Available online at www.sciencedirect.com
SCIENCE DIRECT®
J. Parallel Distrib. Comput. 60 (2005) 519 – 531
Journal of Parallel and Distributed Computing
www.elsevier.com/locate/jpdc

Towards autonomic application-sensitive partitioning for SAMR applications[☆]

Sumir Chandra[*], Manish Parashar

The Applied Software Systems Laboratory, Department of Electrical and Computer Engineering, Rutgers, The State University of New Jersey, 94 Brett Road, Piscataway, NJ 08854 USA

Received 21 August 2003; received in revised form 14 June 2004; accepted 14 September 2004

Abstract

Distributed structured adaptive mesh refinement (SAMR) techniques offer the potential for accurate and cost-effective solutions of physically realistic models of complex physical phenomena. However, the heterogeneous and dynamic nature of SAMR applications results in significant runtime management challenges. This paper investigates autonomic application-sensitive SAMR runtime management strategies and presents the design, implementation, and evaluation of ARMaDA, a self-adapting and optimizing partitioning framework for SAMR applications. ARMaDA monitors and characterizes application runtime state, and dynamically selects and invokes appropriate partitioning mechanisms that match current SAMR state and optimize its computational and communication performance. The advantages of the autonomic partitioning capabilities provided by ARMaDA are experimentally demonstrated.
© 2005 Elsevier Inc. All rights reserved.

Keywords: Autonomic partitioning; Dynamic applications; Runtime application characterization; Structured adaptive mesh refinement

1. Introduction

Dynamic structured adaptive mesh refinement (SAMR) [3] techniques for the solutions of partial differential equations enable computational effort and resources to be dynamically concentrated to appropriate regions of the computational domain by overlaying uniform patch-based refinements on these regions. These techniques can yield highly advantageous ratios for cost/accuracy as compared to techniques based on static approximations. SAMR techniques are being effectively used in several application domains including computational fluid dynamics [1,2,17], numerical relativity [8,10], astrophysics [4,12], subsurface modeling and oil reservoir simulation [16,23].

SAMR-based realizations of complex applications can provide dramatic insights into physical phenomena 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. However, the phenomena being modeled by these applications and their implementations are inherently multi-phased, dynamic, and heterogeneous in time, space, and state. Emerging large-scale parallel/distributed computing environments are similarly complex, heterogeneous, and dynamic. As a result, scalable implementations of SAMR applications in distributed environments present significant challenges in runtime management, adaptation, and optimization. In this paper, we investigate autonomic[1] application-sensitive approaches for

[☆] The research presented in this paper was supported in part by the National Science Foundation via grants numbers ACI 9984357, EIA 0103674 EIA-0120934, ANI 0335244, CNS 0426354 and IIS 0430826, and by DOE ASCI/ASAP via grants numbers PC295251 and 82-1052856 awarded to Manish Parashar.

[*] Corresponding author. Fax: +1 732 445 0593.
E-mail addresses: sumir@caip.rutgers.edu (S. Chandra), parashar@caip.rutgers.edu (M. Parashar).

[1] Autonomic computing is based on the strategies used by biological systems to deal with complexity, heterogeneity, and uncertainty, and enables applications that are self-configuring, optimizing, self-healing, contextually aware, and open.

SAMR runtime management [7], enabling the efficient execution of SAMR applications.

This paper presents the design, implementation, and evaluation of ARMaDA, a self-adapting and optimizing partitioning framework for SAMR applications. The overall goal of ARMaDA is to manage dynamism and space-time heterogeneity, and support the efficient and scalable execution of SAMR implementations. It monitors and characterizes the runtime state of SAMR applications. It then selects, configures, and invokes appropriate partitioning and scheduling mechanisms to match the current state of the application and optimize its computational and communication performance. ARMaDA is composed of three key components: application state monitoring and characterization component, partitioner selection component and policy engine, and runtime adaptation component. Partitioning strategies used includes a selection from broadly used software tools such as GrACE [14,15] and Vampire [21]. The advantages of the autonomic partitioning capabilities provided by ARMaDA are experimentally demonstrated using a selection of SAMR kernels.

ARMaDA builds on our previous work where we defined an approach for characterizing the state of SAMR applications [6,22] and used this approach to experimentally study the behavior of structured domain-based partitioners. In this paper, we use these building blocks to construct an autonomic partitioning framework. This paper makes two key contributions: (1) It defines mechanisms for monitoring, analyzing, and characterizing the state of distributed SAMR applications at runtime, and (2) It defines policies for autonomic application-sensitive partitioner selection, configuration, and invocation to optimize the overall performance of SAMR applications.

The rest of the paper is organized as follows. Section 2 discusses related research and prior work. Section 3 presents the building blocks for constructing an autonomic partitioner for SAMR applications. Section 4 uses a "feasibility study" example to illustrate the operation of adaptive application-sensitive partitioning and to demonstrate its advantages. Section 5 presents the design and implementation of the ARMaDA framework. Section 6 describes the experimental evaluation of ARMaDA using a selection of SAMR kernels. Section 7 presents concluding remarks and future work.

2. Related work

Related research in adaptive partitioning and dynamic load balancing for adaptive mesh refinement (AMR) applications is summarized in Table 1. Partitioners for AMR applications not only have to balance load but also have to address locality, available parallelism, communication overheads, data-migration overheads, partitioning quality, and repartitioning overheads. Note that these factors often have conflicting requirements and must be considered together.

Table 1
Summary of related research in dynamic partitioning and load balancing for AMR applications

Scheme (Type)	Description
PLUM (Unstructured)	Dynamic load balancing for unstructured meshes using a cost-metric model with computation, communication, and remapping weights; accepts new partitioning if gain greater than redistribution cost
Unified Repartitioning Algorithm (Unstructured)	Repartitioning for unstructured meshes based on a relative cost factor that minimizes both communication and data redistribution costs; initial partitioning scheme yielding lowest cost is selected
SAMR Dynamic Load Balancing (Structured)	Defines a moving-grid phase that balances load across processors after adaptation, and splitting-grid phase splits a grid along the longest dimension, if required
Partitioning Advisory System (Unstructured)	Ranks partitioning methods based on problem space, machine speed, and network information; considers processor performance variance assuming linear complexity

Existing approaches typically consider only a subset of these factors. For example, certain techniques (such as SAMR DLB) only consider the computational workload while some others (such as PLUM and URA) combine the load with communication and locality.

Note that SAMR is an alternative to the unstructured approach and partitioners for unstructured AMR cannot be typically used for SAMR. Though there exist several adaptive partitioning approaches for unstructured AMR, there are only a few research efforts investigating runtime adaptations for SAMR. In this paper, we focus on the partitioners that have been shown to be most appropriate for SAMR [22], i.e., those based on space filling curves. However, related work for "unstructured methods" can provide useful insights into several runtime adaptation concepts, such as the metric and policy analysis and execution, while developing similar approaches for SAMR.

The research presented in this paper proposes an autonomic partitioning solution for self-adapting and optimizing SAMR application management. Our approach defines a multi-component objective function that addresses load balance, communication overheads, locality, data-movement overheads, partitioning quality, and repartitioning overheads. Furthermore, the adaptation/optimization process is based on the current state of the application and consists of dynamically selecting the most appropriate partitioning algorithm and configuration that addresses the requirements of the current SAMR grid hierarchy to improves overall application performance.

2.1. PLUM

PLUM (Parallel Load balancing for adaptive Unstructured Meshes) [13] is a dynamic load balancing strategy for adaptive unstructured grid computations that uses a cost-metric model for efficient mesh adaptation. In addition to a flow solver and a mesh adaptor, PLUM includes a partitioner and a remapper that load balances and redistributes the computational mesh, when necessary. Parallel adaptation consists of three phases—initialization, execution, and finalization. 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 metrics that estimate and compare the computational gain and the redistribution cost due to load balancing after each mesh adaptation step. The new partitioning and mapping are accepted if the computational gain is larger than the redistribution cost. The metrics used by PLUM are similar to the metrics proposed in this paper. However, we also consider the speed of the partitioner, the quality of the partition, and partitioning overheads, which are not considered by PLUM. Furthermore, in addition to deciding whether to repartition or not, we also dynamically select and configure the appropriate partitioning algorithm.

2.2. Unified repartitioning algorithm

The Unified Repartitioning Algorithm (URA) [19] is a parallel adaptive repartitioning scheme for the dynamic load balancing of scientific simulations. A key parameter used in URA is the Relative Cost Factor that describes the relative times required for inter-processor communication and data redistribution during load balancing. Using this parameter, it is possible to combine the two minimization objectives (namely, inter-processor communication and data redistribution) of the adaptive graph partitioning problem into a precise cost function. 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, the coarsest graph is repartitioned twice using alternative methods, which are optimized variants of the scratch-remap and global diffusion schemes. The cost functions are then computed for each of these schemes and the one with the lowest cost is selected. Note that, unlike our approach, URA actually repartitions using each of the two available schemes and then chooses the one yielding the lowest cost.

2.3. Dynamic load balancing for SAMR

In investigating dynamic load balancing (DLB) schemes for SAMR [11], Lan et al. analyzed application requirements, identified four unique characteristics (viz., coarse granularity, high dynamism, high imbalance, and different dispersion), and provided an implementation that maintains global information on each processor to manage the dynamic grid hierarchy. Their DLB scheme is composed of two phases: moving-grid phase and splitting-grid phase. During the "moving-grid phase", after each adaptation, the DLB is triggered if load is not balanced, i.e. $\mathit{MaxLoad}/\mathit{AvgLoad} > \mathit{threshold}$. The *MaxProc* moves its grid directly to *MinProc* under the condition that the computational load of this grid does not exceed ($\mathit{threshold} * \mathit{AvgLoad} - \mathit{MinLoad}$), thereby ensuring that an under-loaded processor does not become overloaded. If imbalance still exists, the "splitting-grid phase" is invoked, which splits the grid along the longest dimension into two sub-grids. Eventually, either the load is balanced or the minimum allowable grid size is reached. Both the moving-grid phase and splitting-grid phase may execute in parallel. Note that while this scheme does address SAMR, it only addresses dynamic load balancing and not adaptive partitioning.

2.4. Partitioning advisory system

Crandall and Quinn have developed a partitioning advisory system [9] for networks of workstations. The advisory system has three built-in partitioning methods, viz., contiguous row, contiguous point, and block. Given information about the problem space, the machine speed, and the network, the advisory system provides a 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. This work, however, does not address AMR applications.

3. Enabling autonomic partitioning for SAMR applications

A number of factors contribute to the overall performance of SAMR applications. The relationships between the contributing factors are often complex and depend on dynamic information such as the current state of the application and the system. Dynamic partitioning and load balancing for SAMR applications involves distributing application components on available resources to address parallelism, locality, and current computational, communication, and storage requirements. Partitioners typically optimize a subset of these requirements at the expense of others. Hence, the most appropriate partitioner and associated partitioning parameters depend on the application's configuration and its runtime state. Runtime partitioner selection, configuration, and optimization requires dynamically identifying the most appropriate combination of partitioner and parameters that match the current state of the application/system, and can be a significant challenge. This motivates the investigation of autonomic, self-adapting, and optimizing partitioning and runtime management solutions for SAMR applications. Such a solution consists of 3 steps:

Monitoring and characterization: The application state is monitored at regular intervals and application behavior

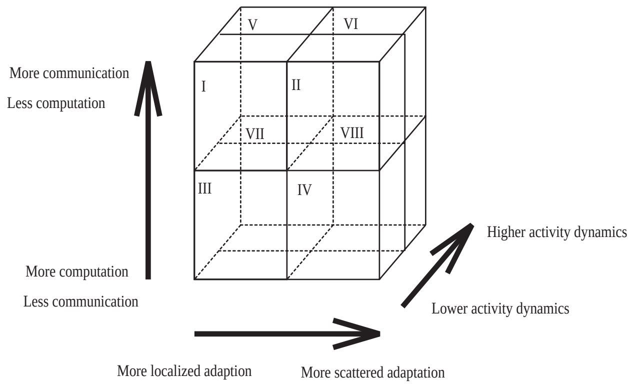

Fig. 1. The octant approach [22] for characterizing application state based on its adaptation pattern, computation/communication requirements, and activity dynamics.

is abstracted in terms of a partitioning quality metric that combines computation/communication requirements, application dynamics, and nature of adaptation, as defined in the following section.

Analysis and deduction: Using the abstracted application state and defined rules, constraints, and heuristics, the most appropriate partitioning algorithm and associated partitioning parameters (e.g., granularity) are selected at runtime from a family of available partitioners.

Adaptation and optimization: The selected partitioner is dynamically configured and invoked. Optimizations are performed to reduce the overheads due to monitoring and characterization and dynamic switching of partitioners.

The autonomic partitioning approach outlined above builds on our prior work on application sensitive characterization of domain-based partitioners [6,22] and application sensitive partitioning [5] for SAMR applications. In this section, we first briefly introduce the key building blocks for enabling autonomic SAMR partitioning, including the partitioning quality metric, the characterization of domain-based partitioners, the octant approach for classifying application state, and the policies for mapping partitioners to application state.

3.1. Characterizing SAMR runtime state

The *octant approach* [22], shown in Fig. 1, provides a means for characterizing the runtime state and corresponding partitioning requirements of a SAMR application. According to the octant approach, application state is classified with respect to (a) the adaptation pattern (scattered or localized), (b) whether runtime is dominated by computations or communications, and (c) the activity dynamics in the solution, e.g., adaptation pattern changes rapidly. Applications may start off in one octant and then migrate to others, as the solution progresses.

3.2. Partitioning quality metric

The dynamic partitioning requirements for a SAMR application and the performance of a partitioner can be abstracted using the five-component quality metric [22] outlined below.

1. *Load imbalance*: This component measures the load imbalance created by a partitioner. It occurs when any processor has more than average load and can cause performance to deteriorate rapidly. This metric component assumes greater significance in a computation-dominated environment.
2. *Communication requirement*: This component is a measure of the inter-processor communication required by the SAMR algorithm for a given partitioning of the SAMR grid hierarchy, and is determined by the geometrical intersections between SAMR grid components. This metric component assumes greater significance in a communication-dominated environment.
3. *Data migration*: This component is a measure of the amount of data that has to be moved between processors when the application transitions from one partitioning to the next. It represents the ability of the partitioner to consider the existing distribution. This metric component assumes greater significance for highly dynamic applications and scattered adaptations.
4. *Partitioning induced overhead*: This component attempts to capture the quality of a partition using the number of sub-grids and their aspect ratio. A large number of small grids may produce better load balance, but the increased communication can cause performance to deteriorate. Similarly, bad aspect ratio can adversely affect the computation-to-communication ratio for a grid component.
5. *Partitioning time*: This component represents how fast the partitioning algorithm computes the new partitioning, given an existing partitioning. It indicates the parallel efficiency of the partitioning algorithms employed.

Optimizing all the above components implies conflicting objectives. For example, optimizing components (1) and (2) together constitutes an *NP*-hard problem. Partitioners typically optimize a subset of the components at the expense of others. The quality metric enables the overall characterization of a partitioner and the definition of dynamic partitioning requirements of the SAMR application.

3.3. Characterizing partitioner behavior

ARMaDA addresses application-sensitive partitioning for SAMR applications, especially those based on variants of space-filling curve [18] based techniques such as the ones from the GrACE distributed AMR infrastructure [14,15] and the Vampire SAMR partitioning library [21]. The three primary partitioners considered in the ARMaDA research

Table 2
Recommendations for mapping octants onto partitioning schemes

Octant	Scheme
I	SFC
II	pBD-ISP, SFC
III	G-MISP+SP, SP-ISP, SFC
IV	G-MISP+SP, SFC
V	pBD-ISP
VI	pBD-ISP
VII	G-MISP+SP, pBD-ISP
VIII	pBD-ISP

are—inverse space-filling curve-based "composite" partitioner (SFC) from GrACE, *p*-way binary dissection inverse space-filling curve partitioner (pBD-ISP) and geometric multilevel inverse space-filling curve partitioner with sequence partitioning (G-MISP+SP) from Vampire. A characterization [6] of these partitioners using the above metric is summarized as follows.

SFC: The SFC scheme is fast, has average load balance, and generates low overhead, low communication and average data migration costs. This scheme is suited for application states associated with low-to-moderate activity dynamics and more computation than communication.

G-MISP+SP: The G-MISP+SP scheme is aimed towards speed and simplicity, and favors simple communication patterns and partitioning speed over amount of data migration. Sequence partitioning significantly improves the load balance but can be computationally expensive.

pBD-ISP: The pBD-ISP scheme is fast and the coarse granularity partitioning generated results in low overheads and communication costs. However, the overall load balance is average and may deteriorate when refinements are strongly localized. This scheme is suited for application states with greater communication and data requirements and lesser emphasis on load balance.

3.4. Mapping partitioners to application state octants

Partitioners can be mapped to application state octants based on the metric components they optimize and their ability to satisfy the partitioning requirements of the octant. Such a mapping, using experiments and heuristics [6,22], is summarized in Table 2.

To illustrate the use of this mapping in partitioner selection, consider an application that tracks an explosion, executing on a computer system with very low communication bandwidth. If the application has high activity dynamics, it will remain in "far part" of the cube (octants V–VIII). The explosion will give rise to many scattered clusters of adaptation, hence putting it in the "right part" of the cube (octants VI and VIII). Due to the low communication bandwidth, the runtime will probably be dominated by communication. Therefore, it will end up in the "upper part" of the cube (octant VI). Based on the available partitioners and their characterization summarized above, pBD-ISP meets the requirements of each of these octants and is a suitable choice for these application states.

4. Adaptive application-sensitive partitioning: a feasibility study

As mentioned previously, the partitioning requirements of a SAMR application depend on the application's configuration and its runtime state. The objective of this section is to manually demonstrate the benefits of adaptive application-sensitive partitioning based on a manual analysis of the SAMR application. In this section, we perform a feasibility study to illustrate the operation of such an adaptive SAMR partitioner. The adaptation policies are manually formulated using a SAMR simulator [20] and these policies are used to adaptively select and invoke the most suitable partitioner from among the available SFC, G-MISP+SP, and pBD-ISP domain-based partitioners. The advantages and benefits of adaptive partitioning are experimentally evaluated. The SAMR application used in this evaluation is a basic, rudimentary version of the RM3D compressible turbulence kernel ($RM3D_b$) solving the Richtmyer–Meshkov instability in 3-D. 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.

4.1. Charaterizing application state

Application characterization consists of two steps. First, the adaptive behavior of the application is captured in an adaptation trace generated using a single processor run. The adaptation trace contains snap-shots of the SAMR grid hierarchy at each regrid step. This trace is then analyzed using the octant approach and the adaptive partitioning strategy is defined. Note that while application characterization in this example is performed manually, this process is automated in ARMaDA as described in Section 5.

The application is executed for 800 coarse level time-steps and the trace consists of over 200 snap-shots. A selection of these snap-shots is shown in Fig. 2 to illustrate the dynamics of the $RM3D_b$ application. $RM3D_b$ starts out with a scattered adaptation, lower activity dynamics, and more computation, placing it initially in octant IV. Either G-MISP+SP or SFC is the most appropriate partitioner for this octant. As the application evolves, its adaptation patterns cause it to move between octants and may require different partitioning schemes. At regrid step 106, the $RM3D_b$ application has high communication requirements, high dynamics, and a scattered adaptation, placing it in octant VI. pBD-ISP is the partitioner of choice in this octant. At the end of the simulation, the application adaptations are localized, with increased computation and lower activity, placing the

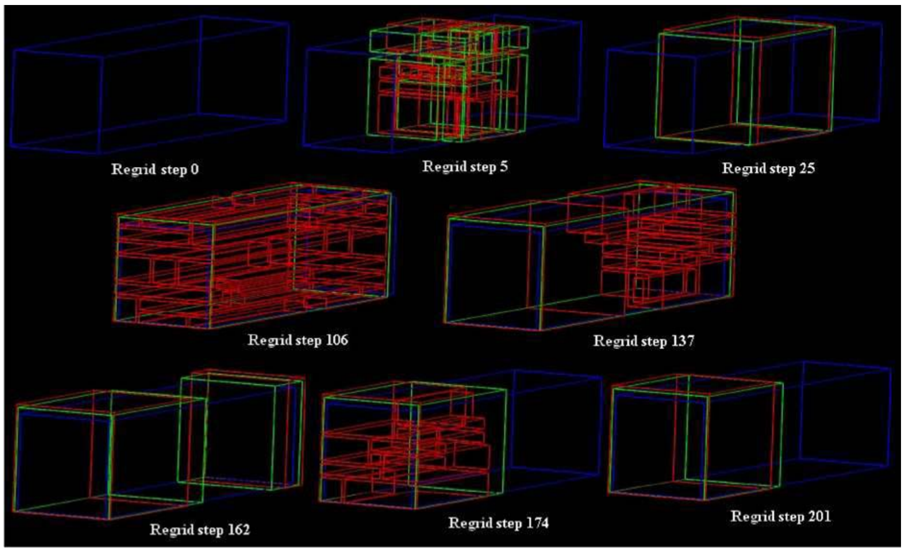

Fig. 2. Sequence of profile views at sampled regrid steps for $RM3D_b$ application.

application in octant III and G-MISP+SP is the appropriate partitioner.

4.2. Formulating partitioner adaptation policy using SAMR simulator

Once the $RM3D_b$ application trace containing snap-shots of the SAMR grid hierarchy is obtained, a SAMR simulator [20] is used to analyze the behavior of a partitioner for the application on 64 processors. Note that the adaptation policy presented here is formulated manually based on a manual analysis of the simulator data. In the discussion below, we only present the simulator analysis for the SFC and pBD-ISP partitioners. As the G-MISP+SP scheme has similar characteristics as the SFC partitioner, its analysis is not presented. The SAMR simulator uses the trace to determine the performance of the partitioners at each regrid step in terms of the primary components of the quality metric (communication, data movement, load imbalance) presented in Section 3.2. Note that the other two metric components, partitioning overhead and partitioning time, are intrinsic characteristics of a partitioner. The partitioning time is directly dependent on the three primary components that determine the runtime performance of the partitioner. Figs. 3–5 plot the maximum values of the total communication, data movement, and load imbalance components of the quality metric respectively for SFC and pBD-ISP partitioners on 64 processors. Analyzing these graphs, five operational zones can be identified.

1. *Zone* I (0 *to* $\approx$ 25): This is the application start-up zone and is typically associated with low communication and data migration. In this zone, we observe that the communication and data movement costs for both partitioners are similar. The deciding factor then becomes the load imbalance. Though SFC starts with high load imbalance, this is quickly corrected to achieve improved load balance. In this zone, the G-MISP+SP scheme starts with and maintains low imbalance. In contrast, the pBD-ISP scheme produces relatively high load imbalance in this zone. Hence, G-MISP+SP or SFC is the preferred partitioner for this zone.
2. *Zone* II ($\approx$ 25 *to* $\approx$ 135): In this zone, there are substantial communication and data movement costs as the application evolves. The pBD-ISP scheme aims at reducing these costs while maintaining moderate-to-low load imbalance. Though the SFC and G-MISP+SP partitioners yield better load balance than pBD-ISP, they produce larger data movement and communication overheads that adversely affect performance. Resolving these conflicting objectives, the pBD-ISP scheme is expected to perform better in this zone.
3. *Zone* III ($\approx$ 135 *to* $\approx$ 150): This zone is characterized by high activity dynamics and greater communication requirements. The SFC and G-MISP+SP schemes outperform pBD-ISP in, both, the load balance and communication metric components, while producing comparable data movement. Clearly, G-MISP+SP or SFC is the appropriate partitioner for this zone.
4. *Zone* IV ($\approx$ 150 *to* $\approx$ 175): In this zone, pBD-ISP gives lower data movement and communication overheads and produces an acceptable load imbalance as compared to the G-MISP+SP or SFC scheme, making it the partitioner of choice for the zone.
5. *Zone* V ($\approx$ 175 *to* $\approx$ 200): This is the relatively “quiet” final stage of the $RM3D_b$ application. The data migration and communication requirements are low and constant. Although all schemes perform equally well with respect to the quality metric, the G-MISP+SP or SFC scheme is slightly preferred as compared to pBD-ISP due to the better load balance produced.

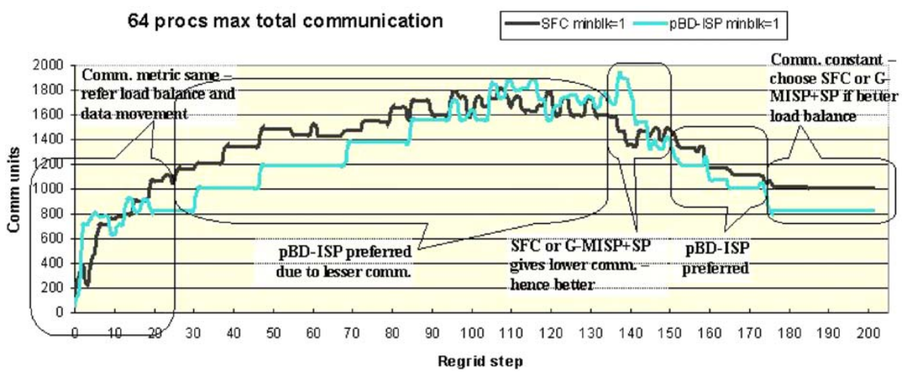

Fig. 3. Simulator analysis: maximum total communication for $RM3D_b$ on 64 processors.

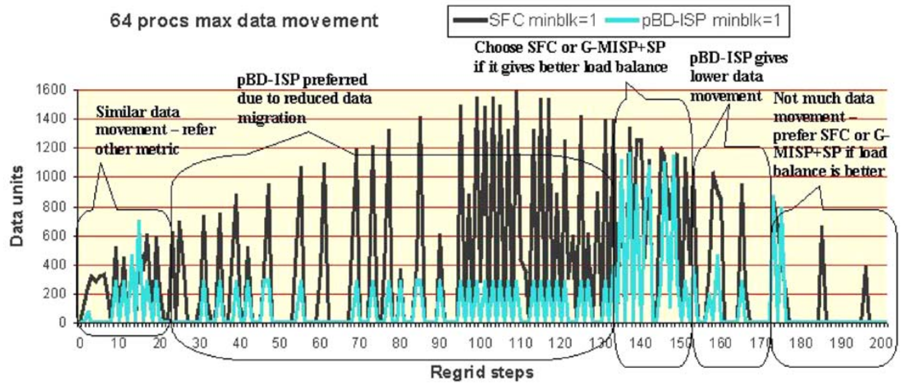

Fig. 4. Simulator analysis: maximum data movement for $RM3D_b$ on 64 processors.

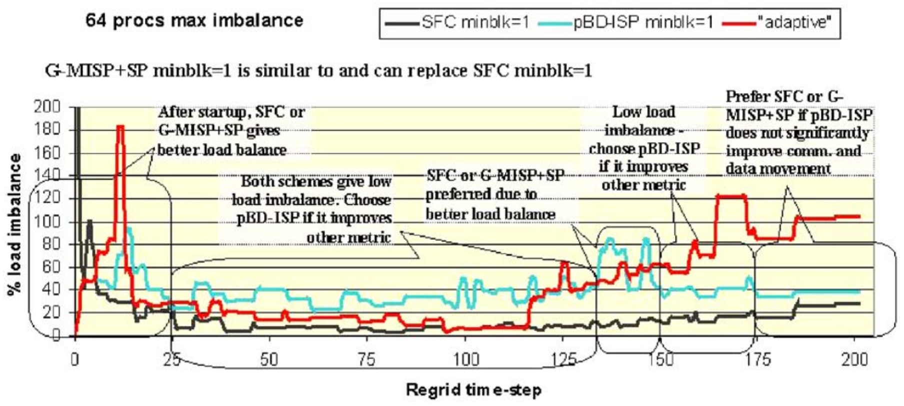

Fig. 5. Simulator analysis: maximum load imbalance for $RM3D_b$ on 64 processors.

An application-sensitive partitioner adaptation policy (denoted by "adaptive" in this example) for $RM3D_b$ can be formulated using the simulator analysis for application-sensitive partitioning as shown in Table 3. The regrid step is used as the switching point where a new partitioner is selected, configured, and invoked.

4.3. Experimental evaluation of the adaptive partitioner

The adaptive partitioning policy, manually formulated above, is experimentally evaluated on the NPACI IBM SP2 "Blue Horizon" at the San Diego Supercomputing Center, using the $RM3D_b$ application with a base grid of size 128*32*32, 3 levels of factor 2 space–time refinements, and regridding performed every 4 time-steps. The experimental evaluation compares the overall execution times for application runs using individual partitioners as well as the adaptive partitioning strategy outlined above. The partitioner as well as the partitioning parameters are switched on-the-fly during application execution. The results of the experiments for 64 processors are listed in Table 4. Note

Table 3
Policy formulation for adaptive partitioning for *RM3D*$_b$ application

Zone	Starting regrid step	Ending regrid step	Partitioning strategy
I	0	22	G-MISP+SP
II	23	134	pBD-ISP
III	135	148	G-MISP+SP
IV	149	174	pBD-ISP
V	175	201	G-MISP+SP

Table 4
Adaptive partitioning evaluation for *RM3D*$_b$ application on 64 processors

Partitioner	Execution time (s)	Max. Load Imbalance (%)	AMR Efficiency (%)
SFC	484.502	24.88	98.82
G-MISP+SP	405.062	11.32	98.78
pBD-ISP	414.952	35.03	98.86
“Adaptive”	352.824	8.12	98.76

that the “adaptive” evaluation in this example uses a combination of pBD-ISP and G-MISP+SP partitioners, rather than the SFC partitioner presented in the simulator analysis. This results in improved load balance during an actual application run with all processors utilized. The basic *RM3D*$_b$ version used in this evaluation serves as a “feasibility study” example and is significantly different from the RM3D kernel evaluated using the ARMaDA framework in Section 6.3, resulting in different execution times.

The above results demonstrate that an adaptive application-sensitive partitioning policy that dynamically switches partitioners (and associated partitioning parameters) at runtime improves application performance and results in reduced execution times. For 64 processors, the improvement is 27.2% over the slowest partitioner. In the adaptive partitioner case, G-MISP+SP improves the load balance when the application is computationally dominated, while pBD-ISP reduces communication and data-migration overheads. This “feasibility study” example demonstrates the potential benefits of adaptive application-sensitive partitioning and motivates the development of the ARMaDA autonomic partitioning framework.

5. ARMaDA autonomic partitioning framework

ARMaDA is an autonomic partitioning framework for SAMR applications that dynamically selects and configures partitioning algorithms at runtime to optimize overall application performance. The ARMaDA framework has three components: application state monitoring and characterization component, partitioner selection component and policy engine, and runtime adaptation component. The state characterization component implements mechanisms that abstract and characterize the current application state in terms of the computation/communication requirements, application dynamics, and the nature of the adaptation. The policy engine defines the associations between application state and partitioner behavior. Using the characterized application state and the defined policies, the appropriate partitioning algorithm is selected at runtime from the available partitioners in the repository. The partitioners used includes a selection from software tools such as GrACE [14,15] and Vampire [21]. The runtime adaptation component dynamically configures the selected partitioner with appropriate partitioning parameters and invokes it at each regrid step. Note that ARMaDA does not require any a priori knowledge about the behavior of the application or its adaptation patterns.

5.1. Automated application state monitoring and characterization

The current state of a SAMR application and its partitioning requirements are determined using the structure of the SAMR grid hierarchy, which is expressed as sets of bounding boxes assigned to the different processors at various levels of refinement. These boxes are used to inexpensively compute the computation/communication requirements, application dynamics, and the nature of the adaptation using simple geometric operations as follows.

5.1.1. Computation/communication requirements

The computation-to-communication ratio (CCratio) determines whether the current application state is computationally intensive or communication-dominated. Since it is quite challenging to determine the exact computation and communication requirements of the SAMR grid hierarchy at runtime, a simple approximation based on geometry of the structured application domain is used to deduce these values. The volume of the bounding boxes at different levels of refinement gives an estimate of the computational work while the surface area gives an estimate of the communication associated with the grid hierarchy. Consequently, the ratio of the total volume to total surface area of the bounding boxes determines CCratio for the current state of the application.

$$\text{CCratio} = \frac{\sum(\text{Volume of bounding boxes})}{\sum(\text{Surface area of bounding boxes})}.$$

5.1.2. Application dynamics

The application dynamics (Dynamics) estimate the rate of change of the application refinement patterns and are determined by comparing the current application state with its previous state. Information about the previous application state is stored within the application state monitoring and characterization component. To compute the Dynamics metric, the overlap between the current set of bounding boxes and the previous set of boxes at each level is computed. A greater overlap region indicates that the application has not changed significantly and has low activity dynamics. A smaller overlap region represents high activity dynamics.

$$\begin{aligned}\text{Dynamics} = \text{Size of }(&\text{Current state boxes}\\ &\cap \text{Previous state boxes}).\end{aligned}$$

5.1.3. Nature of adaptation

The nature of the adaptation (Adapt) captures the adaptation pattern, i.e., whether the refinements are scattered or localized. Scattered refinements can give rise to greater overheads. Determining exact refinement patterns and their location in the SAMR grid hierarchy can once again be quite complex. A simpler, approximate algorithm determines the nature of adaptation based on the number and geometric arrangement of refinement regions within the overall domain under consideration. A higher number of refinement patches is an indicator of possibly scattered refinements. Since differences in adaptation overheads do not significantly affect partitioner performance, this simple scheme provides a reasonable assessment of the nature of adaptation.

$$\text{Adapt} = \frac{\text{Volume of refinement regions}}{\text{Domain volume}} * \text{Number of refinement patches.}$$

5.1.4. Runtime state characterization

The three metrics described above can now be used to effectively estimate the state of the SAMR application at runtime. A key concern during partitioner selection and adaptation in ARMaDA is the possibility of thrashing due to very frequent application state changes which, in turn, can result in frequent switching of partitioners and increased overheads. To avoid thrashing, the ARMaDA framework maintains a history of the application state by storing the structure of the grid hierarchy for two preceding regrid steps. This 3-step sliding window (current and two preceding) is used heuristically to detect transient changes and avoid oscillations in application state by analyzing the magnitude of the metric change (going low or high). Normalized application metric ratios are computed as follows.

Let M_r denote any one of the three metrics that are computed from the state of the grid hierarchy at regrid step r. The normalized metric ratio is then computed as

$$\text{Mratio} = \frac{\text{currM}_r * \text{currM}_{r-2}}{(\text{currM}_{r-1})^2}.$$

Three normalized ratios are computed corresponding to the three metrics, viz., computation/communication ratio “Cratio”, application dynamics ratio “Dratio”, and adaptation ratio “Aratio”.

5.2. Policy-based partitioner selection

The three normalized ratios (Cratio, Dratio, and Aratio) computed by the state characterization component are used, along with application state thresholds and application metric weights, to assign a bit to each metric and adjust metric values to closely match the needs of the dynamic application. The three bits are then combined to map the application to a specific application state octant. The appropriate partitioner is then selected from the set of partitioners that are mapped to the octant and satisfy the partitioning requirements associated with that octant.

Application state thresholds are user-defined parameters and represent the lower and upper limiting values for the normalized metric ratios for a particular application state. A small threshold value can lead to thrashing resulting in frequent state changes and switching of partitioners, causing increased overheads and unstable adaptations. A large threshold value can result in fewer state changes being detected and insufficient adaptations, causing degraded performance. In our evaluation, application state thresholds ranging between 20% and 50% provide reasonable overall performance. The application metric weights are heuristic parameters that can be set by the user to fine-tune metric sensitivity when prior knowledge about the application characteristics is available. For example, in a computationally intensive SAMR application, the weight for Cratio (computation/communication ratio) can be set higher than the weights for other metric ratios, making the partitioner selection strategy more sensitive to changes in the computational requirements of the application. Default metric weights of 1.0 can be used when this information is not known a priori.

Let *Mratio* and *Mbit* represent the normalized ratio and the octant-bit value for a particular metric *M*. Let *lowMwt* and *highMwt* denote the application metric weights for the low and high application state thresholds (represented by *LOW_THRESH* and *HIGH_THRESH*), respectively. The metric ratio to octant bit mapping is given by

$$\begin{aligned}\text{Mbit} &= 0,\ \text{if Mratio} < (\text{lowMwt} * \text{LOW\_THRESH})\\ &= 1,\ \text{if Mratio} >= (\text{highMwt} * \text{HIGH\_THRESH})\\ &\rightarrow \text{else, remain unchanged.}\end{aligned}$$

Three octant bits,“Cbit”, “Dbit”, and “Abit” corresponding to the three metric ratios are computed in this way. A low Cbit ($Cbit = 0$) represents more communication while a high Cbit ($Cbit = 1$) indicates greater computation. A low Dbit ($Dbit = 0$) indicates greater activity whereas a high

Table 5
Association of application state to octants and partitioners within the ARMaDA framework

ARMaDA state C D A	Octant position	Suitable partitioner
0 0 0	V	pBD-ISP
0 0 1	VI	pBD-ISP
0 1 0	I	SFC
0 1 1	II	pBD-ISP, SFC
1 0 0	VII	G-MISP+SP, pBD-ISP
1 0 1	VIII	pBD-ISP
1 1 0	III	G-MISP+SP, SP-ISP, SFC
1 1 1	IV	G-MISP+SP, SFC

Dbit ($Dbit = 1$) represents lesser application dynamics. A low Abit ($Abit = 0$) denotes more localized adaptation while a high Abit ($Abit = 1$) indicates that the nature of adaptation is scattered. These bits are then used to identify the ARMaDA characterized state, which is then mapped to an application state octant and associated partitioners using the policies as shown in Table 5.

5.3. *Adaptive application-sensitive partitioning*

The ARMaDA adaptation component configures the selected partitioner with appropriate partitioning parameters (such as partitioning granularity) and invokes it to partition and load balance the SAMR grid hierarchy. A common interface is defined for the available partitioners allowing them to be uniformly invoked. The partitioner parameters are selected based on the characterization of the application state and the determination of the octant in which the application currently resides. The granularity is based on the requirements of the octant, though it may be overridden by a user defined value.

The overhead associated with ARMaDA is the effort required to (a) determine and characterize the application state by monitoring the SAMR grid hierarchy at runtime, (b) select and configure the appropriate partitioner and partitioning parameters based on the defined policy and thresholds, and (c) invoke the new partitioner to partition the SAMR grid hierarchy. Note that these steps are performed at runtime when the application regrids between two phases of computation.

A key concern in adaptive partitioning is ensuring that the overheads of characterizing the application state and switching partitioners (if necessary) at runtime do not offset the benefits of adaptation. ARMaDA uses efficient and inexpensive mechanisms based on the union and interaction of the structured geometry of the grid components to characterize application state and compute the required metric ratios. Furthermore, ARMaDA provides optimizations and control parameters that can be used to customize the sensitivity, quality, and overheads of application monitoring and partitioner adaptation. For example, if the application state remains unchanged for a specific period (defined by the parameter SAME_REGRID_LIMIT), ARMaDA can be directed to stop sensing application state for a certain number of regrid steps (defined by the parameter SKIP_SAME_REGRID), thereby reducing monitoring overheads. Similarly, when a change in runtime state causes a dynamic switching of partitioners, it is assumed that application characteristics do not change significantly in the immediately succeeding regrids. In this case, runtime state sensing is paused for SKIP_SWITCH steps. These three optimization parameters (SAME_REGRID_LIMIT, SKIP_SAME_REGRID, and SKIP_SWITCH) are user-defined and attempt to control adaptation overheads. Very small or large values of these parameters can, respectively, lead to excessive or insufficient adaptation, resulting in degraded application performance. In our evaluation, 2 or 4 regrid steps are reasonable parameter values for optimization.

6. Evaluation of ARMaDA framework

The experimental evaluation of the ARMaDA framework is performed using various SAMR application kernels. The experiments consist of measuring overall application execution times for different partitioner configurations using the ARMaDA framework. The partitioners used in the evaluation include SFC, G-MISP+SP, pBD-ISP, and adaptive combinations of these partitioners determined at runtime based on application state. With the exception of the partitioning strategy and associated partitioning granularity, all application-specific and refinement-specific parameters are kept constant.

In the adaptive partitioning experiments using ARMaDA, an initial partitioner is selected by the user along with other partitioning parameters and thresholds. The initial partitioner is used for the initial distribution of the SAMR grid hierarchy, while the partitioners used in subsequent distributions are autonomically determined by ARMaDA. The experimental results demonstrate that autonomic application-sensitive partitioning using ARMaDA can improve application performance as compared to non-adaptive partitioning. The results also show that the choice of the initial partitioner does not significantly affect the performance of the application as ARMaDA dynamically adapts to select the most appropriate partitioner based on the application state and its partitioning requirements.

Note that the feasibility study presented in Section 4 uses an application trace and its characterization to manually define the runtime adaptation policy which yields benefits. Since ARMaDA automates the characterization and adaptation processes, a priori knowledge of the runtime characteristics of the SAMR application is not required to make

Table 6
ARMaDA evaluation for TportAMR-2D application on 8 processors on "Discover"

Partitioner	Execution time (s)
SFC alone (no ARMaDA)	352.091
ARMaDA with SFC start	349.286
G-MISP+SP alone (no ARMaDA)	360.946
ARMaDA with G-MISP+SP start and thrashing	353.44
pBD-ISP alone (no ARMaDA)	344.558
ARMaDA with pBD-ISP start	342.646

runtime decisions—ARMaDA autonomically adapts and optimizes the partitioner.

6.1. TportAMR-2D on "Discover"

"Discover" is a 16-node Linux Beowulf cluster comprising 8 dual processor Pentium IIIs at The Applied Software Systems Laboratory at Rutgers University. The Transport AMR 2D (TportAMR-2D) application is a benchmark kernel solving the transport equation and is primarily communication-dominated with high adaptation overheads. This evaluation is performed on 8 processors of Discover using an application base grid of size 64*64 and 3 levels of factor 2 space–time refinements. Regridding is performed every 4 time-steps at each level and the application executes for 40 coarse level time steps. The experiments compare application runs using each individual partitioner in "stand alone" mode without ARMaDA and as initial partitioners with ARMaDA. The application execution times are listed in Table 6.

The TportAMR-2D application used in this experiment is computationally inexpensive but requires considerable communication and data movement. Furthermore, the experiment uses a small domain and few iterations. Since the pBD-ISP partitioner is particularly suited to strongly communication-dominated application states and reduces communication and data migration costs [22], it is expected to perform better than the other partitioners. The results in Table 6 show that the ARMaDA framework with pBD-ISP as the initial partitioner performs the best and its performance is similar to the stand alone pBD-ISP case. When a different initial partitioner is used, the ARMaDA framework dynamically switches to pBD-ISP and results in a slightly better performance than the stand alone case. In this evaluation, the overheads associated with the ARMaDA framework range between 40 and 100 ms for an entire run of approximately 350 s. Though thrashing is detected in the case of G-MISP+SP with switching between G-MISP+SP and pBD-ISP partitioners, its effects are not significant due to the ARMaDA optimizations described earlier. Due to the application characteristics, adaptive partitioning does not produce significantly different execution times in this evaluation.

Table 7
ARMaDA evaluation for VectorWave-2D application on 32 processors on "Frea"

Partitioner	Execution time (s)
SFC	637.478
G-MISP+SP	611.749
pBD-ISP	592.05
ARMaDA with SFC start	470.531

6.2. VectorWave-2D on "Frea"

"Frea" is a 64-node Linux Beowulf cluster also at The Applied Software Systems Laboratory at Rutgers University. The VectorWave-2D application is a coupled set of partial differential equations and forms a part of the Cactus 2-D numerical relativity toolkit, solving Einstein's and gravitational equations. This evaluation is performed on 32 processors of Frea using an application base grid of size 128*128 and 3 levels of factor 2 space–time refinements. Regridding is performed every 4 time-steps at each level and the application runs for 60 coarse level time steps. The application execution times for the individual partitioners and for ARMaDA with SFC start are presented in Table 7.

The VectorWave-2D application is primarily computation-dominated, requiring good load balance and reduced communication and data migration costs. SFC and pBD-ISP partitioners optimize communication and data migration metric components while G-MISP+SP gives good load balance. In this evaluation, the ARMaDA partitioner with SFC start improves performance by 26.19% over the slowest partitioner. The ARMaDA framework overhead for state sensing and adaptation in this case is 0.0616% of the total execution time, which is negligible.

6.3. RM2D & RM3D on "Blue Horizon"

The NPACI IBM SP2 "Blue Horizon" is a Teraflop-scale Power3 based clustered SMP system at the San Diego Supercomputing Center. RM2D and RM3D are 2-D and 3-D versions, respectively, of a compressible turbulence kernel solving the Richtmyer–Meshkov instability. This evaluation is performed on 64 processors of Blue Horizon. The RM2D evaluation uses a base grid of size 128*32 and 60 coarse level time steps, while RM3D uses a base grid of size 128*32*32 and 100 coarse level time steps. Both evaluations use 3 levels of factor 2 space–time refinements and regridding is performed every 4 time-steps at each level. The application execution times for the individual partitioners and ARMaDA for RM2D and RM3D are listed in Tables 8 and 9, respectively.

Both, RM2D and RM3D start out as computation-dominated with low activity dynamics and a more scattered adaptation, placing them in octant IV. As the applications evolve, their communication requirements become more

Table 8
RM2D execution times for ARMaDA partitioners on 64 processors on "Blue Horizon"

Partitioner	Execution time (s)
SFC	264.041
G-MISP+SP	214.745
pBD-ISP	199.738
ARMaDA with G-MISP+SP start	190.431

Table 9
RM3D execution times for ARMaDA partitioners on 64 processors on "Blue Horizon"

Partitioner	Execution time (s)
SFC	5065.51
pBD-ISP	4016.91
ARMaDA with SFC start	3126.3

Table 10
ARMaDA adaptive policy for RM3D application on 64 processors on "Blue Horizon"

Iteration number	RM3D State C D A	Octant position	Appropriate partitioner
1–6	1 1 1	IV	SFC
7–16	0 1 1	II	pBD-ISP
17–26	1 1 0	III	SFC
27–36	0 1 1	II	pBD-ISP
37–46	1 1 0	III	SFC
47–56	0 1 1	II	pBD-ISP
57–66	1 0 0	VII	SFC
67–84	0 0 1	VI	pBD-ISP
84–100	1 1 0	III	SFC

significant. Moreover, the applications exhibit different dynamics and adaptations at different stages of their execution. The pBD-ISP scheme improves communication and data migration metric components and is seen to perform better than the SFC and G-MISP+SP partitioners. The ARMaDA adaptive partitioner starts out with either the G-MISP+SP or SFC partitioner as these partitioners are better suited for the initial stages of the application. As an illustrative example, the ARMaDA settings used for RM3D evaluation are SAME_REGRID_LIMIT = 4, SKIP_SAME_REGRID = 4, SKIP_SWITCH = 4, LOW_THRESH = 0.6, HIGH_THRESH = 1.4, and metric weights (Cwt, Dwt, Awt) = 1.0. The adaptive partitioner selection used by ARMaDA in case of RM3D is listed in Table 10.

The overheads associated with the ARMaDA framework are low. For the RM2D and RM3D evaluations, the overheads are 0.415 and 1.85 s, respectively, and are negligible as compared to the overall application execution times. These experimental results demonstrate that autonomic partitioning using the ARMaDA framework reduces execution time and improves application performance. In the case of RM2D application, improvements due to ARMaDA are 4.66%, 11.32%, and 27.88% over pBD-ISP, G-MISP+SP, and SFC partitioners, respectively. In the case of RM3D application, these improvements are 22.17% over the pBD-ISP partitioner and 38.28% over the SFC scheme. Note that the RM3D kernel used in this evaluation is significantly different from the basic *RM3D*$_b$ version evaluated using a manually orchestrated policy in Section 4.3, exhibiting different physics and, hence, different execution times.

7. Conclusion and future work

This paper presents the design, implementation, and evaluation of ARMaDA, an autonomic application-sensitive partitioning framework for SAMR applications. The overall goal of ARMaDA is to manage the dynamism and space-time heterogeneity, and support the efficient and scalable execution of SAMR implementations for dynamically adaptive scientific and engineering simulations. The ARMaDA framework provides mechanisms for automatically characterizing the dynamic application state at runtime, and abstracting the current computational, communication, and storage requirements. This runtime state is then used by ARMaDA to dynamically select, configure, and invoke the appropriate partitioning strategy from an available repository of partitioners. The performance of the ARMaDA framework is experimentally evaluated using various SAMR application kernels. The experimental results validate the advantages of adaptive application-sensitive partitioning and the ARMaDA autonomic partitioning framework.

The ARMaDA framework focuses on application-sensitive adaptations and assumes uniform system characteristics (no system adaptations considered). Since a fully autonomic system must have awareness of the execution environment also, we plan to investigate runtime machine characteristics as part of the future work. The framework could be extended to include system-sensitive capabilities so as to use information about the CPU, memory, and bandwidth availabilities of the computing elements to perform better adaptations. Permutations of these policies along with system-sensitive rules could be applied at runtime to accurately model the behavior and optimize the performance of SAMR applications.

Acknowledgments

The authors would like to thank Johan Steensland for access to Vampire in this research. The authors are grateful to Ravi Samtaney and Mijan Huq for making the applications available for use.

References

[1] ASCI/ASAP Center, http://www.cacr.caltech.edu/ASAP, California Institute of Technology.

[2] M. Berger, G. Hedstrom, J. Oliger, G. Rodrigue, Adaptive Mesh Refinement for 1-Dimensional Gas Dynamics, Scientific Computing (IMACS/North-Holland Publishing), 1983, pp. 43–47.
[3] M. Berger, J. Oliger, Adaptive Mesh Refinement for Hyperbolic Partial Differential Equations, J. Comput. Phys. 53 (1984) 484–512.
[4] G. Bryan, Fluids in the Universe: Adaptive Mesh Refinement in Cosmology, Comput. Sci. Eng. (March-April 1999) 46–53.
[5] S. Chandra, M. Parashar, ARMaDA: An Adaptive Application-Sensitive Partitioning Framework for SAMR Applications, in: Proceedings of the IASTED International Conference on Parallel and Distributed Computing and Systems, Cambridge, MA, November 2002.
[6] S. Chandra, M. Parashar, An Evaluation of Partitioners for Parallel SAMR Applications, in: R. Sakellariou, J. Keane, J. Gurd, L. Freeman (Eds.), Proceedings of the 7th International Euro-Par Conference, Lecture Notes in Computer Science, vol. 2150, Springer, Berlin, August 2001, pp. 171–174.
[7] S. Chandra, M. Parashar, 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, 10th International Conference on High Performance Computing (HiPC), Hyderabad, India, Elite Publishing, December 2003, pp. 286–295.
[8] M. Choptuik, Experiences with an Adaptive Mesh Refinement Algorithm in Numerical Relativity, in: C. Evans, L. Finn, D. Hobill (Eds.), Frontiers in Numerical Relativity, Cambridge University Press, London, 1989, pp. 206–221.
[9] P. Crandall, M. Quinn, A Partitioning Advisory System for Networked Data-parallel Processing, Concurrency: Practice and Experience 7 (5) (August 1995) 479–495.
[10] S. Hawley, M. Choptuik, Boson Stars Driven to the Brink of Black Hole Formation, Phys. Rev. D 62 (2000) 104024.
[11] Z. Lan, V. Taylor, G. Bryan, Dynamic Load Balancing for Structured Adaptive Mesh Refinement Applications, in: Proceedings of the 30th International Conference on Parallel Processing, Valencia, Spain, 2001.
[12] M. Norman, G. Bryan, Cosmological Adaptive Mesh Refinement, Numerical Astrophysics 1998, Kluwer Academic Publishers Dordrecht, The Netherlands, 1999.
[13] L. Oliker, R. Biswas, PLUM: Parallel Load Balancing for Adaptive Unstructured Meshes, J. Parallel Distrib. Comput. 52 (2) (1998) 150 –177.
[14] M. Parashar, http://www.caip.rutgers.edu/~parashar/TASSL/Projects/GrACE, GrACE homepage, 2001.
[15] M. Parashar, J. Browne, C. Edwards, K. Klimkowski, A Common Data Management Infrastructure for Adaptive Algorithms for PDE Solutions, in: Proceedings of the Supercomputing Conference, ACM/IEEE Computer Society, San Jose, CA, November 1997.
[16] M. Parashar, J. Wheeler, G. Pope, K. Wang, P. Wang, A New Generation EOS Compositional Reservoir Simulator: Part II—Framework and Multiprocessing, in: Proceedings of the Society of Petroleum Engineering Reservoir Simulation Symposium, Dallas, TX, June 1997.
[17] R. Pember, J. Bell, P. Colella, W. Crutchfield, M. Welcome, Adaptive Cartesian Grid Methods for Representing Geometry in Inviscid Compressible Flow, in: Proceedings of the 11th AIAA Computational Fluid Dynamics Conference, Orlando, FL, July 1993.
[18] H. Sagan, Space-filling Curves, Springer, Berlin, 1994.
[19] K. Schloegel, G. Karypis, V. Kumar, A Unified Algorithm for Load-balancing Adaptive Scientific Simulations, in: Proceedings of the International Conference on Supercomputing, Dallas, TX, November 2000.
[20] M. Shee, Evaluation and Optimization of Load Balancing/Distribution Techniques for Adaptive Grid Hierarchies, M.S. Thesis, Graduate School, Rutgers University, NJ, 2000.
[21] J. Steensland, http://www.caip.rutgers.edu/~johans/vampire, Vampire homepage, 2000.
[22] J. Steensland, S. Chandra, M. Parashar, An Application-Centric Characterization of Domain-Based SFC Partitioners for Parallel SAMR, IEEE Transactions on Parallel and Distributed Systems, Vol. 13, No. 12, IEEE Computer Society Press, Silver Spring, MO, December 2002, pp. 1275–1289.
[23] P. Wang, I. Yotov, T. Arbogast, C. Dawson, M. Parashar, K. Sepehrnoori, A New Generation EOS Compositional Reservoir Simulator: Part I—Formulation and Discretization, in: Proceedings of the Society of Petroleum Engineering Reservoir Simulation Symposium, Dallas, TX, June 1997.

Sumir Chandra is a Ph.D. student in the Department of Electrical and Computer Engineering at Rutgers, The State University of New Jersey. He received a B.E. degree in Electronics Engineering from Bombay University, India, and a M.S. degree in Computer Engineering from Rutgers University. Sumir is a researcher at The Applied Software Systems Laboratory at the Center for Advanced Information Processing at Rutgers. His research interests include parallel and distributed computing, autonomic computing, scientific computing, software engineering, runtime management, performance evaluation and prediction. Sumir received the Best Research Poster Award at the Supercomputing Conference at Denver, USA, in 2001. He has co-authored 12 technical papers in international journals and conferences, has co-authored/edited 3 book chapters/proceedings, and has reviewed several technical contributions in prestigious journals and conferences. Additional information is available at http://www.caip.rutgers.edu/~sumir/.

Manish Parashar is Associate Professor of Electrical and Computer Engineering at Rutgers University, where he is also director of the Applied Software Systems Laboratory. He received a BE degree in Electronics and Telecommunications from Bombay University, India, and MS and Ph.D. degrees in Computer Engineering from Syracuse University. He has received the NSF CAREER Award and the Enrico Fermi Scholarship from Argonne National Laboratory. His current research interests include parallel and distributed computing (including autonomic, Grid and peer-to-peer computing) with applications to computational science and engineering, networking and software engineering. Manish is a member of the executive committee of the IEEE Computer Society Technical Committee on Parallel Processing (TCPP), part of the IEEE Computer Society Distinguished Visitor Program (2004-2006), and a member of ACM. He is also the co-founder of the IEEE International Conference on Autonomic Computing (ICAC). Manish has co-authored over 140 technical papers in international journals and conferences, has co-authored/edited 6 books/proceedings, and has contributed to several others in the area of computational science and parallel and distributed computing. For more information please visit http://www.caip.rutgers.edu/~parashar/.