehman and J. H. Kaufman. Optimalgrid: middleware for automatic deployment of distributed fem problems on an internet-based computing grid. In *IEEE International Conference on Cluster Computing*, pages 164–171, 2003.

[60] 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)*, Klagenfurt, Austria, 2003.

[61] X. Li and M. Parashar. Hierarchical partitioning techniques for structured adaptive mesh refinement applications. *The Journal of Supercomputing*, 28(3):265 – 278, 2004.

[62] C. Liu, L. Yang, I. Foster, and D. Angulo. Design and evaluation of a resource selection framework for grid applications. In *11 th IEEE International Symposium on High Performance Distributed Computing HPDC-11 20002 (HPDC'02)*, 2002.

[63] X. Liu, J. Liu, J. Eker, and E. A. Lee. Heterogeneous modeling and design of control systems. In T. Samad and G. Balas, editors, *Software-Enabled Control: Information Technology for Dynamical Systems*. IEEE Press, New York, 2003.

[64] U. Lublin and D. G. Feitelson. The workload on parallel supercomputers: Modeling the characteristics of rigid jobs. *Journal of Parallel and Distributed Computing*, 63(11):1105–1122, 2003.

[65] P. MacNeice. Paramesh, 1999. URL: `http://esdcd.gsfc.nasa.gov/ESS/macneice/paramesh/paramesh.html`.

[66] G. Moore. Moores Law, 1965. URL: `http://www.intel.com/research/silicon/mooreslaw.htm`.

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

[68] J. Pilkington and S. Baden. Dynamic partitioning of non-uniform structured workloads with spacefilling curves. *IEEE Transactions on Parallel and Distributed Systems*, 7(3), 1996.

[69] 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. In *(to be published in) Proc. Combust. Inst.*

[70] 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*, 25(1):219–226, 2000.

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

[72] R. Samtaney. Rm2d. URL: `http://www.galcit.caltech.edu/~ravi/rm.html`.

[73] K. Sankaralingam, S. Sethumadhavan, and J.C. Browne. Distributed pagerank for p2p systems. In *12th IEEE International Symposium on High Performance Distributed Computing*, pages 58–68, 2003.

[74] K. Schloegel, G. Karypis, and V. Kumar. A unified algorithm for load-balancing adaptive scientific simulations. In *Supercomputing*, 2000.

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

[76] H.J. Song, X. Liu, D. Jakobsen, R. Bhagwan, X. Zhang, K. Taura, and A. Chien. The microgrid: A scientific tool for modeling computational grids. In *IEEE Supercomputing (SC2000)*, Dallas, TX, November 2000. IEEE Computer Society Press.

[77] J. Steensland. *Efficient Partitioning of Structured Dynamic Grid Hierarchies*. PhD thesis, Uppsala University, 2002.

[78] J. Steensland. Irregular buffer zone partitioning reducing synchronization cost in samr. In *The 6th Workshop on Parallel and Distributed Scientific and Engineering Computing (PDSEC-05) held in conjunction with The 19th International Parallel and Distributed Processing Symposium (IPDPS-05)*, 2005.

[79] 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*, 13(12):1275–1289, 2002.

[80] J. Steensland, M. Thune, S. Chandra, and M. Parashar. Towards an adaptive meta-partitioner for parallel samr applications. In *IASTED PDCS 2000*, 2000.

[81] J. Steensland, M. Thune, S. Chandra, and M. Parashar. Towards an adaptive meta-partitioner for parallel samr applications. In *IASTED PDCS 2000*, 2000.

[82] D. Talia. Parallel cellular programming for developing massively parallel emergent systems. In *International Parallel and Distributed Processing Symposium*, pages 22–26, 2003.

[83] B. Veeravalli, D. Ghose, V. Mani, and T.G. Robertazzi. *Scheduling Divisible Loads in Parallel and Distributed Systems*. IEEE Computer Society Press, Los Almitos, California, 1996.

[84] B. Wilkinson and M. Allen. *Parallel Programming: Techniques and Applications Using Networked Workstations and Parallel Computers*. Pearson Education, first edition, 1999.

[85] R. Wolski, N. T. Spring, and J. Hayes. The network weather service: a distributed resource performance forecasting service for metacomputing. *Future Generation Computer Systems*, 15(5-6):757–768, 1999.

[86] B. P. Zeigler, S. B. Hall, and H. S. Sarjoughian. Exploiting hla and devs to promote interoperability and reuse in lockheed's corporate environment. *SIMULATION, Special Issue on The High Level Architecture.*, 73(5):288–295, 1999.

# Appendix A

# Glossary

AHMP: Adaptive hierarchical multi-partitioner

AHPA: Adaptive hierarchical partitioning algorithm

AIS: Application information service

ALOC: Application-level out-of-core

ALP: Application-level pipelining

BPA: Bin-packing partitioning algorithm

CGDS: Composite grid distribution strategy

GIS: Grid information service

GMISP+SP: geometric multilevel + sequence partitioning

GPA: Greedy partitioning algorithm

GUL: Grid unit list

HPA: Hierarchical partitioning algorithm

HRMS: Hybrid space-time runtime management strategy

LBC: Level-based clustering algorithm,

pBD+ISP: $p$-way binary dissection algorithm

PDE: Partial differential equation

PDS: Parallel and distributed system

RM: Runtime manager

RMS: Runtime management system

SAMR: Structured adaptive mesh refinement

SBC: Segmentation-based clustering

SFC: Space-filling curve

SHPA: Static hierarchical partitioning algorithm

SPMD: Single program multiple data

T-VCU: Temporal virtual computational unit

# Curriculum Vita

## Xiaolin Li

**2005** PhD, Electrical & Computer Engineering, Rutgers University, USA.

**2001** PhD, Electrical & Computer Engineering, National University of Singapore, Singapore

**1998** MEng, Mechanical & Automation Engineering, Zhejiang University, PRC

**1995** BEng, Mechanical & Electronic Engineering, Qingdao University, PRC

**2001-2005** Graduate Research Assistant, The Applied Software Systems Lab, Center for Advance Information Processing, Rutgers University, USA

**2003-2003** Extreme Blue Intern, Extreme Blue Program, IBM Austin, USA

**2001-2001** Staff R&D Engineer, Mobile Computing and Protocols Group, Center for Wireless Communications, Singapore

**1999-2000** Teaching Assistant, Department of Electrical & Computer Engineering, National University of Singapore

**1998-2001** Research Scholar, Open Source Software Lab and Digital Systems & Applications Lab, National University of Singapore, Singapore

**1995-1998** Research Assistant, National Key Lab of CAD & CG and Modern Design Methodology Lab, Zhejiang University, PRC

### Publications

X. Li and M. Parashar, "Using Clustering to Address the Heterogeneity and Dynamism in Parallel SAMR Application", (accepted) *12th Annual IEEE International Conference on High Performance Computing (HiPC-2005).*

S. Chandra, X. Li, T. Saif and M. Parashar, "Enabling Scalable Parallel Implementations of Structured Adaptive Mesh Refinement Applications", (under revision) *Journal of Supercomputing*, Kluwer Academic Publishers.

S. Chandra, X. Li, T. Saif and M. Parashar, "Addressing the Scalability of Distributed Structured Adaptive Mesh Refinement", (submitted to) *Computing and Visualization in Science*, Springer-Verlag.

X. Li and M. Parashar, "Adaptive Runtime Management of Spatial and Temporal Heterogeneity for Dynamic Grid Applications", *Proceedings of the 13th High Performance Computing Symposium (HPC-2005)*, San Diego, California, pp. 223-228, Apr. 2005.

X. Li, B. Veeravalli, and C.C. Ko, "Distributed Image Processing in a Network of Workstations", *International Journal of Computers and Applications*, ACTA Press, Vol. 25 (2), pp. 136-145, 2003.

X. Li, and M. Parashar , "Hierarchical Partitioning Techniques for Structured Adaptive Mesh Refinement Applications", *Journal of Supercomputing*, Kluwer Academic Publishers, Vol.28(3), pp.265-278, 2004.

S. Chandra, X. Li and M. Parashar, "Engineering an Autonomic Partitioning Framework for Grid-based SAMR Applications", Book chapter in "Hardware/Software Support for Parallel and Distributed Scientific Engineering Computing", Editor: L. T. Yang, Kluwer Academic Publishers, Sep. 2003.

X. Li and M. Parashar, "Dynamic Load Partitioning Strategies for Managing Data of Space and Time Heterogeneity in Parallel SAMR Applications", *Lecture Notes in Computer Science (EuroPar-2003)*, Editors: H. Kosch, L. Boszormenyi, H. Hellwagner, Springer-Verlag, Klagenfurt, Austria, Vol. 2790, pp.181-188, Aug. 2003.

X. Li, S. Ramanathan, and M. Parashar, "Hierarchical Partitioning Techniques for Structured Adaptive Mesh Refinement (SAMR) Applications", *Proceedings of International Conference on Parallel Processing (ICPP-2002), HPSECA Workshop*, Vancouver, Canada, Aug. 2002.

X. Li, B. Veeravalli, and C.C. Ko, "Divisible Load Scheduling in a Hypercube Cluster with Finite-size Buffers and Granularity Constraints", *Proceedings of the First IEEE/ACM International Symposium on Cluster Computing and the Grid (CCGrid-2001)*, Brisbane, Australia, pp. 660-667, May 2001.

B. Veeravalli, X. Li, and C.C. Ko, "On the Influence of Start-up Costs in Scheduling Divisible Loads on Bus Networks", *IEEE Transactions*

*on Parallel and Distributed Systems*, Vol.11 (12), pp.1288-1305, Dec. 2000.

X. Li, B. Veeravalli, and C.C. Ko, "Divisible Load Scheduling on Single-level Tree Networks with Finite-size Buffers", *IEEE Transactions on Aerospace and Electronic Systems*, Vol. 36 (4), pp. 1298-1308, Oct. 2000.