A MULTIOBJECTIVE APPROACH FOR ADDRESSING DYNAMISM AND HETEROGENEITY IN PARALLEL SCIENTIFIC SIMULATIONS

BY SUMIR CHANDRA

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

May, 2007

ABSTRACT OF THE DISSERTATION

A Multiobjective Approach for Addressing Dynamism and Heterogeneity in Parallel Scientific Simulations

by SUMIR CHANDRA

Dissertation Director:

Professor Manish Parashar

Scientific simulations offer the potential for accurate solutions of realistic models of complex physical phenomena. These simulations are based on systems of partial differential equations and are playing an increasingly important role in science and engineering. However, the phenomena underlying scientific simulations are inherently multi-phased, have heterogeneous state, and span multiple time and space scales. The resulting dynamism coupled with the spatiotemporal and computational heterogeneity make parallel implementations of these scientific simulations extremely challenging. Key issues that need to be addressed include algorithmic efficiency, load balancing, coordination and performance management, which lead to conflicting objectives and trade-offs at runtime. In cases when analytical approaches are not feasible, this requires an understanding of application and system characteristics, and the impact of dynamism and heterogeneity on simulation performance.

The overarching goal of this research is to enable large-scale investigative and exploratory science using high-performance "smart" simulations. The key innovation in this research is a multiobjective approach that provides several distribution, coordination, and adaptation strategies to address dynamism and heterogeneity. The partitioning schemes perform dynamic domain decomposition based on either the application geometry or load characteristics to address spatiotemporal and computational heterogeneity. The synchronization algorithm improves communication overheads by reducing messaging frequency in favor of additional computation, when the application is communication-dominated. A runtime infrastructure integrates these strategies and supports the efficient and scalable execution of parallel scientific simulations. Experimental evaluation of the presented strategies using simulations from several application domains demonstrate improvement in overall performance on large systems.

Acknowledgements

I wish to express my sincere gratitude to my advisor, Dr. Manish Parashar, for his invaluable guidance, support and encouragement during the course of this research and throughout my graduate studies at Rutgers University. I am thankful to Dr. Ivan Marsic, Dr. Deborah Silver, Dr. Yanyong Zhang (all at Rutgers University, NJ), and Dr. Jaideep Ray (Sandia National Laboratories, CA) for being on my dissertation committee and for their advice and suggestions.

I thank Dr. Salim Hariri (University of Arizona, AZ), Dr. Xiaolin Li (Oklahoma State University, OK), Dr. Jaideep Ray and Dr. Johan Steensland (Sandia National Laboratories, CA) for valuable research discussions and collaboration related to this research. I thank Dr. Julian Cummings (California Institute of Technology, CA), Dr. Ravi Samtaney (Princeton Plasma Physics Laboratory, NJ), and Dr. Mary Wheeler (University of Texas at Austin, TX) for collaboration on application codes for real-world SAMR simulations. I thank Dr. John Hewson (Sandia National Laboratories, NM) for his help with the methane-air chemical mechanism and generating a self-supporting chemistry module used in this research.

I am grateful to the CAIP technical and administrative staff at Rutgers University for their assistance and support. I thank my colleagues at The Applied Software Systems Laboratory (TASSL) for their cooperation and a wonderful work environment. I thank all my friends and relatives for their support and words of encouragement.

Above all, this dissertation would not have been possible without my family's love, patience and sacrifices. My immediate family to whom this dissertation is dedicated to, has been a constant source of love, concern, support and strength all these years. I wish to express my heartfelt gratitude to my family. My father is my role model and

counsel, and his diligence, pragmatism and dedication to family and work have always been a source of inspiration. My mother's love, selflessness and unequivocal support have always given me comfort and strength. My sister's encouragement, enthusiasm and friendship have been invaluable to me. Last, but not the least, I am thankful to my better half, my wife, whose love, support and patience have been instrumental in the successful completion of my dissertation.

Finally, the research presented in this dissertation was supported in part by the National Science Foundation via grants numbers ACI 9984357, EIA 0103674, EIA 0120934, ANI 0335244, CNS 0305495, CNS 0426354 and IIS 0430826, and by the US Department of Energy via the grant number DE-FG02-06ER54857. I thank these funding agencies for their generous financial support.

Dedication

To my family.

Table of Contents

List of Tables

List of Figures

Chapter 1
Introduction

1.1 Motivation

1.1.1 Large-Scale Simulations for Computational Science

Realistic models of complex physical phenomena are generally structured as systems of partial differential equations (PDEs), which can be solved accurately using scientific simulations. Scientific simulations are playing an increasingly important role in science and engineering. Large-scale parallel simulations offer the potential for dramatic insights into physical systems such as interacting black holes and neutron stars, formations of galaxies, subsurface flows in oil reservoirs and aquifers, and dynamic response of materials to detonation.

Parallel implementations of scientific simulations typically partition the application domain into blocks, enabling processors to perform local computations on these blocks in parallel followed by periodic synchronization. Furthermore, structured grids usually employ regular data structures that are easier to partition and lead to regular access and communication patterns. Consequently, structured formulations of parallel scientific simulations can result in relatively simpler and efficient implementations. Structured grid techniques can be implemented using either uniform mesh (unigrid) [78] or structured adaptive mesh refinement (SAMR) [7] [10] [129] schemes. GrACE [94], SAMRAI [68], Chombo [37], and PARAMESH [76], are examples of structured grid infrastructures that support adaptive implementations of parallel scientific simulations. Simulations on structured grids have been effectively used to solve complex

systems of PDEs in various scientific and engineering application domains, including computational fluid dynamics [2] [9] [101], numerical relativity [34] [55], astrophysics [3] [23] [86], subsurface modeling and oil reservoirs [98] [133], and combustion [107].

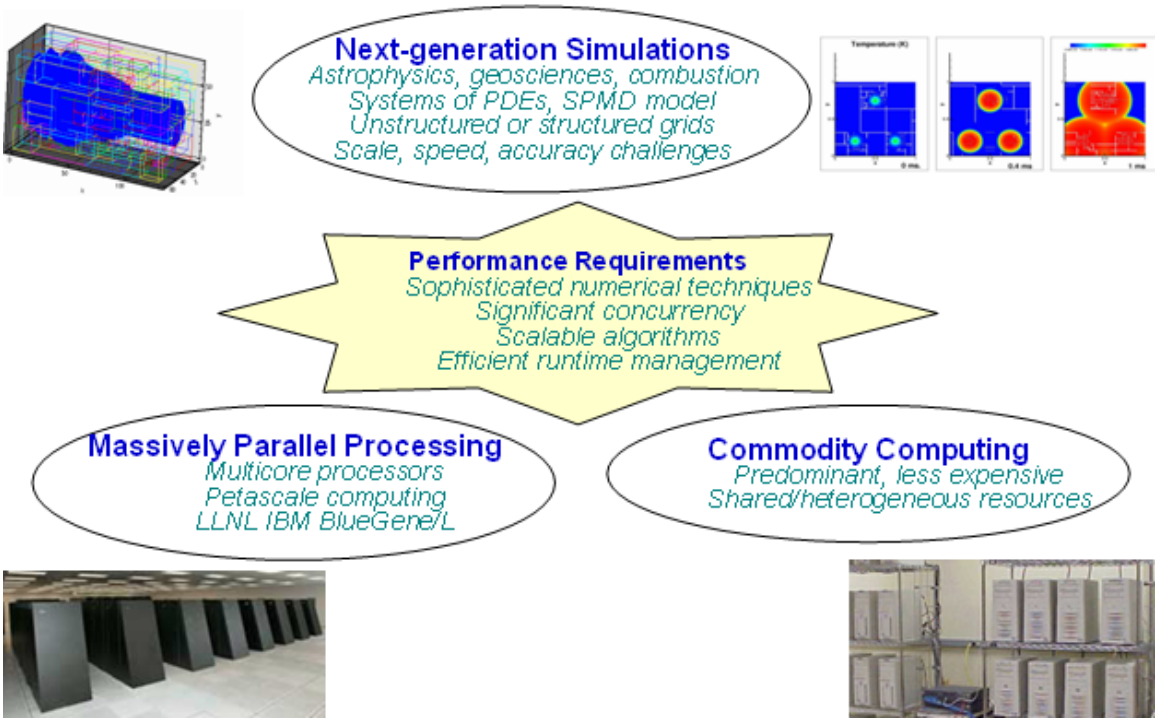

Figure 1.1: Trends and requirements for computational science simulations.

Due to the constant need for increased computational accuracy, scientific simulations can greatly benefit from advances in computing systems. Figure 1.1 illustrates the trends and requirements for enabling computational science simulations on massively parallel processing (MPP) and commodity computing systems, which are described in the following sections.

1.1.2 Architectural Trends and Challenges in High Performance Computing

Over the past decade, processor and networking technologies have demonstrated rapid progress, driven by Moore's Law [82] and Gilder's Law [50] respectively, which have enabled parallel and distributed systems of unprecedented scales. As illustrated in

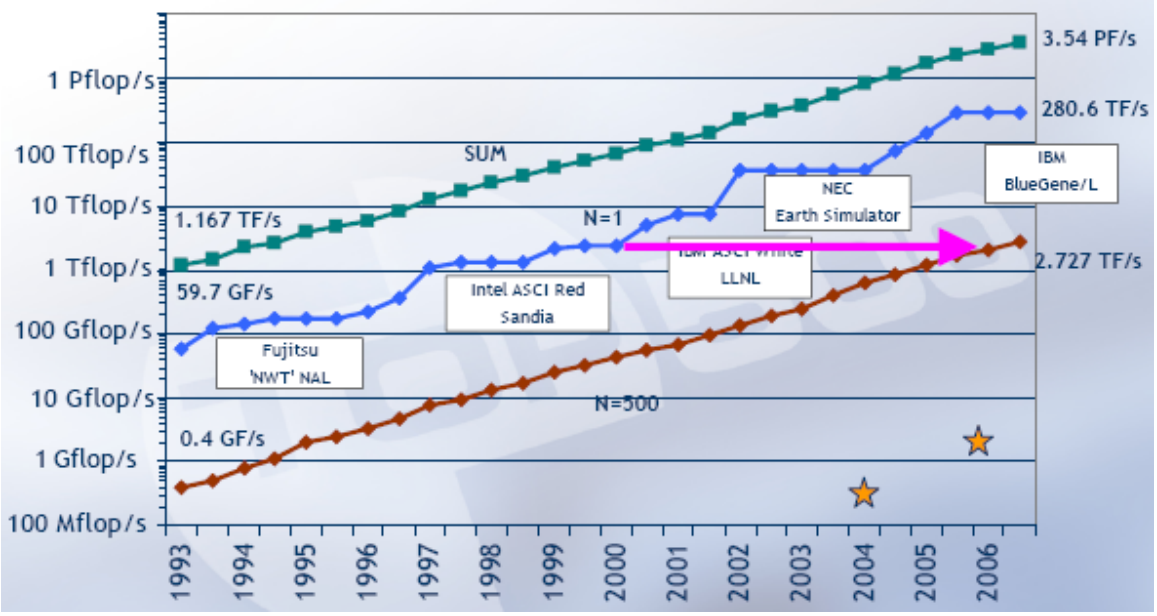

Figure 1.2: Performance development timeline for the Top 500 computing systems in the world as of November 2006 [130].

Figure 1.2 [130], the IBM BlueGene/L [19] supercomputer at the Lawrence Livermore National Laboratory has demonstrated a sustained Linpack performance of 280.6 teraflops (TFlops), and is ranked first on the 28th edition (November 2006) of the TOP500 [132] list of the most powerful commercially available computer systems in the world. The BlueGene/L system boasts a peak speed of over 360 TFlops, a total memory of 32 terabytes (TB), total power of 1.5 megawatts (MW), and machine floor space of 2,500 square feet [19]. The full system, comprising 131,072 700 MHz processors, has 65,536 dual-processor compute nodes that are configured as a $32 \times 32 \times 64$ 3D torus such that each node is connected in six different directions for nearest-neighbor communications. Multiple communications networks enable extreme application scaling and 1024 gigabit-per-second (Gbps) links to a global parallel file system support fast input/output to disk. However, realizing efficient performance for challenging scientific simulations on such a machine is non-trivial.

Furthermore, initiatives have recently been undertaken towards creating a petascale computing environment in the near future for advancing a broad range of science

and engineering. These next-generation computing systems offer the potential for new scientific breakthroughs [43] such as analyzing the structure and function of complex biological molecules, deciphering the origins of the universe and creation of matter, predicting global climatic changes, designing more fuel-efficient and environment-friendly automobiles and aircraft, and understanding the origin, spread and mitigation of contagious diseases. However, harnessing high-performance scientific simulations on computing resources of such scale require sophisticated numerical techniques with significant concurrency, scalable algorithms, and above all, efficient runtime management.

1.1.3 Architectural Trends and Challenges in Commodity Computing

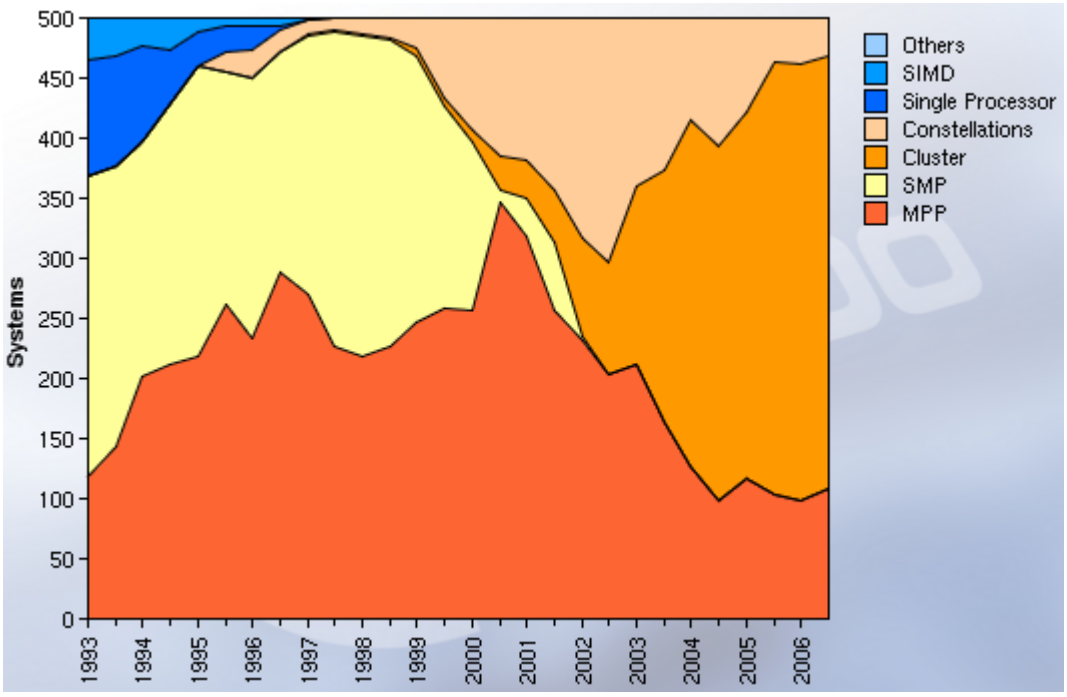

Figure 1.3: Landscape of system architectures for the TOP500 computing systems in the world as of November 2006 [130].

Cluster computing is the predominant, and generally less expensive, execution paradigm in current parallel and distributed systems. Figure 1.3 shows the landscape

of various system architectures [130] that constitute the TOP500 computing systems in the world. Clusters comprise 72.2% of the systems in the 28th TOP500, while 21.6% are massively parallel processing (MPP) systems and the remaining 6.2% are constellations. The 500th ranked system in the TOP500 List is a BL-20P Blade Cluster, comprising 800 Pentium4 Xeon 3.06 GHz processors with a Gigabit Ethernet interconnect, which demonstrates 2.73 TFlops Linpack performance [132]. In contrast, Frea is a 64 node Linux Beowulf symmetric multiprocessing (SMP) cluster at Rutgers University. The nodes of the cluster are 1.7 GHz Pentium4 processors with 512MB main memory, and are interconnected by 100 Mbps Fast Ethernet. Scientific simulations on clusters may encounter shared resources, slower or heterogeneous interconnects, and lower overall performance as compared to dedicated MPP systems. An analysis of the underlying system constraints becomes imperative to ensure efficient performance in such heterogeneous networked environments.

1.2 Problem Description

The performance of parallel scientific simulations can be affected by several factors at runtime. The key challenges include complexity, dynamism and heterogeneity, which are discussed in the following section.

1.2.1 Software Complexity

The complex physical phenomena being modeled by scientific simulations are inherently multi-phased, dynamic, and heterogeneous (in time, space, and state), requiring large numbers of software components. For example, a scientific application modeling combustion [72] can include a linear algebraic or ordinary differential equation (ODE) solver, chemistry integrator, diffusion/physics integrator, time interpolator, error estimator and statistical components, as well as various services/tools for mesh management, discretization, parallel data description and redistribution, performance

evaluation, and visualization. The large numbers of software components present significant challenges in managing the runtime performance of complex scientific simulations.

1.2.2 Dynamics and Spatiotemporal Heterogeneity

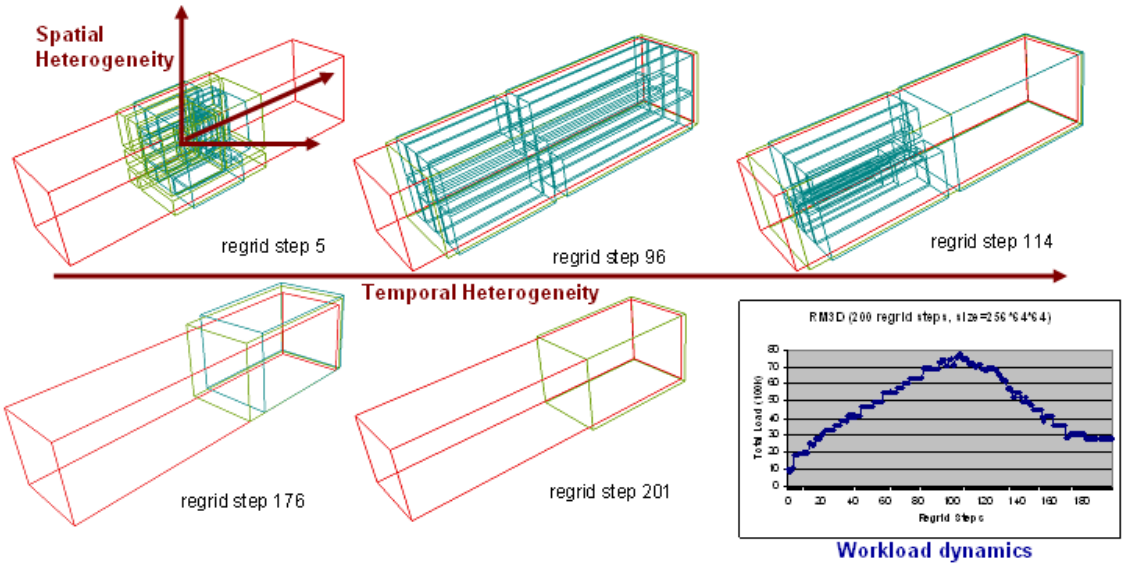

Figure 1.4: Dynamism and spatiotemporal heterogeneity exhibited by an illustrative 3-D compressible turbulence (RM3D) simulation.

The interactions between components in scientific simulations can span several different space and time scales, and the application runtime behavior can change dynamically over time. As an example, Figure 1.4 shows a sequence of snapshots of an illustrative 3-D compressible turbulence (RM3D) SAMR simulation with $256 \times 64 \times 64$ resolution. Changes in RM3D application physics create dynamically varying simulation total workloads, ranging from nearly 3×10^6 at redistribution/regridding step 20 to approximately 7.5×10^6 at regridding step 100. Note that the load at each grid point is assumed to be uniform. The peak total workload is about 8 times larger than the minimum total workload and over two times larger than the average total workload for this simulation. The RM3D application serves as a representative of the class of simulations that exhibit significant dynamism and spatiotemporal heterogeneity,

and is discussed in greater detail in Chapter 4. Dynamism and spatiotemporal heterogeneity in parallel scientific simulations lead to significant challenges in partitioning, load balancing and synchronization.

1.2.3 Computational Heterogeneity

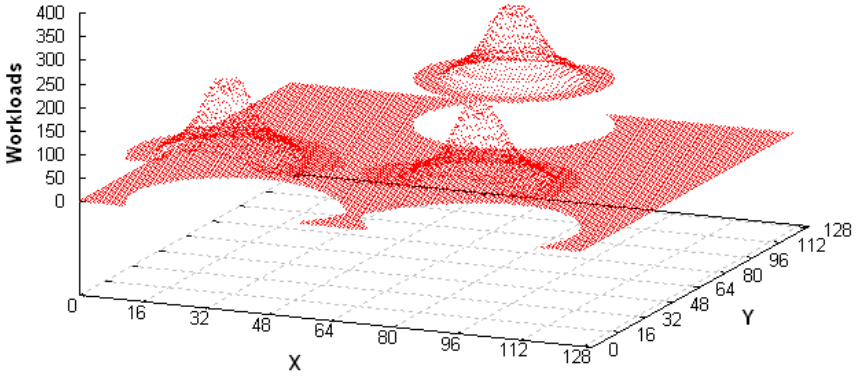

Figure 1.5: Computational heterogeneity exhibited by an illustrative 2-D methane-air combustion (R-D) simulation.

Certain types of scientific applications, such as combustive reacting flows, can change their mathematical and topological characteristics during the simulation to resolve the underlying phenomena. These applications exhibit varying degrees of computational heterogeneity at runtime, generally manifested as pointwise varying workloads. As an example, Figure 1.5 shows a sample distribution of the heterogeneous computational workloads for an illustrative 2-D methane-air combustion (R-D) kernel with 3 hot-spots on a 128×128 structured grid. The reactive processes near the flame fronts have high computational requirements that correspond to large values of workloads ranging from 100 to nearly 250, while the diffusive processes have uniform loads with a value of 1. Efficient partitioning and load balancing for parallel implementations of such simulations are quite challenging due to the computational heterogeneity, and are discussed in greater detail in Chapter 5.

1.2.4 Impact on Runtime Performance

The inherent dynamism coupled with the runtime heterogeneity in parallel scientific simulations lead to significant challenges in ensuring algorithmic efficiency, load balancing, and runtime performance management [30]. Critical among these is the partitioning of the application grid hierarchy to balance load, optimize communication and synchronization, minimize data migration costs, and maximize grid quality (e.g., aspect ratio) and available parallelism. These performance and scalability challenges will be further exacerbated by the advent of petascale and multicore computing that envisage scientific simulations at unprecedented scales in the near future. Moreover, in the case of scientific simulations on heterogeneous clusters where processor speeds typically overshadow network speeds, optimizing communication between simulation components based on the system constraints can lead to efficient performance.

Addressing the runtime challenges of parallel scientific simulations to obtain optimal performance, often, lead to conflicting objectives and trade-offs, which are non-trivial [123] to identify and resolve. In cases where a formal analytical framework exists, e.g., error estimation via Richardson extrapolation [11] or the solution of linear systems with robustness and speed trade-offs [14] [15] [16] [79], optimal behavior can be predicted purely using analytical rigor. However, this is not always the case, and an empirical approach based on runtime calibration and adaptation is the only recourse for predicting performance when analysis of the system is not feasible. This requires an understanding of application characteristics, system constraints, and the impact of heterogeneity on algorithmic, component and overall simulation performance.

1.3 Problem Statement

The complexity, heterogeneity, and dynamism associated with scientific applications has made current programming environments and infrastructure unmanageable and

insecure, and has led researchers to consider alternative programming paradigms and management techniques that are based on strategies used by biological systems. Systems based on this approach, known as autonomic computing [65], have the capabilities of being self-defining, self-healing, self-configuring, self-optimizing, self-protecting, contextually aware, and open.

In this research, we present a multiobjective approach that analyzes application and system state, and provides appropriate distribution, configuration, coordination, and adaptation strategies for simulations on structured grids. Such an autonomic infrastructure formulation can address dynamism and heterogeneity in parallel scientific simulations, and enable their efficient and scalable execution on high-performance or cluster computing architectures. The specific research objectives are as follows.

1.3.1 Application and System Characterization

As discussed in Sections 1.1 and 1.2, the runtime behavior of the scientific application and the underlying system can significantly impact overall simulation performance. Moreover, analytical performance models are generally not feasible for these large-scale simulations. In such cases, an empirical understanding of the application and system dynamism and heterogeneity is critical for analyzing current performance and selecting appropriate adaptations. Addressing this research issue of application and system characterization involves the following tasks.

- Analyze the various characteristics of parallel scientific simulations such as domain features, application locality, computational costs, synchronization behavior, and redistribution overheads.

- Analyze the application synchronization overheads as well as system constraints such as bandwidth to determine the communication costs relative to local computations.

1.3.2 Addressing Dynamism and Heterogeneity

Sections 1.2.2 and 1.2.3 briefly describe the dynamism and runtime heterogeneity typically associated with parallel scientific simulations. Analyzing the cause and effect of these challenges can provide insights into the investigation of smarter/hybrid algorithms and runtime adaptations. For example, the load balancing algorithm may be based on the structure of the problem domain if the formulation is quite homogeneous, or on the simulation workload profile if there is sufficient computational heterogeneity. Different adaptation strategies need to be designed and evaluated. The research tasks required for addressing dynamism and heterogeneity are listed as follows.

- Investigate strategies that maintain application locality, provide good load balance, or reduce synchronization overheads to address dynamism and spatiotemporal heterogeneity.
- Develop partitioning strategies that consider the pointwise varying workloads and perform in-situ global load balancing to address the computational heterogeneity for parallel uniform and adaptive scientific applications.
- Design hybrid schemes that employ runtime calibration to analyze the impact of heterogeneity on load distribution and performance for different application compositions, and use this feedback towards appropriate algorithm selection for domain decomposition.
- Adapt simulation performance by relaxing the synchronization requirements using additional computations on overlapped application sub-domains to enable efficient execution in constrained network/system environments such as heterogeneous clusters.

1.3.3 Autonomic Infrastructure with Multiobjective Formulation

As described in Section 1.2.4, addressing the runtime challenges of parallel scientific simulations to obtain optimal performance, often, lead to conflicting objectives and trade-offs. A multiobjective approach can combine various metrics such as load imbalance, synchronization costs, computation-to-communication ratio (CCR), aspect ratio, and other configurable parameters of the simulation to prescribe performance adaptations at runtime. The multiobjective formulation, runtime characterization, and suite of available adaptation strategies are integrated into a conceptual runtime infrastructure for parallel scientific simulations. The research tasks involved in this modeling process are specified below.

- Devise a multiobjective approach that synthesizes the impact and relative contributions of application and system runtime challenges to define overall performance objectives.
- Conceptualize a runtime infrastructure that provides appropriate distribution, configuration, coordination, and adaptation strategies to optimize simulation performance based on the specified objectives.
- Integrate the algorithmic, application, and system adaptations into the proposed runtime management framework and validate its performance, scalability, and efficiency.

1.4 Research Overview

In this research, we devise an autonomic infrastructure (runtime management framework) for the performance optimization of structured unigrid or adaptive formulations of parallel scientific simulations modeling complex physical phenomena. The framework uses application and system state information to select appropriate distribution,

configuration, coordination, and adaptation strategies at runtime. The overarching goal of this research is to enable large-scale investigative and exploratory science using high-performance "smart" simulations based on a policy- and performance-driven approach.

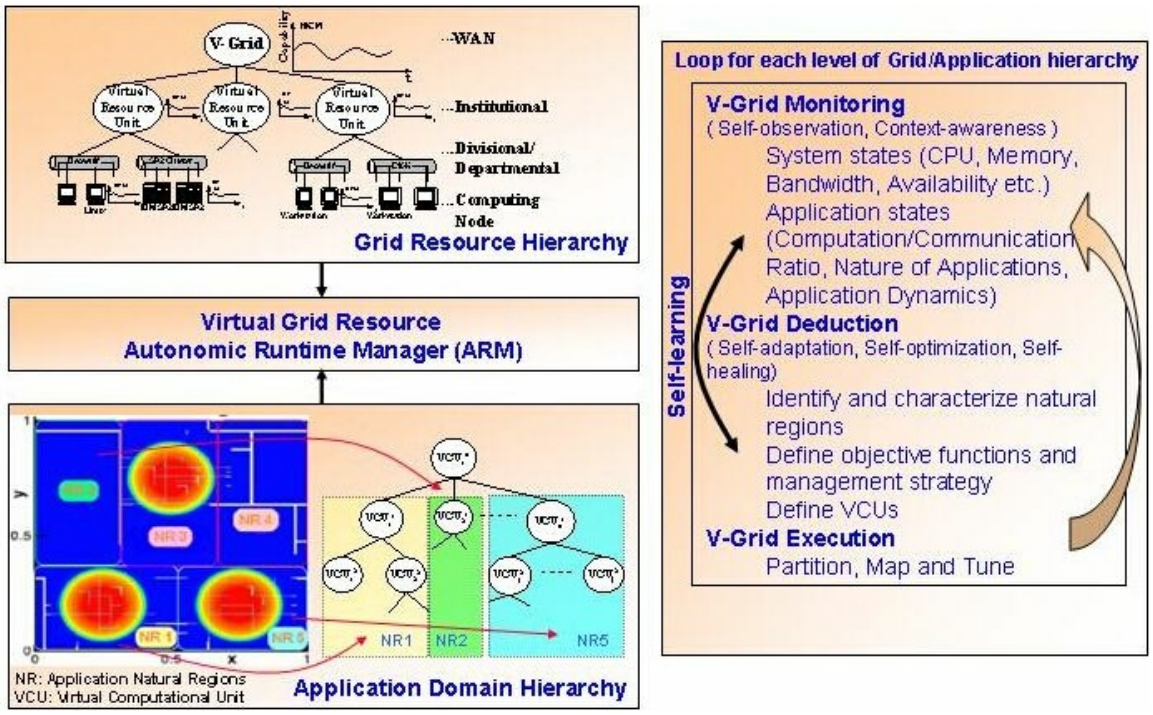

Figure 1.6: Conceptual model of an autonomic runtime infrastructure for parallel scientific simulations.

A conceptual autonomic infrastructure is illustrated in Figure 1.6. The goal of the runtime framework is to reactively and proactively manage and optimize application execution using current system and application state, performance calibration and feedback control, and online multiobjective models for adapting performance. It builds on the concept of *vGrid* [66] proposed by M. Parashar and S. Hariri. The framework defines and maps application "working-sets" across physical resources so as to exploit the space, time, and functional heterogeneity of the simulations and underlying numerical methods. The framework infrastructure services are responsible for collecting and characterizing the operational, functional, and control aspects of

the application. The multiobjective approach uses this information to determine algorithms and adaptations for distribution, configuration, and coordination. As shown in Figure 1.6, the framework has three components: (1) services for monitoring resource capabilities and application dynamics and characterizing the monitored state into natural regions; (2) deduction engine and objective function that define the appropriate optimization strategy based on runtime state and policies; and (3) autonomic runtime manager which is responsible for hierarchically partitioning, scheduling, and mapping the application onto physical resources, and tuning application execution.

We first understand application runtime characteristics and investigate the effects of spatiotemporal heterogeneity on load balancing, performance and scalability of structured adaptive scientific simulations. Next, the effects of computational heterogeneity on simulation performance are addressed by deploying a dynamic partitioning strategy based on pointwise varying workloads. We then use runtime calibration to examine the impact of heterogeneity on load balancing and performance for different orchestrations of scientific applications. This information is used as feedback to devise a hybrid approach that selects the appropriate load balancing algorithm to address heterogeneity at runtime.

Furthermore, we investigate the communication model for parallel scientific simulations in networked environments to identify possible relaxation of synchronization requirements by performing extra computations on overlapped application subdomains. The basic premise for this approach is that processor speeds typically overshadow network speeds for heterogeneous clusters. This behavior is also anticipated for scientific applications when scaled to large numbers of processors on dedicated MPP systems, since such a formulation will involve greater communication and reduced local computations. Subsequently, we analyze these strategies using a multiobjective approach to evaluate the implications of different runtime challenges and to suggest combined algorithmic, application, and system remedies/adaptations that optimize performance. These constituent strategies are integrated into the runtime

management framework described above.

1.5 Contributions and Impact of the Research

The primary contribution of this research is the host of application and system adaptation strategies, which optimize performance and are integrated into the autonomic infrastructure. This research presents a comprehensive approach towards identifying and addressing heterogeneity and runtime challenges for parallel scientific simulations on structured grids. These performance challenges assume greater significance in high-performance petascale systems as well as heterogeneous cluster computing environments. The solutions and adaptations presented herein aim to mitigate performance bottlenecks with the overall goal to enable efficient execution of parallel scientific simulations in such systems. A selection of potential applications of the presented research are highlighted below.

1.5.1 Component-based Scientific Computing

Recent years have seen a steady adoption of component-based technologies for implementing complex scientific simulations [72]. The idea of a monolithic parallel code is replaced by a collection of components, which may be composed into various feasible configurations. Further, these components may be adapted and/or replaced at runtime. Such an approach provides interoperability and flexibility, and can ideally be exploited to significantly improve application performance, especially in cases where the algorithmic and component behaviors as well as overall application execution are not known a priori. For example, a domain-decomposition component for parallel scientific simulations may adapt based on application state and particularly its heterogeneity. However, performing such runtime adaptation is non-trivial and requires an understanding of the requirements of a particular application state as well as a calibration of the impact of the adaptation on overall performance. This research

addresses these issues and can enable efficient component-based scientific computing.

1.5.2 Scientific Simulations in Grid Environments

The Grid [46] has rapidly emerged as the dominant paradigm for wide area distributed computing. Its goal is to provide a service-oriented infrastructure that leverages standardized protocols and services to enable pervasive access to, and coordinated sharing of geographically distributed hardware, software, and information resources. The Grid infrastructure is heterogeneous and dynamic, globally aggregating large numbers of independent computing and communication resources, data stores, and sensor networks. The inherent heterogeneity and dynamism of scientific applications coupled with a similarly heterogeneous and dynamic computational Grid results in significant complexities in application composition and runtime management. This research addresses challenges and provides adaptations for such heterogeneous execution environments.

1.5.3 Insights towards Petascale Computing

As mentioned before, the research community is striving towards achieving petascale computational performance for scientific simulations. The scale and heterogeneity of these systems will necessitate a paradigm shift in application composition and performance optimization. This research comprises important steps towards this objective.

1.6 Outline of the Dissertation

The rest of this dissertation is organized as follows. Chapter 2 outlines structured grid formulations and presents related work in partitioning and load balancing for scientific simulations. Chapter 3 offers an overview of the conceptual autonomic runtime

management infrastructure. Chapter 4 describes prototype application characteristics as well as the design and evaluation of the partitioning/load balancing schemes that address spatiotemporal heterogeneity. Chapter 5 presents the design, operation, and experimental evaluation of partitioning strategies that address computational heterogeneity for scientific applications with pointwise varying workloads. Chapter 6 analyzes the impact of heterogeneity on the performance of reactive flow simulations and motivates the need for a hybrid approach for domain decomposition. Chapter 7 presents the multiobjective formulation that combines different adaptation strategies at runtime to improve the overall performance of scientific simulations. Chapter 8 describes the synchronization adaptation for reducing communication overheads for parallel scientific simulations when the application state is communication-dominated. Chapter 9 presents conclusions and future work.

Chapter 2
Background and Related Work

This chapter first presents an overview of parallel implementations of structured grid formulations of scientific applications. The latter part of this chapter describes the related work in structured grid infrastructures and partitioning and load balancing strategies for scientific simulations.

2.1 PDE and Structured Unigrid Formulations

The numerical solution to a partial differential equation (PDE) can be obtained by first discretizing the problem domain into a mesh/grid comprising small simple shapes called elements [53]. Meshes can either be structured (Cartesian grid of boxes) or unstructured (typically triangulations). The unknowns of the PDE are then approximated numerically at each discrete grid point. The resolution of the grid (or grid spacing) determines the local and global error of this approximation, and is typically dictated by the features of the solution that need to be resolved. The resolution also determines the computational costs and storage requirements for the PDE formulation.

Structured unigrid [78] methods resolve the PDE numerical approximation by discretizing the application problem space onto a single, regular Cartesian grid with uniform resolution. The grid resolution determines the overall workload of the simulation, since the load at each grid point is typically assumed to be homogeneous. Parallel unigrid implementations uniformly decompose the structured grid, assuming homogeneous workloads, to balance load across the processors. Each processor

then performs computations on its local patches and periodically synchronizes its boundaries with neighboring patches.

Note that increasing the resolution to obtain greater solution accuracy in a unigrid application can be expensive. For example, increasing the grid resolution by a factor of two requires a factor of eight more storage in three dimensions. Given a constant Courant factor[1] and homogeneous workloads, the computation time will increase by a factor of sixteen.

2.2 Structured Adaptive Mesh Refinement (SAMR) Formulations

For problems with "well behaved" PDE solutions, it is typically possible to find a grid with uniform resolution that provides the required solution accuracy using acceptable computational and storage resources. However, for problems where the solution contains shocks, discontinuities, or steep gradients, finding such a uniform grid that meets accuracy and resource requirements is not always possible. Furthermore, in these classes of problems, the solution features that require high resolution are localized causing the high resolution and associated computational effort in other regions of the uniform grid to be wasted. Finally, due to solution dynamics in time-dependent problems, it is difficult to estimate the minimum grid resolution required to obtain acceptable solution accuracy.

Compared to numerical techniques based on static uniform discretization, structured adaptive mesh refinement (SAMR) methods employ locally optimal approximations and can yield highly advantageous ratios for cost/accuracy by adaptively concentrating computational effort and resources to regions with large local solution error at runtime. In the case of SAMR methods, dynamic adaptation is achieved by

[1] A time-step/space-step ratio that is normally used as a stability criterion for transient calculations with explicit time-marching schemes.

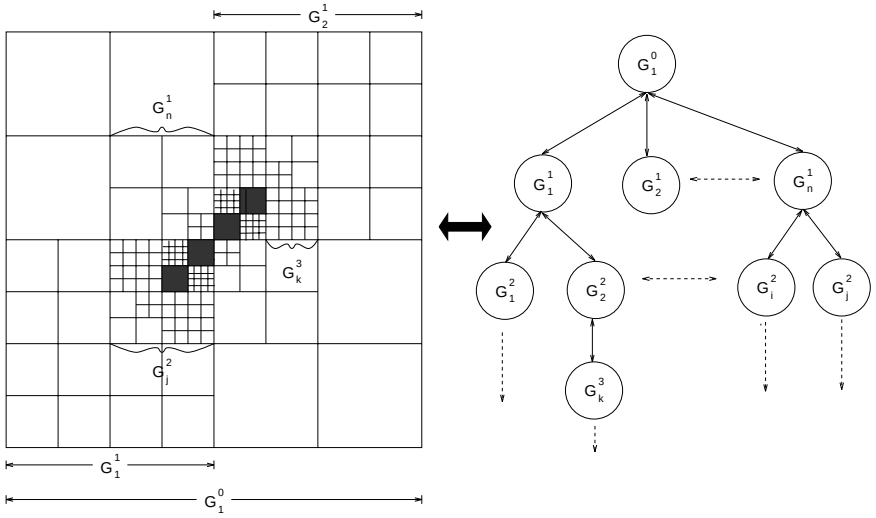

Figure 2.1: Berger-Oliger SAMR formulation of adaptive grid hierarchies.

tracking regions in the domain that require higher resolution and dynamically overlaying finer grids on these regions. These techniques start with a base coarse grid with minimum acceptable resolution that covers the entire computational domain. As the solution progresses, regions in the domain with high solution error, requiring additional resolution, are identified and refined. Refinement proceeds recursively so that the refined regions requiring more resolution are similarly tagged and even finer grids are overlaid on these regions. The resulting grid structure is a dynamic adaptive grid hierarchy. The adaptive grid hierarchy corresponding to the SAMR formulation by Berger and Oliger [10] is illustrated in Figure 2.1.

In SAMR formulations, the grid hierarchy is refined both in space and in time. Refinements in space create finer level grids which have more grid points/cells than their parents. Refinements in time mean that finer grids take smaller time steps and, hence, have to be advanced more often. As a result, finer grids not only have greater computational loads, but also have to be integrated and synchronized more often. This can result in significant spatiotemporal heterogeneity in the SAMR adaptive grid hierarchy. Note that even though finer-resolution grids have more grid points and consequently more work than the coarser grids, the load associated with each grid point throughout the computational domain is still the same since the formulation

typically assumes homogeneous workloads.

2.3 Parallel SAMR Implementations

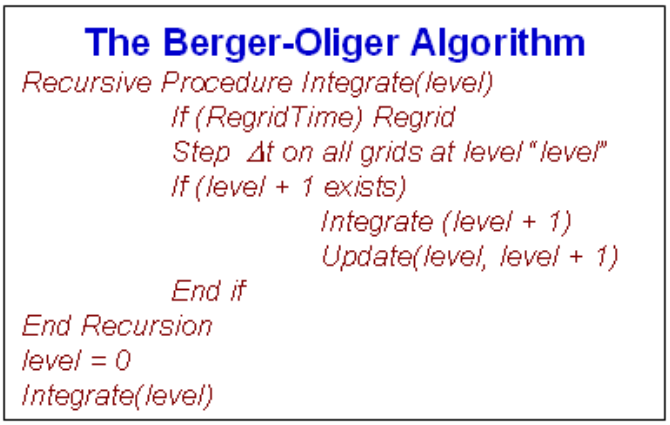

Figure 2.2: Berger-Oliger algorithm with recursive integration at each level of the SAMR grid hierarchy.

Parallel implementations of SAMR applications typically partition the dynamic heterogeneous grid hierarchy across participating processors, with each processor operating on its local portions of the computational domain in parallel. As illustrated in Figure 2.2, each processor starts at the coarsest level, integrates the patches at this level, and performs intra-level or "ghost" communications for boundary updates. It then recursively operates on the finer grids using the refined timesteps - i.e., for each step on a parent grid, there are multiple steps (equal to the time refinement factor) on a child grid. When the parent and child grids are at the same physical time, inter-level communications are used to inject information from finer levels onto coarser ones. The solution error at different levels of the SAMR grid hierarchy is evaluated at regular intervals and this error is then used to determine the regions where the hierarchy needs to be locally refined or coarsened. Dynamic re-partitioning and regridding/redistribution is typically required after this step.

2.3.1 SAMR Algorithmic Requirements

The overall efficiency of parallel SAMR applications is limited by the ability to partition the underlying grid hierarchies at runtime to expose all inherent parallelism, minimize communication and synchronization overheads, and balance load. A critical requirement while partitioning these adaptive grid hierarchies is the maintenance of logical locality, both across different levels of the hierarchy under expansion and contraction of the adaptive grid hierarchy, and within partitions of grids at all levels when they are decomposed and mapped across processors. The former enables efficient computational access to the grids while the latter minimizes the total synchronization overheads. Furthermore, the grid hierarchy is dynamic and adaptation results in application grids being dynamically created, moved, and deleted.

Figure 2.3 illustrates a 2-D snapshot of a sample SAMR application representing a combustion simulation [107] that investigates the ignition of a H_2-Air mixture. The top half of Figure 2.3 depicts the temperature profile of the non-uniform temperature field with 3 hot-spots, while the bottom half of the figure shows the mass-fraction plots of various radicals produced during the simulation. SAMR applications exhibit high dynamism and spatiotemporal heterogeneity, which can present significant runtime and scalability challenges [26].

2.3.2 Computation and Communication Behavior for Parallel SAMR

The timing diagram [74] (note that this diagram is not to scale) in Figure 2.4 illustrates the operation of the SAMR algorithm described above using a 3-level grid hierarchy. For simplicity, only the computation and communication behaviors of processors $P1$ and $P2$ are shown. The three components of communication overheads (described in Section 2.3.3) are illustrated in the enlarged portion of the time line. Note that the timing diagram shows that there is one time step on the coarsest level

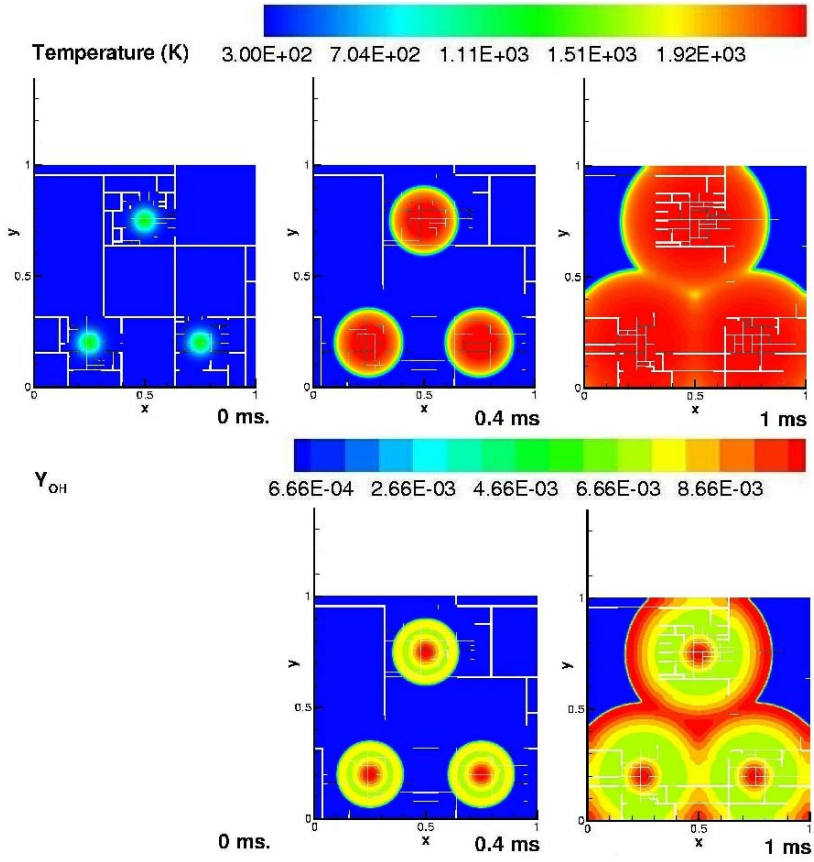

Figure 2.3: 2-D snapshots of a combustion simulation illustrating the ignition of H_2-Air mixture in a non-uniform temperature field with three hot spots (Courtesy: J. Ray, et al, Sandia National Laboratory). The top half of the figure shows the temperature profile while the bottom half shows the mass-fraction plots of radicals.

(level 0) of the grid hierarchy followed by two time steps on the first refinement level and four time steps on the second level, before the second time step on level 0 can start. Also, note that the computation and communication for each refinement level are interleaved.

2.3.3 Communication Overheads for Parallel SAMR

As shown in Figure 2.4, the communication overheads of parallel SAMR applications primarily consist of three components: (1) *Inter-level communications* are defined

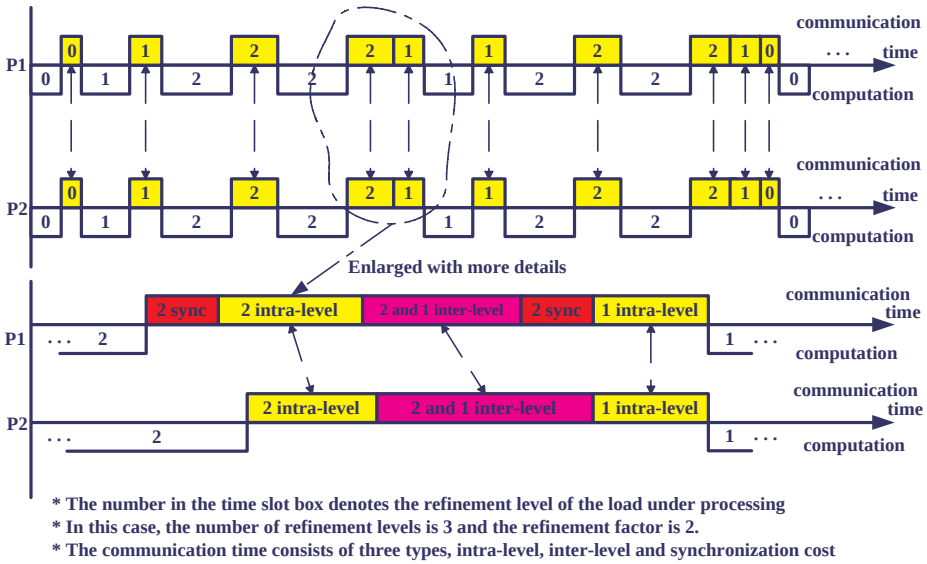

Figure 2.4: Timing diagram for a parallel implementation of Berger-Oliger SAMR algorithm, showing computation and communication behaviors for two processors [74].

between component grids at different levels of the grid hierarchy and consist of prolongations (coarse to fine grid data transfer and interpolation) and restrictions (fine to coarse grid data transfer and interpolation); (2) *Intra-level communications* are required to update the grid elements along the boundaries of local portions of a distributed grid and consist of near-neighbor exchanges. These communications can be scheduled so as to be overlapped with computations on the interior region; and (3) *Synchronization costs* occur when load is not balanced among processors at any time step and at any refinement level. Note that there are additional communication costs due to the data movement required during dynamic load-balancing and redistribution.

Clearly, an optimal partitioning of the SAMR grid hierarchy and scalable SAMR implementations require a careful consideration of the timing pattern and the communication overheads described above. A critical observation from Figure 2.4 is that, in addition to balancing the total load assigned to each processor, the load balance at each refinement level and the communication/synchronization costs within a level need to be addressed. This leads to a trade-off between (i) maintaining parent-child locality for component grids that results in reduced communication overheads but

possibly high load imbalance, and (ii) "orphaning" component grids (i.e., isolating grid elements at different refinement levels of the SAMR hierarchy) resulting in better load balance with slightly higher synchronization costs.

2.4 Taxonomy for Partitioning and Load Balancing

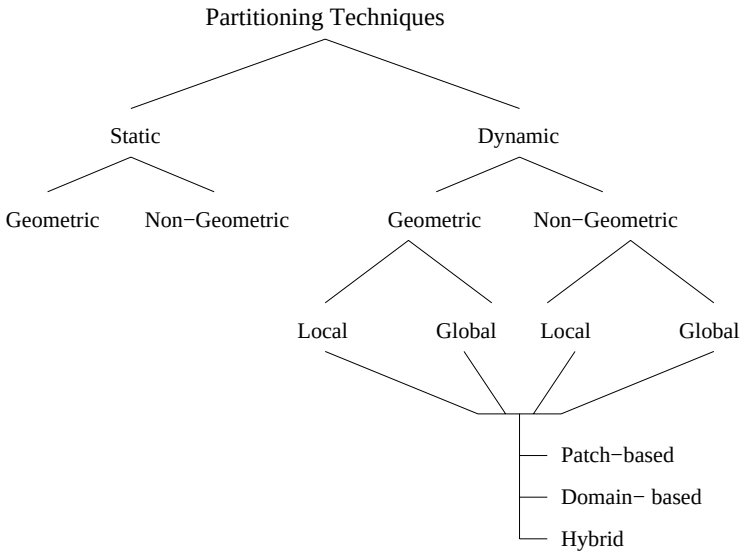

Figure 2.5: A general taxonomy of partitioning and load balancing approaches.

A general taxonomy of partitioning and load balancing approaches is presented in Figure 2.5. The plethora of research on partitioning and load balancing techniques can be broadly divided into static and dynamic techniques. **Static partitioning** techniques are used when the application domain is partitioned only once (or very few times) and there is no dynamic redistribution involved. In this case, the initial partitioning is maintained through the execution of the application. Static techniques tend to focus on the partitioning quality rather than partitioning speed. **Dynamic partitioning** routines are used by adaptive applications to repartition and redistribute the dynamic application domain at runtime. Moreover, these techniques have to minimize data movement overheads and partitioning time since grid adaptations occur at regular intervals. Consequently, partitioning quality is often sacrificed for

speed and efficiency by these partitioners. **Dynamic scheduling** [117, 104], a variant of the dynamic techniques, is a more general way of dealing with the load balancing problem. It requires some basic partitioning of the domain into "tasks" that can be created at runtime. These tasks are then scheduled dynamically. This approach is closer to the operating system and mimics thread-level scheduling as compared to the application-level strategies discussed in this section.

Static and dynamic partitioners can be further sub-divided into structured (geometric) and unstructured (graph-based) techniques. **Structured techniques** take the geometry of the grids into account during partitioning, and include strategies such as binary dissection [8] [21], ISP [89], and geometric mesh partitioning [49] [25]. **Unstructured techniques** use a graph representation of the problem domain and partition this graph among processors. Examples of schemes in this category include recursive spectral bisection [5] and the multilevel algorithms in the software partitioning library METIS [62] and ParMETIS [63]. Unstructured techniques can also be used for partitioning domains where geometry plays an important role. In such cases, the geometrical information is coded into the graph which is used as the input to the partitioner.

Dynamic partitioning and load balancing techniques can be either global or local. **Global techniques** maintain a global view of the problem domain and use this global information to partition the domain. Global techniques include space-filling curve (SFC) partitioners [94, 102] and diffusion schemes based on global workload [113]. SFC-based global techniques are used by PARAMESH [75] and GrACE [90] [94] SAMR infrastructures. While these global techniques typically lead to a better and more balanced distribution, the global synchronization required in these schemes may be expensive. The hierarchical multithreaded model proposed in [105] uses multiple threads and a hierarchical redistribution to reduce these overheads. A similar approach is used in PaLaBer [104] which additionally uses preemptive and non-preemptive process migration to balance load. **Local techniques**, such as the

molecular dynamics load balancing algorithm in [20] and diffusion schemes using local workloads [113], base their partitioning on localized views of the problem domain. Typically, each processor only knows about its workload and that of its immediate neighbors. Local schemes tend to be faster but the simulation workload may not be as well-balanced as in the case of global techniques.

Partitioners for SAMR grid hierarchies can be classified as patch-based, domain-based, or hybrid. In the case of **patch-based partitioners**, distribution decisions are independently made for each newly created grid. A grid may be kept on the local processor or entirely moved to another processor. If the grid is too large, it may be split (see Figure 2.6). **Domain-based partitioners** partition the physical domain rather than the grids themselves. The domain is partitioned along with all contained grids on all refinement levels. **Hybrid partitioners** use a two-step partitioning approach. The first step uses domain-based techniques to generate meta-partitions which are mapped to a group of processors. The second step uses a combination of domain and patch based techniques to optimize the distribution of each meta-partition within its processor group.

The SAMR framework SAMRAI [68] (based on the LPARX [4] and KeLP [45] model) fully supports patch-based partitioning. The distribution scheme maps the patches at a refinement level of the AMR hierarchy across processors. Domain-based partitioners tend to be more scalable [92] [13] [131] [106] [121] [124] and are the primary focus of the research presented in this thesis. Some hybrid approaches are presented in [92].

Patch-based techniques result in good load balance, independently of the depth and complexity of the grid hierarchy, as long as each patch has a sufficient amount of work. Moreover, they do not require redistribution when grids are created or deleted. These techniques, however, can lead to substantial parent-child communication overheads. In a SAMR grid hierarchy, a fine grid typically corresponds to a relatively small region of the underlying coarse grid. If both the fine and coarse grids are distributed

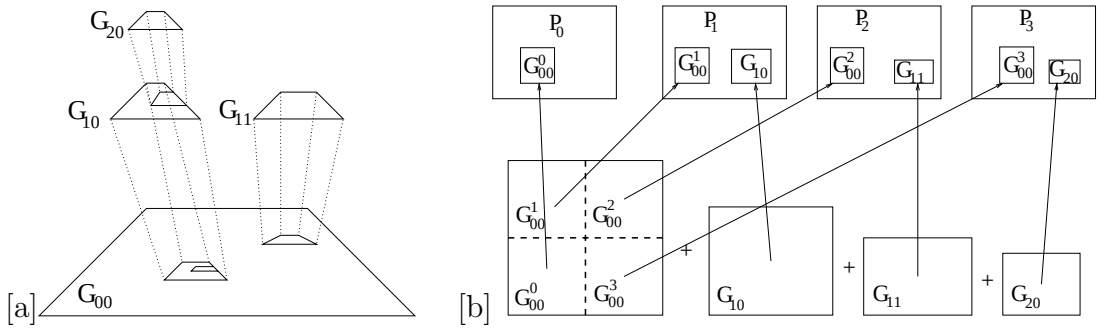

Figure 2.6: Patch-based partitioning: patch G_{00}^0 in (a) split into $G_{00}^0 - G_{03}^0$ in (b) (Courtesy: J. Steensland [123]).

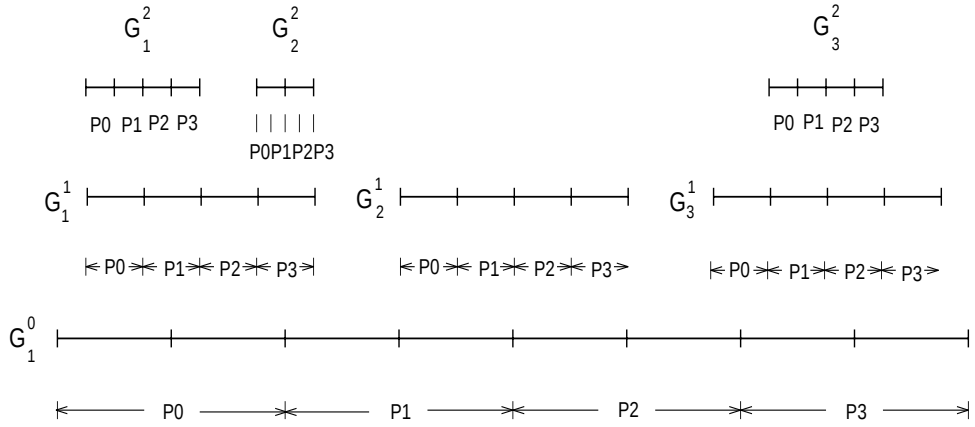

Figure 2.7: Patch-based partitioning of a 1-D SAMR grid hierarchy (Courtesy: J. Steensland [123]).

over the entire set of processors (corresponding to a patch-based partitioning), all the processors will communicate with the small set of processors corresponding to the associated coarse grid region, thereby causing a serialization bottleneck. For example, in Figure 2.7, a restriction from grid G_2^2 to grid G_1^1 requires all the processors to communicate with processor $P3$. Another problem with patch-based distributions is that parallelism across multiple grids at a level is not exploited. In most SAMR algorithms, updates are performed level by level in a sequential manner. That is, the solution on all grids at refinement level l have to be advanced before the solution on any overlaid grid at level $l+1$ can be advanced. Thus, parallelism in SAMR applications primarily comes from operating on component grids at a level in parallel. For example, in Figure 2.7, grids G_1^1, G_2^1, and G_3^1 are distributed across the same set

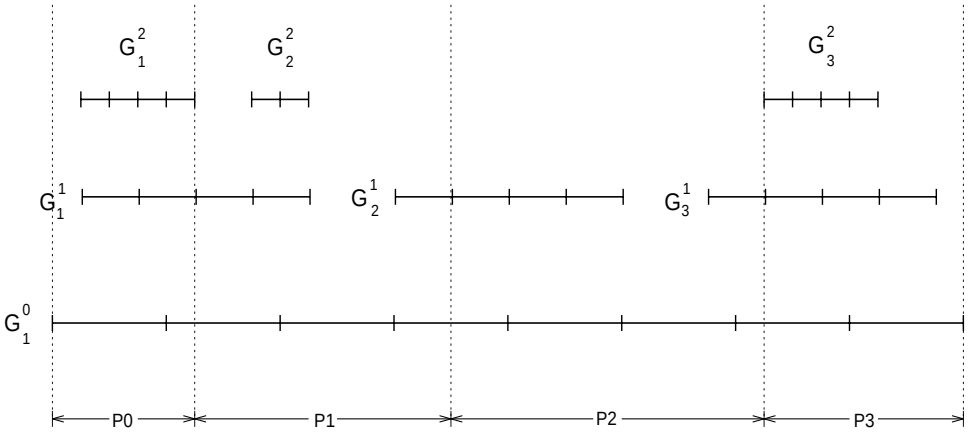

Figure 2.8: Domain-based partitioning of a 1-D SAMR grid hierarchy (Courtesy: J. Steensland [123]).

of processors and have to be integrated sequentially. This causes processors to be temporarily idle and parallel efficiency to deteriorate.

Domain-based partitioners (Figure 2.8), on the other hand, eliminate "depth-wise" communication by maintaining parent-child locality. Furthermore, these partitioners fully exploit the available parallelism at each level of the SAMR grid hierarchy. However, domain-based partitioners require load redistribution when grids are created or destroyed. But this redistribution can be performed incrementally [92], thus minimizing overheads. Although domain-based partitioners can effectively support parallel SAMR, these techniques require the overall structure of the grid hierarchy to be maintained and partitioned, which can be challenging. Furthermore, the load balance produced by domain-based partitioners for deep SAMR hierarchies with strongly localized regions of refinement can deteriorate significantly. In these cases, hybrid partitioners, combining patch-based and domain-based approaches, can be used.

2.5 Space-Filling Curve Based Partitioning

Inverse space-filling curve partitioners (ISP) [92] [120] are widely used to implement domain-based partitioning of SAMR grid hierarchies. Space-filling curves (SFC) [57]

[100] [109] [17] [110] are computationally efficient, locality preserving recursive mappings from N-dimensional space to 1-dimensional space i.e. $N^d \rightarrow N^1$, such that each point in N^d is mapped to a unique point or index in N^1. Two such mappings, the Morton order and the Peano-Hilbert order, are shown in Figure 2.9.

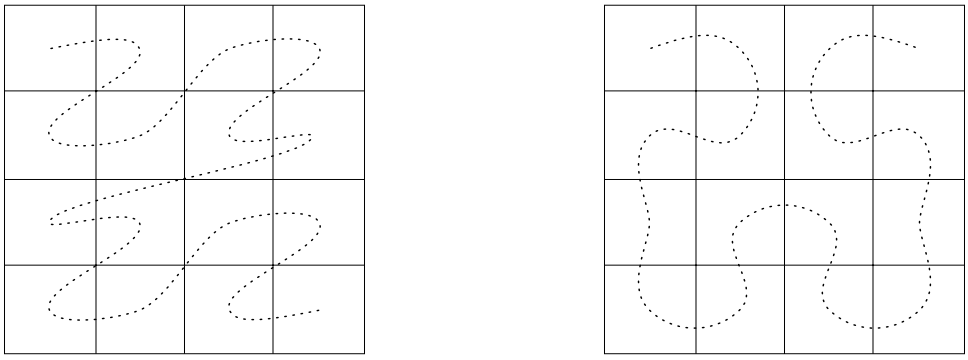

Figure 2.9: Space-filling curves: Morton (left) and Peano-Hilbert (right) [123].

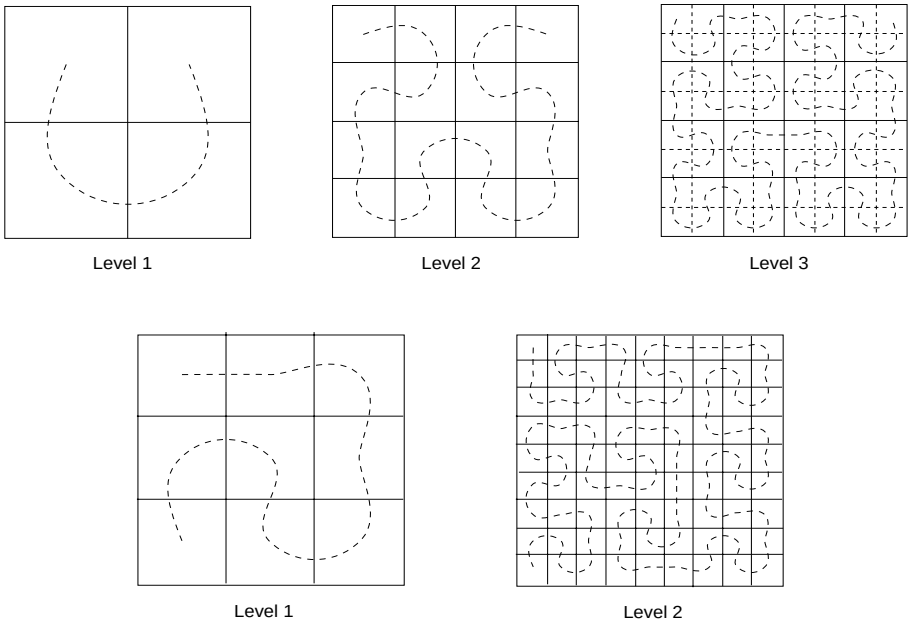

Figure 2.10: Self-similarity property of space-filling curves [123].

Two properties of space filling curves, viz. digital causality and self-similarity, make them particularly suitable for partitioning SAMR grid hierarchies. Digital causality implies that points close together in d-dimensional space will be mapped to points close together in 1-dimensional space, i.e. the mapping preserves locality [44]

[51] [52] [80] [93]. Self-similarity implies that, as a d-dimensional region is refined, the refined sub-regions 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, this self-similar or recursive nature of SFC mappings is exploited to represent the hierarchical structure and to maintain locality across different hierarchy levels. Finally, these techniques are computationally efficient and are suited to SAMR algorithms, which require regular regriding steps. ISP techniques can be effectively used to support parallel applications [89] [102] and are used by SAMR infrastructures such as PARAMESH [75] and GrACE [93].

2.6 AMR Infrastructures

Currently, there exist a wide spectrum of software infrastructures that have been developed and deployed to support parallel and distributed implementations of AMR applications. A survey on AMR applications is presented in [41]. Each system represents a unique combination of design decisions in terms of algorithms, data structures, decomposition schemes, mapping and distribution strategies, and communication mechanisms. Table 2.1 summarizes a selection of the existing AMR software packages/infrastructures.

2.6.1 PARAMESH

PARAMESH [76] is a FORTRAN library designed to facilitate the parallelization of an existing serial code that uses structured grids. The package builds a hierarchy of sub-grids to cover the computational domain, with spatial resolution varying to satisfy the demands of the application. These sub-grid blocks form the nodes of a tree data-structure (quad-tree in 2-D or oct-tree in 3-D). Each grid block has a logically

Table 2.1: Summary of AMR infrastructures.

AMR Infrastructures		
Infrastructure	Description and Method Supported	Strategy
GrACE	Object-oriented toolkit built on semantically specialized DSM substrate implementing a hierarchical distributed dynamic array. Supports SAMR and multigrid methods	Dynamic, global, domain-based, geometric
SCOREC	Hierarchical representation of finite element mesh with operators to query and update mesh structure. Supports adaptive finite element methods	Dynamic, global, geometric
PARAMESH	Extends serial code to parallel based on partitioning hierarchical tree representation of adaptive grid structure. Supports SAMR	Dynamic, geometric, domain-based
BATSRUS	Block-based approach with the adaptation distributed over processors in computational pool in phases. Supports AMR	Dynamic, domain-based
SAMRAI	Object-oriented SAMR framework based on LPARX and KeLP model with patches mapped separately at refinement levels	Dynamic, global, patch-based
Chombo	Distributed infrastructure for implementing finite difference calculations on block-structured, adaptively refined rectangular grids	Dynamic, domain-based, geometric

Cartesian structured mesh.

2.6.2 SAMRAI

Structured Adaptive Mesh Refinement Application Infrastructure (SAMRAI) [68] [134] provides computational scientists with general and extensible software support for the prototyping and development of parallel SAMR applications. The framework contains modules for handling tasks such as visualization, mesh management, integration algorithms, geometry, etc. SAMRAI makes extensive use of C++ object-oriented techniques and various design patterns such as abstract factory, strategy and chain of responsibility. Load-balancing in SAMRAI occurs independently at each refinement level resulting in convenient handling but increased parent-child communication.

2.6.3 Chombo

The Chombo [37] package provides a distributed infrastructure and set of tools for implementing parallel calculations using finite difference methods for the solution of partial differential equations on block-structured, adaptively refined rectangular grids. Chombo includes both elliptic and time-dependent modules as well as support for parallel platforms and standardized self-describing file formats.

2.6.4 GrACE

Grid Adaptive Computational Engine (GrACE) [92] is an adaptive computational and data-management framework for enabling distributed adaptive mesh refinement computations on structured grids. It is built on a virtual, semantically specialized distributed shared memory substrate with multifaceted objects specialized to distributed adaptive grid hierarchies and grid functions. The development of GrACE is based on systems engineering focusing on abstract types, a layered approach, and separation of concerns. GrACE allows the user to build parallel SAMR applications and provides support for multigrid methods.

2.6.5 SCOREC

SCOREC Parallel Mesh Database [115] provides a generic mesh database for the topological, geometric and classification information that describes a finite element mesh. The database supports meshes of non-manifold models and multiple meshes on a single model or multiple models. Operators are provided to retrieve, store and modify the information available in the database. Parallel Mesh Database (PMDB) provides three static partitioning procedures for initial mesh distribution, three dynamic load-balancing schemes, and mesh migration operators.

2.6.6 BATSRUS

BATSRUS [6] is implemented in Fortran90 using a block-based domain-decomposition approach. Cell blocks, stored as 3-D Fortran90 arrays, are locally stored on each processor to achieve reasonably balanced load. The application starts out with a pool of processors with some processors being possibly unused. Each utilized processor has a block of equal memory size but possibly at a different resolution and/or different partition size of the physical space. As the application adapts, new (adapted) grids are allocated in units of the same fixed block size to the unused processors. No additional refinement can occur once all the virtual processors have been utilized.

2.7 Partitioning Libraries

In addition to the AMR infrastructures, there exist several partitioning tools that are used in domain decomposition and load balancing for scientific simulations. Table 2.2 summarizes a selection of existing partitioning libraries.

Table 2.2: Summary of partitioning libraries.

Partitioning Libraries		
Library	Description and Method Supported	Strategy
Vampire	Research tool with dedicated partitioners combining structured and unstructured techniques. Supports SAMR	Dynamic, global, domain-based
METIS ParMETIS	Partitioning unstructured graphs based on parallel multilevel k-way graph-partitioning algorithms	Dynamic, global, graph-based
PART ParaPART	Tool for distributed systems considering heterogeneity and application computational complexity, using simulated annealing	Dynamic, global
Zoltan	Object-oriented library including a suite of algorithms for dynamically computing the partitions over sets of processors	Dynamic, geometric/ graph-based

2.7.1 Vampire

Vampire [119] is a dedicated SAMR partitioning library (2D/3D) involving variable-grain adaptive mesh partitioner/repartitioner strategies. It uses reverse multilevel ISP techniques combining structured and unstructured schemes and consists of a set of highly parameterized domain-based algorithms written in C. The prominent features of Vampire include fast, high-quality partitioning and variable grain size allowing for experimentation with granularity. In Vampire, granularity is determined by the size of the "atomic unit" and the "weight limit".

2.7.2 METIS and ParMETIS

METIS [62] is a set of programs for partitioning graphs, partitioning finite element meshes, and for producing fill reducing orderings for sparse matrices. The algorithms implemented in METIS are based on the multilevel graph partitioning schemes. Key features of METIS include high quality partitions, fast execution, and better fill-orderings. ParMETIS [63] is an MPI-based parallel library that extends the functionality provided by METIS and includes routines that are especially suited for parallel AMR computations and large scale numerical simulations. The algorithms implemented in ParMETIS are based on the parallel multilevel k-way graph-partitioning algorithms.

2.7.3 PART and ParaPART

PART [32] is a tool for automatic mesh partitioning for distributed systems. PART takes into consideration the heterogeneities in processor performance, network performance, and application computational complexities to achieve a balanced estimate of execution time across the processors in the distributed system. Simulated annealing is used in PART to perform the backtracking search for desired partitions. ParaPART [33] is a parallel version of PART and significantly improves its performance by using

the asynchronous multiple Markov chains approach of parallel simulated annealing.

2.7.4 Zoltan

Zoltan [40] is a load-balancing tool that provides an interface between an application and existing load balancing routines. This object-oriented library, written in C++, uses established algorithms to compute partitions over a set of processors for a given parallel application. Though the new distribution is computed transparently by Zoltan, the data-movement features have to be supported by the user or the interfacing application.

2.8 Dynamic Adaptive Partitioning Strategies

Dynamic adaptive partitioning and load balancing for AMR applications on structured and unstructured computational meshes is an active research area. A summary of these schemes is presented in Table 2.3. Whereas dynamic adaptive partitioning techniques have been extensively investigated for unstructured meshes, such schemes for structured grids are relatively unexplored. Our goal in this research is to adaptively manage dynamic and heterogeneous applications on structured grids based on a multiobjective approach and a characterization of the runtime state.

2.8.1 PLUM

PLUM (Parallel Load-balancing for adaptive Unstructured Meshes) [87] is a dynamic load balancing strategy for adaptive unstructured grid computations that uses a cost-metric model for efficient mesh adaptation. PLUM consists of a flow solver and a mesh adaptor, with a partitioner and a remapper that load balances and redistributes the computational mesh when necessary. Parallel adaptation code consists of three phases – initialization, execution, and finalization.

Table 2.3: Research summary on dynamic partitioning/load-balancing schemes.

Dynamic Adaptive Partitioning and Load Balancing	
Scheme	Description
PLUM	Dynamic load balancing for unstructured meshes using cost-metric model with computation, communication and remapping weights; accepts new partitioning if gain greater than redistribution cost
URA	Repartitioning for unstructured meshes based on relative cost factor to minimize both communication and data redistribution cost; initial partitioning scheme yielding lowest cost is selected
SAMR DLB	Moving-grid and splitting-grid phases in parallel where load is balanced across processors after adaptation, or grid is split along the longest dimension, if imbalance still exists
Partitioning Advisory System	Ranking of partitioning methods based on the problem space, machine speed, and network information; considers processor performance variance, but assumes linear complexity

A dual graph representation of the initial computational mesh keeps complexity and connectivity constant. The PLUM model uses computation (W_{comp}), communication (W_{comm}), and data-remapping (W_{remap}) weights to implement accurate metrics that estimate and compare the computational gain and the redistribution cost of having a balanced workload after each mesh adaptation step. W_{comp} and W_{comm} determine how dual graph vertices should be grouped to form partitions that minimize both the disparity in the partition weights and the runtime communication. W_{remap} determines how partitions should be quickly assigned to processors using a heuristic remapping algorithm such that the cost of data redistribution is minimized. The new partitioning and mapping are accepted if the computational gain is larger than the redistribution cost.

2.8.2 Unified Repartitioning Algorithm

The Unified Repartitioning Algorithm (URA) [114] is a parallel adaptive repartitioning scheme for the dynamic load-balancing of scientific simulations that attempts to solve the precise multi-objective optimization problem. A key parameter used in URA is the Relative Cost Factor (RCF) that describes the relative times required for

performing the inter-processor communications incurred during parallel processing and to perform the data redistribution associated with balancing the load. Using this parameter, it is possible to unify the two minimization objectives of the adaptive graph partitioning problem into the precise cost function

$$|E_{cut}| + \alpha|V_{move}|$$

where α is the RCF, $|E_{cut}|$ is the edge-cut of the partitioning, and $|V_{move}|$ is the total amount of data redistribution. URA attempts to compute a repartitioning while directly minimizing this cost function.

URA has three phases: graph coarsening, initial partitioning, and uncoarsening/-refinement. In the initial partitioning phase of URA, repartitioning is performed on the coarsest graph twice by alternative methods which are optimized variants of scratch-remap and global diffusion schemes. The cost functions are then computed for each of these and the one with the lowest cost is selected. The algorithm is extremely fast and scalable to very large problems; however, it is primarily targeted towards unstructured meshes.

2.8.3 Dynamic Load Balancing for SAMR

In investigating dynamic load balancing (DLB) schemes for SAMR [70], Lan, Taylor, and Bryan analyzed the requirements imposed by the applications. The analysis provided four unique characteristics: coarse granularity, high dynamism, high imbalance and different dispersion, and an implementation that maintains some global information. Their DLB scheme is composed of two steps: moving-grid phase and splitting-grid phase.

In the "moving-grid phase", after each adaptation, the DLB is triggered if the load is imbalanced, i.e. $MaxLoad/AvgLoad > threshold$. The $MaxProc$ moves its grid directly to $MinProc$ under the condition that the computational load of this grid does not exceed $(threshold * AvgLoad - MinLoad)$, thereby ensuring that an

underloaded processor does not become overloaded. If imbalance still exists, the "splitting-grid phase" is invoked and it splits the grid, along the longest dimension, into two subgrids. Eventually, either the load is balanced or the minimum allowable grid size is reached. Both the moving-grid phase and splitting-grid phase execute in parallel.

A modified DLB scheme on distributed systems [71] considers the heterogeneity of processors and the heterogeneity and dynamic load of the network. The scheme employs global load-balancing and local load-balancing, and uses a heuristic method to evaluate the computational gain and redistribution cost for global redistribution.

2.8.4 Partitioning Advisory System

Crandall and Quinn developed a partitioning advisory system [38] for a network of workstations. The advisory system has three built-in partitioning methods (contiguous row, contiguous point, block). Given information about the problem space, the machine speed, and the network, the advisory system provides ranking of the three partitioning methods. The advisory system takes into consideration the variance in processor performance among the workstations but variance in network performance is not considered. The problem, however, is that linear computational complexity is assumed for the application.

2.9 Addressing Multiple Objectives in Scientific Simulations

Multiobjective optimization problems [1, 118] require the simultaneous optimization of more than one objective function in a multi-dimensional criteria search space. Multiobjective optimization techniques are based on evolutionary [135] or combinatorial [22] formulations, and have been extensively studied in the field of operations research. A comprehensive survey of multiobjective optimization techniques is presented in [35].

The set of criteria that produces the optimal outcome is a set of non-inferior or non-dominated solutions, designated as the Pareto optimal set or Pareto front. When the behaviors of the objective functions are adequately known and the criteria search space is convex, aggregating functions can be used to combine multiple objectives into a single function. Examples of this approach include weighted aggregation methods, goal programming or attainment, and ϵ-constraint methods. Alternative techniques for multiobjective optimization can be based on population policies or specialized operations for objective functions, and include genetic algorithms and their variants, hierarchical or lexicographic ordering, weighted min-max approach, and use of game theory principles.

Despite the recent advances in multiobjective optimization techniques and their widespread use in several science and engineering domains, these strategies have been sparingly adopted in high-performance computing. This observation can be attributed to three key factors. First, the runtime behaviors of adaptive scientific simulations are not known a priori, making it difficult to characterize, adapt and optimize performance. Second, analytical models of the performance objectives for complex, multi-phased scientific simulations are typically not feasible, and scientists resort to heuristics and empirical approaches to estimate and optimize performance. Third, even if an accurate performance model existed, it is generally very time-consuming to compute the exact Pareto optimal solution that satisfies all performance objectives for each state of the dynamic parallel scientific simulation. Moreover, an exact Pareto analysis tends to be an overkill for high-performance simulations since the cost to determine and deploy the optimization may be greater than the benefit provided by the optimization itself. Nonetheless, multiobjective optimization techniques have been applied in certain frameworks supporting scientific simulations, which are described in Chapter 7. In this research, we focus on the dynamism and heterogeneity challenges that impact the runtime performance of adaptive structured grid formulations of various parallel scientific simulations.

Chapter 3
Autonomic Runtime Management Infrastructure

3.1 Application and System Heterogeneity in SAMR

Unlike static applications where requirements are typically known a priori, the dynamic behavior of structured adaptive mesh refinement (SAMR) applications is based on the current state of the physical phenomenon being simulated and can only be determined at runtime. Thus, it is important to abstract the state of the SAMR application in order to determine its current computational, communication, and storage requirements. This information can then be used to determine an appropriate decomposition of the application and mapping of the computations to available processing elements of the computational environment, and to drive the selection of appropriate algorithms and implementations, both at the application level (solvers, preconditioners) as well as the system level (communication mechanism).

Furthermore, the execution environment such as a high-performance or networked computational system may exhibit dynamic or constrained behavior at runtime. The underlying system heterogeneity introduces a new level of complexity and makes the selection of a "best" match between system resources, application algorithms, problem decompositions, mappings and load distributions, communication mechanisms, etc., non-trivial. System dynamics coupled with application adaptation makes application composition, runtime management, and optimization a significant challenge.

To address these challenges, we propose a multiobjective approach that analyzes application and system state at runtime and provides appropriate distribution, configuration, coordination, and adaptation strategies. Such an autonomic infrastructure

formulation can address dynamism and heterogeneity in parallel scientific simulations on structured grids, and enable their efficient and scalable execution on high-performance or cluster computing architectures.

3.2 Runtime Infrastructure Requirements

Parallel/distributed implementations of scientific simulations lead to interesting computational and computer science challenges in dynamic resource allocation, data-distribution and load balancing, communications and coordination, and runtime management. As a result, key requirements for an efficient runtime management framework for SAMR applications include:

- **Dynamic Partitioning Support:** The overall efficiency of the algorithms is limited by the ability to partition the underlying data-structures at runtime so as to expose all inherent parallelism, minimize communication and synchronization overheads, and balance load.

- **Adaptive Communication Support:** A critical requirement while partitioning adaptive grid hierarchies is the maintenance of logical locality, both across different levels of the hierarchy under expansion and contraction of the adaptive grid structure, and within partitions of grids at all levels when they are decomposed and mapped across processors. The former enables efficient computational access to the grids while the latter minimizes the total communication and synchronization overheads.

- **Dynamic Application Configuration Support:** Application adaptation results in application grids being dynamically created, moved and deleted on-the-fly, making it necessary to efficiently re-partition the hierarchy so that it continues to meet performance goals. Furthermore, the heterogeneity in the

underlying execution environment may require selecting and configuring application components based on system state.

3.3 Autonomic Infrastructure: Model and Operation

3.3.1 Monitoring and Characterization

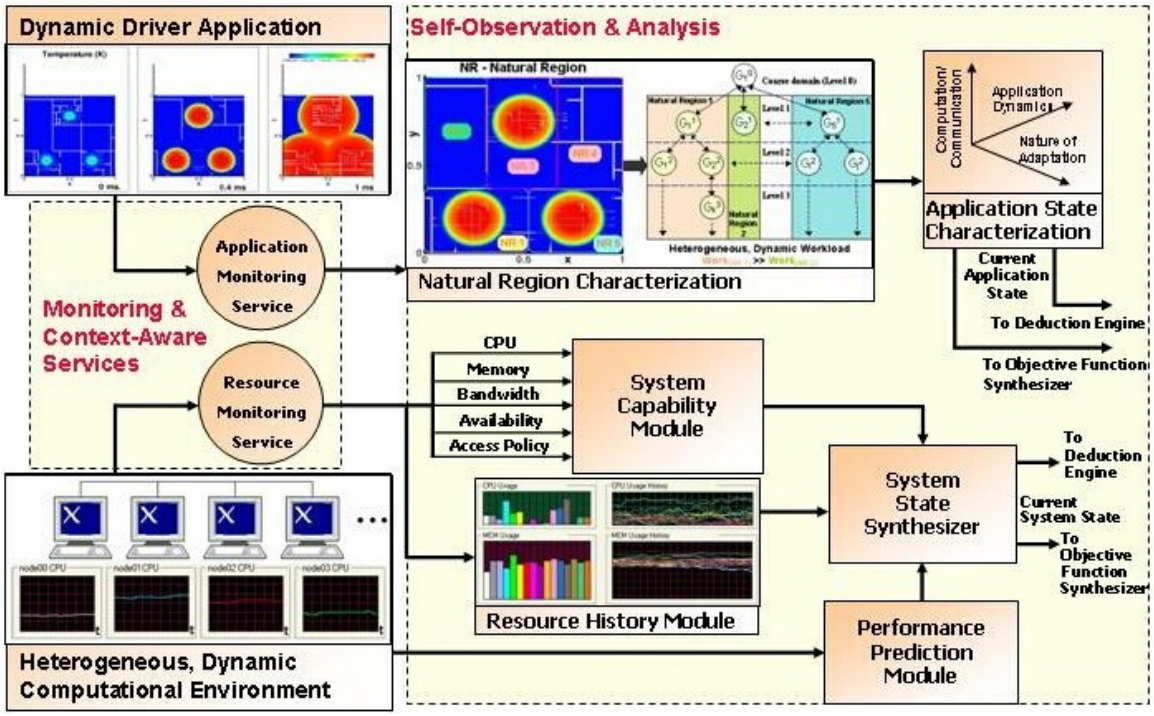

Figure 3.1: Autonomic infrastructure operation: monitoring and characterization.

The monitoring and characterization mechanisms in the runtime management framework consist of embedded application-level and system-level sensors/actuators and are illustrated in Figure 3.1. The application is characterized into "natural regions" (NRs) which are regions of relatively homogeneous activity in the application domain and can span various levels of the SAMR grid hierarchy. Application sensors monitor the structure and state of the SAMR grid hierarchy and the nature of the refined regions. One way to track such natural regions for SAMR applications is using the refinement patterns based on local truncation errors. For example, a

"hot-spot" in the flame simulation application described in Section 2.3.1 represents a natural region in the application domain. The application state is abstracted using these natural regions and is characterized in terms of application-level metrics such as computation/communication requirements, storage requirements, activity dynamics, and the nature of adaptations [123].

Similarly, system sensors, which may be built on existing infrastructures such as NWS (Network Weather Service) and MDS (Metacomputing Directory Service), sense the current state of underlying computational resources in terms of CPU, memory, bandwidth, availability, and access capabilities. These are fed into the system state synthesizer along with history information (current state stored over time in the history module) and performance estimates (obtained using performance functions from the prediction module) to determine the overall system runtime state. The current application and system state are provided as inputs to the deduction engine and are used to define the autonomic runtime objective function.

3.3.2 Deduction and Objective Function

The deduction engine and the autonomic runtime manager provide the primary decision making capabilities within the runtime management framework. As shown in Figure 3.2, the current application and system state and the overall "decision space" are the inputs to the deduction engine. The decision space comprises the adaptation policies, rules, and constraints defined in terms of application metrics, and enables autonomic configuration, adaptation, and optimization. Application metrics may include application locality, communication mechanism, data migration, load balancing, memory requirements/constraints, adaptive partitioning, adaptation overheads, and granularity control. Based on current runtime state and policies/constraints within the decision space, the deduction engine formulates prescriptions for algorithms, configurations, and parameters that are used to define the objective function for adapting the behavior of the SAMR application. The deduction engine may be made capable

of self-learning by augmenting its decision space with new rules and constraints. The prescriptions provided by the deduction engine along with the objective function yield two metric – normalized work metric (NWM) and normalized resource metric (NRM) that characterize the current application state and current system state, respectively. These metric are self-defined based on the current application/system context and enable autonomic runtime management by helping to configure the SAMR application with appropriate parameters and execute optimally within the computing environment, which may be heterogeneous.

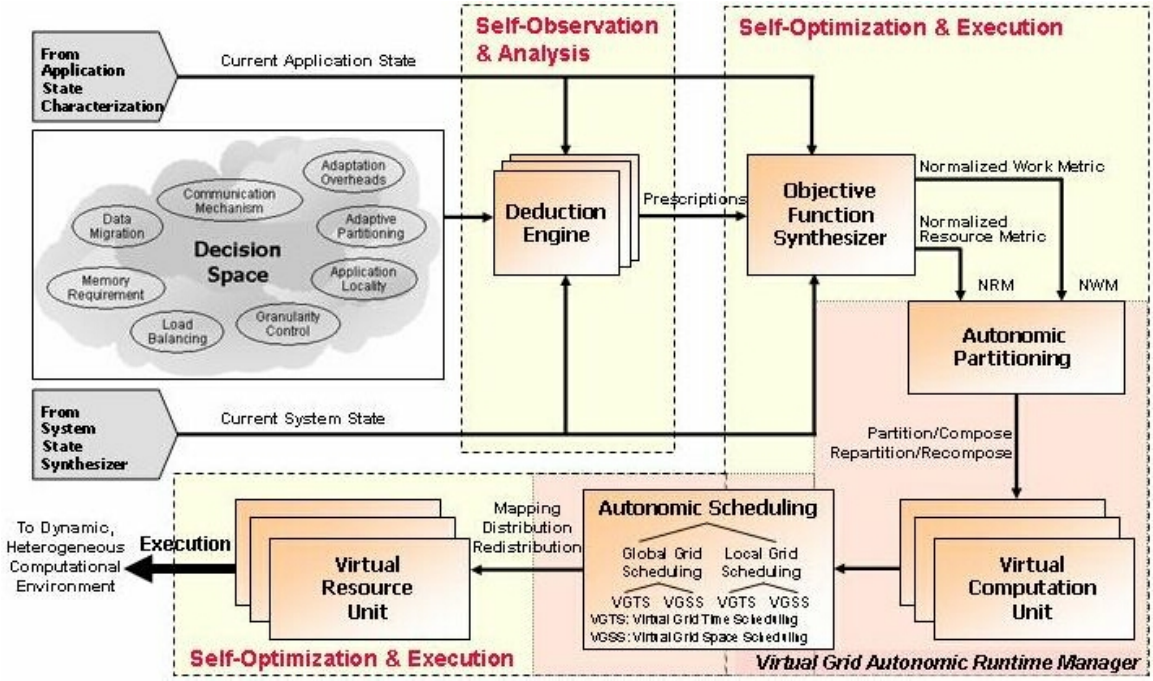

Figure 3.2: Autonomic infrastructure operation: deduction and optimization.

3.3.3 Autonomic Runtime Manager

The normalized metric, NWM and NRM, form the inputs to the autonomic runtime manager (ARM). Using these inputs, ARM defines a hierarchical distribution mechanism, configures and deploys appropriate partitioners at each level of the hierarchy, and maps the application domain onto virtual computational units. A virtual computational unit (VCU) is the basic application work unit that is scheduled by

the runtime management framework and may consist of computation patches on a single refinement level of the SAMR grid hierarchy or composite patches that span multiple refinement levels. VCUs are dynamically defined at runtime to match the natural regions (NRs) in the application. Using natural regions to define VCUs can significantly reduce coupling and synchronization costs.

Subsequent to partitioning, scheduling operations are performed first across VRUs (Global-Grid Scheduling (GGS)) and then within a VRU (Local-Grid Scheduling (LGS)). During GGS, VCUs are hierarchically assigned to sets of VRUs, whereas LGS is used to schedule one or more VCU within a single VRU. The entire process is first spatial and then temporal, and could possibly combine a range of partitioning techniques (domain-based, patch-based, tree-based, etc.) and scheduling techniques (gang, backfilling, migration, etc.). A virtual resource unit (VRU) may be an individual resource (compute, storage, instrument, etc.) or a collection (cluster, supercomputer, etc.) of physical system resources. A VRU is characterized by its computational, memory, and communication capacities and by its availability and access policy. Finally, the VRUs are dynamically mapped onto physical system resources at runtime and the SAMR application is tuned for execution within the dynamic computing environment.

Note that the work associated with a VCU depends on the state of the computation, the configuration of the components (algorithms, parameters), and the current ARM objectives (optimize performance, minimize resource requirements, etc.). Similarly, the capability of a VRU depends on its current state as well as the ARM objectives (minimizing communication overheads implies a VRU with high bandwidth and low latency has higher capability). The normalized metric NWM and NRM are used to characterize VRUs and VCUs based on current ARM objectives.

Chapter 4
Addressing Spatiotemporal Heterogeneity

4.1 Overview

Parallel implementations of SAMR applications with highly dynamic and localized features require high SAMR efficiencies[1] [10], which can lead to deep hierarchies and small regions of refinement with unfavorable computation-to-communication ratios. The resulting dynamism and spatiotemporal heterogeneity present significant scalability challenges. In this chapter, we present a SAMR runtime engine, consisting of a stack of partitioners, which can address the space-time heterogeneity and dynamism of the SAMR grid hierarchy. The partitioners build on a locality-preserving space-filling curve based representation of the SAMR grid hierarchy [93] and enhance it based on localized requirements to minimize synchronization costs within a level (*level-based partitioning*), balance load (*bin-packing based partitioning*), or reduce partitioning costs (*greedy partitioner*). The SAMR runtime engine as well as its individual components are experimentally evaluated on an IBM SP2 supercomputer using a 3-D Richtmyer-Meshkov (RM3D) compressible turbulence kernel. Results demonstrate that the techniques presented individually improve runtime performance, and collectively enable the scalable execution of applications with localized refinements and high SAMR efficiencies on large numbers of processors (upto 1024 processors).

[1] AMR efficiency is the measure of effectiveness of AMR and is computed as one minus the ratio of the number of grid points using AMR to that required if a uniform mesh is used.

4.2 Driving SAMR Application – RM3D Kernel

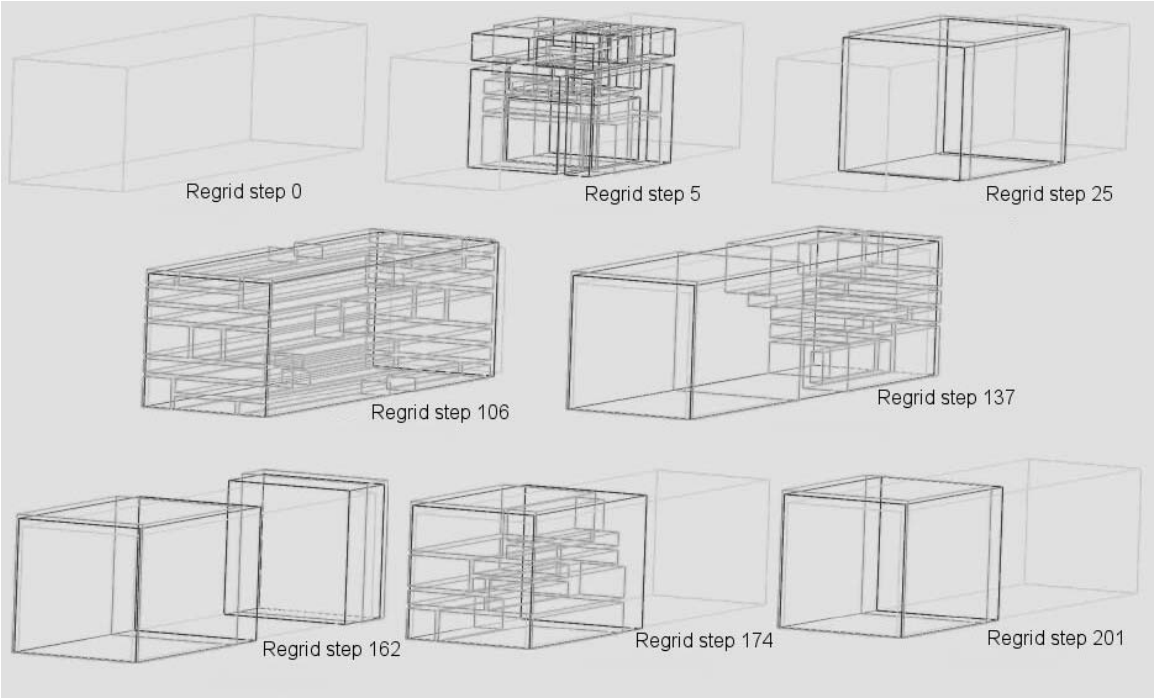

Figure 4.1: Snapshots of the grid hierarchy for a 3-D Richtmyer-Meshkov simulation. Note the dynamics of the SAMR grid hierarchy as the application evolves.

The 3-D compressible turbulence application kernel solving Richtmyer-Meshkov (RM3D) instability is used as the driving SAMR application in this chapter. The RM3D application is a part of the virtual test facility (VTF) developed at the ASCI/-ASAP Center at the California Institute of Technology[2]. The Richtmyer-Meshkov instability 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. The RM3D application is dynamic in nature and exhibits space-time heterogeneity and is representative of the simulations targeted by this research. A selection of snapshots for the RM3D adaptive SAMR grid hierarchy are shown in Figure 4.1.

[2]Center for Simulation of Dynamic Response of Materials - http://www.cacr.caltech.edu/ASAP/

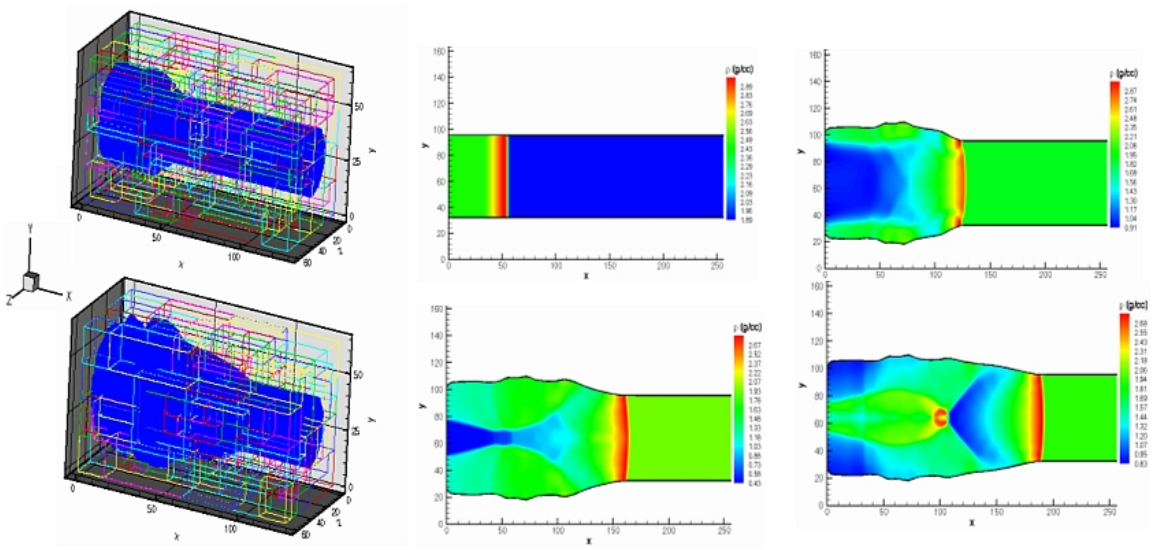

Figure 4.2: Richtmyer-Meshkov detonation in a deforming tube modeled using SAMR with 3 levels of refinement. The Z=0 plane is visualized on the right (Courtesy: R. Samtaney, VTF+GrACE, Caltech).

In particular, RM3D is characterized by highly localized solution features resulting in small patches and deep application grid hierarchies (i.e., a small region is very highly refined in space and time). As a result, the application has increasing computational workloads and greater communication/synchronization requirements at higher refinement levels with unfavorable computation to communication ratios. The physics modeled by the RM3D application for detonation in a deforming tube is illustrated in Figure 4.2. Samples of the SAMR statistics for RM3D execution on 32, 64, and 128

Table 4.1: Sample SAMR statistics for RM3D application on 32, 64, and 128 processors of IBM SP2 "Blue Horizon" using a 128*32*32 coarse grid, executing for 50 iterations with 3 levels of factor 2 space-time refinements and regridding performed every 4 steps.

Application Parameter	32 processors	64 processors	128 processors
AMR efficiency	87.50%	85.95%	85.29%
Avg. blocks per processor per regrid	15	12	9
Avg. memory per processor (MB)	360.64	202.72	106.08
Minimum block size	4	4	4
Maximum block size	128	128	128

processors of IBM SP2 "Blue Horizon" are listed in Table 4.1. The resulting SAMR grid hierarchy characteristics significantly limit RM3D scalability on large numbers of processors. The SAMR runtime engine described in Section 4.3 addresses these runtime challenges in synchronization, load balance, locality, and communication to realize scalable RM3D implementations.

4.3 SAMR Hierarchical Partitioning Framework

The hierarchical partitioning framework within the SAMR runtime engine consists of a stack of partitioners that can manage the space-time heterogeneity and dynamism of the SAMR grid hierarchy. These dynamic partitioning algorithms are based on a core Composite Grid Distribution Strategy (CGDS) belonging to the GrACE [90] SAMR infrastructure [92]. This domain-based partitioning strategy performs a composite decomposition of the adaptive grid hierarchy using Space-filling Curves (SFCs) [57] [80] [100] [109]. SFCs are locality preserving recursive mappings from n-dimensional space to 1-dimensional space. At each regridding stage, the new refinement regions are added to the SAMR domain and the application grid hierarchy is reconstructed. CGDS uses SFCs and partitions the entire SAMR domain into sub-domains such that each sub-domain keeps all refinement levels in the sub-domain as a single composite grid unit. Thus, all inter-level communication are local to a sub-domain and the inter-level communication time is greatly reduced. The resulting composite grid unit list (GUL) for the overall domain must now be partitioned and balanced across processors. However, certain SAMR applications with localized refinements and deep hierarchies (such as RM3D) have substantially higher computational requirements at finer levels of the grid hierarchy. As described in Section 2.3.3, maintaining a GUL with composite grid units may result in high load imbalance across processors in such cases. To address this concern, CGDS allows a composite grid unit with high workload (greater than the load-balancing threshold) to be orphaned/separated

into multiple sub-domains, each containing a single level of refinement. An efficient load-balancing scheme within CGDS can use this "orphaning" approach to alleviate processor load imbalances and provide improved application performance despite the increase in inter-level communication costs.

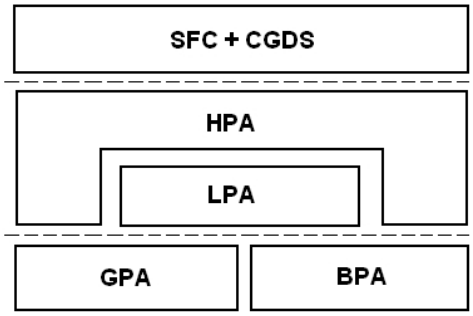

Figure 4.3: Layered design of hierarchical partitioning framework within SAMR runtime engine (SFC: Space-Filling Curves, CGDS: Composite Grid Distribution Strategy, HPA: Hierarchical Partitioning Algorithm, LPA: Level-based Partitioning Algorithm, GPA: Greedy Partitioning Algorithm, BPA: Bin-packing based Partitioning Algorithm).

The layered structure of the SAMR hierarchical partitioning framework is shown in Figure 4.3. Once the grid hierarchy index space is mapped using the SFC+CGDS scheme, higher-level partitioning techniques may be applied in a hierarchical manner using the hierarchical partitioning algorithm (HPA). In HPA [74], processor groups are defined based on the dynamic grid hierarchy structure and correspond to regions of the overall computational domain. The top processor group partitions the global GUL obtained initially and assign portions to each processor sub-group in a hierarchical manner. In this way, HPA further localizes the communication to sub-groups, reduces global communication and synchronization costs, and enables concurrent communication. Within each processor sub-group, higher-level partitioning strategies are then applied based on the local requirements of the SAMR grid hierarchy sub-domain. The objective could be to minimize synchronization costs within a level using the level-based partitioning algorithm (LPA), or efficiently balance load using the bin-packing based partitioning algorithm (BPA), or reduce partitioning costs using the greedy

partitioning algorithm (GPA), or combinations of the above. GPA and BPA form the underlying distribution schemes that can work independently or can be augmented using LPA and/or HPA.

4.3.1 Greedy Partitioning Algorithm

GrACE uses a default greedy partitioning algorithm to partition the global GUL and produce a local GUL for each processor. The GPA scheme performs a rapid partitioning of the SAMR grid hierarchy as it scans the global GUL only once while attempting to distribute the load equally among all processors, based on a linear assignment of grid units to processors. If the workload of a grid unit exceeds processor threshold, it is assigned to the next successive processor and the threshold is adjusted. GPA helps in reducing partitioning costs and works quite well for a relatively homogeneous computational domain with few levels of relatively uniform refinement, and small-to-medium scale application runs. However, due to the greedy nature of the algorithm, GPA tends to result in overloading of processors encountered near the end of the global GUL, since the load imbalances from previous processors have to be absorbed by these latter processors. Scalable SAMR applications require a good load balance during the computational phase between two regrids of the dynamic grid hierarchy. In applications with localized features and deep grid hierarchies, the load imbalance in GPA at higher levels of refinement can lead to large synchronization delays, thus limiting SAMR scalability.

4.3.2 Level-based Partitioning Algorithm

The computational workload for a certain patch of the SAMR application is tightly coupled to the refinement level at which the patch exists. The computational workload at a finer level is considerably greater than that at coarser levels. The level-based partitioning algorithm (LPA) [74] attempts to simultaneously balance load and

minimize synchronization cost. LPA essentially preprocesses the global application computational units represented by a global GUL, disassembles them based on their refinement levels, and feeds the resulting homogeneous units at each refinement level to GPA (or any other partitioning/load-balancing scheme). The GPA scheme then partitions this list to balance the workload. Due to the preprocessing, the load on each refinement level is also balanced.

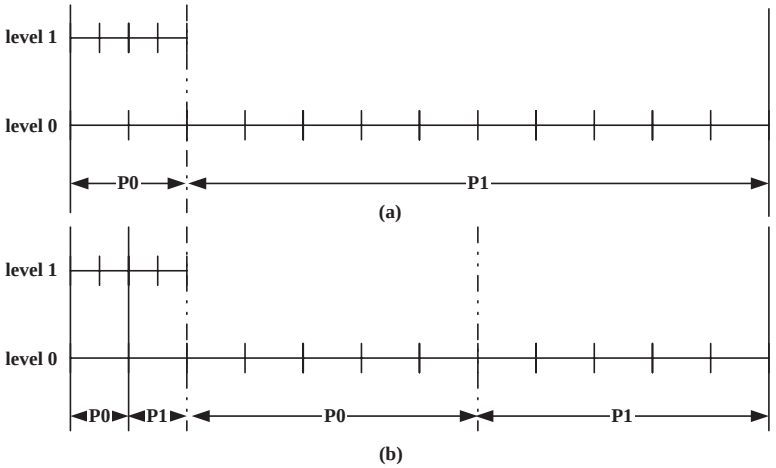

Figure 4.4: Partitions of a 1-D grid hierarchy for (a) GPA, (b) LPA.

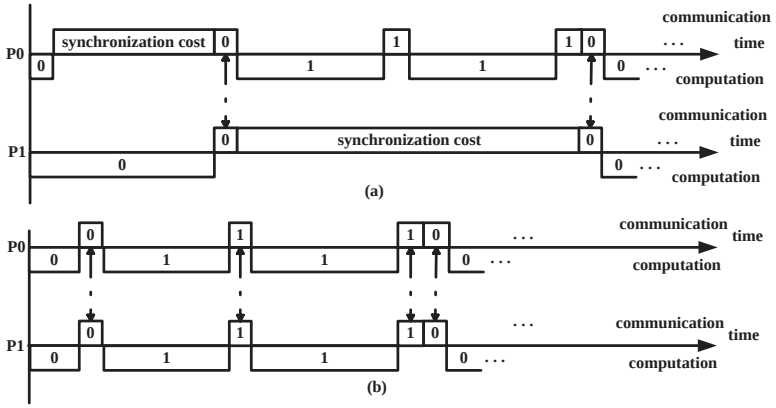

Figure 4.5: Timing diagrams showing computation and communication behaviors for a 1-D grid hierarchy partitioned using (a) GPA, (b) LPA.

LPA benefits from the SFC-based technique by maintaining parent-children relationship throughout the composite grid and localizing inter-level communications, while simultaneously balancing the load at each refinement level, which reduces the

synchronization cost as demonstrated by the following example. Consider the partitioning of a one-dimensional grid hierarchy with two refinement levels, as illustrated in Figure 4.4. For this 1-D example, GPA partitions the composite grid unit list into two sub-domains. These two parts contain exactly the same load: the load assigned to $P0$ is $2 + 2 \times 4$ while the load assigned to $P1$ is 10 units. From the viewpoint of GPA scheme, the partition result is perfectly balanced as shown in Figure 4.4(a). However, due to the heterogeneity of the SAMR algorithm, this distribution leads to large synchronization costs as shown in the timing diagram (Figure 4.5(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 4.4(b) which results in the computation and communication behavior depicted in Figure 4.5(b). As a result, there is an improvement in overall execution time and a reduction in communication and synchronization time.

4.3.3 Bin-packing based Partitioning Algorithm

The bin-packing based partitioning algorithm (BPA) improves the load-balance during the SAMR partitioning phase. The computational workload associated with a GUL at different refinement levels of the SAMR grid hierarchy is distributed among available processors. The distribution is performed under constraints such as minimum block size (granularity) and aspect ratio. Grid units with loads larger than the threshold limit are decomposed geometrically along each dimension into smaller grid units, as long as the granularity constraint is satisfied. This decomposition can occur recursively for a grid unit if its workload exceeds processor threshold. If the workload is still high and the orphaning strategy is enabled, the grid units with minimum granularity are separated into multiple uni-level grid elements for better load balance.

Initially, BPA distributes the global GUL workload among processors based on processor load threshold, in a manner similar to GPA. A grid unit that cannot be

allocated to the current processor, even after decomposition and orphaning, is assigned to the next consecutive processor. However, no processor accepts work greater than the threshold in the first phase. Grid units representing unallocated loads after the first phase are distributed among processors using a "best-fit" approach. If no processor matches the load requirements of an unallocated grid unit, the "most-free" approach (i.e., the processor with least load accepts the unallocated work) is adopted until all the work in the global GUL is assigned.

BPA allows the user to set a tolerance value that determines the acceptable workload imbalance for the SAMR application. In case of BPA, the load imbalance, if any, is low since it is bounded by the tolerance threshold. Due to the underlying bin-packing algorithm, the BPA technique provides overall better load balance for the grid hierarchy partitions among processors as compared to the GPA scheme. However, a large number of patches may be created as a result of multiple patch divisions. Also, the load distribution strategy in BPA can result in multiple scans of the grid unit list that marginally increases the partitioning overheads.

A combined approach using LPA and BPA can provide good scalability benefits for SAMR applications since LPA reduces the synchronization costs and BPA yields good load balance at each refinement level. The experimental evaluation presented in Section 4.4 employs this combined approach. It disassembles the application global GULs into uniform patches at various refinement levels using level-based partitioning, which is subsequently followed by bin-packing based load-balancing for the patches at each refinement level of the SAMR grid hierarchy.

4.4 Scalability Evaluation of the SAMR Runtime Engine

The experimental evaluations of individual components of the SAMR runtime engine such as LPA and BPA, as well as the scalability for SAMR applications are performed using the RM3D compressible turbulence application kernel (described in Section

4.2) on the NPACI IBM SP2 "Blue Horizon" supercomputer. Blue Horizon [18] is a teraflop-scale Power3 based clustered Symmetric Multiprocessing (SMP) system from IBM, installed at the San Diego Supercomputing Center (SDSC). The machine contains 1,152 processors running AIX that are arranged as 144 SMP compute nodes. The nodes have a total of 576 GB of main memory such that each node is equipped with 4 GB of memory shared among its eight 375 MHz Power3 processors. Each node also has several gigabytes of local disk space. Nodes are connected by the Colony switch, a proprietary IBM interconnect.

4.4.1 Evaluation: LPA and BPA Techniques

The LPA and BPA partitioning and load-balancing techniques are evaluated for the RM3D application on 64 processors of Blue Horizon using the GrACE infrastructure. RM3D uses a base grid of size $128 \times 32 \times 32$ and a 3-level hierarchy with factor 2 space-time refinements and regridding performed every 8 time-steps at each level. The RM3D application executes for 100 iterations and the total execution time, synchronization (Sync) time, recompose time, average maximum load imbalance, and the number of boxes are measured for each of the following three configurations, namely: (i) default GrACE partitioner (GPA), (ii) LPA scheme using GrACE, and (iii) the LPA+BPA technique using GrACE, i.e., the combined approach.

Figure 4.6 illustrates the effect of LPA and BPA partitioning schemes on RM3D application performance. Note that the values plotted in the figure are normalized against the corresponding maximum value. The results demonstrate that the LPA scheme helps to reduce application synchronization time while the BPA technique provides better load balance. A combined approach reduces the overall execution time by around 10% and results in improved application performance. The LPA+BPA strategy improves load balance by approximately 70% as compared to the default GPA scheme. The improvements in load balance and synchronization time outweigh the overheads (increase in number of boxes) incurred while performing the optimizations

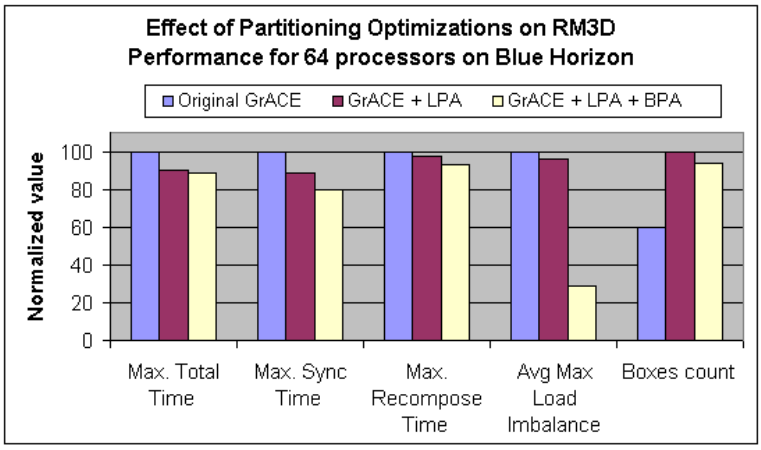

Figure 4.6: Evaluation of RM3D performance (normalized values) for LPA and BPA partitioning schemes on 64 processors of Blue Horizon.

within LPA and BPA. This evaluation is performed on a reduced scale on fewer processors, and hence the performance benefits of the SAMR partitioning framework are lower. However, these hierarchical partitioning strategies become critical while addressing scalable SAMR implementations, as shown in the next section.

4.4.2 Evaluation: Overall RM3D Scalability

The evaluation of overall RM3D SAMR scalability uses a base coarse grid of size $128 \times 32 \times 32$ and the application executes for 1000 coarse level time-steps. The experiments are performed on 256, 512, and 1024 processors of Blue Horizon using a 4-level SAMR hierarchy with factor 2 space-time refinements, and regridding is performed every 64 time-steps at each level. The partitioning algorithm chosen from the hierarchical SAMR partitioning framework for these large-scale evaluations is LPA+BPA since a combination of these two partitioners results in reduced synchronization costs and better load balance, as described in Sections 4.3 and 4.4.1. The orphaning strategy and a local refinement approach are used in conjunction with LPA+BPA in this evaluation since the RM3D application exhibits localized patterns and deep hierarchies with large computational and communication requirements. The minimum block size for a patch on the grid is maintained at 16.

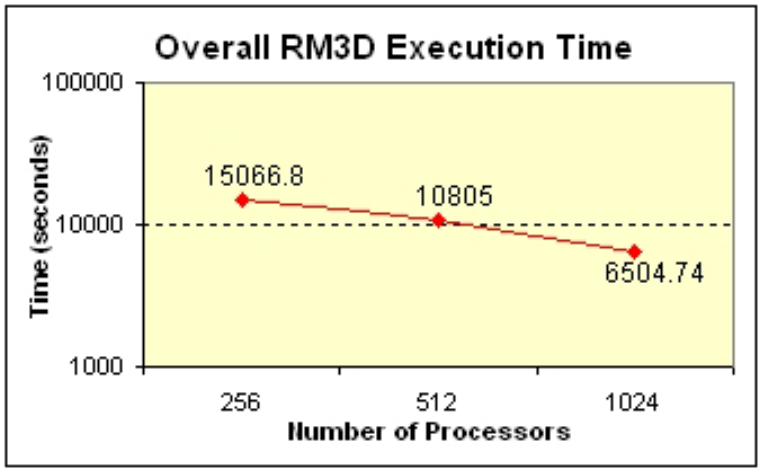

Figure 4.7: Scalability evaluation of the overall execution time for 1000 coarse-level iterations of the RM3D application with a 4-level SAMR hierarchy and $128 \times 32 \times 32$ base grid – log graphs for 256, 512, and 1024 processors of Blue Horizon.

The scalability tests on 256, 512, and 1024 processors measure the runtime metrics for RM3D application. For these three experiments, the overall execution time, average computation time, average synchronization time, and average regridding/-redistribution time are illustrated in Figures 4.7, 4.8, 4.9, and 4.10 respectively. The vertical error bars in Figures 4.8, 4.9, and 4.10 represent the standard deviations of the corresponding metrics. Table 4.2 presents the coefficients of variation[3] (CV) for computation, synchronization and regridding times, and the average time per synchronization and regrid operation.

As seen in Figure 4.7, the scalability ratio of overall application execution time from 256 to 512 processors is 1.394 (ideal scalability ratio is 2) yielding a parallel efficiency of 69.72%. The corresponding values of scalability ratio and parallel efficiency as processors increase from 512 to 1024 are 1.661 and 83.05% respectively. These execution times and parallel efficiencies indicate reasonably good scalability, considering the high computation and communication runtime requirements of the RM3D application. The LPA partitioning and bin-packing based load-balancing techniques

[3]The coefficient of variation is a dimensionless number that is defined as the ratio of the standard deviation to the mean for a particular metric, and is usually expressed as a percentage.

Table 4.2: Scalability evaluation for RM3D application on 256, 512, and 1024 processors of Blue Horizon in terms of coefficients of variation (CV) and additional metrics. The RM3D evaluation is performed for 1000 coarse-level iterations on a $128 \times 32 \times 32$ base grid and a 4-level SAMR hierarchy.

Application Parameter	256 processors	512 processors	1024 processors
Computation CV	11.31%	15.5%	12.91%
Synchronization CV	8.48%	6.62%	7.41%
Regridding CV	2.89%	7.52%	9.1%
Avg. time per sync	0.43 sec	0.38 sec	0.22 sec
Avg. time per regrid	5.35 sec	7.74 sec	9.46 sec

collectively enable scalable RM3D runs with multiple refinement levels on large numbers of processors.

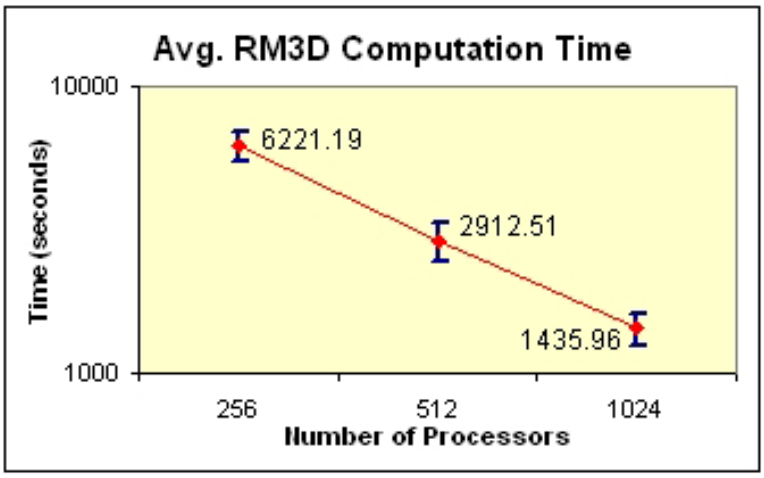

Figure 4.8: Scalability evaluation of the average computation time for 1000 coarse-level iterations of the RM3D application with a 4-level SAMR hierarchy and $128 \times 32 \times 32$ base grid – log graphs for 256, 512, and 1024 processors of Blue Horizon. The vertical error bars are standard deviations of the computation times.

The RM3D average computation time, shown in Figure 4.8, scales quite well. The computation CV in Table 4.2 is reasonably low for the entire evaluation, implying good overall application load-balance provided by the SAMR runtime engine (LPA+BPA strategy). Note that in the case of large numbers of processors, the

RM3D application has few phases in which there is not enough workload in the domain that can be distributed among all processors. In such cases, some processors remain idle during the computation phase which affects the standard deviations of the computation times. Hence, the computation CV for 512 and 1024 processors are slightly higher than for 256 processors, as shown in Table 4.2. However, this lack of computation on large numbers of processors is an intrinsic characteristic of the application and not a limitation of the SAMR runtime engine.

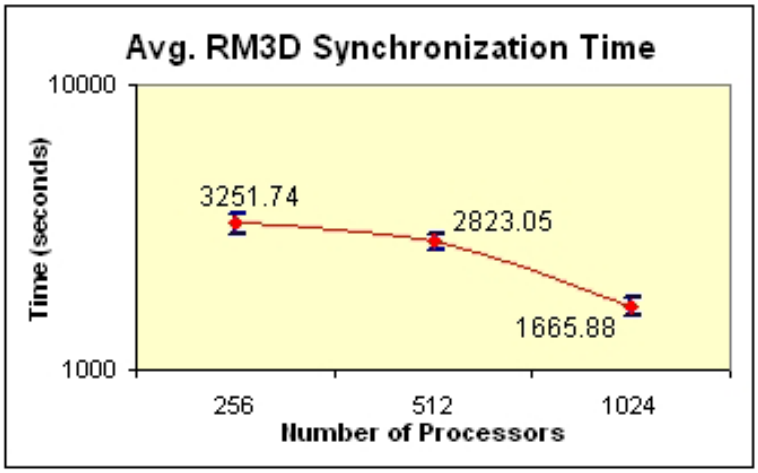

Figure 4.9: Scalability evaluation of the average synchronization time for 1000 coarse-level iterations of the RM3D application with a 4-level SAMR hierarchy and $128 \times 32 \times 32$ base grid – log graphs for 256, 512, and 1024 processors of Blue Horizon. The vertical error bars are standard deviations of the synchronization times.

The scalability of synchronization time is limited by the highly communication-dominated application behavior and unfavorable computation to communication ratios, as described in Section 4.2 and observed in Figure 4.9. The evaluation exhibits reasonably low synchronization CV in Table 4.2, which can be attributed to the communication improvements provided by the SAMR runtime engine (LPA and delayed waits techniques). As observed in Table 4.2, the average time taken per synchronization operation reduces with an increase in the number of processors. This is primarily due to the reduction in size of the grid units owned by processors, resulting in smaller message transfers for boundary updates. The decrease in synchronization time is

proportionately greater when processors increase from 512 to 1024 due to smaller processor workloads, further reduced message sizes, and greater parallelism among message transfers. Moreover, as described earlier, some application phases may not have sufficient workload that can be allocated to all processors during redistribution in case of large numbers of processors, forcing some processors to remain idle. In such cases, this can result in reduced synchronization times since the idle processors do not participate in the synchronization process. However, this is not a limitation of the SAMR runtime engine, but is a direct consequence of the lack of application dynamics.

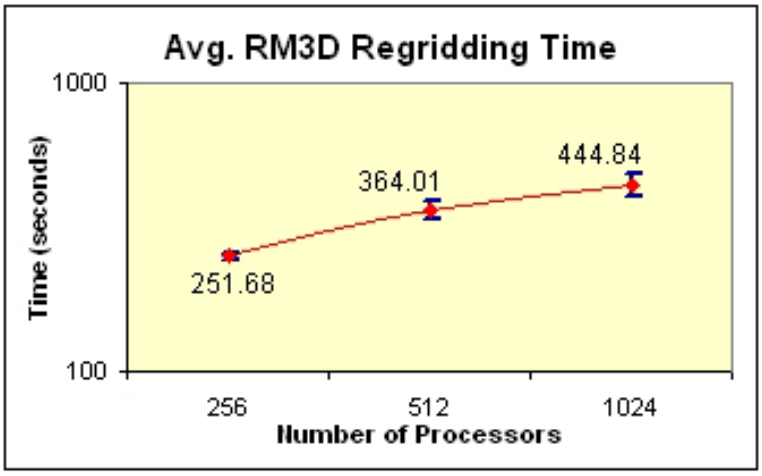

Figure 4.10: Scalability evaluation of the average regridding time for 1000 coarse-level iterations of the RM3D application with a 4-level SAMR hierarchy and $128 \times 32 \times 32$ base grid – log graphs for 256, 512, and 1024 processors of Blue Horizon. The vertical error bars are standard deviations of the regridding times.

The scalability evaluation of the regridding costs for the RM3D application is plotted in Figure 4.10. SAMR regridding/redistribution entails error estimation, clustering and refinement of application sub-domains, global domain decomposition/-partitioning, and reconstruction of the application grid hierarchy followed by data migration among processors to reflect the new distribution. The average regridding time increases from 256 to 1024 processors since it is more expensive to perform global decomposition operations for larger number of processors. However, the partitioning

overheads induced by the SAMR hierarchical partitioning framework are miniscule compared to the average regridding costs. As an example, the overall partitioning overhead for the SAMR runtime engine on 512 processors is 4.69 seconds which is negligible compared to the average regridding time of 364.01 seconds. The regridding CV and the average time per regrid, noted in Table 4.2, show a similar trend as the average regridding time – these metrics increase in value for larger number of processors. To avoid prohibitive regridding costs, the redistribution of the SAMR grid hierarchy is performed less frequently, in periods of 64 time-steps on each level.

This scalability evaluation on 256-1024 processors analyzed the runtime behavior and overall performance for the RM3D application with a 4-level SAMR hierarchy and a $128 \times 32 \times 32$ base grid. Note that a unigrid RM3D implementation for the same experimental settings will use a domain of size $1024 \times 256 \times 256$ with approximately 67 million computational grid points. Such a unigrid implementation will require extremely large overall memory availability (a total of 600 GB) that accounts for the spatial resolution ($1024 \times 256 \times 256$), grid data representation (8 bytes for "double" data), local copies in memory for grid functions and data structures (approximately 50), storage for previous, current, and next time states (total 3 sets), and temporal refinement corresponding to the number of time-steps at the finest level (factor of 8). With 0.5 GB memory availability per processor on Blue Horizon, this unigrid implementation will require 1200 processors to complete execution, which exceeds the system configuration of 1152 processors. Thus, unigrid implementations are not even feasible for large-scale execution of the RM3D application on the IBM SP2. Moreover, even if such unigrid RM3D implementations were possible on large systems, they would entail a tremendous waste of computational resources since the RM3D application exhibits localized refinement patterns with high SAMR efficiencies. Consequently, a SAMR solution that addresses spatiotemporal heterogeneity is the only viable alternative to realize such large-scale implementations, which underlines the primary motivation for this research in this chapter.

4.4.3 Evaluation: SAMR Benefits for RM3D Performance

To additionally evaluate the benefits of using SAMR, another experiment compares the RM3D application performance for configurations with identical finest-level resolution but different coarse/base grid size and levels of refinement. This evaluation is performed on 512 processors of Blue Horizon and all other application-specific and refinement-specific parameters kept constant. Note that for every step on the coarse level (level 0), two steps are taken at the first refinement level (level 1), four steps on level 2, and so on. Each configuration has the same resolution and executes for the same number of time-steps (in this case, 8000) at the finest level of the SAMR grid hierarchy. The RM3D domain size at the finest level is $1024 \times 256 \times 256$ and the evaluation uses the LPA+BPA technique with a minimum block size of 16. The first configuration (4-Level Run) uses a coarse grid of size $128 \times 32 \times 32$ with a 4-level SAMR hierarchy and factor 2 space-time refinements, and executes for 1000 coarse-level iterations which correspond to 8000 steps at the finest level. The second configuration (5-Level Run) uses a 5-level SAMR hierarchy with a $64 \times 16 \times 16$ base grid and runs for 500 coarse-level iterations, corresponding to 8000 steps at the finest level, to achieve the same resolution.

Table 4.3: RM3D performance evaluation of SAMR on 512 processors of Blue Horizon for different application base grids and varying refinement levels. Both experiments have the same finest resolution and execute for 8000 steps at the finest level.

Application Parameter	4-Level Run (128*32*32 base grid)	5-Level Run (64*16*16 base grid)	Performance Improvement
Overall Execution time	10805 sec	6434.62 sec	40.45%
Avg. Computation time	2912.51 sec	1710.15 sec	41.28%
Avg. Synchronization time	2823.05 sec	1677.18 sec	40.59%
Avg. Regridding time	364.01 sec	255.34 sec	29.85%

Table 4.3 presents the runtime metrics for 4-Level and 5-Level configurations and the performance improvement obtained with SAMR. Table 4.4 lists the coefficients

of variation for computation, synchronization and regridding times for the two configurations, and the average time per synchronization and regrid operation. In this evaluation, the error estimator used for determining refinement at each level of the hierarchy is an absolute threshold of 0.005. Note that the quality of solutions obtained in the two configurations are comparable; however, the grid hierarchies are not identical. It is typically not possible to guarantee identical grid hierarchies for different application base grids with varying refinement levels, if the application uses localized refinements and performs runtime adaptations on the SAMR grid hierarchy. However, we reiterate that the two grid hierarchies in this evaluation are comparable, which is validated by the similar average time per regrid (shown in Table 4.4) for 4-Level and 5-Level runs.

Table 4.4: Coefficients of variation (CV) and additional metrics for evaluating RM3D SAMR performance on 512 processors of Blue Horizon using 4 and 5 level hierarchies.

Application Parameter	4-Level Run (128*32*32 base)	5-Level Run (64*16*16 base)
Computation CV	15.5%	14.27%
Synchronization CV	6.62%	7.67%
Regridding CV	7.52%	4.39%
Avg. time per sync	0.38 sec	0.32 sec
Avg. time per regrid	7.74 sec	7.74 sec

SAMR techniques seek to improve the accuracy of the solution by dynamically refining the computational grid only in the regions with large local solution error. The 5-Level Run has fewer grid points in the application grid hierarchy at the finest resolution since the refinements are localized, and hence uses fewer resources than the corresponding 4-Level Run. Consequently, the average times for all runtime metrics shown in Table 4.3 and the CV values for computation and regridding (in Table 4.4) are lower for the 5-Level Run. Though the average time for each synchronization operation is lesser for the 5-Level Run (as seen in Table 4.4), the standard deviation values per synchronization operation are similar for both configurations. Consequently, the

5-Level Run has a relatively higher synchronization CV than the 4-Level Run due to deeper SAMR hierarchies requiring more inter-level communications. As observed from Table 4.3, the overall RM3D execution time shows around 40% improvement for the 5-Level hierarchy due to the efficiency of the basic Berger-Oliger SAMR algorithm. These results corroborate our claim that the RM3D application exhibits localized patterns with high SAMR efficiencies, and can derive substantial performance benefits from scalable SAMR implementations.

Chapter 5
Addressing Computational Heterogeneity

5.1 Overview

Structured grid frameworks typically assume that the computational effort at each grid point is the same and the workload at any level of the grid hierarchy is uniformly distributed, resulting in homogeneous load balancing. However, there exist certain SAMR applications involving reactive flows, such as the simulation of hydrocarbon flames with detailed chemistry [107], where the physical models include transport/-structured processes with uniform computational loads as well as reactive/pointwise processes having varying workloads. In these simulations, the solution of pointwise processes at each iteration requires a different number of sub-cycles to complete the computation at each grid point within a single global timestep. As a result, the computational load varies at different grid points and is only known locally, and at runtime. Therefore, traditional parallelization approaches are not suitable for these simulations, and their parallel/distributed implementations present significant partitioning and runtime challenges due to the inherent computational heterogeneity.

In this chapter, we present the *Dispatch* dynamic structured partitioning strategy for parallel scientific applications with computational heterogeneity. *Dispatch* maintains distributed computational weights associated with pointwise processes, computes local workloads in parallel, and performs in-situ global load balancing to determine processor allocations proportional to the computational weights and data redistribution that preserves application locality. The experimental evaluation of *Dispatch* is performed using an illustrative 2-D reactive-diffusion (R-D) kernel and

demonstrates improvement in load distribution and overall application performance.

5.2 Parallel Formulations of Simulations with Heterogeneous Workloads

5.2.1 Partitioning Challenges for Pointwise Varying Workloads

Parallel structured implementations of PDE-based simulations typically assume a uniform workload per grid point, and use numerical schemes that require all points to march forward, together in time, in lock-step. Such an approach is exact since it preserves the spatial coupling in a straightforward manner. However, there exist certain classes of problems, such as reactive flows, which have physical point processes that are coupled to other similar point processes through a second process. Reaction/chemistry is such a point process – its models (and operators in the evolution equations for reactive flows) do not contain any spatial derivatives. Reactive processes at a point in space affect others around them through convective and diffusive processes, which are separate physical processes and usually operate at different timescales. Thus, over a time period that is far smaller than the transport (convection and diffusion) timescale, the reactive processes can be considered as approximately decoupled. This approximation is exploited in operator-split [77] integration methods.

Operator-split methods are used in PDEs where the physical processes can be approximately decoupled over a global timestep Δt_g. Consequently, the physical processes can be advanced in time over Δt_g using separate integrators since they do not have to be marched in lock-step, and are usually chained in a certain sequence for accuracy reasons. If one of these physical processes happens to be a purely point process, viz. reaction, then separate (systems of) ordinary differential equations (ODEs) for different points in space are obtained. While these ODEs are advanced up to Δt_g

for all points, different points (decoupled point processes) can adopt various independent paths to reach there. Thus, points in the domain with little reaction take a few large timesteps to reach Δt_g, while points with significant reactive processes take miniscule timesteps in order to time-resolve the fastest reactive processes. Although such an approach is efficient as the non-reactive regions do not necessarily march in lock-step with explosive regions, it results in a highly uneven distribution of computational load as a function of space. Any domain partitioner that ignores this uneven load distribution often incurs a stiff load imbalance penalty.

Since the reactive processes are approximately decoupled in space (over Δt_g) and there are no spatial terms in the reaction operator, a conceptually simple solution exists. The grid points are distributed arbitrarily across all processors as long as the loads are equalized. Such a solution has no spatial couplings, and hence does not consider communication costs or preserve the connections between a point and its neighbors. As a result, this approach incurs significant communication costs as the data is redistributed at every global timestep. In combusting flows, where one strives to capture subtle effects of the simulation by preserving as many chemical species as possible (leading to 50-100 variables per grid point), this communication cost can be prohibitive. This motivates the requirement to calculate the reactive processes *in situ*, and achieve load balance by a prudent domain decomposition that incorporates the uneven nature of load distribution.

5.2.2 Reactive-Diffusion (R-D) Application

Combustion applications modeling the properties of hydrocarbon flames [107] are highly complex. The physical processes in such simulations interact in a strongly non-linear fashion and accurate solutions can only be obtained using highly detailed models that include complex chemistry and transport processes [81]. The transport (or structured) processes have uniform computational workloads while the chemical/reactive (or pointwise) processes require different amounts of computation

(i.e., "weights") at each point.

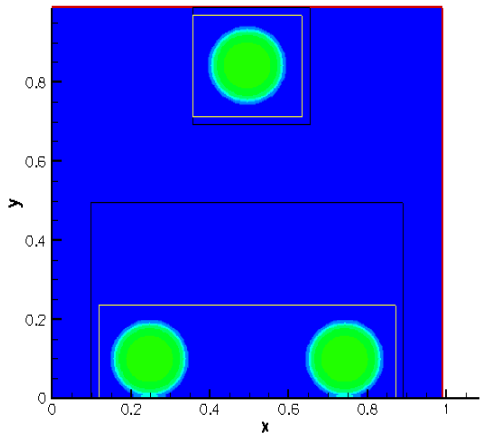

Figure 5.1: 2-D snapshot of R-D kernel's temperature field with 3 hot-spots at time t=50.

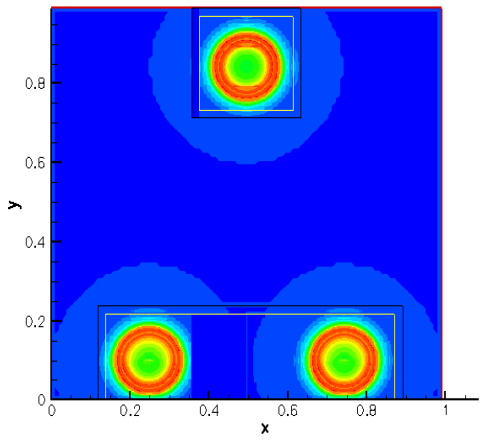

Figure 5.2: 2-D snapshot of R-D kernel's temperature field with 3 hot-spots at time t=100.

A model problem approximating the ignition of a CH_4-Air mixture is used as an illustrative example to evaluate the *Dispatch* partitioning strategy presented in this chapter, and is referred to as the *reactive-diffusion* (R-D) application or kernel. Figs. 5.1, 5.2, and 5.3 are 2-D snapshots of the R-D kernel and illustrate the application dynamics during the ignition of a CH_4-Air mixture in a non-uniform temperature field with 3 "hot-spots" at 50, 100 and 150 timesteps, respectively. The application exhibits high dynamism, space-time heterogeneity and varying computational workloads, and is representative of the class of simulations targeted by this research. Figure

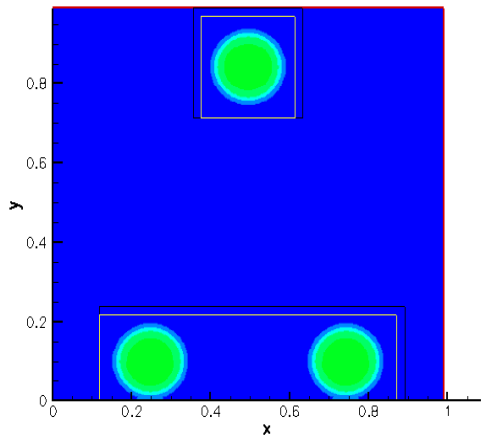

Figure 5.3: 2-D snapshot of R-D kernel's temperature field with 3 hot-spots at time t=150.

5.4 shows a sample distribution of the heterogeneous computational workloads associated with pointwise processes for the R-D kernel on a 128×128 structured grid. The reactive processes near the flame fronts have high computational requirements that correspond to large values of workloads (over 100) at the 3 hot-spots, while the diffusive processes have uniform loads with a value of 1, as illustrated in Figure 5.4.

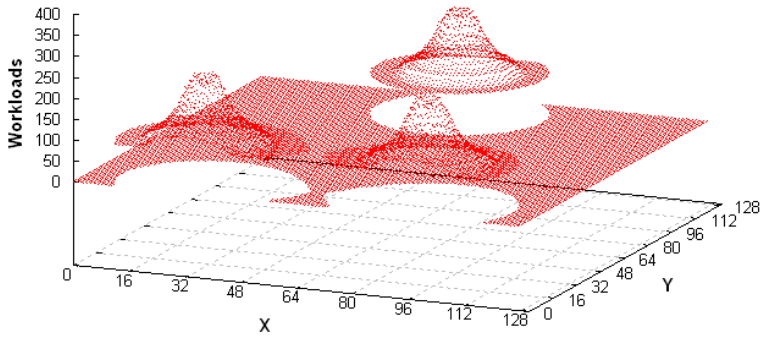

Figure 5.4: Distribution of heterogeneous loads for R-D kernel on a 128×128 structured grid.

Briefly, the R-D kernel solves an equation of the form

$$\frac{\partial \mathbf{\Phi}}{\partial t} = \nabla^2 \mathbf{\Phi} + R(\mathbf{\Phi}) \tag{5.1}$$

where $\mathbf{\Phi}$ is a vector consisting of the temperature and the mass fraction of 26 chemical species at a given point in space. $R(\mathbf{\Phi})$ models the production of heat and chemical

species by 92 reversible chemical reactions [56]. ∇^2 is approximated using second-order central differences. Eqn. 5.1 is evolved in the following manner:

1. Over a timestep of $\Delta t_g/2$, we advance $\mathbf{\Phi}^n$ (solution at timestep n) using $\mathbf{\Phi}_t = \nabla^2\mathbf{\Phi}$ to $\mathbf{\Phi}'$ with Heun's method (second order Runge-Kutta scheme). For this step, the integration is done either on one level for a unigrid implementation or in a recursive manner for all levels in case of SAMR, so that the CFL condition is preserved on each patch of the SAMR hierarchy for the Berger-Oliger formulation [10].

2. Using $\mathbf{\Phi}'$ as initial condition, we solve $\mathbf{\Phi}_t = R(\mathbf{\Phi})$ over Δt_g to get $\mathbf{\Phi}''$. Since there are no spatial coupling terms, this system is solved on a point-by-point basis. At certain points, especially near flame fronts and ignition points, this ODE system exhibits very fast kinetics and has to be advanced using small timesteps (for accuracy reasons). This is done using BDF3 from the CVODE [36] package. This step does not require recursive integration in the case of SAMR, and accounts for the heterogeneity in the application workloads.

3. Using $\mathbf{\Phi}''$ as initial condition, we solve $\mathbf{\Phi}_t = \nabla^2\mathbf{\Phi}$ over $\Delta t_g/2$ to get $\mathbf{\Phi}^{n+1}$, exactly as in Step 1.

5.3 Related Work on Partitioning Heterogeneous Computational Workloads

In [81], Moon et al performed experiments to evaluate the performance of simulations of hydrocarbon flames using Multiblock PARTI (structured) and CHAOS (unstructured) runtime support libraries. The physical processes are classified as structured (e.g., heat conduction) or pointwise (e.g., radiation, chemistry, etc.) processes. As there is no spatial coupling, the block-partitioned data is redistributed across processors to balance the workload of pointwise processes, and is then moved back into

the original locations for the next structured process. This results in a substantial amount of communication as the application redistributes data at every timestep. Heuristic schemes partly alleviate redistribution costs by generating a load balancing plan for each of the participating processors and reducing the communication volume during workload distribution. However, the *Dispatch* scheme, implemented as part of this research and presented in Chapter 5, accounts for heterogeneous workload in the current distribution itself and performs in-situ global partitioning, without requiring additional data migration for load balancing purposes. Moreover, the use of distributed grid functions enables parallel generation of load schedules for participating processors.

Adaptive MPI (AMPI) [58] extends MPI to support processor virtualization and provides dynamic measurement-based load balancing strategies for automatic load redistribution, based on object/thread migration in CHARM++. AMPI is evaluated using an artificial benchmark involving non-uniform 2-D stencil calculations, where the load on 1/16 of the processors is much heavier than on the other 15/16 processors. Unlike AMPI, *Dispatch* addresses adaptive meshing, domain decomposition, and runtime support for scientific applications with computational heterogeneity.

5.4 Dynamic Partitioning for Applications with Computational Heterogeneity

This section presents *Dispatch*, a dynamic structured partitioning strategy for scientific applications with pointwise varying workloads. *Dispatch* has been integrated with the GrACE [94] computational framework and enables parallel uniform and adaptive simulations. *Dispatch* augments the structured grid formulations outlined in Section 2.3 by combining an inverse space-filling curve based partitioner (ISP) [94] with in-situ weighted load balancing using global thresholds. The reactive-diffusion (R-D) kernel, presented in Section 5.2.2, is used to illustrate *Dispatch*. The parallel

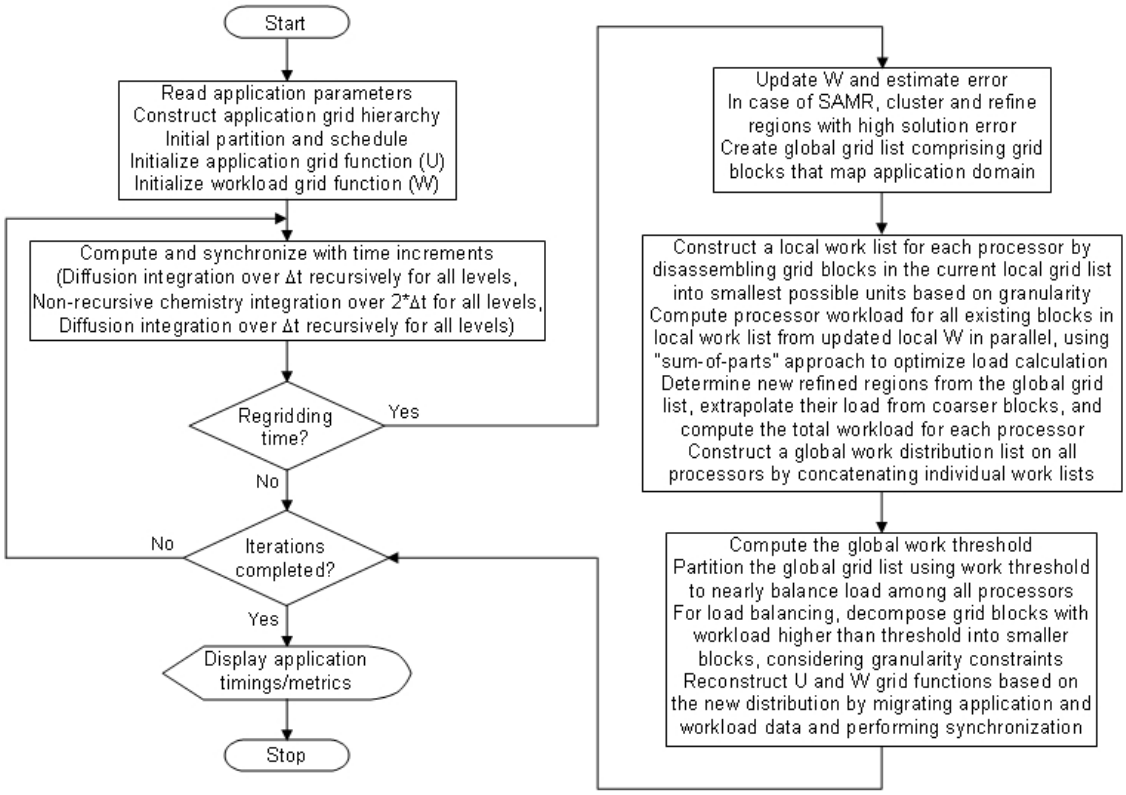

Figure 5.5: R-D kernel execution illustrates the Dispatch scheme for heterogeneous workloads.

execution of the R-D application is shown in Figure 5.5 and described in the next section.

5.4.1 Parallel R-D Application Execution

The execution of the R-D application consists of three major phases: (i) initialization, (ii) computation and synchronization at each timestep, and (iii) periodic load balancing followed by redistribution. The structured grid domain/hierarchy is constructed at the start of the simulation based on input parameters. The initial partitioning is a simple geometric decomposition of the application domain among the available processors. The application grid function (U) and workload grid function (W) are then initialized. Grid functions are distributed entities that represent application variables denoting physical entities (e.g., pressure, temperature, density, etc.), and use the grid

hierarchy as a template to define their structure and distribution.

The computation component consists of two diffusion integration methods over an application timestep Δt. These two integration methods are recursively invoked for each level of the (single level or adaptive) grid hierarchy and are separated by a non-recursive chemistry integration routine over $2 * \Delta t$ for all levels. This is followed by boundary updates and timestepping. During the redistribution phase, computational weights in W corresponding to existing grid points are first updated. In case of SAMR, a truncation error estimate is used to identify regions requiring additional resolution, which are then clustered and refined. As described in Section 5.4.2, a global grid list mapping the entire application domain is created and the *Dispatch* strategy is invoked to dynamically partition the grid. Since unigrid can be viewed as a special case of SAMR with only one level, the description of *Dispatch* focuses on the general SAMR case.

5.4.2 Parallel Workload Computation

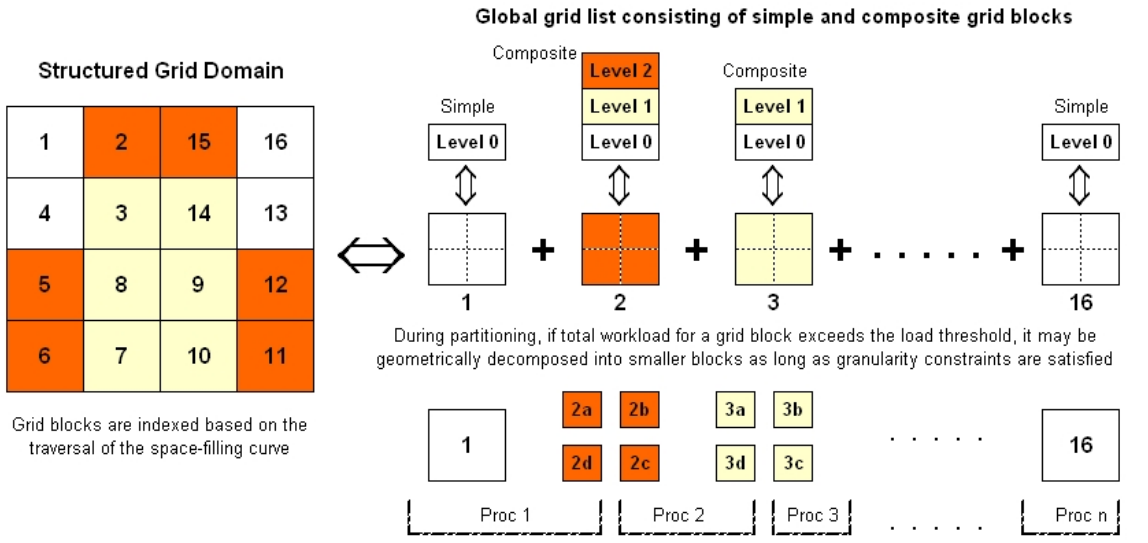

Figure 5.6: The structured grid domain is mapped to a global grid list for load balancing in Dispatch.

As depicted in Figure 5.6, traditional inverse space-filling curve based partitioners

(ISP) [94] preserve application locality by indexing the grid blocks that map the structured grid domain in the order of traversal of the Hilbert space-filling curve (SFC) [109] to form a one-dimensional global grid list. The global grid list consists of simple or composite grid blocks representing portions of the application domain. Simple grid blocks are strictly base grid regions while composite grid blocks contain regions that span multiple levels of refinement in the grid hierarchy. Decomposing a grid block entails a geometrical bisection of the block along all axes as long as the minimum block dimension (granularity) constraints are satisfied. A grid block that attains minimum block dimension is called a "grain". As an example, a 2-D grid block, if divisible, will decompose into 4 smaller blocks. Similarly, a 64×64 size grid block, when recursively decomposed, will result in 256 grains for a minimum application granularity of 4.

Each simple/composite grid block in the global grid list is assigned a cost corresponding to its computational load, which is determined by the load at the grid points contained in the block at each level and the level of the block in the SAMR grid hierarchy. Since the load per grid point is uniform for homogeneous simulations, the total computational work is proportional to the number of grid points and is relatively easy to calculate. However, in the heterogeneous case such as the R-D kernel, the load of a grid block at a level is obtained as the sum of the computational weights in the workload grid function (W) corresponding to the grid points in the grid block at that level.

Since the computational weights for existing grid blocks in W as well as the global grid list may have been updated during regridding, the *Dispatch* strategy first compares the current and previous global grid lists to identify existing grid blocks and new refinement regions. Each processor then operates on its local grid list (portions of the global grid list owned by it) in parallel, and constructs a local work list comprised of grains obtained by disassembling the simple/composite grid blocks in the local grid list. This decomposition of the local grid list is performed to speed up load

calculation using the "sum-of-parts" approach (overall load of the grid block is simply the sum of the loads of all its grains) and to fine-tune the load balancing algorithm (decomposing a grid block during partitioning does not involve recalculation of loads of the constituent parts). The owner computes the updated workload for each grain in the local work list from its corresponding local W, and determines the workloads for existing grid blocks in the list. When a grain contains a newly refined level, the load at each grid point for that level is interpolated as the average workload of its parent (the immediate coarser level). Each processor stores the loads for all grains in its local work list and computes the total processor workload, in parallel. All processors then collectively construct a global work distribution list by concatenating individual local work lists.

5.4.3 Global Load Balancing

The global work threshold is locally computed at each processor from the global work distribution list. *Dispatch* performs domain decomposition by appropriately partitioning the global grid list based on the global work threshold so that the total computational load on each processor is approximately balanced. Processor allocation is based on a linear assignment of loads in the order of occurrence in the global grid list. This is done to preserve application locality which, in turn, reduces communication and data migration overheads during redistribution. If the workload for a grid block in the global grid list exceeds the processor threshold, the block is decomposed (possibly recursively) and replaced by smaller grid blocks whose loads are already known. If the load is still high, the block is assigned to the next available processor and the processor work thresholds are updated. The load imbalance generated during this phase is due to application granularity and aspect ratio constraints for each grid block that need to be satisfied.

The application (U) and workload (W) grid functions are reconstructed after the dynamic partitioning phase based on the new distribution. This step involves

data migration, communication, and synchronization among participating processors, which updates the structure of the grid hierarchy. The new distribution is used in subsequent computation stages until the next regridding stage.

5.5 Experimental Evaluation

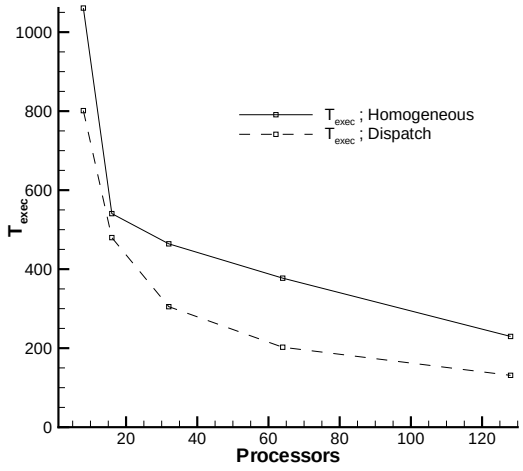

Figure 5.7: Execution time for 200 timesteps of a simulation on a 256×256 uniform grid plotted as a function of number of executing processors. All times are in seconds.

The experimental evaluation of the *Dispatch* strategy is performed using unigrid and SAMR implementations of the 2-D reactive-diffusion (R-D) kernel. The evaluation is performed on the IBM SP4 "DataStar" [39] at the San Diego Supercomputer Center (SDSC). Datastar is SDSC's largest IBM terascale machine that is especially suited for data intensive computations, and has 272 (8-way) P655+ compute nodes with 16-32 GB of memory. The experiments consist of comparing the performance of the *Dispatch* scheme and the default GrACE partitioner (*Homogeneous*) by measuring overall application execution time, load imbalance, synchronization time, and redistribution overheads. The *Homogeneous* strategy assumes that all grid points have the same workload requirements, and hence does not consider computational heterogeneity during load balancing.

5.5.1 Unigrid Evaluation

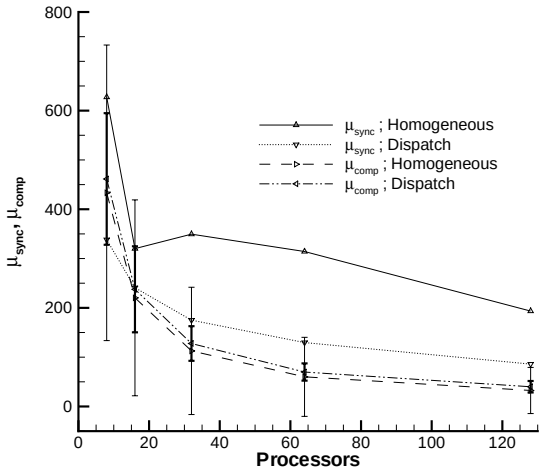

Figure 5.8: Comparison of compute (μ_{comp}) and synchronization (μ_{sync}) times averaged across all processors. Error bars (dark: *Dispatch*; light: *Homogeneous*) are standard deviations of the compute times. All times are in seconds.

The unigrid (1-level) evaluation for the R-D application is performed using *Homogeneous* and *Dispatch* strategies on 8-128 processors on DataStar using 2-D application base grids with resolutions of 256×256 and 512×512. The application executes for 200 iterations, with all other application-specific parameters kept unchanged. Figure 5.7 plots the total execution time $\mathrm{T_{exec}}$ for *Homogeneous* and *Dispatch* schemes. The *Dispatch* scheme improves overall application execution time by 11.23% for 16 processors up to 46.34% for 64 processors. To further analyze load distribution and application runtime behavior, Figure 5.8 plots the average (across all processors) compute (μ_{comp}) and synchronization (μ_{sync}) times for 8-128 processor runs. The standard deviation σ_{comp} in compute time is plotted as error bars. The average compute times are roughly similar for both strategies while the *Dispatch* scheme achieves smaller average synchronization times than the *Homogeneous* scheme. *Dispatch* considers the weights of pointwise processes while performing load balancing and achieves a consistently smaller σ_{comp}. This leads to reduced synchronization times (since processors finish computation closer together in time) and ultimately improved execution times,

as compared to the *Homogeneous* strategy.

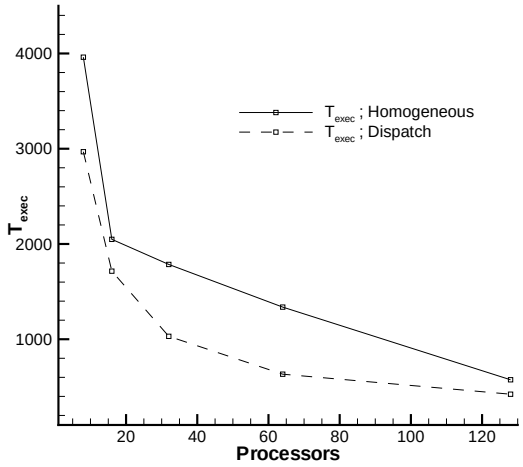

Figure 5.9: Execution time for 200 timesteps of a simulation on a 512×512 uniform grid plotted as a function of number of executing processors. All times are in seconds.

The *Homogeneous* scheme does not repartition the base grid beyond the initial decomposition since it does not consider computational heterogeneity. Using the *Dispatch* strategy does lead to increased partitioning overheads that include the costs to extract the existing pointwise weights and interpolate new ones, compute the weights of local grid blocks for each processor, determine the global workload, and perform an appropriate domain decomposition based on the heterogeneous loads. However, the cumulative partitioning overheads are of $O(10)$ seconds, which is an order of magnitude smaller than the application execution time. Hence, the partitioning overheads for *Dispatch* are considered negligible compared to the performance improvement in the uniform grid case. Figures 5.9 and 5.10 plot the same metrics for a 512×512 uniform grid run, and similar performance improvement is observed as for the 256×256 uniform grid case. *Dispatch* improves application execution times by about 15-50% and provides better load balance. The partitioning overheads are, once again, negligible compared to the overall performance gain. Note that application execution times for 512×512 uniform grid in Figure 5.9 are approximately 4 times larger than

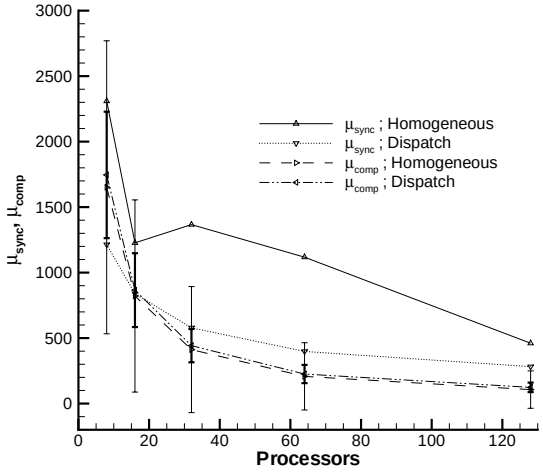

Figure 5.10: Comparison of compute (μ_{comp}) and synchronization (μ_{sync}) times averaged across all processors. Error bars (dark: *Dispatch*; light: *Homogeneous*) are standard deviations of the compute times. All times are in seconds.

the corresponding times for the 256×256 uniform grid, since the domain has been scaled by a factor of 4. Increasing the resolution on uniform grids to obtain greater accuracy can be expensive, and hence SAMR methods are suitable for this class of applications with localized features.

5.5.2 Evaluation of Dispatch SAMR Formulations

2-level SAMR: The *Dispatch* strategy is evaluated using a 2-level SAMR implementation of the R-D kernel on 8-128 processors of DataStar with an application base grid of resolution 512×512 and factor 2 space-time refinement. The application uses a timestep (Δt) value of 2.1875e-9 and executes for 200 iterations, performing 10 regrids. All other application-specific and refinement-specific parameters are kept constant. The minimum block size for this set of experiments is set to 4. Figures 5.11 and 5.12 respectively plot the execution time T_{exec} and the compute μ_{comp} and synchronization times μ_{sync} averaged across processors. Error bars in Figure 5.12 are the standard deviation of the compute times across processors. The *Dispatch* strategy

improves application execution time (upto 32 processors) and achieves a more uniform load balance, though the overall improvement obtained using *Dispatch* reduces as the computation-to-communication ratio (roughly, the ratio of average compute to average synchronization times) for the R-D application decreases. Since the R-D application has localized spiked loads, *Dispatch* generates more patches to yield a better load balance. If there is not enough computation per grid block on each processor and the application is communication-dominated, the *Dispatch* strategy can, in fact, perform worse than the *Homogeneous* scheme. This is seen in the case of 64 and 128 processors for granularity of 4, due to large number of grid blocks created.

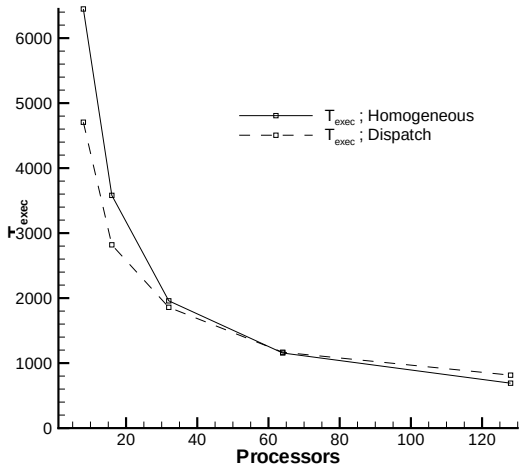

Figure 5.11: R-D application times for a 2-level SAMR hierarchy with a 512×512 base grid executing for 200 timesteps. All times are in seconds. *Dispatch* scheme shows a improvement in execution times, though the improvement reduces as the computation-to-communication ratio decreases.

However, if an appropriate granularity is chosen so that the domain does not contain a large number of tiny blocks, the *Dispatch* strategy can be expected to perform better. Note that this performance variation is a direct result of the application/-workload characteristics and the "computation-communication trade-off", and not a restriction for the *Dispatch* strategy. Furthermore, Table 5.1 shows that *Dispatch* gives better runtime performance as compared to *Homogeneous* for the 2-level SAMR

evaluation on 64 and 128 processors with a base grid resolution of 512×512, if the granularity is set to 32 or 16 instead of 4.

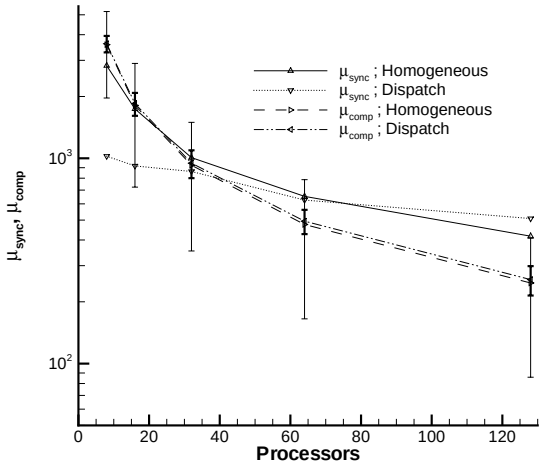

Figure 5.12: Average R-D compute and synchronization times for 512×512 2-level SAMR simulation. Error bars (dark: *Dispatch*; light: *Homogeneous*) are standard deviations of compute times. All times are in seconds.

Table 5.1: 2-level SAMR evaluation of *Dispatch* and *Homogeneous* schemes for R-D application with varying granularity on 64 and 128 processors of DataStar.

Number of processors	Grain size	*Homogeneous* Time (sec)	*Dispatch* Time (sec)
64	4	1155.53	1167.35
	32	1742.11	1436.36
128	4	691.13	814.32
	16	765.63	664.15

3-level SAMR: The *Dispatch* strategy is evaluated for the R-D application on 16 and 64 processors of DataStar using an application base grid of resolution 256×256 and a 3-level SAMR hierarchy with factor 2 space-time refinements. The application uses a timestep (Δt) value of 8.75e-9 and executes for 200 iterations, performing 10 regrids. All other application-specific and refinement-specific parameters are kept constant. The minimum block size for this set of experiments is set to 4 for the 16 processor run and 16 for the 64 processor run. The application execution times

are listed in Table 5.2. The improvements are lower for the 64 processor run due to the higher synchronization costs. However, the 16 processor evaluation has more computational load, and its performance is improved by the *Dispatch* strategy.

Table 5.2: Dispatch 3-level SAMR evaluation for R-D application with 256×256 base grid on 16 and 64 processors on DataStar.

Number of processors	Grain size	*Homogeneous* Time (sec)	*Dispatch* Time (sec)	Percentage Improvement
16	4	1324.57	1120.1	15.44
64	16	678.55	668	1.55

Benefits of SAMR: To additionally evaluate the benefits of using SAMR for the R-D kernel, we present another set of experiments comparing the performance of *Dispatch* and *Homogeneous* schemes for three configurations that have identical finest-level resolution, but different coarse/base grid size and number of refinement levels. Each configuration has 512×512 resolution at the finest level of the SAMR grid hierarchy and executes for 400 finest-level timesteps. The first configuration (Unigrid) is a simple uniform grid case with 512×512 grid resolution executing for 400 iterations. The second configuration (2-Level) is a SAMR hierarchy comprising 2 levels with a coarse grid of size 256×256 and executes for 200 coarse-level iterations, which correspond to 400 steps at the finest level (factor 2 space-time refinement). The final configuration (3-Level) uses a 128×128 base grid for a 3-level SAMR hierarchy and executes for 100 coarse-level iterations, corresponding to 400 steps at the finest level. These experiments are performed on 32 processors of DataStar and the application granularity is set to 4. All other application-specific and refinement-specific parameters are kept constant.

The R-D application execution times as well as average and standard deviation values for computation, synchronization, and regridding times for various unigrid and SAMR configurations are listed in Table 5.3. Table 5.3 also presents the coefficients

Table 5.3: Evaluation of *Dispatch* and *Homogeneous* schemes on 32 processors of DataStar for different R-D application base grids and refinement levels. All configurations have 512×512 finest-level resolution and execute for 400 finest-level steps. *H*:*Homogeneous*, *D*:*Dispatch*.

Application Parameters	Unigrid		2-Level SAMR		3-Level SAMR	
	H	*D*	*H*	*D*	*H*	*D*
Overall execution time (sec)	3546.04	2049.17	607.73	566.83	95.31	77.77
Average compute time (sec)	820.32	880.65	285.73	300.35	36.22	39.27
Standard deviation compute time (sec)	956.21	252.26	163.17	45.05	19.12	5.14
Compute coefficient of variation	1.17	0.29	0.57	0.15	0.53	0.13
Average sync time (sec)	2712.1	1155.89	314.42	254.82	56.99	35.7
Standard deviation sync time (sec)	955.61	248.36	163.58	45.11	18.98	5.18
Sync coefficient of variation	0.35	0.21	0.52	0.18	0.33	0.15
Average regrid time (sec)	0	7.42	6.27	10.69	1.81	2.61
Standard deviation regrid time (sec)	0	4.24	0.28	0.23	0.41	0.14
Regrid coefficient of variation	-	0.57	0.04	0.02	0.23	0.05

of variation[1] (CV) for computation, synchronization and regridding times. The *Dispatch* strategy outperforms the *Homogeneous* scheme in all cases, with significant improvements in standard deviation for compute and sync times. The average compute times for *Dispatch* are slightly higher than for *Homogeneous* due to higher but well-balanced individual loads for each processor, resulting in a higher overall average. However, the standard deviation and CV values for compute times truly reflect the performance gain due to more equitable load distribution. Similar improvements are observed for synchronization times as well. Moreover, the average regrid times for

[1]The coefficient of variation is a dimensionless number that is defined as the ratio of the standard deviation to the mean for a given metric.

Dispatch are only slightly higher than for the *Homogeneous* scheme. Consequently, the overheads for the *Dispatch* strategy are negligible compared to the overall performance improvement. Also, the R-D scheme is amenable for SAMR implementations as can be observed by a factor of 6 improvement in execution times for SAMR configurations with successively higher levels of refinement. However, deep SAMR hierarchies can lead to unfavorable computation-to-communication ratios resulting in performance degradation. As a result, appropriate trade-offs between computation and synchronization components are necessary for efficient SAMR execution.

Chapter 6

Impact of Heterogeneity in Parallel Scientific Components

6.1 Overview

Component-based technologies for scientific simulations provide interoperability and flexibility, and can ideally be exploited to significantly improve application performance, especially in cases where the algorithmic and component behaviors as well as overall application execution are not known a priori. However, the inherent dynamism coupled with the runtime heterogeneity in scientific simulations lead to significant challenges in ensuring algorithmic efficiency, load balancing, and runtime performance management [30]. In such cases, performing runtime adaptation is non-trivial and requires an understanding of the requirements of a particular application state as well as a calibration of the impact of the adaptation on overall performance.

In this chapter, we use runtime calibration to analyze the impact of computational heterogeneity on application performance, and investigate load balancing trade-offs when applied to different orchestrations of component-based scientific simulations. The analysis is based on a 2-D methane-air reaction-diffusion model solved using an operator-split method, which employs an iterative implicit time-integration scheme for the (stiff) reactive terms and creates a variable computational load. A domain-decomposition component for this simulation may be either based on the application structure such as domain geometry, or on the application characteristics such as load heterogeneity. The overarching goal of this research is to enable performance-enhancing runtime adaptations of the load balancing component based on the level of

heterogeneity and application behavior. The characterization of computational heterogeneity and its performance implications, presented in this chapter, are important steps towards achieving this objective.

6.2 Component-Based Scientific Computing

High-performance scientific simulations often leverage the combined expertise of multidisciplinary research teams comprising scientists, engineers, mathematicians, and computer scientists. However, code incompatibilities and lack of standardization among contributing teams lead to significant challenges in managing the complexity and performance of such simulation codes. As a result, component-based software engineering [12] has been widely adopted as the paradigm of choice in designing and managing large-scale scientific simulations. One such approach is the Common Component Architecture (CCA) [12] that offers greater flexibility, interoperability, and reuse in comparison to large, unwieldy monolithic codes.

The CCA paradigm envisages the creation of "components" [24], which generally embody a particular scientific model or a numerical, data-decomposition or I/O functionality. These components are designed with standard, clearly-defined interfaces that help to protect them from external changes in the software environment. Applications can be composed or assembled at runtime from components selected from a component pool. These selected components are peers and can interact with one another only through well-defined interfaces. Therefore, when an application needs to be modified, a single component can be modified (or exchanged for a similar component), without affecting the other components making up the application. Moreover, various compositions of the constituent components can result in different runtime scenarios for scientific applications.

6.3 The Reaction-Diffusion System

Gaseous, combusting flows, in the absence of significant sooting, consist of convection, diffusion and reactions involving a fuel, an oxidizer and a number of intermediate radicals. They are modeled by laws governing the conservation of mass, momentum, energy and the conservation of each of the radicals [84]. Transport processes refer to processes (typically, convection and diffusion) that transport various species (fuel, oxidizer or intermediate radicals) around in the domain of interest. The combustion model presented in this paper does not consider convection, and hence the transport processes are purely diffusive. On the other hand, reaction processes are processes that are governed by chemical reactions occuring in situ, i.e., based entirely on the state at a point. Reaction processes at two adjoining points interact with each other via transport processes.

In this study, we approximate the true behavior (for performance purposes) of this highly non-linear system by a set of reaction-diffusion partial differential equations (PDEs). When solved with realistic chemical mechanisms, this PDE system is plagued by a wide spectrum of timescales, ranging from nanoseconds for reactive processes to tens of milliseconds for transport processes. A common technique to address this is operator-splitting [67] that advances chemistry implicitly, while transport may be dealt with either explicitly or implicitly. In either case, the integrators for chemistry and transport are very different. The implicit chemistry time-advancement leads to higher loads at grid points in reactive regions. A non-linear system is iteratively solved at each grid point; reactive regions with fast processes converge slowly as they have to be time-resolved. This leads to a spatially heterogeneous computational load. We consider the case of an igniting methane-air mixture and use the CFRFS [72, 73] Toolkit, which is a collection of CCA components, to simulate the problem. Two different chemical mechanisms are used, as described later in Section 6.5. Briefly, the

reaction-diffusion system is of the form

$$\frac{\partial \phi_i}{\partial t} = \frac{\nabla P_i \cdot \nabla \phi_i}{Q_i} + R(\phi_j) \tag{6.1}$$

$\phi_i, i = 1 \ldots N_{species} + 1$ at a grid point is the temperature or the mass fraction of $N_{species}$ chemical species at a given point in space. $R(\phi_j)$ models the production of heat and chemical species by reversible chemical reactions. Spatial derivatives are approximated using central finite differences. The case $i = 1$ corresponds to the temperature equation with $P_i = \rho C_p \alpha$ and $Q_i = \rho C_p$, where ρ is the mixture density, C_p is the specific heat at constant pressure of the mixture and α is the thermal diffusivity. For $i = 2 \ldots N_{species} + 1$, $P_i = \rho D_i$ and $Q_i = \rho$, where D_i is the diffusivity of each of the $N_{species}$ species. α and D_i are obtained from a mixture-averaged formulation. ρ, α, D_i and C_p couple the temperature and all the species together. Below, we will refer to $\phi_i, i = 1 \ldots N_{species} + 1$ as Φ. Equation 6.1 can be rewritten as

$$\frac{\partial \Phi}{\partial t} = T(\Phi) + R(\Phi) \tag{6.2}$$

where $T(\Phi)$ contains all the spatial derivatives. The system is time-evolved in the following manner:

1. Over a timestep of $\Delta t_g/2$, we advance Φ^n (solution at timestep n) using $\Phi_t = T(\Phi)$ to Φ' with a second-order Runge-Kutta-Chebyshev scheme [72]. The gradients are computed using a central-difference scheme.

2. Using Φ' as initial condition, we solve $\Phi_t = R(\Phi)$ over Δt_g to Φ''. Since there are no spatial coupling terms, this system is solved on a point-by-point basis. At certain points, especially near flame fronts and ignition points, this ODE system exhibits very fast kinetics and has to be advanced using small timesteps (for accuracy reasons). This is done using BDF3 from the CVODE [36] package. This step accounts for the heterogeneity of workloads.

3. Using Φ'' as initial condition, we solve $\Phi_t = T(\Phi)$ over $\Delta t_g/2$ to get Φ^{n+1}. This is done exactly as in Step 1.

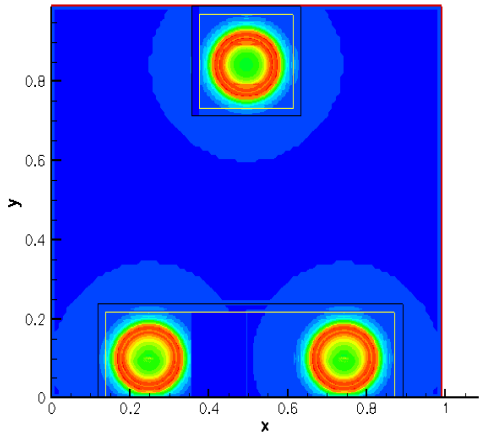

Figure 6.1: Illustrative snapshot of a 2-D methane-air combustion simulation with 3 hot-spots [29].

This scheme is also referred to as diffusion-reaction-diffusion (D-R-D) splitting. A R-D-R splitting is also possible and changes the computational load. The problem is solved in a square domain using a uniform (square) mesh. Zero-gradient boundary conditions (adiabatic system) are enforced. Three high-temperature Gaussian kernels are initialized in a stoichiometric methane-air mixture and serve as the ignition sites, as observed in Figure 6.1.Ignition fronts propagate out into the unburnt gas. Due to the wide spectrum of the timescales of the active (chemical) processes in the ignition front, the iterative solver embedded in the implicit time-integration scheme (Step 1 and 3 above) takes long to converge, as it resolves all the timescales. In contrast, the integration in the burnt and unburnt region is computationally light. Further, the computational load of the diffusion step (Step 1 above) is usually lighter than the reaction step. However, it involves ghost cell updates and incurs communication costs. Such an uneven loading of the domain requires a non-uniform decomposition across processors. Furthermore, the domain has to be redistributed as the ignition fronts propagate.

6.4 Load Balancing for Parallel Reaction-Diffusion Simulations

Parallel simulations of the reaction-diffusion model exhibit conflicting load balancing requirements for reactive and transport processes. The degree of heterogeneity impacts the performance of the domain decomposition strategy and the overall execution. In this section, we discuss load balancing schemes for structured meshes that address two extremes. *Blocked* distribution is a homogeneous scheme based on the domain geometry, whereas the *Dispatch* strategy addresses the computational heterogeneity. Note that these partitioning strategies are two different domain decomposition algorithms that have been integrated into the GrACE [94] infrastructure, which serves as the mesh and data management component in component-based implementations of high-performance scientific simulations.

6.4.1 *Blocked* Distribution

The *Blocked* [94] distribution strategy is based on application geometry and provides a spatially uniform decomposition along each axis of the structured domain. This widely-used scheme is relatively simple, has low overheads, and results in roughly the same number of grid points per processor when the domain can be favorably partitioned. This approach does not consider variations in computational heterogeneity and assumes equal load at each grid point. As a result, uniform resolution (unigrid) simulations do not perform domain redistribution for this scheme, since the domain structure remains unchanged.

6.4.2 *Dispatch* Strategy

Dispatch [29] combines inverse space-filling curve based partitioning (ISP) [94] with pointwise varying, in-situ global load balancing to address the dynamic partitioning and heterogeneous computational requirements of structured uniform or adaptive

scientific applications.

The decomposition of the structured grid hierarchy is performed using space-filling curves (SFCs) [109]. SFCs are locality preserving recursive mappings from N-dimensional space to 1-dimensional space. *Dispatch* uses SFCs to map the entire application domain into a global grid list comprising grid blocks (representing sub-domains that keep all contained refinement levels), as illustrated in Figure 5.6. Moreover, *Dispatch* maintains the loads associated with pointwise reactive processes, which represent computational heterogeneity, using a workload grid function that is distributed among processors. Since *Dispatch* addresses computational heterogeneity that can change as the simulation progresses over time, this strategy is invoked at periodic intervals during application execution to perform redistribution and data remapping among processors.

During redistribution, the *Dispatch* scheme analyzes previous and current decompositions to determine the updated loads for existing grid blocks (representing portions of the application domain) and to generate interpolated workloads for new refinement regions in case of adaptive meshes. Each processor simultaneously computes and stores the loads for all grid blocks in its local work list and determines the cumulative workload for its local workload grid function. All processors construct a global work distribution list by concatenating individual local work lists and compute the global work threshold. The partitioning algorithm then decomposes the global grid list based on the threshold so that the total workload on each processor is nearly balanced. The load imbalance generated during this phase is due to granularity (minimum grid block dimension) and aspect ratio constraints that need to be satisfied. *Dispatch* balances the pointwise varying computational loads across processors and can result in complicated partitions with unequal number of grid points.

6.4.3 Runtime Calibration

To analyze the impact of computational heterogeneity on overall performance, we augment the application reaction component with sensors and timers that enable runtime calibration. The sensors profile the reaction characteristics and express heterogeneity in terms of the number of iterative solves performed at each pointwise process. The timers instrument the chemistry adapter (comprising the reaction compute section) at a fine level and measure the time spent in reaction time-advancement, implicit solves, and other data operations. During redistribution, the calibration component gathers the cumulative reaction computation times for all processors and computes the average, standard deviation, and coefficient of variation (defined as a dimensionless ratio of the standard deviation to the average) metrics for the heterogeneous loads. This coefficient of variation can serve as a useful indicator to analyze the impact of computational heterogeneity on runtime performance, especially in non-intuitive cases.

Note that the runtime calibration for component-based combustion simulations presented in this research are performed by modifying the existing interfaces of various components. Though components should ideally be enhanced with specialized interfaces for obtaining application-specific performance characteristics, these are not available in current implementations, which require significant software engineering efforts to provide these calibration interfaces. For our proof-of-concept study, we focus on using the runtime calibration to analyze the impact of heterogeneity on load balancing and performance of parallel scientific simulations.

6.5 Experimental Evaluation

The experimental evaluation is performed using structured unigrid implementations of the 2-D methane-air reaction-diffusion model. Two different orchestrations of this combustion application can be enabled by using either a reduced chemical mechanism

(R-D kernel) or the more detailed/accurate but computationally heavy GRI 1.2 [48] mechanism (CFRFS kernel). The variation in execution times for the two compositions can be attributed to the different computational requirements of the underlying chemical mechanisms.

Both models are evaluated on 64 processors of "Jacquard" [85] at the National Energy Research Scientific Computing Center (NERSC) using *Blocked* and *Dispatch* load balancing strategies, presented in Section 6.4. Jacquard is a 712-CPU Opteron cluster, arranged as 356 dual-processor nodes, running a Linux operating system. Each processor runs at a clock speed of 2.2GHz and has a theoretical peak performance of 4.4 GFlop/s. Processors on each node share 6GB of memory. The nodes are interconnected with a high-speed InfiniBand network.

The experiments consist of analyzing the computational heterogeneity and comparing the performance of the two partitioners by measuring overall application execution time, spans (defined as the difference between the maximum and minimum values) for reaction and diffusion compute time and synchronization time, and the redistribution overheads. Note that the synchronization time metric in this evaluation is the total time required to complete the synchronization operation and includes processor "wait" time as well as the time taken to perform ghost cell updates. Consequently, higher load imbalances among processors in the computation section can result in larger synchronization spans.

6.5.1 Methane-Air Model using a Reduced Chemical Mechanism

The experimental evaluation of the R-D kernel is performed on a 2-D uniform mesh with 512×512 resolution using a diffusion-reaction-diffusion (D-R-D) splitting strategy. A second-order central difference scheme is used to evaluate the spatial derivatives on the uniform mesh. A reduced methane-air mechanism with 25 species and 92

reversible reactions is used in this experiment. The application granularity is set to 4 and the simulation executes for 200 timesteps, with other application parameters kept unchanged. Due to splitting, a spatially variable load-distribution is achieved, as shown in Figure 6.2. The reactive processes near the flame fronts have high computational requirements that correspond to large values of workloads at the 3 hot-spots (ranging around 100-125), while in the bulk of the domain, they have a value near 1.

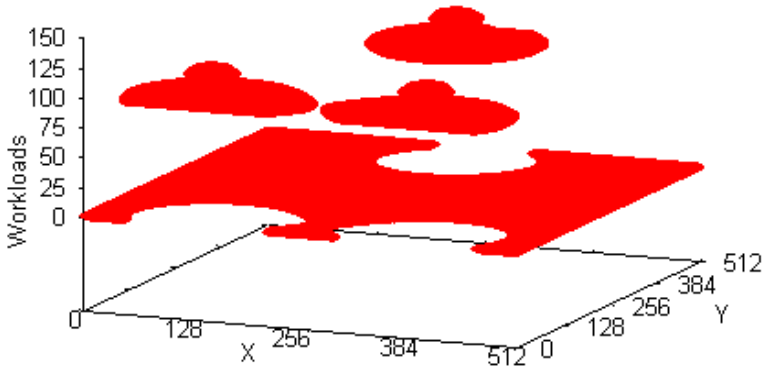

Figure 6.2: Computational load heterogeneity at timestep 78 for the R-D simulation with 512×512 resolution.

Figure 6.3 illustrates the application execution times for the R-D kernel obtained with *Blocked* and *Dispatch* load balancing strategies. The average redistribution time as well as the spans for reaction and diffusion compute times and application synchronization (sync) times are also shown in Figure 6.3. Table 6.1 presents further details on the performance metrics and the heterogeneity analysis for the R-D simulation, viz. the average, standard deviation and coefficient of variation for reaction computation times (μ^R_{comp}, σ^R_{comp}, CV^R_{comp}), diffusion computation times (μ^D_{comp}, σ^D_{comp}, CV^D_{comp}), and synchronization times (μ_{sync}, σ_{sync}, CV_{sync}). Even though the R-D kernel uses a reduced chemical mechanism, the simulation exhibits large variation in computational heterogeneity, as observed in Figure 6.2, and deduced from the high values of reaction span and CV^R_{comp}. The calibration component reports that the pointwise reactive loads vary by an order of nearly 100-125, and the coefficient of variation for the reaction component exceeds 100%.

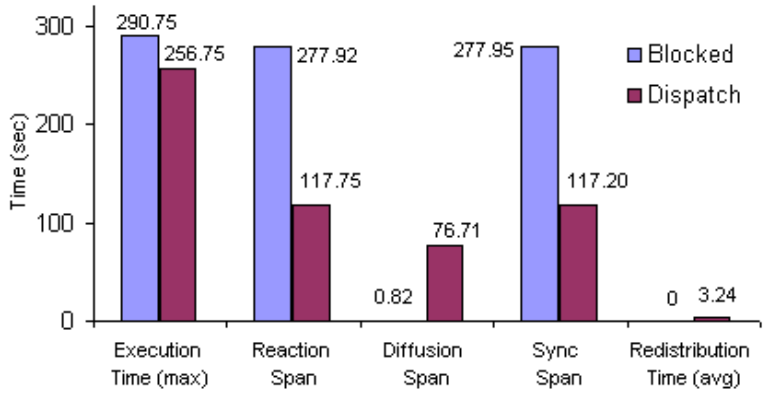

Figure 6.3: Performance evaluation of *Blocked* and *Dispatch* load balancing strategies for the R-D kernel on 64 processors.

In this scenario, the *Blocked* scheme performs decomposition based on domain geometry that results in very high imbalance in the reaction component. This high imbalance is also reflected in the large sync span, since lightly-loaded processors have to wait during the synchronization stage for the overloaded processors to complete their local computations. However, the diffusion component is well-balanced by the *Blocked* strategy since the diffusion phenomenon is based on the number of grid points rather than the load at each point. Also, there are no redistribution costs since the domain structure remains unchanged.

The R-D performance of *Dispatch* is antipodal to that of the *Blocked* strategy. *Dispatch* considers the variation in computational heterogeneity and improves overall R-D performance by about 12%, while maintaining low redistribution overheads (1.26% of total execution time). While *Dispatch* provides better load balance and improved sync times (evident from the low values of CV^R_{comp} and CV_{sync}), the diffusion compute times suffer due to the non-equitable distribution of grid points. However, the magnitude and variation in heterogeneity within the reaction component overshadow the runtime effects of diffusion, and *Dispatch* provides overall better performance as compared to the *Blocked* scheme. Therefore, the *Dispatch* partitioner is well-suited for load balancing parallel simulations of the R-D methane-air model.

Table 6.1: Heterogeneity analysis for *Blocked* and *Dispatch* schemes applied to different models of the 2-D methane-air combustion application on 64 processors.

Runtime Parameters	R-D kernel		CFRFS kernel	
	Blocked	*Dispatch*	*Blocked*	*Dispatch*
μ^R_{comp} (s)	77.13	81.6	1423.18	1590.02
σ^R_{comp} (s)	102.14	28.48	108.19	200.3
CV^R_{comp}	1.327	0.349	0.076	0.126
μ^D_{comp} (s)	9.03	9.15	279.42	314.73
σ^D_{comp} (s)	0.14	15.64	10.01	40.29
CV^D_{comp}	0.016	1.709	0.036	0.128
μ_{sync} (s)	204.42	161.69	13.57	95.94
σ_{sync} (s)	102.11	27.77	9.73	31.09
CV_{sync}	0.5	0.172	0.717	0.324

6.5.2 Methane-Air Model using GRI 1.2

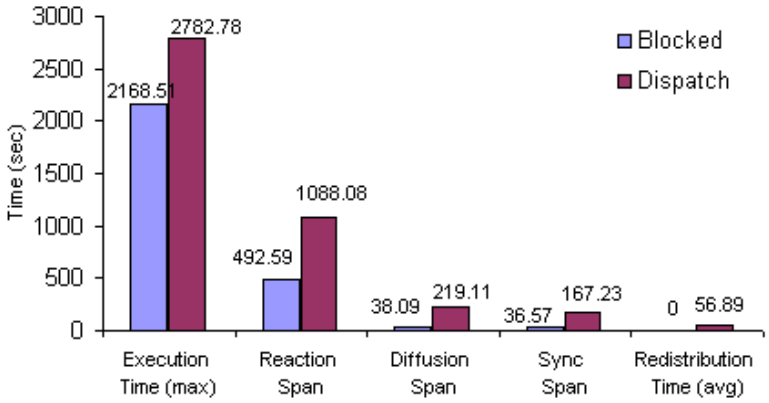

Figure 6.4: Performance evaluation of *Blocked* and *Dispatch* load balancing strategies for the CFRFS kernel on 64 processors.

This experiment is conducted on a 2-D uniform mesh with 500×500 resolution using the CFRFS [72] Toolkit, a component-based toolkit for simulating reacting flows. The simulation executes for 20 timesteps, with other application parameters kept unchanged. We use a reaction-diffusion-reaction (R-D-R) splitting, unlike Section 6.5.1, which increases the overall computational load due to the two reaction steps. The spatial derivatives in the diffusion step are computed using fourth-order

central differences - this also requires one to keep a broader border of ghost cells on each processor and, hence, the application granularity is set to 8. The diffusion coefficients are computed using DRFM [99]. The GRI 1.2 chemical mechanism [48] is used in this evaluation and consists of 32 species and 177 reversible reactions.

The computational load due to chemistry is estimated to be nearly five times more than due to diffusion. Intuitively, one may posit that *Dispatch* should outperform the *Blocked* scheme, based on the empirical analysis of the R-D kernel presented in Section 6.5.1. However, this is not the case, as illustrated in Figure 6.4 that shows the comparative performance of the two load balancing strategies for the CFRFS kernel. The *Blocked* partitioner improves overall application performance by nearly 22% as compared to *Dispatch*. *Dispatch* exhibits larger spans for reaction and diffusion compute times and synchronization times, but maintains low redistribution overheads (2% of the total execution time).

Table 6.1 presents the runtime performance metrics and the heterogeneity analysis for the CFRFS simulation. Though the reaction component dominates diffusion in terms of computation time (μ^R_{comp} is roughly five times greater than μ^D_{comp}), there is low variation among the pointwise varying loads apparent from the low values of CV^R_{comp} (7.6% for *Blocked* and 12.6% for *Dispatch*). Similar low values are observed for CV^D_{comp} as well. The calibration component reports that the pointwise loads differ by a factor of 2 approximately, which is very low compared to the large variation observed in the R-D kernel. Moreover, the reaction coefficient of variation computed by the calibration component ranges around 4-7% during the course of the simulation. In such simulations with relatively low heterogeneity, *Dispatch* may well prove to be an overkill due to more complicated partitions and possibly higher imbalance that can increase both reaction and diffusion component times, resulting in degraded performance. This indicates that the evaluated CFRFS kernel may be better suited to partitioning schemes based on domain geometry, since the simulation has relatively homogeneous computational characteristics, even though the reaction component is

quite compute-intensive.

6.5.3 Inferences

In the experimental evaluation presented above, R-D and CFRFS are combustion kernels that exhibit different execution times due to different complexities of the chemical mechanisms. Moreover, runtime heterogeneity impacts the performance of the domain decomposition component for both kernels in different ways. It can, therefore, be inferred that the overall performance of component-based scientific simulations is highly dependent on the problem characteristics and the implementation and particular connectivity of the components in the simulation code, none of which are known before runtime. This study illustrates these uncertainties that can be introduced by componentization as well as heterogeneity, and motivates the need for an empirical approach using runtime calibration, heuristics or prediction to ensure a degree of performance in such poorly characterized environments.

The research presented in this paper motivates the investigation of a hybrid domain decomposition approach that can select an appropriate load balancing component (from a pool of available load balancers) at runtime, based on calibration of heterogeneity and performance for a particular orchestration of the component-based simulation. Such an approach can be augmented with feedback and other control-theoretic methods to perform runtime adaptations, thereby reducing the uncertainties due to component behaviors and improving overall simulation performance.

Chapter 7

Addressing Heterogeneity and Dynamism in Parallel Reactive Flow Simulations

7.1 Overview

Reactive flows involve the interaction of chemical reactions with fluid flow, and are seen by a wide spectrum of complex physical phenomena in various scientific domains such as combustion, oceanography, atmospheric modeling, astrophysics, particle cosmology, and bio-engineering. The dynamic, and often unsteady, interactions underlying these physical phenomena span several different time and space scales, resulting in significant spatiotemporal complexity. Moreover, the reaction mechanisms, key physical processes, and the strength and type of coupling among processes vary substantially in these systems. As a result, depending on the numerical algorithms used to simulate such flows, the intrinsic interactions can lead to significant computational heterogeneity. This heterogeneity is typically manifested as pointwise varying workloads for chemical (reaction) processes and homogeneous computational requirements for fluid dynamics (diffusion) processes.

Furthermore, the reaction and diffusion components within parallel reactive flow simulations, such as combustion, exhibit incompatible characteristics at runtime. The load balance in the reaction phase is dependent on the average computational load on each processor, which can differ from the domain geometry due to the presence of pointwise varying workloads. On the other hand, the diffusion component assumes uniform requirements at each grid point and benefits from an equitable spatial decomposition of the application domain. While balancing these conflicting requirements is

necessary for efficient execution of combustion applications, identifying appropriate trade-offs is non-trivial [123].

This chapter presents a performance calibration and hybrid partitioning approach for addressing heterogeneity and dynamism in parallel reactive flow simulations on structured grids. The calibration component quantifies the relative performance of the reaction and diffusion processes and the heterogeneity manifested at runtime. The domain decomposition component uses this knowledge to partition the application domain based on spatial structure or load heterogeneity. Experimental evaluation of our approach using a 2-D reactive flow combustion application (CFRFS [72] Toolkit) demonstrates improvement in overall simulation performance.

7.2 Related Work in Runtime Management of Scientific Simulations

In this section, we summarize related efforts that apply multiobjective optimization techniques to support scientific simulations.

PLUM (Parallel Load-balancing for adaptive Unstructured Meshes) [87] is a dynamic load balancing strategy for adaptive unstructured grid computations that uses computation, communication, and data-remapping weights to implement accurate metrics that estimate and compare the computational gain and the redistribution cost of having a balanced workload after each mesh adaptation step. The new partitioning and mapping are accepted if the computational gain is larger than the redistribution cost.

The Unified Repartitioning Algorithm [114] is a parallel adaptive scheme for scientific simulations on unstructured meshes that attempts to compute a repartitioning while minimizing a cost function combining the edge-cut of the partitioning and the total amount of data redistribution. The cost functions are computed for variants of scratch-remap and global diffusion schemes and the one with the lowest cost is

selected.

A multi-constraint algorithm for contact/impact computations [61] reduces the communication overheads across the two computation phases while ensuring that the overall computation remains well-balanced. This strategy models the unstructured mesh as a graph with two vertex weights (for the contact and impact phases) and uses a decision tree along with heuristics to compute hyperplanes that partition the application domain.

In case of simulations on unstructured meshes solved using sparse direct factorization methods, an approach is presented in [83] that uses a predictor-corrector approach to simultaneously balance the number of elements assigned to each processor and the amount of time required to factor the local subproblem using direct factorization. Different refinement algorithms emphasize various parameters controlling edge cut and vertex separators, and the strategy reduces the fill-in of the overweight sub-domains achieving better load balance.

In [103], a general framework for addressing graph partitioning problems is described, in which the work per processor is a complex function of the partition. In this approach, the cost of partition with respect to the desired objective is evaluated and a global schedule for moving cost between sub-domains is determined. Vertex transfers and cost updates are performed until a heuristic threshold is reached. The utility of the framework has been investigated for partitioning to balance overlapped sub-domains and to minimize the sum of computation and communication times.

The majority of these related efforts have been investigated for unstructured meshes. In this part of the research, we focus on the challenges in dynamic partitioning for adaptive structured grid formulations and target scientific simulations with computational heterogeneity, specifically the CFRFS application.

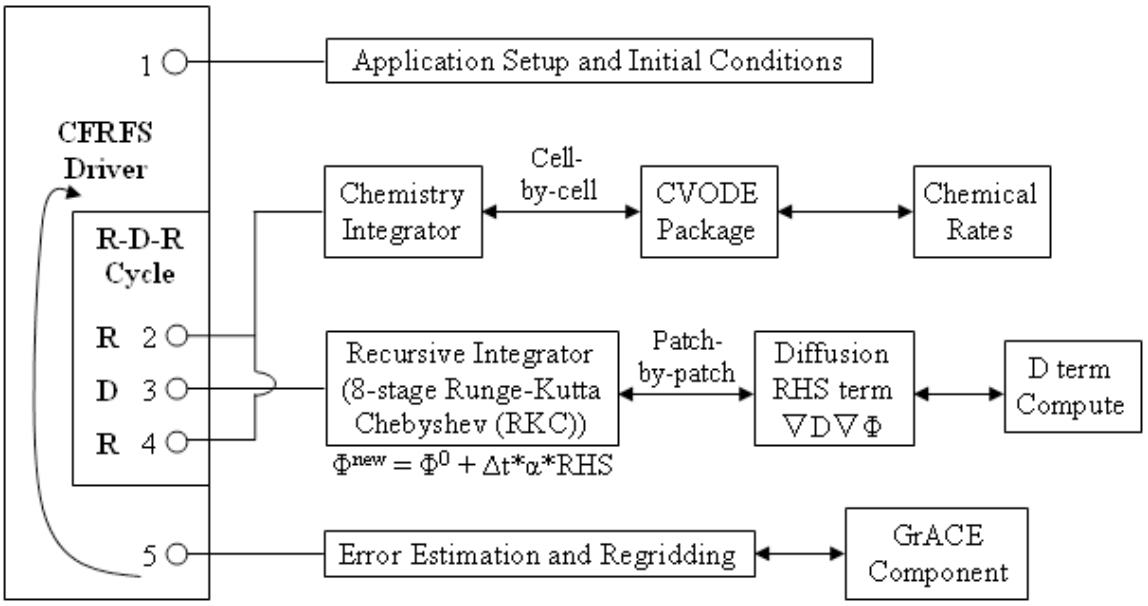

Figure 7.1: The reaction-diffusion cycle within the CFRFS reaction flow application.

7.3 CFRFS Application

The CFRFS combustion application framework accepts user input, initializes the SAMR grid hierarchy and applies initial conditions (step 1). The main component of the CFRFS application is the reaction-diffusion (R-D-R) cycle and is illustrated in Figure 7.1. The CFRFS application uses the operator-split approach for handling the reaction and diffusion components at different timescales. The R-D-R cycle consists of two non-recursive reaction/chemistry integration methods (steps 2 and 4) over an application global timestep $\Delta t_g/2$ for all levels, separated by a diffusion integration routine (step 3) over Δt_g, which is recursively invoked for each level of the (single level or adaptive) grid hierarchy, followed by boundary updates and time-stepping. During the redistribution phase (step 5), computational weights corresponding to existing grid points in the workload grid function are first updated and weights for new refinement regions, in case of SAMR, are extrapolated from their parent blocks. In SAMR formulations, a truncation error estimate is used to identify regions requiring additional resolution, which are then clustered and refined. A global grid list (consisting of grid blocks that correspond to various sub-domains) mapping the entire

application domain is created and the domain decomposition strategy (based on either application geometry or reaction load) is invoked to dynamically partition the grid hierarchy. The application grid hierarchy is then reconstructed and the application execution cycles over to the next iteration at the successive timestep.

CFRFS application geometry impacts the performance of the "Diffusion" component, which consists of the computation of D and synchronization during the recursive integration procedure in Figure 7.1, and involves time and space staggering due to SAMR. Optimal partitioning for "Diffusion" involves optimizing the surface area for grid blocks as well as the shared boundary lengths among participating processors. The computational weights/workloads in CFRFS affect the performance of the "Chemistry/Reaction" component, which is currently optimized in the Dispatch formulation. Optimal partitioning for "Reaction" involves optimizing the workload distribution for grid blocks among participating processors. However, efficient runtime behavior necessitates a blend of these two different objectives, based on previous application performance and current application state.

As illustrated in Figure 7.2, the CFRFS application is constructed using various components for chemistry, diffusion, mesh management, visualization, etc., within the Common Component Architecture (CCA) [12] framework. As described in Chapter 6, we enable performance calibration by augmenting the application reaction component with sensors and timers. The sensors profile the reaction characteristics and express heterogeneity in terms of the number of iterative solves performed at each pointwise process. The timers instrument the chemistry adapter (comprising the reaction compute section) at a fine level and measure the time spent in reaction time-advancement, implicit solves, and other data operations. Note that the performance calibration for CCA-based combustion simulations, presented above, is implemented by extending/-modifying the existing interfaces of various components. Our focus in this paper is on using the runtime calibration to address the impact of heterogeneity on load balancing and performance of parallel reactive flow simulations.

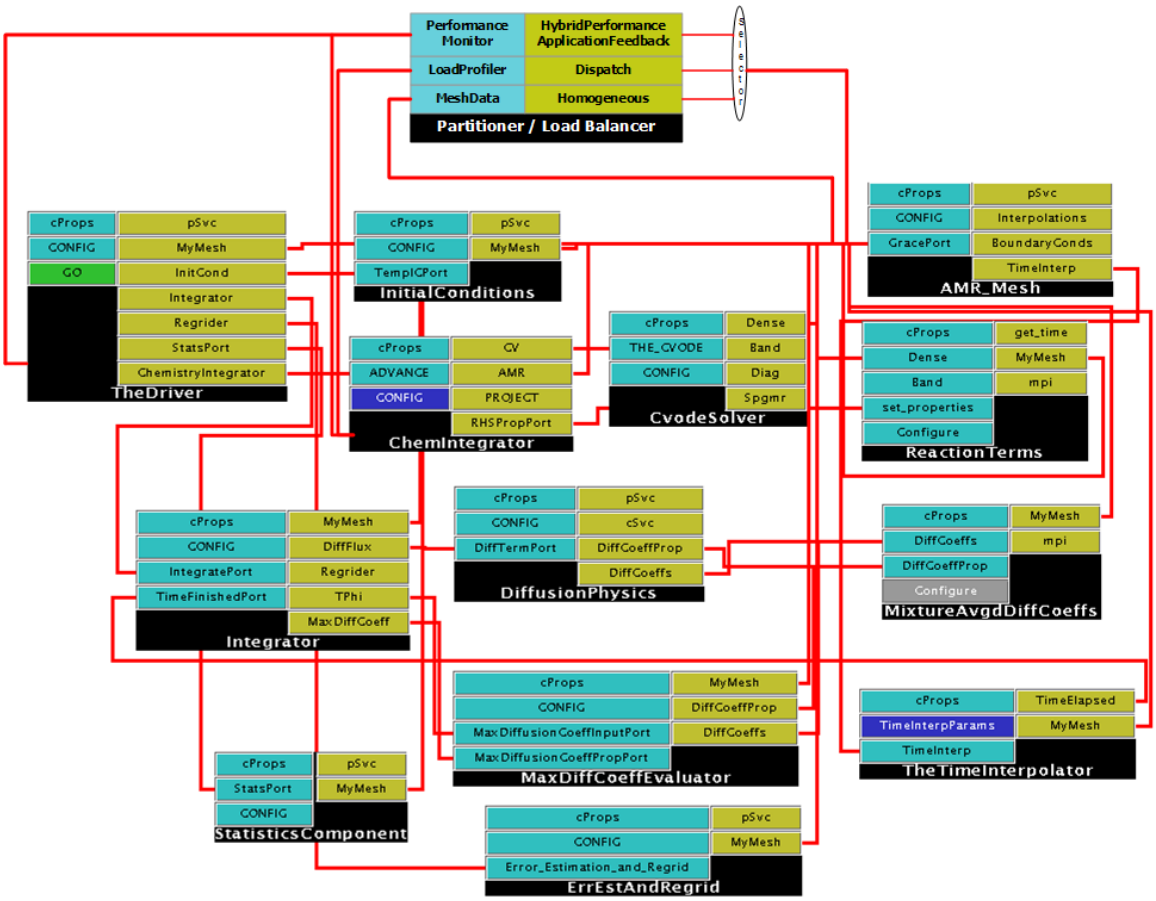

Figure 7.2: Composition of the CFRFS application using various CCA components.

7.4 Recapitulation of Partitioning Algorithms for Parallel Reactive Flow Simulations

In this section, we briefly discuss various partitioners (presented previously in this research) that can manage the spatiotemporal and computational heterogeneity and dynamism in SAMR implementations of parallel reactive flow simulations. These dynamic partitioning algorithms are based on a core Composite Grid Distribution Strategy (CGDS) belonging to the GrACE [90] SAMR infrastructure [92]. This domain-based partitioning strategy performs a composite decomposition of the adaptive grid hierarchy using Space-filling Curves (SFCs) [57] [80] [100] [109], and is based on the application geometry (referred to as "Geometry" in this chapter). The *Geometry* strategy uses SFCs and partitions the entire SAMR domain into sub-domains

such that each sub-domain keeps all refinement levels in the sub-domain as a single composite grid unit. The resulting composite grid unit list (GUL) for the overall domain must now be partitioned and balanced across processors. On the other hand, "Dispatch" [29] combines inverse space-filling curve based partitioning with pointwise varying, in-situ global load balancing to address the dynamic partitioning and heterogeneous computational requirements of structured uniform or adaptive scientific applications. *Dispatch* maintains the loads associated with pointwise reactive processes, which represent computational heterogeneity, using a workload grid function that is distributed among processors.

We have previously presented 3 decomposition algorithms, namely the greedy partitioning algorithm (GPA), bin-packing based partitioning algorithm (BPA) and level-based partitioning algorithm (LPA). The *Geometry* and *Dispatch* strategies use any one of these decomposition algorithms to distribute the SAMR application domain among processors at runtime. The GPA scheme partitions the global GUL and produces a local GUL for each processor, while attempting to distribute the load equally among all processors, using a greedy approach and based on a linear assignment of grid units to processors. The BPA scheme distributes the global GUL workload among processors based on processor load threshold in a manner similar to GPA, but no processor accepts work greater than the threshold in the first phase. Grid units representing unallocated loads after the first phase are distributed among processors using a "best-fit" approach. If no processor matches the load requirements of an unallocated grid unit, the "most-free" approach (i.e., the processor with least load accepts the unallocated work) is adopted until all the work in the global GUL is assigned. The LPA+BPA scheme attempts to simultaneously balance load and minimize synchronization cost. LPA essentially preprocesses the global GUL, disassembles the grid units based on their refinement levels, and feeds the resulting homogeneous units at each refinement level to BPA, which then partitions this list to balance the workload. Due to the preprocessing, the load on each refinement level is

also balanced.

7.5 Hybrid Partitioning Approach

Parallel reactive flow simulations (such as CFRFS) can exhibit conflicting load balancing requirements for reactive and transport processes. Furthermore, the degree of runtime heterogeneity impacts the performance of the domain decomposition strategy and the overall execution. Consequently, the overall runtime performance of a parallel reactive flow application is a function of the problem domain, number of computing elements, and the current application state. To address this issue, we formulate a hybrid partitioning (referred to as "Hybrid") approach, which is illustrated in Figure 7.3 and described below.

7.5.1 Initialization

At the outset, the application domain is partitioned using *Geometry+GPA* scheme, since there is no previous knowledge about the application characteristics. Subsequently, initial conditions are applied and redistribution is performed on the SAMR grid hierarchy. The *Hybrid* scheme starts with an initial partitioner, which is accepted as input from the user, for the first regridding. Since there are no reaction loads as yet in the simulation, the appropriate choice for the initial partitioner can be either *Geometry+GPA* or *Dispatch+GPA* as they would have similar behavior at the initial stage. Moreover, since it may be possible for the application to exhibit substantial pointwise varying workloads before the next regridding stage, it may be preferable to use *Dispatch+GPA* as the initial partitioner.

7.5.2 Analysis of Previous Distribution

We then analyze the previous and current distribution to determine which decomposition strategy should be suitable as part of the next partitioner. Using previously

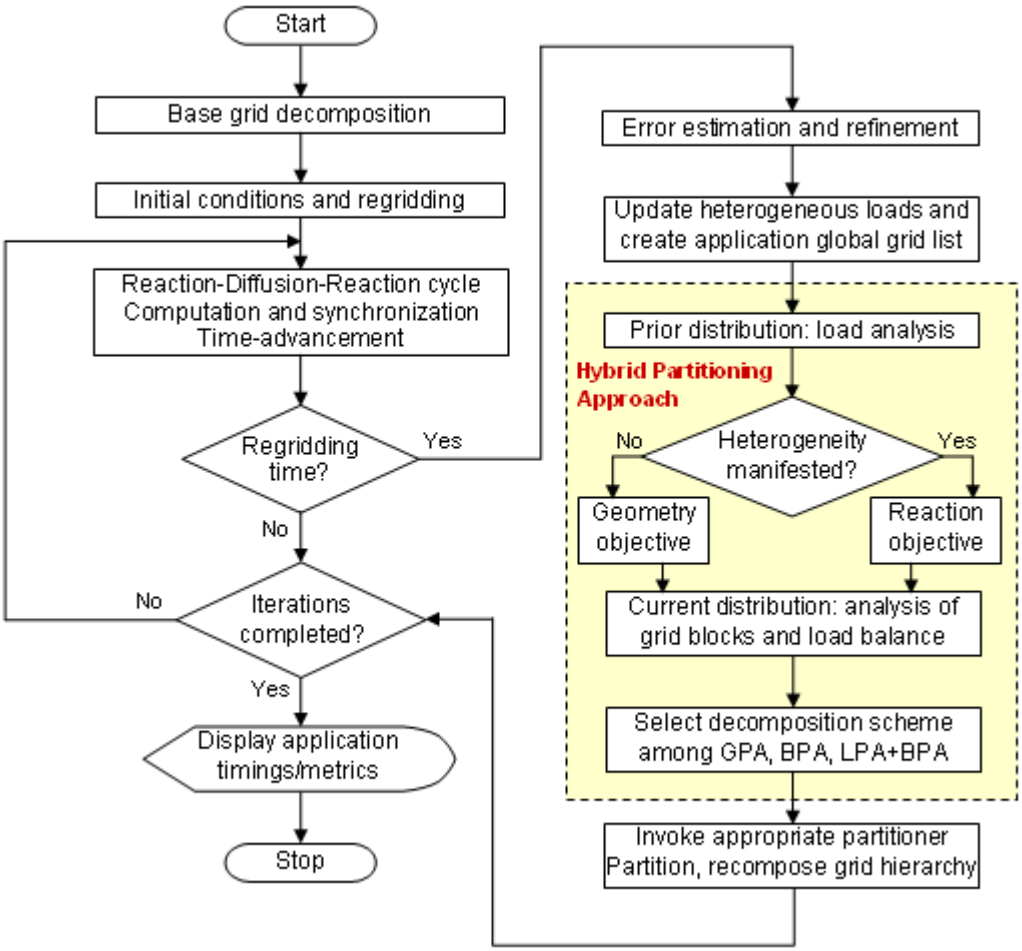

Figure 7.3: Execution flowchart for the CFRFS application illustrating the hybrid partitioning approach.

stored domain information, we compute the average, standard deviation, and coefficient of variation or CV (defined as a dimensionless ratio of the standard deviation to the average) metrics for the processor loads (which may be due to geometry or reaction) in the previous distribution. We also compute the peak-to-average ratio (PAR) for the old distribution since it helps to quantify the maximum imbalance for the previous stage. If the workload is increasing and the PAR is greater than MAX_IMBALANCE_THRESHOLD (set to 25% in our evaluation), we select the BPA decomposition strategy.

7.5.3 Heterogeneity Analysis

For the current distribution, we analyze the loads with respect to geometry (equivalent to number of points in grid blocks) as well as loads with respect to reaction (pointwise varying loads for reactive processes). We compute the PAR values for both geometry and reaction loads normalized with respect to the grid block size at each level of the SAMR hierarchy for the current distribution. The maximum size-normalized PAR at any level for the reaction loads is an indicator of the level of heterogeneity in the simulation at that time. If this level of computational heterogeneity exceeds a predefined threshold, RD_SCALEUP (set to 20% in our evaluation), the simulation exhibits substantial computational heterogeneity and should benefit from the *Dispatch* partitioning scheme. If the computational heterogeneity is less than the threshold, *Geometry* scheme is selected as the next partitioner.

7.5.4 Analysis of Current Distribution

We also determine the number of grid blocks in the current distribution that are at minimum granularity size and have loads that exceed the threshold load at that level. Such blocks can result in high load imbalance and greater synchronization costs, and hence are addressed by using the LPA+BPA decomposition strategy. If the number of such non-decomposable and heavily-loaded grid blocks exceed a preset threshold, BAD_GU_THRESHOLD (set to 15% in our evaluation), then LPA+BPA is selected as the decomposition technique that augments whichever scheme is selected to address heterogeneity (*Geometry* or *Dispatch*).

To analyze the workload behavior, we determine the "work ratio" which is defined as the workload for the current distribution with respect to the metric used in the previous distribution. If the work ratio is greater than the sum of 1 and LOAD_INCREASE_THRESHOLD (set to 30% in our evaluation) and the CV for the previous work is greater than LOW_VARIATION_THRESHOLD (set to 10%

in our evaluation), the application is assumed to heading towards high imbalance and hence BPA scheme is selected as the decomposition strategy of choice. If the work ratio is greater than 1 and CV for the previous work distribution is greater than HIGH_VARIATION_THRESHOLD (set to 30% in our evaluation), the decomposition scheme is upgraded by one level, i.e. from GPA to BPA or from BPA to LPA+BPA. If the work ratio is less than 1 and the previous work CV is less than LOW_VARIATION_THRESHOLD, then the decomposition scheme is downgraded by one level.

7.5.5 Partitioner Selection

Based on the selection of the heterogeneity partitioner (*Geometry* or *Dispatch*) and the decomposition strategy (GPA, BPA, LPA+BPA), the appropriate combination is invoked at runtime as the selected partitioner and domain decomposition is performed. At the next regridding stage, the cycle repeats itself.

7.6 Experimental Evaluation

The experimental evaluation of hybrid partitioning is performed using a 3-level SAMR implementation of the 2-D methane-air reaction-diffusion model using the GRI 1.2 [48] mechanism (referred to as the CFRFS kernel). The experiments are performed on 8 and 64 processors of "Jacquard" [85] at NERSC using the *Geometry* and *Dispatch* strategies along with GPA, BPA and LPA+BPA decomposition schemes, summarized in Section 7.4. The experiments compare the performance of the individual partitioners and analyze the effectiveness of the hybrid approach by measuring overall and normalized application execution time, load imbalance, synchronization time, regridding time, and adaptation overheads.

7.6.1 Test Evaluation on 8 Processors

This experiment is conducted on a 2-D 3-level SAMR mesh with 64×64 base grid resolution using the CFRFS [72] Toolkit, a component-based toolkit for simulating reacting flows. The simulation executes for 10 timesteps, with regridding performed every 4 timesteps and other application parameters kept unchanged, and uses a reaction-diffusion-reaction (R-D-R) splitting. The spatial derivatives in the diffusion step are computed using fourth-order central differences. This also requires a broader border of ghost-cells on each processor and, hence, the application granularity is set to 8. The diffusion coefficients are computed using DRFM [99]. The GRI 1.2 chemical mechanism [48] is used in this evaluation and consists of 32 species and 177 reversible reactions.

Figure 7.4 illustrates the normalized application execution times for 6 individual strategies (namely, *Geometry*+GPA, *Geometry*+BPA, *Geometry*+LPA+BPA, *Dispatch*+GPA, *Dispatch*+BPA and *Dispatch*+LPA+BPA) and 2 hybrid schemes (namely, Hybrid with *Dispatch*+GPA start and Hybrid with *Geometry*+GPA as initial partitioner) that are used in the evaluation of the CFRFS simulation. The normalized spans (defined as the difference between the maximum and minimum values) for reaction compute, diffusion compute, total compute, synchronization (sync) and regrid times for each partitioner are also shown in Figure 7.4.

Note that the different partitioners listed above produce different numbers of grid cells after domain decomposition, since they are typically based on different runtime objectives that may involve either balancing load aggressively or keeping low overhead or managing synchronization costs. Since different placements of cuts produce different application sub-domains for the various partitioners at runtime, we normalize the performance of all partitioners by the number of grid cells produced by a baseline scheme such as *Geometry*+GPA. Note that the normalization does not adversely affect the simulation behavior and is useful for gauging the comparative performance of the individual and hybrid schemes presented in this research.

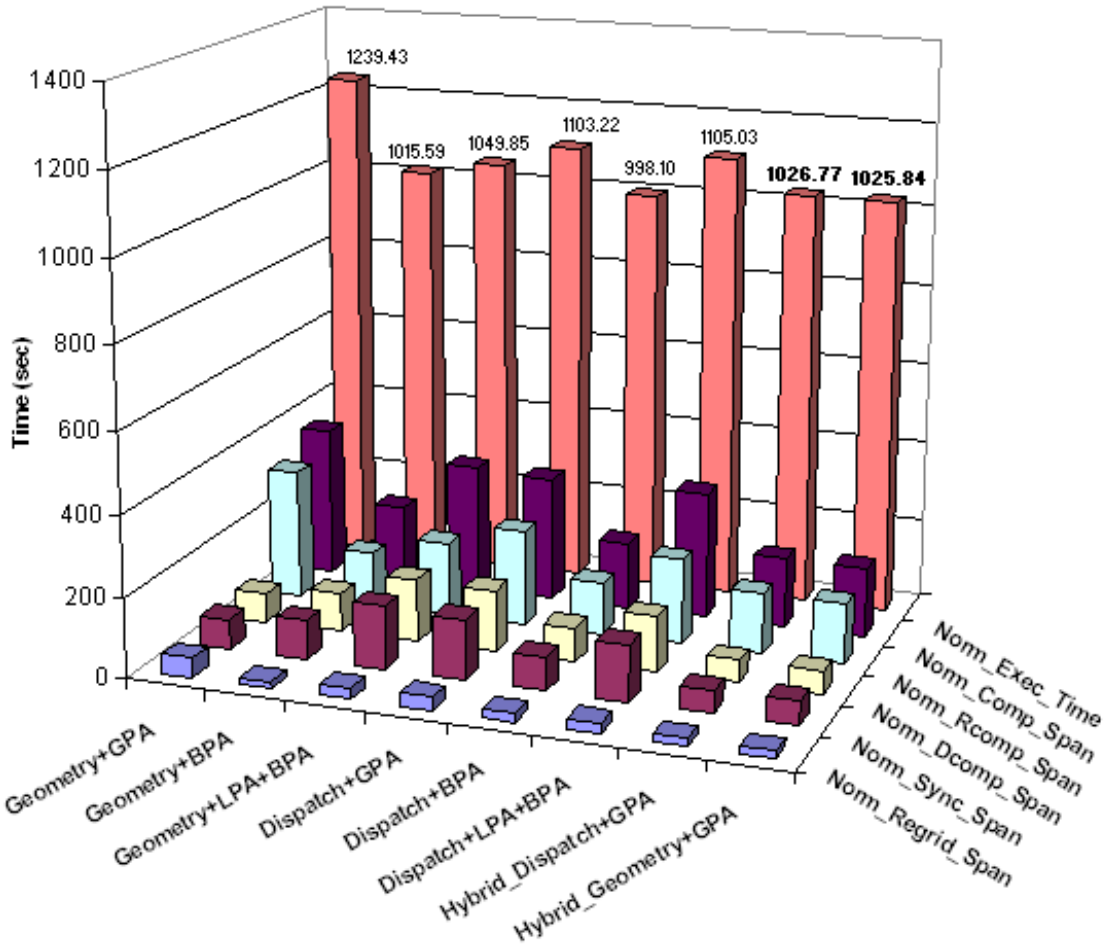

Figure 7.4: Performance comparison of six individual and two hybrid partitioners used in the 3-level SAMR evaluation of the CFRFS application on 8 processors on Jacquard.

The hybrid partitioner performs the initial domain decomposition using either *Geometry*+GPA or *Dispatch*+GPA. Then, at each successive regridding stage, the appropriate partitioner is selected at runtime based on the current and previous application state. Table 7.1 lists the sequence of partitioners selected by the *Hybrid* strategy at runtime for our current CFRFS evaluation executing for 10 iterations with regridding performed every 4 timesteps. Note that during the initialization stages, we do not switch between *Geometry* and *Dispatch* partitioners simply because the reaction loads are not manifested until the application has set up the reactive processes and advanced in time over a couple of timesteps. Therefore, as observed in

Table 7.1, no partitioner state transition occurs until the first time-advancement regrid (at timestep 4). However, the load imbalance characteristics are known while the simulation is performing regridding during initialization and it is possible for the decomposition strategy (GPA or BPA or LPA+BPA) to be adapted during this stage.

Table 7.1: Sequence of partitioners selected at runtime by the *Hybrid* strategy for the 3-level SAMR CFRFS evaluation on 8 processors with 64*64 base grid and executing for 10 iterations.

Regrid Count	Application Stage	*Hybrid* with initial state	
		Dispatch+GPA	*Geometry*+GPA
1	Initialization 1	*Dispatch*+GPA	*Geometry*+GPA
2	Initialization 2	*Dispatch*+BPA	*Geometry*+BPA
3	Timestep 4	*Geometry*+BPA	*Geometry*+BPA
4	Timestep 8	*Geometry*+BPA	*Geometry*+BPA

As observed in Figure 7.4, the best performance among the various partitioners is provided by *Dispatch*+BPA due to the good load balance and also partly due to the large number of grid cells that are a consequence of the complicated partitions created by this scheme. Similar good performance is also observed for *Geometry*+BPA and the hybrid strategy. Since the simulation does not exhibit substantial load heterogeneity and pointwise varying loads for reaction vary by only 4%, the hybrid scheme selects the *Geometry* strategy and then augments it intially with GPA and subsequently with BPA to address load imbalance.

Tables 7.2 and 7.3 present further details on the performance metrics for the CFRFS application using individual and hybrid partitioners respectively. These metrics include the maximum, average, standard deviation, coefficient of variation and the peak-to-average ratio (PAR) for overall execution times (Max_{exec}, μ_{exec}, σ_{exec}, CV_{exec}, PAR_{exec}), total computation times (Max_{comp}, μ_{comp}, σ_{comp}, CV_{comp}, PAR_{comp}), reaction computation times (Max^R_{comp}, μ^R_{comp}, σ^R_{comp}, CV^R_{comp}, PAR^R_{comp}), diffusion computation times (Max^D_{comp}, μ^D_{comp}, σ^D_{comp}, CV^D_{comp}, PAR^D_{comp}), synchronization times (Max_{sync}, μ_{sync}, σ_{sync}, CV_{sync}, PAR_{sync}), and regridding times (Max_{regrid}, μ_{regrid}, σ_{regrid}, CV_{regrid}, PAR_{regrid}).

Table 7.2: Normalized performance metrics for individual partitioners applied to a 3-level SAMR formulation of the CFRFS application, with 64*64 base grid resolution and executing for 10 iterations on 8 processors on Jacquard.

Performance	*Geometry* augmented with			*Dispatch* augmented with		
Metrics	GPA	BPA	LPA+BPA	GPA	BPA	LPA+BPA
Max_{exec} (s)	1239.43	1015.59	1049.85	1103.22	998.10	1105.03
μ_{exec} (s)	1148.92	987.03	1013.29	1046.67	976.43	1051.82
σ_{exec} (s)	38.95	19.48	25.31	35.09	16.15	27.81
CV_{exec}	0.03	0.02	0.02	0.03	0.02	0.03
PAR_{exec}	1.08	1.03	1.04	1.05	1.02	1.05
Max_{comp} (s)	973.89	832.58	925.40	832.98	821.61	937.28
μ_{comp} (s)	693.33	747.47	783.12	667.73	738.87	791.21
σ_{comp} (s)	111.17	66.48	124.57	102.63	56.68	115.13
CV_{comp}	0.16	0.09	0.16	0.15	0.08	0.15
PAR_{comp}	1.40	1.11	1.18	1.25	1.11	1.18
Max^R_{comp} (s)	680.58	535.93	579.63	547.51	512.56	620.53
μ^R_{comp} (s)	429.05	465.72	496.74	409.60	457.53	502.85
σ^R_{comp} (s)	103.27	49.53	67.95	78.21	38.80	70.52
CV^R_{comp}	0.24	0.11	0.14	0.19	0.08	0.14
PAR^R_{comp}	1.59	1.15	1.17	1.34	1.12	1.23
Max^D_{comp} (s)	297.81	322.73	364.43	351.99	316.54	351.85
μ^D_{comp} (s)	264.28	281.75	286.38	258.12	281.34	288.36
σ^D_{comp} (s)	28.90	29.42	58.86	48.88	23.98	49.45
CV^D_{comp}	0.11	0.10	0.21	0.19	0.09	0.17
PAR^D_{comp}	1.13	1.15	1.27	1.36	1.13	1.22
Max_{sync} (s)	286.42	194.18	189.43	265.16	186.23	189.73
μ_{sync} (s)	243.76	138.88	111.98	205.71	137.03	114.83
σ_{sync} (s)	28.58	29.44	58.84	46.64	23.98	49.43
CV_{sync}	0.12	0.21	0.53	0.23	0.18	0.43
PAR_{sync}	1.18	1.40	1.69	1.29	1.36	1.65
Max_{regrid} (s)	49.67	21.58	32.01	41.10	23.36	38.01
μ_{regrid} (s)	38.67	13.04	20.28	26.42	12.40	25.72
σ_{regrid} (s)	14.53	5.90	10.28	10.36	6.05	9.11
CV_{regrid}	0.38	0.45	0.51	0.39	0.49	0.35
PAR_{regrid}	1.28	1.66	1.58	1.56	1.88	1.48

From Table 7.3, we observe that the coefficient of variation (CV) values for all performance metrics, except for regridding times, for the *Hybrid* scheme are among the lowest across all listed partitioning strategies. As a result, we can deduce that the *Hybrid* strategy provides a good domain decomposition and good overall performance, especially since no information about runtime behavior was known a priori. Table 7.3 also lists the adaptation overheads incurred by the hybrid schemes in choosing the appropriate partitioner at a regridding stage. The simplicity and efficiency of our hybrid partitioning approach ensures that these overheads are kept quite low, in the order of 0.2-0.3 milliseconds, which is negligible compared to the regridding and overall execution times for the CFRFS simulation.

Table 7.3: Normalized performance metrics for hybrid partitioners based on *Dispatch* and *Geometry* strategies for 3-level CFRFS application, with 64*64 base grid resolution and executing for 10 iterations on 8 processors on Jacquard.

CFRFS Performance Metrics	*Hybrid* with initial state		CFRFS Performance Metrics	*Hybrid* with initial state	
	Dispatch + GPA	*Geometry* + GPA		*Dispatch* + GPA	*Geometry* + GPA
Max_{exec} (s)	1026.77	1025.84	Max_{comp} (s)	857.93	858.07
μ_{exec} (s)	990.63	989.75	μ_{comp} (s)	744.09	744.18
σ_{exec} (s)	17.42	17.41	σ_{comp} (s)	52.62	52.61
CV_{exec}	0.02	0.02	CV_{comp}	0.07	0.07
PAR_{exec}	1.04	1.04	PAR_{comp}	1.15	1.15
Max^R_{comp} (s)	559.21	559.34	Max^D_{comp} (s)	302.55	302.46
μ^R_{comp} (s)	463.24	463.43	μ^D_{comp} (s)	280.85	280.75
σ^R_{comp} (s)	44.68	44.66	σ^D_{comp} (s)	16.88	16.88
CV^R_{comp}	0.10	0.10	CV^D_{comp}	0.06	0.06
PAR^R_{comp}	1.21	1.21	PAR^D_{comp}	1.08	1.08
Max_{sync} (s)	178.85	178.47	Max_{regrid} (s)	20.60	20.82
μ_{sync} (s)	143.87	143.51	μ_{regrid} (s)	14.39	14.59
σ_{sync} (s)	16.88	16.88	σ_{regrid} (s)	5.72	5.74
CV_{sync}	0.12	0.12	CV_{regrid}	0.40	0.39
PAR_{sync}	1.24	1.24	PAR_{regrid}	1.43	1.43
Adaptation overheads (ms)				0.2-0.3	0.2-0.3

7.6.2 Evaluation on 64 Processors

This experiment evaluates the performance of the *Hybrid* strategy on 64 processors on Jacquard using a 3-level SAMR formulation of the CFRFS application with 512×512 base grid resolution. The simulation executes for 20 timesteps, with regridding performed every 8 timesteps and other application parameters kept unchanged. The partitioners evaluated in this experiment include *Geometry*+GPA, *Dispatch*+GPA, *Dispatch*+BPA, *Dispatch*+LPA+BPA, and the *Hybrid* strategy. The hybrid partitioner performs initial decomposition using *Dispatch*+GPA.

Table 7.4: Performance comparison of individual and hybrid partitioners for 3-level CFRFS simulation on 64 processors with 512*512 base grid and executing for 20 iterations.

Partitioning Strategy	Normalized Max. Execution Time (sec)
Geometry+GPA	5843.80
Dispatch+GPA	5003.93
Dispatch+BPA	4997.99
Dispatch+LPA+BPA	4665.04
Hybrid	4535.67

Table 7.4 lists the performance of the individual and hybrid partitioners for this 64-processor experiment. The *Hybrid* scheme has the lowest overall execution time normalized with respect to the total number of grid cells in the simulation, and outperforms the individual partitioners. The *Hybrid* scheme has low overheads in this experiment, ranging from 0.5 - 0.8 milliseconds for different processors, which is negligible compared to regridding and overall execution times.

As observed in Table 7.5, the *Hybrid* strategy starts with *Dispatch*+GPA as the initial partitioner. At the second initialization stage, though the coefficient of variation (CV) for the previous state is 0.248, the simulation load is decreasing (work ratio is less than 1). Hence, the decomposition strategy is maintained as GPA. During the regrid stage at timestep 8, the maximum normalized peak-to-average ratio for reaction is 1.22 which exceeds the RD_SCALEUP threshold of 1.2. Since there is substantial

Table 7.5: Partitioner selection in *Hybrid* approach for the 3-level SAMR CFRFS evaluation on 64 processors with 512*512 base grid and executing for 20 iterations. Regridding is performed every 8 timesteps.

Application Stage	Partitioner Selected	Max. Normalized PAR for Reaction	Work Ratio	Previous State CV
Initialization 1	*Dispatch*+GPA	-	-	-
Initialization 2	*Dispatch*+GPA	-	0.91	0.248
Timestep 8	*Dispatch*+BPA	1.22	1.73	0.279
Timestep 16	*Geometry*+BPA	1.13	0.96	0.145

heterogeneity manifested in the current application state, the *Dispatch* partitioner is selected. Also, due to increasing workloads and high CV for the previous state, the BPA decomposition strategy is preferred over GPA. Therefore, *Dispatch*+BPA is selected as the partitioner of choice during the regrid at timestep 8. At timestep 16, there is lesser heterogeneity manifested at runtime and the *Geometry* partitioner is selected. Due to to similar load characteristics, BPA is maintained as the decomposition strategy. The experimental evaluation of the hybrid partitioning approach demonstrates that *Hybrid* provides a good domain decomposition and good overall performance, especially with lack of prior knowledge about runtime behavior.

7.7 Inferences

The *Hybrid* partitioning approach presented in this research appears promising and can provide good performance on average (and average performance in the worst case) for parallel reactive flow simulations, especially when application characteristics are not known a priori. However, the applicability of this approach as a general solution for performance optimization of most parallel scientific simulations cannot be easily ascertained, since it is typically not feasible to obtain Pareto optimal solutions for such simulations. Furthermore, to keep adaptation overheads low, any hybrid approach should be lightweight in its complexity and reasonably efficient in its evaluation. As a result, brute force schemes cannot be applied to optimize dynamic adaptive scientific simulations and one has to resort to either statistical inference or machine

learning or predictor-corrector approaches to address multiple conflicting objectives at runtime. As part of future work for the *Hybrid* partitioning approach, one can conduct a sensitivity analysis to study the impact and effectiveness of various runtime parameters that affect simulation performance. Moreover, the *Hybrid* partitioning approach can be augmented with learning mechanisms to predict an appropriate next state based on past experiences and historical observations. However, guaranteeing the optimality and effectiveness of non-heuristic approaches for addressing parallel scientific simulations is non-trivial.

Chapter 8

Algorithmic Adaptations for Reducing Synchronization in Scientific Simulations

8.1 Overview

Scientific simulations modeling complex physical phenomena can be targeted on dedicated massively parallel processing (MPP) systems or commodity computing systems such as clusters. Compared to MPP systems, cluster environments may be heterogeneous and dynamic, aggregating large numbers of dedicated or shared computing and communication resources with varying capabilities. Moreover, clusters typically have lower installation and maintenance costs, better performance per unit cost, and are more feasible for a majority of institutions and research/commercial organizations as compared to MPP systems. Consequently, clusters have emerged as the dominant execution paradigm for high performance distributed computing, comprising 72.2% of the 500 most powerful commercially available computer systems in the world (28th TOP500 [132] List).

Parallel adaptive implementations of scientific simulations are inherently dynamic and heterogeneous and their runtime performance is sensitive to the computation and communication costs. The computation-to-communication ratios (CCR) vary significantly in cluster environments due to system configurations and loads, which can further exacerbate the challenges in partitioning, synchronization and runtime management and impact the overall performance and scalability of scientific applications, especially when computing resources are either shared or heterogeneous. While these performance challenges can be partially addressed for uniform resolution simulations

by using granularity control (blocking) to ameliorate unfavorable computation-to-communication ratios (CCR), this approach is not sufficient. Furthermore, granularity control using predetermined block sizes is not efficient for structured adaptive mesh refinement (SAMR) applications. In such applications, the block sizes should be determined dynamically by the application features to obtain good runtime performance.

This chapter investigates algorithmic adaptations based on computation and communication trade-offs and their impact on runtime performance for unigrid and SAMR formulations of parallel scientific simulations. We present a reformulation of the underlying finite difference algorithm in these simulations to address unfavorable CCR by adaptively trading off reduced synchronization for additional computation when the application is communication-dominated. The goal is to reduce the frequency of synchronization/messaging but still maintain numerical correctness of the finite difference scheme as well as valid domain data. This is achieved by scheduling larger data transfers than required during the synchronization phase and then performing extra computations locally on each processor. Experimental evaluation demonstrates the improvement in overall performance of parallel scientific simulations in cluster environments due to these synchronization adaptations.

8.2 Related Work

Ghost cell expansion, proposed by Ding and He [42] and analyzed in detail in [108], is a form of domain decomposition with overlapping boundaries. Ding and He present a unigrid formulation for finite difference methods that can be used to solve PDEs with a timestep iteration, such as time-dependent PDEs (parabolic or hyperbolic) or elliptic problems solved with a timestep-like iteration (e.g. Jacobi, Gauss-Seidel or ADI). In this formulation, the overlaps (known as "ghost" regions/cells) between subdomains are increased to reduce the frequency of synchronization among processors

in favor of extra local computations.

A similar approach called the SuperBoundary Exchange [69] reduces communication in distributed implementations of iterative computations by sending a larger ghost boundary less often. The algorithm has been implemented and evaluated for uniform 2-D finite difference problems.

We extend the basic premise of "ghost cell expansion" to support reduced synchronization for structured adaptive mesh refinement (SAMR) formulations by allowing for additional local computations of the expanded overlapped regions across various levels of the SAMR grid hierarchy in lieu of greater communication frequency among processors.

8.3 Synchronization Algorithm

Structured grid formulations (either unigrid or SAMR) consider a regular rectangular domain, which is partitioned across processors such that the decomposition overlaps the sub-domains by the width of the spatial stencil. These overlapped regions are known as "ghost" regions/cells. Each local sub-domain on a processor stores in its own ghost cells the values of those adjoining cells in neighboring sub-domains (that may be local or off-processor) that are in the numerical domain of dependence of the sub-domains own cells. The width of this ghost region is defined by the spatial stencil for the formulation. Adjacent to the ghost cells along each dimension in the local sub-domain is a region which corresponds to the neighbor's ghost cells.

Communication is required after every timestep, when the values contained in the ghost cells change. In the communication phase, the local domain sends its local near-boundary values to its neighbors ghost cells, and receives its neighbors local near-boundary values into its own ghost cells. Ghost cell expansion widens the ghost regions by a small number of cells and allows communication to be delayed over as many timesteps as the number of expansion cells. In unigrid and SAMR formulations,

the delay reduces the frequency of communication at the cost of increasing the volume of data in a single transmission as well as incurring additional local computations on processors to maintain numerical correctness. Since the message volume is not an issue for small boundary sizes which are typical for simulations on large numbers of processors, this research focuses on the gain in synchronization overheads at the cost of additional computations.

The primary consideration in the reduced synchronization model for SAMR implementations is that the additional local computations are not performed for those boundaries of sub-domains (referred to as grid patches/blocks) which coincide with the physical domain boundary or an adaptive boundary in case of finer level grids. This is due to the fact that the ghost regions are not expanded at the physical domain boundary or an adaptive boundary for any grid patch. We have applied this synchronization algorithm to both uniform and adaptive formulations of a 2-D application solving the transport equation (Transport2D kernel), described in the following section.

8.4 Transport Equation Model

In physics, chemistry and engineering, a transport phenomenon is any of various mechanisms by which particles or quantities move from one place to another. Three common examples of transport phenomena are diffusion, convection, and radiation. A scalar transport equation is a general partial differential equation that describes transport phenomena such as heat transfer, mass transfer, fluid dynamics (momentum transfer), etc. All the transfer processes express a certain conservation principle. In this respect, any differential equation addresses a certain quantity as its dependent variable and thus expresses the balance between the phenomena affecting the evolution of this quantity. Briefly, the 2-D transport equation is of the form

$$\partial_t u(x,y,t) + c_1 \partial_x u(x,y,t) + c_2 \partial_y u(x,y,t) = 0 \tag{8.1}$$

This equation can be used to model air pollution, dye dispersion, or even traffic flow with u representing the density of the pollutant (or dye or traffic) at position (x,y) and time t.

In this research, we use a structured grid approach to model the transport equation onto a 2-dimensional, regular Cartesian computational domain. We investigate both uniform and SAMR implementations of the 2-D transport model. A finite difference scheme is used to solve the model numerically and time evolution is performed using the MacCormack (predictor-corrector) method. Furthermore, for sake of simplicity and numerical correctness, boundary conditions (at the physical domain or an adaptive boundary) are not applied within the Transport2D model since these conditions do not allow synchronization requirements to be relaxed across successive timesteps due to the required boundary update procedures. Note that the boundary constraints are restrictions posed by the underlying numerical formulation to maintain solution correctness and not a limitation of the synchronization algorithm.

8.5 Reduced Synchronization Adaptation for 2-D Transport Model

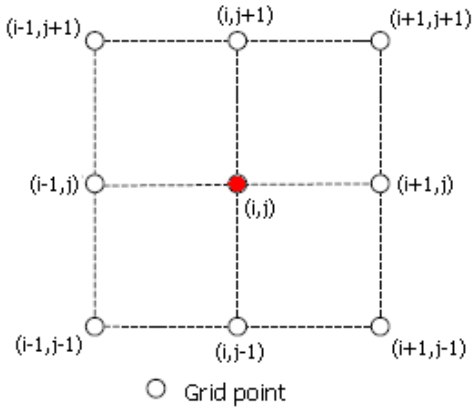

Figure 8.1: 9-point stencil used by the finite difference scheme within the 2-D transport application.

The finite difference scheme within the Transport2D application uses a 9-point

spatial stencil of radius 1 (ghost region width is set to 1), as shown in Figure 8.1. Each grid point computes its updated value using its current value and the values obtained from 1-away neighboring grid points. In case of our reduced synchronization algorithm illustrated in Figure 8.2, we expand the size of the ghost regions to 2 since the Transport2D application is communication-dominated, and reduce the communication requirements by alternating the synchronization calls within the predictor and corrector computation sections, while maintaining numerical correctness.

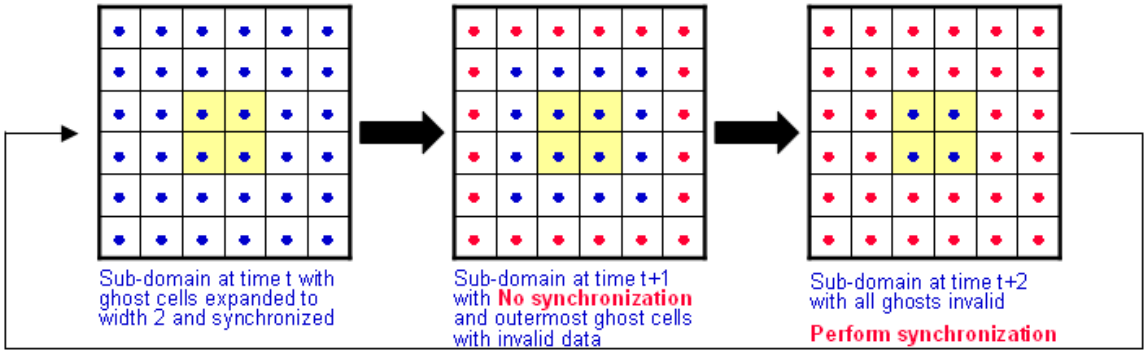

Figure 8.2: Operation of the reduced synchronization algorithm illustrating synchronization and computation trade-offs.

At timestep t at any level, assume that all domain data is valid and all ghost regions are synchronized. Since the ghost width in our new approach is 2, we perform additional local computations on the expanded ghost cell regions for timestep t when these regions are valid. At the next timestep $(t+1)$, synchronization is skipped and the computation region is shrunk by 1 to avoid the outermost ghost cells that are invalidated due to lack of synchronization. However, the interior domain data (local to a processor) remains unaffected and can be correctly computed. At the successive timestep $t+2$, the entire ghost region contains invalid data and the synchronization phase needs to be reapplied on the expanded ghost region to obtain the correct ghost data from the interior domain data for the neighboring processor. The application state after synchronization at timestep $t+2$ is now identical to the synchronized state at timestep t, and the cycle repeats itself.

Note that that the expansion factor for the ghost region is set to 2 in our evaluation since it is constrained by the space-time refinement factor used to create the SAMR grid hierarchy. Refinements in space create finer level grids, which have more grid points than their parents (coarser grids), and hence greater computational loads. Refinements in time mean that finer grids take smaller timesteps, and hence have to be advanced and synchronized more often. Due to restriction (fine to coarse grid data transfer and interpolation) operations occurring in the SAMR hierarchy, the parent grids (grids at a coarse level) receive updated data from child grids (grids at one level finer than the parent) and need to synchronize their boundaries with other parent grids at the same level to ensure numerical correctness. The frequency of restriction across levels is determined by the space-time refinement factor, which is set to 2 in our evaluation. Therefore, the ghost cell expansion factor is also set to 2 and is the maximum factor by which synchronization requirements can be relaxed without affecting the numerical correctness of the simulation. Note that the load at each grid point is homogeneous in the Transport2D evaluation.

8.6 Experimental Evaluation

The experimental evaluation of our reduced synchronization approach is performed using structured unigrid and adaptive implementations of the Transport2D kernel that solves the transport equation in 2-D. The experiments are performed on the 64-node "Frea" cluster at Rutgers University and on "Jacquard" which is a 712-CPU Opteron cluster at NERSC. We compare the performance (in terms of overall execution) of the reduced synchronization and traditional (original) lock-step approaches, and verify the numerical correctness of the reduced synchronization algorithm.

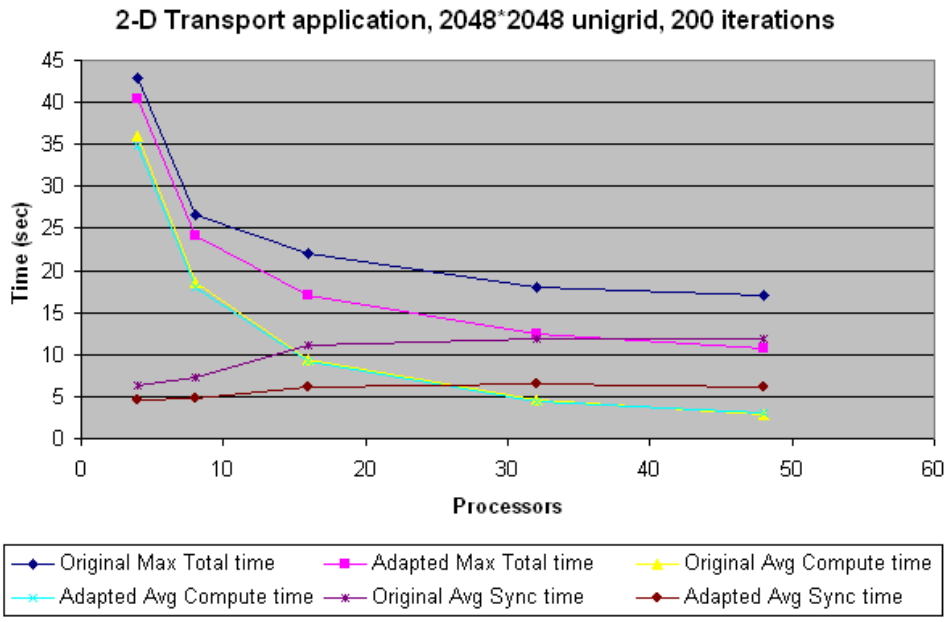

Figure 8.3: Performance of the reduced synchronization algorithm for 2048 × 2048 unigrid Transport2D application on 4-48 processors on Frea.

8.6.1 Unigrid Evaluation

The first experiment evaluates the performance of the original and reduced synchronization schemes for a unigrid implementation of the Transport2D application on 4-48 processors on Frea. The application uses a 2048 × 2048 grid and executes for 200 iterations (or timesteps). Figure 8.3 illustrates the performance of the reduced synchronization scheme. Since the application is communication-dominated, there is overall improvement in the simulation performance due to the reduction in the communication times.

8.6.2 2-level SAMR Evaluation

The second experiment evaluates the performance of the original and reduced synchronization schemes for a 2-level SAMR formulation of the Transport2D application on 8 processors on Frea. The application uses a 128 × 128 base grid and executes for

Table 8.1: Performance comparison of original and reduced synchronization algorithms for a 2-level Transport2D application with 128×128 base grid on 8 processors on Frea.

Execution Parameter	Original Scheme	Reduced Synchronization	Percentage Improvement
Max. Execution Time (sec)	17.49	12.323	29.54%
Max. Compute Time (sec)	0.06	0.062	-3.33%
Min. Compute Time (sec)	0.039	0.042	-7.69%
Avg. Compute Time (sec)	0.052	0.056	-7.69%
Total Sync Count	535	334	37.57%
Max. Sync Time (sec)	14.911	9.442	36.68%
Min. Sync Time (sec)	13.574	8.491	37.45%
Avg. Sync Time (sec)	14.386	9.012	37.36%

50 iterations with regridding performed every 8 timesteps. Table 8.1 and Figure 8.4 illustrate the performance and numerical correctness of our approach respectively. Due to the small application domain and lack of dynamics, the computation time for the Transport2D application is dwarfed by the synchronization time. Since the simulation is heavily communication-dominated, the reduced synchronization algorithm helps to lower the communication overheads by reducing the number of synchronization calls across timesteps in favor of additional local computations performed at each processor due to expanded ghost regions. As observed in Table 8.1, the overheads of extra computations range from approximately 3-8% while the improvement in communication costs is nearly 37%. As a result, the overall simulation performance improves by nearly 30% due to the reduced synchronization approach. Furthermore, this approach maintains the numerical correctness of the solution (similar to the original scheme), as demonstrated by the coincident graphs for the maximum, minimum and second norm values obtained at each level for the simulation, illustrated in Figure 8.4.

8.6.3 3-level SAMR Evaluation

The next set of experiments compare the performance of the original and reduced synchronization schemes for different configurations of a 3-level SAMR formulation of the

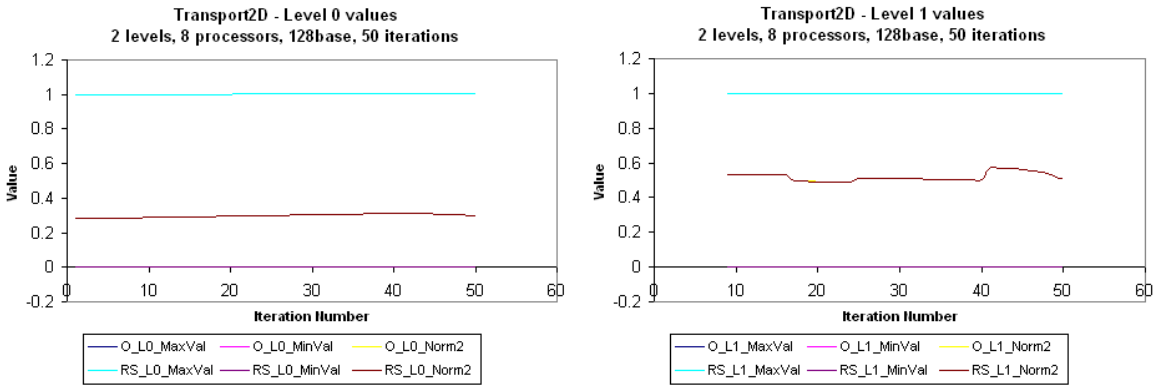

Figure 8.4: Numerical correctness for the reduced synchronization algorithm using a 2-level SAMR Transport2D implementation on 8 processors. Graphs for the maximum, minimum and second norm values at each level are coincident for original and reduced synchronization schemes.

Transport2D application on Frea and Jacquard. While Frea is a shared-node cluster environment with a 100Mbps Fast Ethernet interconnect, Jacquard is a dedicated-node MPP system with InfiniBand interconnect having a peak MPI unidirectional bandwidth of 620MB/s. Besides evaluating the performance of the reduced synchronization scheme on both systems, these experiments aim to demonstrate that communication-dominated SAMR applications can potentially benefit from relaxing the synchronization requirements even when executing on MPP systems, which have higher network capabilities and lower latencies as compared to cluster environments.

Table 8.2: Reduced synchronization algorithm performance for 3-level SAMR formulation of the Transport2D application with 256×256 base grid and executing for 500 iterations on 8 processors on Frea.

Execution Parameter	Original Scheme	Reduced Synchronization	Percentage Improvement
Max. Execution Time (sec)	160.866	134.358	16.48%
Max. Compute Time (sec)	1.745	1.801	-0.032%
Min. Compute Time (sec)	1.128	1.156	-0.025%
Avg. Compute Time (sec)	1.528	1.578	-0.032%
Total Sync Count	5891	4693	20.34%
Max. Sync Time (sec)	145.415	119.157	18.06%
Min. Sync Time (sec)	127.283	102.96	19.11%
Avg. Sync Time (sec)	139.846	114.218	18.33%

Table 8.3: Reduced synchronization algorithm performance for 3-level SAMR formulation of the Transport2D application with 256×256 base grid and executing for 500 iterations on 8 processors on Jacquard.

Execution Parameter	Original Scheme	Reduced Synchronization	Percentage Improvement
Max. Execution Time (sec)	12.947	11.334	12.46%
Max. Compute Time (sec)	0.897	0.932	-3.9%
Min. Compute Time (sec)	0.729	0.744	-2.06%
Avg. Compute Time (sec)	0.794	0.826	-4.03%
Total Sync Count	5891	4693	20.34%
Max. Sync Time (sec)	10.021	8.195	18.22%
Min. Sync Time (sec)	8.552	7.007	18.07%
Avg. Sync Time (sec)	9.387	7.706	17.91%

We first evaluate the two schemes on a 3-level SAMR grid hierarchy for the Transport2D application with a 256×256 base grid and executing for 500 iterations with regridding performed every 8 timesteps. Figure 8.5 illustrates the SAMR domain structure at iterations 0, 48, 208 and 496 for this 3-level formulation of the Transport2D application. The application dynamics lead to high synchronization costs and result in a communication-dominated behavior. The comparative performance of the original and reduced synchronization schemes on 8 processors on Frea and Jacquard are listed in Tables 8.2 and 8.3 respectively. The overall improvement in execution time for this experiment is 16.48% on Frea and 12.46% on Jacquard. Due to the reduced synchronization algorithm, the synchronization time component improves by nearly 18% on both systems, while the overheads of additional local computations is marginal (ranging from 0.025% to 4%) for both cases.

The next 3-level SAMR experiment compares the impact of original and reduced synchronization strategies on the performance of the Transport2D application with a 512×512 base grid and executing for 1000 coarse-level iterations on 48 processors on Frea and Jacquard. Though the base grid in this evaluation is 4 times larger than the 256×256 base grid in the previous experiment, the increase in the number of processors by a factor of 6 offsets the increase in computation time and, therefore, the compute times for both experiments are nearly similar. However, the decomposition

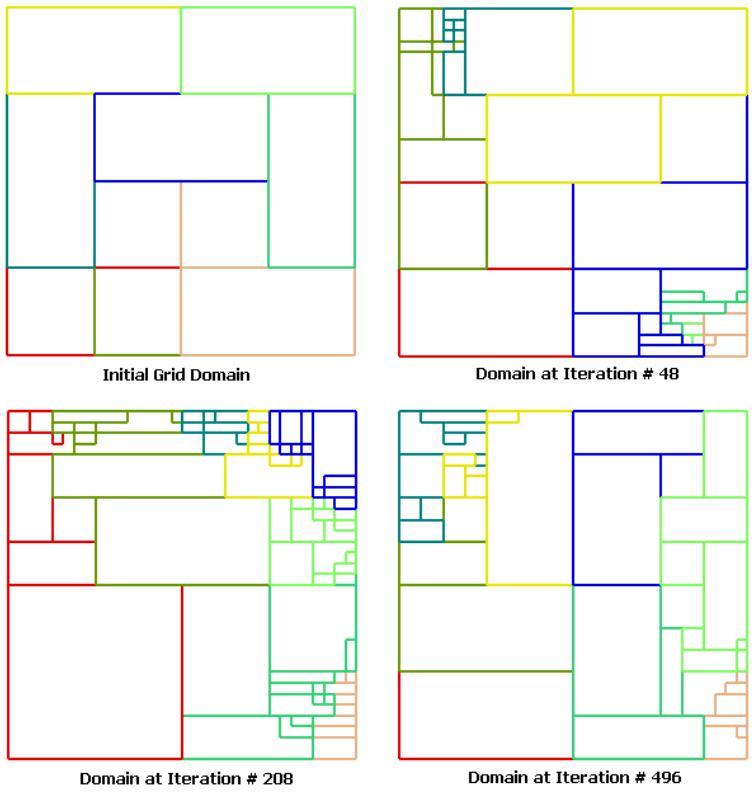

Figure 8.5: Snapshots of the 3-level Transport2D application at iterations 0, 48, 208 and 496 illustrating the SAMR domain structure on 8 processors on Frea and Jacquard.

of the SAMR domain among higher number of processors increases the synchronization costs on both systems, as illustrated in Tables 8.4 and 8.5 for Frea and Jacquard respectively. By trading off frequent synchronization in favor of additional local computations, the reduced synchronization algorithm improves the overall simulation execution time by nearly 19% and 13% for Frea and Jacquard respectively. The large synchronization times observed for Frea can be attributed to application load imbalance, slower interconnect and shared-usage environment. In addition to providing considerable performance benefit in such cluster environments, the reduced synchronization scheme also improves performance in MPP systems when the simulation is communication-dominated.

Table 8.4: Reduced synchronization algorithm performance for 3-level SAMR formulation of the Transport2D application with 512×512 base grid and executing for 1000 iterations on 48 processors on Frea.

Execution Parameter	Original Scheme	Reduced Synchronization	Percentage Improvement
Max. Execution Time (sec)	1734.68	1398.69	19.37%
Max. Compute Time (sec)	2.335	2.412	-3.3%
Min. Compute Time (sec)	0.682	0.695	-1.91%
Avg. Compute Time (sec)	1.235	1.27	-2.83%
Total Sync Count	10649	8477	20.4%
Max. Sync Time (sec)	1613.25	1280.55	20.62%
Min. Sync Time (sec)	192.209	132.081	31.28%
Avg. Sync Time (sec)	977.171	767.634	21.44%

Table 8.5: Reduced synchronization algorithm performance for 3-level SAMR formulation of the Transport2D application with 512×512 base grid and executing for 1000 iterations on 48 processors on Jacquard.

Execution Parameter	Original Scheme	Reduced Synchronization	Percentage Improvement
Max. Execution Time (sec)	39.136	34.142	12.76%
Max. Compute Time (sec)	0.791	0.827	-4.55%
Min. Compute Time (sec)	0.435	0.443	-1.84%
Avg. Compute Time (sec)	0.597	0.622	-4.19%
Total Sync Count	10649	8477	20.4%
Max. Sync Time (sec)	32.51	26.995	16.96%
Min. Sync Time (sec)	2.043	1.785	12.63%
Avg. Sync Time (sec)	21.815	18.082	17.11%

8.7 Inferences

Experimental evaluation of the reduced synchronization algorithm demonstrates its potential benefits when applied to uniform or adaptive implementations of parallel scientific simulations that use finite difference numerical methods and do not have synchronization limitations due to boundary condition restrictions or specialized stencils.

Several extensions to the reduced synchronization approach are possible. If the simulation permits different ghost region widths along different boundaries of the same grid patch and one-sided inter-processor communication is possible, then the communication algorithm can be transformed to be completely asynchronous by using

communication schedules or a multi-threaded implementation. Moreover, the additional local computations on each processor due to expanded ghost regions creates redundancy in the computational domain that can be potentially exploited to provide fault tolerance for the simulation at runtime. Once a certain processor fails, the other processors can detect this failure and re-create the application sub-domains owned by the failed processor from the redundant versions that reside on other functioning processors.

Chapter 9
Conclusions and Future Work

9.1 Summary

Simulations are playing an increasingly important role in science and engineering. Multi-scale and coupled scientific simulations enable realistic modeling of complex physical phenomena and have been effectively used to solve systems of partial differential equations in several application domains including astrophysics, combustion, computational fluid dynamics, numerical relativity, plasma physics, and subsurface modeling.

Parallel scientific simulations are typically implemented using uniform or adaptive techniques on structured or unstructured grid formulations. The multi-phased phenomena underlying these simulations span different time and space scales resulting in significant spatiotemporal heterogeneity. Moreover, these simulations can exhibit varying degrees of computational heterogeneity due to changes in their mathematical and topological characteristics. The inherent dynamism coupled with the runtime heterogeneity lead to significant challenges in ensuring algorithmic efficiency, coordination, load balancing, runtime management, and scalability of parallel scientific simulations on high-performance architectures.

Furthermore, the advent of multi-core processors and petascale computing systems offer the potential for new scientific breakthroughs such as analyzing the structure and function of complex biological molecules, deciphering the origins of the universe and creation of matter, predicting global climatic changes, designing more fuel-efficient and environment-friendly automobiles and aircraft, and understanding the origin,

spread and mitigation of contagious diseases [43]. However, the heterogeneity and dynamism manifested at unprecedented scales in these next-generation simulations will substantially exacerbate their performance and scalability in petascale environments. These challenges necessitate a paradigm shift in application composition, management and performance optimization, which underlines the motivation for this research.

Our previous work investigated the application-sensitive characterization for dynamically selecting the appropriate partitioner at runtime from a suite of partitioning algorithms. In this research, we present a multiobjective approach that analyzes application and system state, and provides appropriate distribution, configuration, coordination, and adaptation strategies for simulations on structured grids. Prototypes of individual partitioning and load balancing schemes that address spatiotemporal and computational heterogeneity have been developed and evaluated. We have examined the impact of heterogeneity on load balancing and performance using runtime calibration for different orchestrations of scientific applications. The synchronization algorithm improves communication overheads by reducing the messaging frequency in favor of additional computation, when the application is communication-dominated. The presented strategies have been integrated into an autonomic infrastructure that can address dynamism and heterogeneity in parallel scientific simulations, and enable their efficient and scalable execution on high-performance or cluster computing architectures.

9.2 Conclusions and Contributions

Addressing the runtime challenges associated with scientific applications to obtain optimal performance, often, lead to conflicting objectives and trade-offs. Since analytical approaches are typically not feasible, this research focuses on understanding

application and system characteristics, developing solutions to address heterogeneity and dynamism, and performing runtime adaptations to enhance performance and scalability for uniform and structured adaptive mesh refinement (SAMR) implementations of scientific simulations. Experimental evaluation of the presented strategies using simulations from several application domains demonstrate improvement in overall performance on large systems.

Key contributions of this research are summarized below.

9.2.1 Addressing Spatiotemporal Heterogeneity

In this research, we present a hierarchical partitioning framework, consisting of a stack of partitioners, which can address the space-time heterogeneity and dynamism in adaptive simulations. The partitioners build on a locality-preserving inverse space-filling-curve based representation of the SAMR grid hierarchy and enhance it based on localized requirements to minimize synchronization costs within a level (level-based partitioning), balance load (bin-packing based partitioning), or reduce partitioning costs (greedy partitioner). These schemes mitigate computation and synchronization limitations caused by unfavorable computation-to-communication ratios at runtime, and have enabled efficient scalable implementations of SAMR applications on thousands of processors.

9.2.2 Addressing Computational Heterogeneity

We have developed a dynamic partitioning strategy called Dispatch to address the computational heterogeneity for applications that exhibit non-uniform workload distributions in the computational domain. Dispatch maintains distributed computational weights associated with pointwise processes, computes local workloads in parallel, and performs in-situ global load balancing to determine processor allocations

proportional to computational weights and data redistribution that preserves application locality. Dispatch has shown improvements for uniform and adaptive methane-air combustion simulations that use operator-split reaction-diffusion models.

9.2.3 Synchronization Adaptation

Efficient execution of scientific simulations in heterogeneous networked environments necessitates a system-sensitive approach for adapting runtime performance. In this research, we have adapted the synchronization algorithm so that communication requirements can be relaxed by using additional computations on overlapped application sub-domains while reducing the frequency of synchronization. The basic premise is that processor speeds typically overshadow network speeds for heterogeneous clusters and hence this adaptation is beneficial when the application is communication-dominated. This behavior is also anticipated for scientific applications when scaled to large numbers of processors on dedicated massively parallel processing (MPP) systems, since such a formulation will involve more communication and reduced local computations.

9.2.4 Calibration, Analysis and Multiobjective Formulation

Due to the inherent dynamism and runtime heterogeneity in scientific simulations, performing runtime adaptation is non-trivial and requires an understanding of the application requirements as well as a calibration of the impact of the adaptation on overall performance. In this research, we have used runtime calibration to analyze the impact of heterogeneity on load distribution and performance for different orchestrations of scientific simulations. The heterogeneity analysis is then used as feedback towards appropriate algorithm selection for domain decomposition. We also present a multiobjective optimization approach that analyzes the severity and relative contributions of various runtime challenges and enables suitable algorithmic, application,

and system adaptations to mitigate the ill-effects of these challenges.

9.2.5 Autonomic Infrastructure

We have devised a conceptual autonomic runtime infrastructure with the aim to reactively and proactively manage and optimize application execution using current system and application state, performance calibration and feedback control, and online multiobjective models for adapting performance. The framework defines and maps application "working-sets" across physical resources so as to exploit the space, time, and functional heterogeneity of the simulations and underlying numerical methods. The various adaptation strategies presented in this research are integrated within this framework and help to improve simulation performance in different scenarios.

9.3 Research Publications

The research ideas presented in this dissertation have resulted in the following publications.

- *Analyzing the Impact of Computational Heterogeneity on Runtime Performance of Parallel Scientific Components.* S. Chandra, M. Parashar and J. Ray. Proceedings of the 15th High Performance Computing Symposium (HPC), March 2007.

- *Enabling Scalable Parallel Implementations of Structured Adaptive Mesh Refinement Applications.* S. Chandra, X. Li, T. Saif and M. Parashar. Journal of Supercomputing, vol. 39, no. 2, pages 177-203, Kluwer Academic Publishers, February 2007.

- *Addressing Spatiotemporal and Computational Heterogeneity for Structured Adaptive Mesh Refinement.* S. Chandra and M. Parashar. Computing and Visualization in Science, vol. 9, no. 3, pages 145-163, Springer Berlin/Heidelberg,

November 2006.

- *Dynamic Structured Partitioning for Parallel Scientific Applications with Pointwise Varying Workloads.* S. Chandra, M. Parashar and J. Ray. Proceedings of the 20th IEEE/ACM International Parallel and Distributed Processing Symposium (IPDPS), IEEE Computer Society Press, Rhodes Island, Greece, April 2006.

- *Dynamic Structured Partitioning of Adaptive Applications with Computational Heterogeneity.* S. Chandra, M. Parashar and J. Ray. Technical Report Number TR-281, Center for Advanced Information Processing, Rutgers University, Piscataway, NJ, USA, December 2005.

- *Investigating Autonomic Runtime Management Strategies for SAMR Applications.* S. Chandra, M. Parashar, J. Yang, Y. Zhang and S. Hariri. Special Issue on the NSF Next Generation Software (NGS) Program, International Journal of Parallel Programming (IJPP), editor: F. Darema, Springer Science+Business Media B.V., vol. 33, no. 2-3, pages 247-259, June 2005.

- *Towards Autonomic Application-Sensitive Partitioning for SAMR Applications.* S. Chandra and M. Parashar. Journal of Parallel and Distributed Computing (JPDC), Academic Press, vol. 65, no. 4, pages 519-531, April 2005.

- *A Simulation Framework for Evaluating the Runtime Characteristics of Structured Adaptive Mesh Refinement Applications.* S. Chandra and M. Parashar. Technical Report Number TR-275, Center for Advanced Information Processing, Rutgers University, Piscataway, NJ, USA, September 2004.

- *Autonomic Proactive Runtime Partitioning Strategies for SAMR Applications.* Y. Zhang, J. Yang, S. Hariri, S. Chandra and M. Parashar. Proceedings of the NSF Next Generation Software (NGS) Program Workshop, held in conjunction with the 18th IEEE/ACM International Parallel and Distributed Processing

Symposium, Santa Fe, NM, USA, IEEE Computer Society Press, 8 pages on CD-ROM, April 2004.

- *Engineering an Autonomic Partitioning Framework for Grid-based SAMR Applications.* S. Chandra, X. Li and M. Parashar. Book chapter in "High Performance Scientific and Engineering Computing: Hardware/Software Support", editors: L. T. Yang and Y. Pan, Kluwer Academic Publishers, ISBN: 104920-7580-4, pages 169-187, March 2004.

- *GridARM: An Autonomic Runtime Management Framework for SAMR Applications in Grid Environments.* S. Chandra, M. Parashar and S. Hariri. Proceedings of the Autonomic Applications Workshop, held in conjunction with the 10th International Conference on High Performance Computing (HiPC), Hyderabad, India, Elite Publishing, pages 286-295, December 2003.

9.4 Future Work

The primary contribution of this research is a comprehensive approach towards identifying and addressing dynamism and heterogeneity, using a gamut of application and system adaptation strategies, to optimize the performance of parallel scientific simulations on structured grids. However, these challenges assume greater significance in heterogeneous cluster computing and high-performance petascale environments. Harnessing scientific simulations on computing resources of such scale require sophisticated numerical techniques with significant concurrency, scalable algorithms, and above all, efficient runtime management. The ideas presented in this research constitute important steps towards these objectives, and envision several potential research directions that is the focus of future work.

9.4.1 Expanding the Runtime Decision Space

The research presented in this dissertation focuses on heterogeneity and dynamism for parallel scientific simulations and provides partitioning and synchronization strategies to address these challenges at runtime. As part of future work, the decision space for the deduction engine within the autonomic infrastructure can be expanded by investigating other runtime parameters for parallel scientific simulations such as locality and clustering properties, memory constraints, and adaptive granularity control. However, care must be taken to ensure that the overheads of characterization and adaptation are kept low, especially for an expanded decision space, since the cost of deploying the adaptation may overshadow the benefit of the adaptation itself. This assumes greater significance in high-performance computing environments.

9.4.2 Error Convergence and Fault Tolerance

The synchronization adaptation algorithm presented in this research lowers the communication overheads for scientific simulations, especially when the application is communication-dominated, by employing a trade-off between messaging frequency and additional local computation. By utilizing greater boundary overlaps that increase local computations on processors while reducing overall synchronization overheads, this approach also creates a certain degree of redundancy within the application domain, which can be possibly exploited for fault tolerance to enhance simulation survivability on large computational systems. Furthermore, it is possible to devise novel modifications to existing numerical schemes that can enable adaptations based on the quantity and accuracy of the available data for better error convergence at runtime, while simultaneously reducing the frequency of lock-step synchronization typically observed in SAMR algorithms.

9.4.3 Machine Learning in Scientific Simulations

Another extension of this research is to augment the calibration and feedback component within the runtime infrastructure with transductive inference and reinforcement techniques drawn from machine learning theory to provide a robust mapping of application and system states with performance-enhancing adaptations. Such a mapping can exploit the similarity in runtime characteristics of different classes of scientific simulations, and can prove extremely useful in predicting and optimizing performance in cases where application behaviors are not known a priori. Moreover, it is conceivable to use formal methods and models for performance prediction and optimization of parallel scientific simulations in various domains.

9.4.4 Addressing Performance for Component-based Scientific Computing

Component-based scientific computing is a promising domain with significant benefits from advances in runtime optimization. Recent years have seen a steady adoption of component-based technologies for implementing complex scientific simulations. The idea of a monolithic parallel code is replaced by a collection of components, which may be composed into various feasible configurations. Furthermore, these components may be adapted and/or replaced at runtime. Such an approach provides interoperability and flexibility, and can ideally be exploited to significantly improve application performance, especially in cases where algorithmic and component behaviors are not known a priori. This research can be extended to address issues related to efficient component-based scientific computing in a parallel or distributed environment. Specifically, it may be worthwhile to develop unique performance components (including calibrators, timers, rational agents) that can be used in a plug-and-play fashion in componentized high-performance simulations. Furthermore, it is possible to formulate an algebra that can combine various performance objectives based on the outputs

of these performance components, and determine the appropriate composition of the application from a pool of components at runtime.

9.4.5 Grid-based Scientific Computing

A recent and popular application area is the execution of scientific simulations on the computational Grid. The Grid has rapidly emerged as the dominant paradigm for wide area distributed computing. The inherent heterogeneity and dynamism of scientific applications coupled with a similarly heterogeneous and dynamic computational Grid results in significant complexities in runtime management. Future work can investigate the unique challenges of Grid environments and apply the research ideas related to heterogeneity and synchronization presented in this dissertation to provide performance adaptations for scientific simulations deployed on a wide-area or even global scale, similar to the SETI@Home model or protein-folding simulations.

9.4.6 Addressing Next-Generation Simulations

The overarching goal of this research is to enable large-scale scientific investigation and discovery using high-performance "smart" simulations. The ideas presented in this research have been successfully applied to simulations in basic sciences including domains in physics and chemistry. It will be worthwhile to investigate potential applications in domains such as bioinformatics and computational finance, which can greatly benefit from runtime performance optimization.

Furthermore, extending the ideas in this research pose several challenging questions such as: How should an application developer write scientific codes to maintain modularity and reuse while minimizing performance degradation and complexity? To what extent can a computer scientist perform optimizations to compensate for algorithmic limitations? How do we transform the performance optimization process

for adaptive computational simulations from an art into an exact science? Exploring answers to these issues as well as application, system and numerical adaptations are critical towards enabling next-generation scientific simulations at unprecedented scales.

References

[1] A. Abraham, L. Jain, and R. Goldberg. *Evolutionary Multiobjective Optimization: Theoretical Advances and Applications*. Springer, USA, 2005.

[2] ASCI/ASAP Center at the California Institute of Technology. URL: http://-www.cacr.caltech.edu/ASAP.

[3] ASCI Alliance, University of Chicago. http://www.llnl.gov/asci-alliances/asci-chicago.html.

[4] S. Baden, S. Kohn, and S. Fink. Programming with LPARX. CSE Technical Report CS94-377 (also in Proceedings of the Intel Supercomputer User's Group Meeting), Department of Computer Science & Engineering, University of California, San Diego, CA, 1994.

[5] S. Barnard and H. Simon. A Fast Multilevel Implementation of Recursive Spectral Bisection for Partitioning Unstructured Problems. *Concurrency: Practice and Experience*, 6:101–107, 1994.

[6] BATSRUS Homepage. URL: http://hpcc.engin.umich.edu/HPCC/codes/2/-BATSRUSv2.html.

[7] J. Bell, M. Berger, J. Saltzman, and M. Welcome. Three-Dimensional Adaptive Mesh Refinement for Hyperbolic Conservation Laws. *SIAM Journal on Scientific Computing*, 15(1):127–138, January 1994.

[8] M. Berger and S. Bokhari. A Partitioning Strategy for Non-Uniform Problems on Multiprocessors. *IEEE Transactions on Computers*, C-36(5):570–580, May 1987.

[9] M. Berger, G. Hedstrom, J. Oliger, and G. Rodrigue. Adaptive Mesh Refinement for 1-Dimensional Gas Dynamics. *Scientific Computing*, 3(17):43–47, 1983.

[10] M. Berger and J. Oliger. Adaptive Mesh Refinement for Hyperbolic Partial Differential Equations. *Journal of Computational Physics*, 53:484–512, March 1984.

[11] M. J. Berger and P. Colella. Local Adaptive Mesh Refinement for Shock Hydrodynamics. *Journal of Computational Physics*, 82:64–84, 1989.

[12] D. E. Bernholdt, B. A. Allan, R. Armstrong, and et al. A Component Architecture for High-Performance Scientific Computing. *International Journal of High-Performance Computing Applications*, 20:162–202, 2006.

[13] S. Bhavsar, M. Shee, and M. Parashar. Characterizing the Performance of Dynamic Distribution and Load-Balancing Techniques for Adaptive Grid Hierarchies. In *Proceedings of the IASTED International Conference on Parallel and Distributed Computing and Systems*, pages 782–787, Cambridge, MA, November 1999.

[14] S. Bhowmick, D. Kaushik, L. C. McInnes, B. Norris, and P. Raghavan. Parallel Adaptive Solvers in Compressible PETSc-FUN3D Simulations. Argonne National Laboratory preprint ANL/MCS-P1283-0805, 2005. Submitted to Proceedings of the 17th International Conference on Parallel CFD, August 2005.

[15] S. Bhowmick, L. C. McInnes, B. Norris, and P. Raghavan. The Role of Multi-Method Linear Solvers in PDE-based Simulations. In *Proceedings of the International Conference on Computational Science and its Applications,* Lecture Notes in Computer Science, Vol. 2667, pages 828–839, Montreal, Canada, 2003.

[16] S. Bhowmick, P. Raghavan, L. C. McInnes, and B. Norris. Faster PDE-Based Simulations Using Robust Composite Linear Solvers. *Future Generation Computer Systems*, 20:373–387, 2004.

[17] T. Bially. *A Class of Dimension Changing Mappings and its Application to Bandwidth Compression*. PhD thesis, Department of Electrical Engineering, Polytechnic Institute of Brooklyn, 1967.

[18] Parallel Environment (PE) for AIX V3R2.0: MPI Programming Guide, 2nd edition, December 2001.

[19] BlueGene/L. Terascale Simulation Facility, Lawrence Livermore National Laboratory, Livermore, CA. URL: http://www.llnl.gov/asc/computing_resources/-bluegenel/bluegene_home.html, 2006.

[20] J. Boillat, F. Brugé, and P. Kropf. A Dynamic Load-Balancing Algorithm for Molecular Dynamics Simulation on Multi-Processor Systems. *Journal of Computational Physics*, 96:1–14, 1991.

[21] S. Bokhari, T. Crockett, and D. Nicol. Binary Dissection: Variants & Applications. Technical Report ICASE Report No. 97-29, NASA Langley Research Center, Hampton, VA, 1997.

[22] P. C. Borges and M. P. Hansen. A Basis for Future Successes in Multiobjective Combinatorial Optimization. Technical Report IMM-REP-1998-8, Department of Mathematical Modeling, Technical University of Denmark, 1998.

[23] G. Bryan. Fluids in the Universe: Adaptive Mesh Refinement in Cosmology. *Computing in Science & Engineering*, 1(2):46–53, March/April 1999.

[24] Common Component Architecture (CCA) Forum. URL: http://www.cca-forum.org, 2004.

[25] T. Chan, J. Gilbert, and S. Teng. Geometric Spectral Partitioning. Technical Report CSL-94-15, Xerox Palo Alto Research Center, Palo Alto, CA, 1995.

[26] S. Chandra, X. Li, and M. Parashar. Engineering an Autonomic Partitioning Framework for Grid-based SAMR Applications. In L. T. Yang and Y. Pan, editors, *High Performance Scientific and Engineering Computing: Hardware/Software Support*, pages 169–187. Kluwer Academic Publishers, March 2004.

[27] S. Chandra and M. Parashar. An Evaluation of Partitioners for Parallel SAMR Applications. In *Proceedings of the 7th International Euro-Par Conference,* Springer-Verlag Lecture Notes in Computer Science, Vol. 2150, pages 171–174, August 2001.

[28] S. Chandra and M. Parashar. Towards Autonomic Application-Sensitive Partitioning for SAMR Applications. *Journal of Parallel and Distributed Computing*, 65(4):519–531, April 2005.

[29] S. Chandra, M. Parashar, and J. Ray. Dynamic Structured Partitioning for Parallel Scientific Applications with Pointwise Varying Workloads. In *Proceedings of the 20th IEEE/ACM International Parallel and Distributed Processing Symposium*, Rhodes Island, Greece, 10 pages on CD-ROM, IEEE Computer Society Press, April 2006.

[30] S. Chandra, S. Sinha, M. Parashar, Y. Zhang, J. Yang, and S. Hariri. Adaptive Runtime Management of SAMR Applications. In S. Sahni, V. Prasanna, and U. Shukla, editors, *Proceedings of the 9th International Conference on High Performance Computing,* Springer-Verlag Lecture Notes in Computer Science, Vol. 2552, pages 564–574, Bangalore, India, December 2002.

[31] 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)*, 12 pages on CDROM, October 2001.

[32] J. Chen and V. Taylor. PART: An Automatic Partitioning Tool. In *Proceedings of the 11th International Conference on Application-Specific Systems, Architectures, and Processors*, pages 328–337, Sweden, August 1997.

[33] J. Chen and V. Taylor. ParaPART: Parallel Mesh Partitioning Tool for Distributed Systems. *Concurrency: Practice and Experience*, 12(2-3):111–123, 2000.

[34] M. Choptuik. Experiences with an Adaptive Mesh Refinement Algorithm in Numerical Relativity. *Frontiers in Numerical Relativity*, pages 206–221, 1989.

[35] C. A. Coello Coello. A Comprehensive Survey of Evolutionary-based Multiobjective Optimization Techniques. *Knowledge and Information Systems*, 1(3):269–308, 1999.

[36] S. D. Cohen and A. C. Hindmarch. CVODE, A Stiff/Nonstiff ODE Solver in C. *Computers in Physics*, 10(2):138–143, 1996.

[37] P. Colella and et al. Chombo Infrastructure for Adaptive Mesh Refinement. URL: http://seesar.lbl.gov/ANAG/chombo/, 2005.

[38] P. Crandall and M. Quinn. A Partitioning Advisory System for Networked Data-Parallel Processing. *Concurrency: Practice and Experience*, 7(5):479–495, August 1995.

[39] DataStar User Guide at the San Diego Supercomputer Center. URL: http://-www.sdsc.edu/user_services/datastar/, 2005.

[40] K. Devine, B. Hendrickson, E. Boman, M. St. John, and C. Vaughan. Design of Dynamic Load-Balancing Tools for Parallel Applications. In *Proceedings of the International Conference on Supercomputing*, pages 110–118, Santa Fe, NM, May 2000.

[41] L. Diachin, R. Hornung, P. Plassman, and A. Wissink. Parallel Adaptive Mesh Refinement. In M. Heroux, P. Raghavan, and H. Simon, editors, *Frontiers of Parallel Processing for Scientific Computing*. SIAM book series on Software, Environments, and Tools (Philadelphia, PA), 2005.

[42] C. Ding and Y. He. A Ghost Cell Expansion Method for Reducing Communications in Solving PDE Problems. In *Proceedings of the Supercomputing Conference*, Denver, CO, November 2001.

[43] T. Dunning. Petascale Computing – Coming to a Supercomputer Center Near You. URL: http://www.hpcwire.com/hpc/1086279.html, 2006.

[44] C. Edwards. *A Parallel Infrastructure for Scalable Adaptive Finite Element Methods and its Application to Least Squares C-infinity Collocation*. PhD thesis, Department of Computational & Applied Mathematics, University of Texas at Austin, May 1997.

[45] S. Fink, S. Baden, and S. Kohn. Flexible Communication Mechanisms for Dynamic Structured Applications. In *Proceedings of the 3rd International Workshop on Parallel Algorithms for Irregularly Structured Problems*, pages 203–215, Santa Barbara, CA, August 1996.

[46] I. Foster and C. Kesselman. Computational Grids. In I. Foster and C. Kesselman, editors, *The Grid: Blueprint for a New Computing Infrastructure*. Morgan Kaufmann Publishers, 1998.

[47] I. Foster and C. Kesselman. *The Grid: Blueprint for a Future Computing Infrastructure*. Morgan Kaufmann Publishers, San Francisco, CA, 1998.

[48] M. Frenklach, H. Wang, M. Goldenberg, G. P. Smith, D. M. Golden, C. T. Bowman, R. K. Hanson, W. C. Gardiner, and V. Lissianski. GRI-Mech— An Optimized Detailed Chemical Reaction Mechanism for Methane Combustion. Technical Report GRI-95/0058, Gas Research Institute, Chicago, IL, November 1995.

[49] J. Gilbert, G. Miller, and S. Teng. Geometric Mesh Partitioning: Implementation and Experiments. *SIAM Journal of Scientific Computing*, 19:2091–2110, 1998.

[50] G. Gilder. *Gilder's Law on Network Performance. Telecosm: The World After Bandwidth Abundance*. Touchstone Books, 2002.

[51] M. Griebel and G. Zumbusch. Hash-Storage Techniques for Adaptive Multilevel Solvers and their Domain Decomposition Parallelization. *Contemporary Mathematics*, 218:279–286, 1998.

[52] M. Griebel and G. Zumbusch. Parallel Multigrid in an Adaptive PDE Solver Based on Hashing. *Proceedings of Parallel Computing '97*, pages 589–599, 1998.

[53] B. Hamann and R. J. Moorhead II. Survey of grid generation methodologies and scientific visualization efforts. In *Scientific Visualization: Overviews, Methodologies, and Techniques*, pages 59–101, IEEE Computer Society, Washington, DC, 1997.

[54] S. Hariri and et al. A Hierarchical Analysis Approach for High Performance Computing and Communication Applications. In *Proceedings of the 32nd Hawaii International Conference on System Sciences*, pages 112–120, January 1999.

[55] S. Hawley and M. Choptuik. Boson Stars Driven to the Brink of Black Hole Formation. *Phys. Rev. D,*, 62(10):104024, October 2000.

[56] J. C. Hewson and F. A. Williams. Rate-Ratio Asymptotic Analysis of Methane-Air Diffusion Flame Structure for Predicting Production of Oxides of Nitrogen. *Combustion and Flame*, 117:441–476, 1999.

[57] D. Hilbert. Uber die stetige Abbildung einer Linie auf Flachenstuck. *Math. Ann.*, 38:459–460, 1891.

[58] C. Huang, O. Lawlor, and L. V. Kale. Adaptive MPI. In *Proceedings of the 16th International Workshop on Languages and Compilers for Parallel Computing,* Springer-Verlag Lecture Notes in Computer Science, Vol. 2958, pages 306–322, College Station, TX, October 2003.

[59] H. Johansson. *Performance Characterization and Evaluation of Parallel PDE Solvers.* PhD thesis, Licentiate thesis, University of Uppsala, Uppsala, Sweden, 2006.

[60] J. Douglas Jr. and T. Dupont. Alternating-Direction Galerkin Methods on Rectangles. In B. Hubbard, editor, *Symposium on Numerical Solution of Partial Differential Equations - II*, pages 133–164. Academic Press (New York), 1971.

[61] G. Karypis. Multi-Constraint Mesh Partitioning for Contact/Impact Computations. Technical Report 03-022, Department of Computer Science and Engineering, University of Minnesota, Minneapolis, MN, May 2003.

[62] G. Karypis and V. Kumar. METIS: Unstructured Graph Partitioning and Sparse Matrix Ordering System, version 2.0. Technical Report TR_95-064, Department of Computer Science, University of Minnesota, Minneapolis, MN, 1995.

[63] G. Karypis, K. Schloegel, and V. Kumar. ParMeTiS: Parallel Graph Partitioning and Sparse Matrix Ordering Library. Technical Report TR_97-014, Department of Computer Science, University of Minnesota, Minneapolis, MN, 1997.

[64] J. P. Kenny, S. B. Benson, Y. Alexeev, J. Sarich, C. L. Janssen, L. C. McInness, M. Krishnan, J. Nieplocha, E. Jurrus, C. Fahlstrom, and T. L. Windus. Component-based Integration of Chemistry and Optimization Software. *Journal of Computational Chemistry*, 25:1717–1725, 2004.

[65] J. Kephart and D. Chess. The Vision of Autonomic Computing. *IEEE Computer*, 36(1):41–50, 2003.

[66] B. Khargharia, S. Hariri, M. Parashar, L. Ntaimo, and B. Kim. vGrid: A Framework For Building Autonomic Applications. In *Proceedings of the International Workshop on Challenges of Large Applications in Distributed Environments*, pages 19–26, IEEE Computer Society, Washington, DC, 2003.

[67] O. M. Knio, H. N. Najm, and P. S. Wyckoff. A Semi-Implicit Numerical Scheme for Reacting Flow II - Stiff, Operator-Split Formulation. *Journal of Computational Physics*, 154:428–467, 1999.

[68] S. Kohn. SAMRAI Homepage, Structured Adaptive Mesh Refinement Applications Infrastructure, URL: http://www.llnl.gov/CASC/SAMRAI, 2005.

[69] H. Kuang, L. F. Bic, and M. B. Dillencourt. Superboundary Exchange: A Technique for Reducing Communication in Distributed Implementations of Iterative Computations. In *Proceedings of the 4th International Conference on Algorithms and Architectures for Parallel Processing*, Hong Kong, December 2000.

[70] Z. Lan, V. Taylor, and G. Bryan. Dynamic Load Balancing for Structured Adaptive Mesh Refinement Applications. In *Proceedings of the 30th International Conference on Parallel Processing*, pages 571–579, Valencia, Spain, September 2001.

[71] Z. Lan, V. Taylor, and G. Bryan. Dynamic Load Balancing of SAMR Applications on Distributed Systems. In *Proceedings of the Supercomputing Conference*, Denver, CO, 2001.

[72] S. Lefantzi, J. Ray, C. A. Kennedy, and H. N. Najm. A Component-based Toolkit for Reacting Flows with High Order Spatial Discretizations on Structured Adaptively Refined Meshes. *Progress in Computational Fluid Dynamics*, 5(6):298–315, 2005.

[73] S. Lefantzi, J. Ray, and H. N. Najm. Using the Common Component Architecture to Design High Performance Scientific Simulation Codes. In *Proceedings of the International Parallel and Distributed Processing Symposium*, Nice, France, April 2003.

[74] X. Li and M. Parashar. Dynamic Load Partitioning Strategies for Managing Data of Space and Time Heterogeneity in Parallel SAMR Applications. In H. Kosch, L. Boszormenyi, and H. Hellwagner, editors, *Proceedings of the 9th International Euro-Par Conference,* Springer-Verlag Lecture Notes in Computer Science, Vol. 2790, pages 181–188, Klagenfurt, Austria, August 2003.

[75] P. MacNeice. PARAMESH Homepage. URL: http://esdcd.gsfc.nasa.gov/ESS/-macneice/paramesh/paramesh.html, 1999.

[76] P. MacNeice, K. M. Olson, C. Mobarry, R. deFainchtein, and C. Packer. PARAMESH: A Parallel Adaptive Mesh Refinement Community Toolkit. *Computer Physics Communications*, 126:330–354, 2000.

[77] G. Marchuk. Splitting and Alternating Direction Methods. In P. Ciarlet and J. Lions, editors, *Handbook of Numerical Analysis, Volume I*, pages 197–462. Elsevier Science Publishers B.V. (North-Holland, Amsterdam), 1990.

[78] S. McCormick and J. Ruge. Unigrid for Multigrid Simulation. *Mathematics of Computation*, 41(163):43–62, 1986.

[79] L. C. McInnes, B. Norris, S. Bhowmick, and P. Raghavan. Adaptive Sparse Linear Solvers for Implicit CFD Using Newton-Krylov Algorithms. In *Proceedings of the Second MIT Conference on Computational Fluid and Solid Mechanics,* Vol. 2, pages 1024–1028, Elsevier, Cambridge, MA, June 2003.

[80] B. Moon, H. Jagadish, C. Faloutsos, and J. Saltz. Analysis of the Clustering Properties of the Hilbert Space-Filling Curve. *IEEE Transactions on Knowledge and Data Engineering*, 13(1):124–141, January/February 2001.

[81] B. Moon, G. Patnaik, R. Bennett, D. Fyfe, A. Sussman, C. Douglas, J. H. Saltz, and K. Kailasanath. Runtime Support Dynamic Load Balancing Strategies for Structured Adaptive Applications. In *Proceedings of the 7th SIAM Conference on Parallel Processing for Scientific Computing*, pages 575–580, San Francisco, CA, February 1995.

[82] G. Moore. Moore's Law. URL: http://www.intel.com/research/silicon/mooreslaw.htm, 1965.

[83] I. Moulitsas and G. Karypis. Partitioning Algorithms for Simultaneously Balancing Iterative and Direct Methods. Technical Report 04-014, Department of Computer Science and Engineering, University of Minnesota, Minneapolis, MN, March 2004.

[84] H. N. Najm, R. W. Schefer, R. B. Milne, C. J. Mueller, K. D. Devine, and S. N. Kempka. Numerical and Experimental Investigation of Vortical Flow-Flame Interaction. SAND Report SAND98-8232, UC-1409, Sandia National Laboratories, Livermore, CA, February 1998.

[85] NERSC. Jacquard Homepage. URL: http://www.nersc.gov/nusers/resources/-jacquard, 2006.

[86] M. Norman and G. Bryan. Cosmological Adaptive Mesh Refinement. *Numerical Astrophysics,* eds. S. Miyama and K. Tomisaka, Kluwer: Dordrecht, pages 19–28, 1999.

[87] L. Oliker and R. Biswas. PLUM: Parallel Load Balancing for Adaptive Unstructured Meshes. *Journal of Parallel and Distributed Computing*, 52(2):150–177, 1998.

[88] B. Olstad and F. Manne. Efficient Partitioning of Sequences. *IEEE Transactions on Computers*, 44(11):1322–1326, November 1995.

[89] C. Ou and S. Ranka. Parallel Remapping Algorithms for Adaptive Problems. *Journal of Parallel and Distributed Computing*, 42:109–121, 1997.

[90] M. Parashar. GrACE: Grid Adaptive Computational Engine. URL: http://-www.caip.rutgers.edu/~parashar/TASSL/Projects/GrACE, 2005.

[91] M. Parashar and J. Browne. Distributed Dynamic Data Structures for Parallel Adaptive Mesh Refinement. In *Proceedings of the International Conference for High Performance Computing*, pages 22–27, December 1995.

[92] M. Parashar and J. Browne. On Partitioning Dynamic Adaptive Grid Hierarchies. In *Proceedings of the 29th Annual Hawaii International Conference on System Sciences*, pages 604–613, January 1996.

[93] M. Parashar and J. Browne. System Engineering for High Performance Computing Software: The HDDA/DAGH Infrastructure for Implementation of Parallel Structured Adaptive Mesh Refinement. *IMA Volume on Structured Adaptive Mesh Refinement (SAMR) Grid Methods*, pages 1–18, January 2000.

[94] M. Parashar, J. Browne, C. Edwards, and K. Klimkowski. A Common Data Management Infrastructure for Adaptive Algorithms for PDE Solutions. In *Proceedings of the ACM/IEEE Conference on Supercomputing*, 15 pages on CDROM, San Jose, CA, November 1997.

[95] M. Parashar and S. Hariri. Compile-Time Performance Interpretation of HPF/Fortran 90D. *IEEE Parallel and Distributed Technology*, 4(1):57–73, Spring 1996.

[96] M. Parashar and S. Hariri. Interpretive Performance Prediction for Parallel Application Development. *Journal of Parallel and Distributed Computing*, 60(1):17–47, January 2000.

[97] M. Parashar and S. Hariri. PRAGMA: An Infrastructure for Runtime Management of Grid Applications. In *NSF Next Generation Systems Program Workshop, IEEE/ACM International Parallel and Distributed Processing Symposium*, FL, April 2002.

[98] M. Parashar, J. Wheeler, G. Pope, K. Wang, and P. Wang. A New Generation EOS Compositional Reservoir Simulator: Framework and Multiprocessing. In *Proceedings of the Society of Petroleum Engineers, Reservoir Simulation Symposium,* Paper No. 37977, pages 31–38, Dallas, TX, June 1997.

[99] P. H. Paul. DRFM: A New Package for the Evaluation of Gas-Phase-Transport Properties. Sandia Report SAND98-8203, Sandia National Laboratories, Albuquerque, NM, November 1997.

[100] G. Peano. Sur une Courbe qui Remplit Toute une Aire Plane. *Math. Ann.*, 36:157–160, 1890.

[101] R. B. Pember, J. B. Bell, W. Y. Crutchfield, M. L. Welcome, and P. Colella. Adaptive Cartesian Grid Methods for Representing Geometry in Inviscid Compressible Flow. In R. C. Swanson and E. Turkel, editors, *11th International AIAA Computational Fluid Dynamics Conference, 28th Thermophysics Conference and 23rd Fluid Dynamics, Plasma Dynamics and Laser Conference*, pages 5–7, Orlando, FL, July 1993.

[102] J. Pilkington and S. Baden. Partitioning with Space-filling Curves. Technical Report CS94-34, Department of Computer Science & Engineering, University of California, San Diego, CA, 1994.

[103] A. Pinar and B. Hendrickson. Partitioning for Complex Objectives. In *Workshop on Solving Irregularly Structured Problems in Parallel, Proceedings of the 15th International Parallel and Distributed Processing Symposium*, pages 1232–1237, San Francisco, CA, 2001.

[104] R. Pollak. A Hierarchical Load Balancing Environment for Parallel and Distributed Supercomputer. In *Proceedings of the International Symposium on Parallel and Distributed Supercomputing*, pages 34–40, Fukuoka, Japan, September 1995.

[105] S. Ramanathan. Performance Optimization for Dynamic Adaptive Grid Hierarchies. Master's thesis, Department of Electrical & Computer Engineering, Graduate School, Rutgers University, 2000.

[106] J. Rantakokko. *Data Partitioning Methods and Parallel Block-oriented PDE Solvers*. PhD thesis, Department of Scientific Computing, Uppsala University, 1998.

[107] J. Ray, H. Najm, R. Milne, K. Devine, and S. Kempka. Triple Flame Structure and Dynamics at the Stabilization Point of an Unsteady Lifted Jet Diffusion Flame. *Proceedings of Combustion Institute*, 28(1):219–226, 2000.

[108] C. Zambrana Rojas and M. Hoemmen. Communication Savings with Ghost Cell Expansion for Domain Decompositions of Finite Difference Grids. Class Report SAND98-8203, University of California, Berkeley, CA, Spring 2004.

[109] H. Sagan. *Space-filling Curves*. Springer-Verlag, New York, NY, 1994.

[110] H. Samet. *The Design and Analysis of Spatial Data Structures*. Addison-Wesley, Reading, MA, 1989.

[111] R. Samtaney. Suppression of the Richtmyer-Meshkov Instability in the Presence of a Magnetic Field. *Physics of Fluids*, 15(8):53–56, August 2003.

[112] R. Samtaney, S. C. Jardin, P. Colella, and D. F. Martin. 3D Adaptive Mesh Refinement Simulations of Pellet Injection in Tokamaks. *Computer Physics Communication*, 164:220–228, 2004.

[113] K. Schloegel and et al. A Performance Study of Diffusive vs. Remapped Load Balancing Schemes. Technical Report 98-018, Department of Computer Science, University of Minnesota, Minneapolis, MN, 1998.

[114] K. Schloegel, G. Karypis, and V. Kumar. A Unified Algorithm for Load-balancing Adaptive Scientific Simulations. In *Proceedings of the Supercomputing Conference*, Dallas, TX, 11 pages on CDROM, November 2000.

[115] SCOREC. Parallel Scientific Computation Software Package, SCOREC Homepage, http://www.scorec.rpi.edu/programs/parallel/ParallelScientific.htm.

[116] M. Shee. Evaluation and Optimization of Load Balancing/Distribution Techniques for Adaptive Grid Hierarchies. Master's thesis, Department of Electrical & Computer Engineering, Graduate School, Rutgers University, 2000.

[117] S. Smith. *Adaptive Asynchronous Parallel Algorithms in Distributed Computation*. PhD thesis, Department of Computer Science, Graduate School, University of Colorado., 1991.

[118] R. B. Statnikov and J. B. Matusov. *Multicriteria Optimization and Engineering*. Chapman and Hall, New York, USA, 1995.

[119] J. Steensland. Vampire Library Homepage. URL: http://www.caip.rutgers.edu/~johans/vampire, 2000.

[120] J. Steensland. Domain-based Partitioning for Parallel SAMR Applications. Licentiate Thesis, 2001-002, Department of Scientific Computing, Uppsala University, Uppsala, Sweden, 2001.

[121] J. Steensland. Dynamic Structured Grid Hierarchy Partitioners using Inverse Space-filling Curves. Technical Report 2001-002, Department of Scientific Computing, Uppsala University, Uppsala, Sweden, 2001.

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

[123] 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, December 2002.

[124] J. Steensland, S. Chandra, M. Thuné, and M. Parashar. Characterization of Domain-based Partitioners for Parallel SAMR Applications. In *Proceedings of the IASTED International Conference on Parallel and Distributed Computing and Systems*, pages 425–430, Las Vegas, NV, November 2000.

[125] J. Steensland and J. Ray. A Heuristic Re-Mapping Algorithm Reducing Inter-Level Communication in SAMR Applications. In *Proceedings of the 15th IASTED International Conference on Parallel and Distributed Computing and Systems,* Vol. 2, ACTA Press, pages 707–712, 2003.

[126] J. Steensland and J. Ray. A Partitioner-Centric Model for SAMR Partitioning Trade-Off Optimization: Part II. In *Proceedings of International Conference on Parallel Processing (ICPP) Workshops*, pages 231–238, 2004.

[127] J. Steensland and J. Ray. A Partitioner-Centric Model for SAMR Partitioning Trade-Off Optimization: Part I. *International Journal of High Performance Computing Applications*, 19(4):409–422, 2005.

[128] J. Steensland, S. Soderberg, and M. Thuné. A Comparison of Partitioning Schemes for Blockwise Parallel SAMR Algorithms. In *Proceedings of the 5th International Workshop on Applied Parallel Computing,* Springer-Verlag Lecture Notes in Computer Science, Vol. 1947, pages 160–169, June 2000.

[129] E. Steinthorsson and D. Modiano. Advanced Methodology for Simulation of Complex Flows using Structured Grid Systems. In *Surface Modeling, Grid Generation, and Related Issues in Computational Fluid Dynamic (CFD) Solutions,* NASA Conference Publication 3291, pages 697–710, May 1995.

[130] E. Strohmaier. Highlights of the 28th TOP500. Presented at Supercomputing Conference, Tampa, FL, 2006.

[131] M. Thuné. Partitioning Strategies for Composite Grids. *Parallel Algorithms and Applications*, 12(9):325–348, September 1998.

[132] TOP500 List. URL: http://www.top500.org/lists/2006/11, 2006.

[133] P. Wang, I. Yotov, T. Arbogast, C. Dawson, M. Parashar, and K. Sepehrnoori. A New Generation EOS Compositional Reservoir Simulator: Formulation and Discretization. In *Proceedings of the Society of Petroleum Engineers, Reservoir Simulation Symposium,* Paper No. 37979, pages 55–64, Dallas, TX, June 1997.

[134] A. M. Wissink, R. D. Hornung, S. R. Kohn, S. S. Smith, and N. Elliott. Large Scale Parallel Structured AMR Calculations using the SAMRAI Framework. In *Proceedings of the 2001 ACM/IEEE Conference on Supercomputing*, 6 pages on CD-ROM, ACM Press, Denver, CO, 2001.

[135] E. Zitzler and L. Thiele. Multiobjective Evolutionary Algorithms: A Comparative Case Study and the Strength Pareto Approach. *IEEE Transactions on Evolutionary Computation*, 3(4):257–271, November 1999.

Vita

Sumir Chandra

2007 Ph.D. Candidate in Computer Engineering, GPA 3.56/4.0; Rutgers University, NJ, USA.

2002 MS in Computer Engineering, GPA 3.56/4.0; Rutgers University, NJ, USA.

1998 BE in Electronics Engineering, GPA 3.7/4.0; University of Mumbai, Mumbai, India.

2000-2007 Graduate Assistant, Center for Advance Information Processing, Rutgers University, Piscataway, NJ, USA

2006-2006 Visiting Researcher, Advanced Software Research and Development, Sandia National Laboratories, Livermore, CA, USA

2000-2000 Programmer Analyst Intern, Goldman, Sachs & Co., New York, NY, USA

1999-2000 Part-time Lecturer, Computer Science Department, Rutgers University, Piscataway, NJ, USA

1999-2000 Laboratory Assistant and Grader, Department of Electrical and Computer Engineering, Rutgers University, Piscataway, NJ, USA

1998-1999 Associate Systems Engineer, Tata Infotech Ltd., Mumbai, India

1997-1998 Engineering Intern, Onida (MIRC Electronics), Mumbai, India

Publications: Journal Articles

S. Chandra, X. Li, T. Saif and M. Parashar, "Enabling Scalable Parallel Implementations of Structured Adaptive Mesh Refinement Applications", *Journal of Supercomputing*, vol. 39, no. 2, pages 177-203, Kluwer Academic Publishers, February 2007.

S. Chandra and M. Parashar, "Addressing Spatiotemporal and Computational Heterogeneity in Structured Adaptive Mesh Refinement", *Computing and Visualization in Science*, vol. 9, no. 3, pages 145-163, Springer Berlin/Heidelberg, November 2006.

S. Chandra, M. Parashar, J. Yang, Y. Zhang and S. Hariri, "Investigating Autonomic Runtime Management Strategies for SAMR Applications", *International Journal of Parallel Programming*, Special Issue on the NSF Next Generation Software Program, editor: F. Darema, vol. 33, no. 2-3, pages 247-259, Springer, June 2005.

S. Chandra and M. Parashar, "Towards Autonomic Application-Sensitive Partitioning for SAMR Applications", *Journal of Parallel and Distributed Computing*, vol. 65, issue 4, pages 519-531, Academic Press, April 2005.

J. Steensland, S. Chandra and M. Parashar, "An Application-Centric Characterization of Domain-Based Inverse Space-Filling Curve Partitioners for Parallel SAMR Applications", *IEEE Transactions on Parallel and Distributed Systems*, vol. 13, no. 12, pages 1275-1289, IEEE Computer Society Press, December 2002.

Publications: Conference/Workshop Papers

S. Chandra, M. Parashar and J. Ray, "Analyzing the Impact of Computational Heterogeneity on Runtime Performance of Parallel Scientific Components", *Proceedings of the 15th High Performance Computing Symposium* (HPC-07), part of the SCS Spring Simulation Multiconference, Norfolk, VA, USA, March 2007.

S. Chandra, M. Parashar and J. Ray, "Dynamic Structured Partitioning for Parallel Scientific Applications with Pointwise Varying Workloads", *Proceedings of the 20th IEEE/ACM International Parallel and Distributed Processing Symposium* (IPDPS-06), Rhodes Island, Greece, 10 pages on CD-ROM, IEEE Computer Society Press, April 2006.

Y. Zhang, J. Yang, S. Hariri, S. Chandra and M. Parashar, "Autonomic Proactive Runtime Partitioning Strategies for SAMR Applications", *Proceedings of the NSF Next Generation Software Program Workshop* (NGS-04), held in conjunction with *18th IEEE/ACM International Parallel and Distributed Processing Symposium*, Santa Fe, NM, USA, 8 pages on CD-ROM, IEEE Computer Society Press, April 2004.

S. Chandra, M. Parashar and S. Hariri, "GridARM: An Autonomic Runtime Management Framework for SAMR Applications in Grid Environments", New Frontiers in High-Performance Computing, *Proceedings of the Autonomic Applications Workshop* (AAW-03), held in conjunction with *10th IEEE International Conference on High Performance Computing*, Hyderabad, India, pages 286-295, Elite Publishing, December 2003.

S. Chandra, S. Sinha, M. Parashar, Y. Zhang, J. Yang and S. Hariri, "Adaptive Runtime Management of SAMR Applications", *Proceedings of the 9th IEEE International Conference on High Performance Computing* (HiPC-02), Bangalore, India, editors: S. Sahni, V. K. Prasanna and U. Shukla,

Lecture Notes in Computer Science, vol. 2552, pages 564-574, Springer-Verlag, December 2002.

S. Chandra and M. Parashar, "ARMaDA: An Adaptive Application-Sensitive Partitioning Framework for Structured Adaptive Mesh Refinement Applications", *Proceedings of the 14th IASTED International Conference on Parallel and Distributed Computing and Systems* (PDCS-02), Cambridge, MA, USA, pages 446-451, ACTA Press, November 2002.

S. Chandra, J. Steensland, M. Parashar and J. Cummings, "An Experimental Study of Adaptive Application Sensitive Partitioning Strategies for SAMR Applications", *Proceedings of the 2nd Los Alamos Computer Science Institute Symposium* (LACSI-01), Santa Fe, NM, USA, 12 pages on CD-ROM, October 2001.

S. Chandra and M. Parashar, "An Evaluation of Partitioners for Parallel SAMR Applications", *Proceedings of the 7th International European Conference on Parallel Computing* (EuroPar-01), Manchester, UK, editors: R. Sakellariou, J. Keane, J. Gurd and L. Freeman, Lecture Notes in Computer Science, vol. 2150, pages 171-174, Springer-Verlag, August 2001.

J. Steensland, M. Thune, S. Chandra and M. Parashar, "Characterization of Domain-Based Partitioners for Parallel SAMR Applications", *Proceedings of the 12th IASTED International Conference on Parallel and Distributed Computing and Systems* (PDCS-00), Las Vegas, NV, USA, pages 425-430, ACTA Press, November 2000.

Publications: Book Chapters

S. Chandra, X. Li and M. Parashar. "Engineering an Autonomic Partitioning Framework for Grid-based SAMR Applications". Chapter in "High Performance Scientific and Engineering Computing: Hardware/Software Support", editors: L. T. Yang and Y. Pan, pages 169-187, Kluwer Academic Publishers, March 2004.

M. Parashar and S. Chandra. "Distributed Shared Memory Tools". Chapter in "Tools and Environments for Parallel and Distributed Computing", editors: S. Hariri and M. Parashar, pages 57-78, John Wiley and Sons, February 2004.

Publications: Technical Reports

S. Chandra, M. Parashar and J. Ray. "Dynamic Structured Partitioning of Adaptive Applications with Computational Heterogeneity". Technical Report Number TR-281, Center for Advanced Information Processing, Rutgers University, Piscataway, NJ, USA, December 2005.

S. Chandra and M. Parashar. "A Simulation Framework for Evaluating the Runtime Characteristics of Structured Adaptive Mesh Refinement Applications". Technical Report Number TR-275, Center for Advanced Information Processing, Rutgers University, Piscataway, NJ, USA, September 2004.