Self-Adapting, Self-Optimizing Runtime Management of Grid Applications using PRAGMA*

H. Zhu and M. Parashar
The Applied Software System Laboratory
Dept. of Electrical and Computer Engineering
Rutgers, The State University of New Jersey
Piscataway, NJ 08854, USA
hailan,parashar@caip.rutgers.edu

J. Yang, Y. Zhang, S. Rao and S. Hariri
High Performance Distributed Computing
Dept. of Electrical and Computer Engineering
University of Arizona
Tucson, AZ 85721, USA
jm_yang,zhang,sojanya,hariri@ece.arizona.edu

Abstract

The emergence of the computational Grid and the potential for seamless aggregation, integration and interactions has made it possible to conceive a new generation of realistic, scientific and engineering simulations of complex physical phenomena. The inherently heterogeneous and dynamic nature of these application and the Grid presents significant runtime management challenges. In this paper we extend the PRAGMA framework to enable self adapting, self optimizing runtime management of dynamically adaptive applications. Specifically, we present the design, prototype implementation and initial evaluation of policies and mechanisms that enable PRAGMA to autonomically manage, adapt and optimize structured adaptive mesh refinement applications (SAMR) based on current system and application state and predictive models for system behavior and application performance. We use the 3-D adaptive Richtmyer-Meshkov compressible fluid dynamics application and Beowulf clusters at Rutgers University, University of Arizona, and NERSC to develop our performance models, and define and evaluate our adaptation policies. In our prototype, the predictive performance models capture computational and communicational loads and, along with current system state, adjust processors capacities at runtime to enable the application to adapt and optimize its performance.

1 Introduction

The emergence of the computational Grid and the potential for seamless aggregation, integration and interactions has made it possible to conceive a new generation of realistic, scientific and engineering simulations of complex physical phenomena. These next-generation scientific and engineering simulations will be built on widely distributed computational Grids and will provide new and important insights into complex 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. These applications however, are inherently heterogeneous, dynamic and combine multiple physics, computational models and phases. Furthermore, the underlying Grid infrastructure is similarly heterogeneous and dynamic. As a result, configuring, managing and optimizing the execution of these applications to exploit the underlying computational power in spite of its heterogeneity and dynamism presents significant runtime management challenges.

The overall goal of the PRAGMA project is to realize a next-generation adaptive runtime infrastructure capable of support self managing, self adapting and self optimizing applications on the Grid - i.e., the runtime can sense current system and application state, anticipate system and application behavior and proactively and reactively adapt application execution to optimize its execution. This approach has been inspired by human autonomic nervous system that has the ability to self-configure, self-tune and even repair themselves without conscience human involvement.

In this paper we extend our prototype PRAGMA framework[1, 2] to enable self adapting, self optimizing runtime management of dynamically adaptive applications. Specifically, we present the design, prototype implementation and initial evaluation of policies and mechanism that enable PRAGMA to autonomically manage, adapt and optimize structured adaptive mesh refinement applications (SAMR) based on current system, application state and predictive models for system behavior and application per-

*The work presented in this paper is supported by the National Science Foundation NGS program via grant EIA-0103674

formance. We use the 3-D adaptive Richtmyer-Meshkov (RM3D) compressible fluid dynamics application and Beowulf clusters at Rutgers University, University of Arizona, and NERSC to develop performance models and formulate, implement and evaluate our autonomic adaptation and optimization policies. Specifically, we define and evaluate policies that will enable the applications to adapt to variability in computational and communication load and available memory. In our prototype, the predictive performance models capture computational load, communication loads and along with current system state, adjust processor capacities at runtime to enable the application to redistribute its computations in order to adapt and optimize its performance.

The rest of this paper is organized as follows. Section 2 outlines the characteristics of dynamically adaptive applications and specifically RM3D. Section 3 presents an overview of PRAGMA and describes its components. Section 4 defines and evaluates policies for system sensitive autonomic adaptations. Section 5 presents concluding remarks.

2 Enabling Realistic Simulations Using Structured Adaptive Mesh Refinement

The design of the PRAGMA adaptive runtime framework is driven by large-scale dynamically adaptive Grid simulations based on Structured Adaptive Mesh Refinement (SAMR) techniques. In this paper, we use the 3-D Richtmyer-Meshkov (RM3D[1]) instability encountered in compressible fluid dynamics. The RM instability occurs when a plane shock interacts with a corrugated interface between two fluids of different densities. As a result of such an interaction, interface perturbation starts to grow because the transmitted shock is converging at the wave peak and diverging at the valley. Converging shock increases pressure and accelerates perturbation peak into the second fluid. RM instabilities occur over a wide range of scales, from nearly microscopic objects, such as laser fusion pellets, to objects of astronomical size, such as supernovae.

A key challenge in such a simulation is that the physics exhibits multiple scales of length and time. If one were to employ zoning, which resolves the smallest scales, the required number of computational zones would be prohibitive. One solution is to use adaptive mesh refinement with multiple independent timesteps, which allows the grid resolution to adapt to a local estimate of the error in the solution. With AMR, the number of zones along with their location in the problem space is continuously changing. Besides dynamic communication and storage requirements, another challenge is that the local physics may change significantly from zone to zone as fronts move through the system.

Structured adaptive mesh refinement (SAMR) methods are based on uniform patch-based refinements overlaid on a structured coarse grid. In SAMR methods, dynamic adaptation is achieved by tracking regions in the domain that require higher resolution and dynamically overlaying finer grids on these regions. These techniques start with a coarse base grid with minimum acceptable resolution that covers the entire computational domain. As the solution progresses, regions in the domain with large solution error, requiring additional resolution, are identified and refined. Refinement proceeds recursively so that the refined regions requiring higher resolution are similarly tagged and even finer grids are overlaid on these regions. The resulting grid structure is a dynamic adaptive grid hierarchy.

Distributed implementations of these SAMR-based simulations lead to interesting challenges in dynamic resource allocation, data-distribution and load balancing, communications and coordination, and resource management. Furthermore, the complexity and heterogeneity of the Grid make 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 adaptivity makes application configuration and runtime management a significant challenge. In this paper, we address self-adaptation and optimization using dynamic system-sensitive partitioning and load-balancing.

3 PRAGMA: A Framework for Adaptive Proactive & Reactive Runtime Management of Grid Applications

The overall goal of the PRAGMA project is to realize a next-generation adaptive runtime infrastructure capable of support self managing, self adapting and self optimizing applications on the Grid - i.e., the runtime can sense current system and application state, anticipate system and application behavior and proactively and reactively adapt application execution to optimize its execution. This approach has been inspired by the human autonomic nervous system that has the ability to self-configure, self-tune and even repair themselves without conscience human involvement. PRAGMA addresses 3 key research challenges:

1. Formulation of predictive performance functions that hierarchically combine analytical, experimental and empirical performance models for individual elements of a heterogeneous, distributed computational environment, and use these functions along with current system/network state information to anticipate the opera-

[1] RM3D has been developed by Ravi Samtaney as part of the virtual test facility at the Caltech ASCI/ASAP Center (http://www.cacr.caltech.edu/ASAP).

tions and expected performance of applications for a given workload and system configuration.

2. Development of mechanisms for monitoring and characterizing the state of adaptive applications and abstracting their current computational, communication and storage requirements, and using this information to maximize application efficiency and performance.

3. Design, development and deployment of an active control network combining application sensors and actuators, policies, and application management agents capable of configuring application and execution environment at runtime, allocating and setting up required resources, monitoring application and system state, proactively and reactively adapting the application and execution environment to satisfy application requirements, maintaining application quality of service, improving performance and/or respond to system failures.

The runtime management framework is composed of three key components: a system characterization and abstraction component, a performance analysis module, and an active control network module, described as follows.

3.1 System Characterization and Abstraction

The objective of the system characterization/abstraction component is to monitor, abstract and characterize the current state of the underlying computational environment, and use this information to drive the predictive performance functions and models that can estimate its performance in the near future. Networked computational environments such as the computational "grid" are highly dynamic in nature. Thus, it is imperative that the application management system be able to react to this dynamism and make runtime decisions to satisfy application requirements and optimize performance. These decisions include selecting the appropriate number, type, and configuration of the computing elements, appropriate distribution and load-balancing schemes, the most efficient communication mechanism, as well as the right algorithms and parameters at the application level. Furthermore, proactive application management by predicting system behavior will enable a new generation of applications that can tolerate the dynamics of the grid and truly exploit its computational capabilities.

3.2 Performance Analysis Module

The performance analysis module is built on *Performance Functions*. Performance Functions (PF) describe the behavior of a system component, subsystem or compound system in terms of changes in one or more of its attributes. Using the PF concept, we can characterize the operations and performance of any resource in a distributed environment. Once the PFs of each resource used by an application are defined, we compose these PFs to generate an overall end-to-end PF that characterizes and quantify application performance.

Our PF-based modeling approach includes three steps. First, we identify the attributes that can accurately express and quantify the operation and performance of a resource (e.g., Clock speed, Error, Capacity). The second step is to use experimental and analytical techniques to obtain the PF that characterizes and quantifies the performance of each system component in terms of these attributes. The final step is to compose the component PFs to generate an overall PF that can be used during runtime to estimate and project the operation and performance of the application for any system and network state. This composition approach is based on the performance interpretation approach for parallel and distributed applications [9, 10].

3.3 Active Control Network

The underlying mechanisms for adaptive runtime management of SAMR applications are realized by an active control network of sensors, actuators, and management agents. This network overlays the application data-network and allows application components to be interrogated, configured, and deployed at runtime to ensure that application requirements are satisfied. Sensors and actuators are embedded within the application and/or system software and define interfaces and mechanisms for adaptation. This approach has been successfully used to embed and deploy sensors and actuators for interactive computational steering of large, distributed and adaptive applications [7].

3.4 Autonomic Runtime Management in PRAGMA

Autonomic management, adaptation and optimization in PRAGMA are based on three processes (see Figure 1): Monitoring, Analysis, and Adaptation. Monitoring is responsible for detecting conditions under which the parameters affecting application execution deviate from their acceptable behavior or operation. For example, the application performance may degrades severely due to increased computational and/or network load, low available memory, or due to software or hardware failures. Analysis encapsulates the adaptation policy and determines the appropriate application reconfiguration strategy, and the resources required to optimize the application performance. Once an appropriate adaptation/optimization strategy is identified, the Adaptation process initiated its execution. In what follows we present SAMR Grid applications.

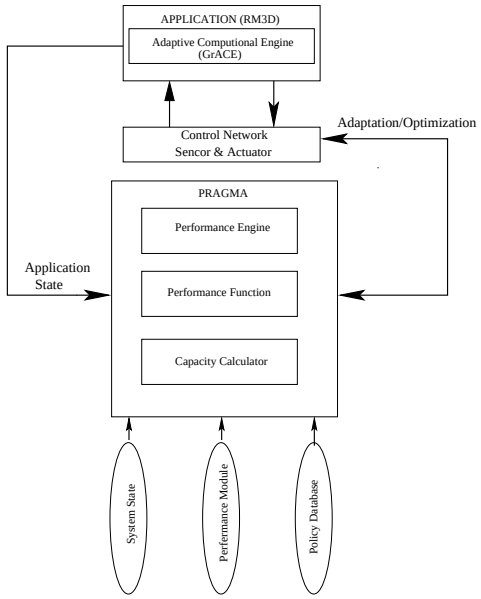

Figure 1. Autonomic Runtime Management in PRAGMA.

4 Policies for System Sensitive Adaptation and Optimization for SAMR Applications

Autonomic system sensitive adaptation in PRAGMA uses system state information including computational and communication load and available memory to select and tune distribution parameters and to dynamically partition and load balance the SAMR grid hierarchies. It builds on our earlier work on adaptive runtime management[1, 2] and system sensitive partitioning[11]. In what follows we first summarize the operation of the adaptive system-sensitive partitioner and then define and evaluate policies for enabling autonomic adaptation to variations in computational load, available memory and available bandwidth. The prototype PRAGMA implementation has been integrated into the GrACE (Grid Adaptive Computational Engine)[8] infrastructure's adaptive runtime system. GrACE is a data-management framework for parallel/distributed AMR. The policies are evaluated using the RM3D CFD kernel on Beowulf clusters at Rutgers University, University of Arizona, and NERSC.

4.1 Adaptive, System Sensitive Runtime Partitioning and Load Balancing

The adaptive system sensitive partitioner uses current system parameters, obtained using predictive performance functions and the resource-monitoring tools, to compute their relative computational capacities at each of the processors as follows [11]. Let us assume that there are K processors in the system among which the partitioner distributes the workload and C_K is the relative capacity of each processor such that,

$$\sum_{k=1}^{K} C_k = 1 \tag{1}$$

If the total work to be assigned to all the processors is denoted by L, then the work L_k assigned to the kth processor can be computed as $L_k = C_k L$. In order to enable the application self-optimize at runtime, we need to dynamically adjust the capacity of each processor based on its current computational load and the state of its computing and communication resources (e.g., available memory and communication bandwidth). In what follow, we describe in detail how to dynamically adjust the capacity C_k for each processor for the RM3D application. At runtime PRAGMA monitors application and system state on each processor. Application state includes the levels of refinement, the number, shape and aspect ration of the refined patches and the dynamism of the application[3, 4]. System state include computational load available memory and link bandwidth. The performance engine uses the application configuration and this state information to select the appropriate performance function and to predict the execution time of the application for the next time step on each processor. Let $t_k(t)$ be the execution time on processor k at time t, and $t_{avg}(t)$ be the average execution time for the next iteration on all the processors, i.e.,

$$t_{avg} = \sum_{k=1}^{K} t_k(t)/K \tag{2}$$

The performance function for predicting the execution time for the RM3D application on a processor k for a given application load X_1 and AMR level X_2 is empirically defined as follows:

$$\begin{aligned} t_k \;=\; & a_0 + a_1 X_1 + a_2 X_2 + a_3 X_1 X_2 + \\ & a_4 {X_1}^2 + a_5 {X_2}^2 + a_6 {X_1}^2 X_2 + \\ & a_7 X_1 {X_2}^2 + a_8 {X_1}^2 {X_2}^2 \end{aligned} \tag{3}$$

where

$$\begin{aligned} a_0 &= 0.004928425 \\ a_1 &= 1.1312091e-005 \\ a_2 &= -0.098176435 \\ a_3 &= 8.264314e-006 \\ a_4 &= 7.8054074e-013 \end{aligned}$$

$$
\begin{aligned}
a_5 &= 0.086556022 \\
a_6 &= 2.89973974e-012 \\
a_7 &= -0.3955486e-005 \\
and & \\
a_8 &= -1.71450838e-012
\end{aligned}
$$

The execution time should be adjusted based on the current load on the processor. Our experimental results show that the execution time of a program increased linearly with the current system load. As a result, the execution time for one iteration of the RM3D application as a function of the system load is given as:

$$t_k{}' = t_k x L_d \tag{4}$$

Where L_d is the current system workload.

To balance the load on each processors, the execution time for the next iteration should be the same on all processors. We can adjust the capacity of each processor such that their execution time during the next one or more iterations will be identical within an acceptable tolerance. The adjustment factor associated with each processor can be computed as,

$$F_i(t) = t_{avg}(t)/t_i(t) \tag{5}$$

Consequently, if the execution time on a given processor is less than the average time, this adjustment factor will be greater than 1 and more work will be assigned to this processor. However, if the execution time is greater than the average time, the adjustment factor will be less than 1 and the work assigned to this processor will be reduced. Once the adjustment factor is determined, we can compute the capacity of processor k for the next iteration as follows:

$$C_k(t) = C_k(t-1) x F_I(t) \tag{6}$$

To make sure that the sum of the capacities on all processors is equal to 1, the new capacities are adjusted as follows:

$$C_k{}'(t) = C_k(t) / \sum_{k=1}^{K} C_k(t) \tag{7}$$

The frequency of capacity adjustment depends on the rate at which system/application dynamics and SAMR adaptivity lead to an overall imbalance. The frequency of capacity adjustment typically depends on the application regridding frequency, and is usually done just just before regridding so that any subsequent regridding can use the new capacities during redistribution of load. In a similar approach, we can use the performance functions that predict the application execution time with respect to available communication delays as well as available memory, in conjunction with system monitoring tools, to adjust the relative capacity associated with each processor as outlined in[11].

4.2 Autonomic Adaptations to Computational Load

In this subsection, we experimentally demonstrate the application of the approach outlined above to autonomically adapt to system load dynamics. Current load is obtained using the on-line monitoring and performance prediction as discussed in Section 4.1. In this experiment we compare the performance of the application with and without the self-optimizing runtime. Table 1 presents the performance gains for different base grid sizes on 4 processors, while Table 2 presents the performance gains for 8 processors for a base grid size of $64 * 64 * 32$. We are currently benchmarking the performance gain on larger number of processors.

Table 1. Self-optimizing performance gain for different base grid sizes on 4 processor cluster.

Problem size	Execution time without self-optimization	Execution time with self-optimization	Percentage improvement
64*16*16	438.83	362.97	17.29%
64*32*16	780.95	660.67	15.40%
64*32*32	2326.1	1681.68	27.70%
64*64*32	4165.28	3535.3	15.12%

Table 2. Self-optimizing performance gain for base grid size of $64 * 64 * 32$ on 8 processor cluster.

Execution time without self-optimization	2109.43
Execution time with self-optimization	1651.33
Percentage improvement	21.72%

4.3 Autonomic Adaptations to Memory Availability

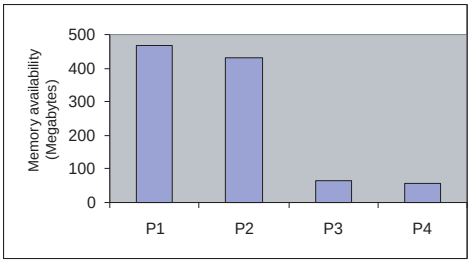

Figure 2. Memory availability of 4 processors.

Memory availability of each processor is captured using system monitoring tools and is used by PRAGMA to adapt and optimize GrACE partitioning parameters, i.e. the upper bound on the size of SAMR patches is tuned so that it conforms to the available memory. In the case of processors with higher memory availability, the PRAGMA attempts to maximize the size of SAMR patches at each level so as to increase cache locality and reduce overall communication. However, when the available memory is limited, PRAGMA constrains the size of the patches to conform to the available memory. As smaller patches can lead to increased communication overheads, PRAGMA tries to assign the smaller refined patches and assigns them to the processors with limited memory before breaking the larger patches. The capability of PRAGMA to autonomically adapt to variation in memory availability is evaluated using RM3D on 4 processors. A "rogue" program that random hogs memory is executed on each of the processors and simulates the variability in available memory. In our experiment, two processors (P1 and P2) are set to have higher memory availability and the other two processors (P3 and P4) have limited memory availability. Figure 2 presents the memory availability for the 4 processors. As the application executes, processors P3 and P4 are assigned smaller patches to match their limited available memory. P1 and P2 on the other hand, are assigned relatively large patches. Figure 3 and Figure 4 illustrate the adaptation of the maximum patch size (in terms of grid points) assigned to each processor based on the relative memory availability at the processor.

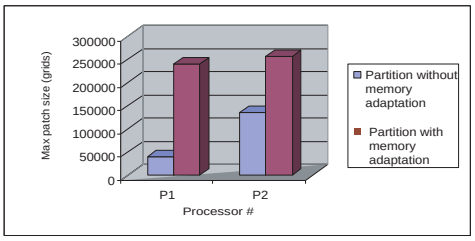

Figure 3. Max patch size (grid points) variance for P1, P2 with two partitioning scheme.

5 Conclusions

The emergence of the computational Grid and the potential for seamless aggregation, integration and interactions has made it possible to conceive a new generation of realistic, scientific and engineering simulations of complex physical phenomena. The inherently heterogeneous and dynamic nature of these application and the Grid presents significant runtime management challenges. In this paper

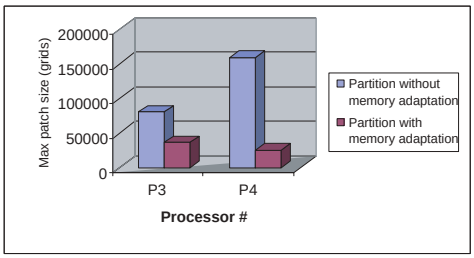

Figure 4. Max patch size (grid points) variance for P3, P4 with two partitioning scheme.

we extended the PRAGMA framework to enable self adapting, self optimizing runtime management of dynamically adaptive applications. Specifically, we presented the design, prototype implementation and initial evaluation of policies and mechanisms that enable PRAGMA to autonomically manage, adapt and optimize structured adaptive mesh refinement applications (SAMR) based on current system and application state and predictive models for system behavior and application performance. We used the 3-D adaptive Richtmyer-Meshkov compressible fluid dynamics application and Beowulf clusters at Rutgers University, University of Arizona, and NERSC to develop our performance models, and define and evaluate our adaptation policies. In our prototype, the predictive performance models capture computational and communicational loads and, along with current system state, and adjust processor capacities at runtime to enable the application to adapt and optimize its performance. We are currently extending our policies to simultaneous adapt to variations load, available memory and communication bandwidth. We are also experimenting with different (and larger) system configurations and working with other applications.

References

[1] M. Parashar, and S. Hariri. PRAGMA: An Infrastructure for Runtime Management of Grid Applications. Proceedings of the *NSF Next Generation Systems Program Workshop, IEEE/ACM International Parallel and Distributed Processing Symposium*, Fort Lauderdale, FL, CDROM, IEEE Computer Society Press, 8 pages April 2002.

[2] S. Chandra, S. Sinha, M. Parashar, Y. Zhang, J. Yang, and S. Hariri. Adaptive Runtime Management of SAMR Applications. Proceedings of the *9th International Conference on High Performance Computing (HiPC 2002), Lecture Notes in Computer Science* , Editors: S. Sahni, V.K. Prasanna, U. Shukla, Springer-

Verlag, Bangalore, India, Vol. 2552, pp 564 - 574, December 2002.

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

[4] 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* (also best research poster at *Supercomputing Conference 2001*), October 2001.

[5] S. Hariri, P. K. Varshney, L. Zhou, H. Xu and S. Ghaya", A Hierarchical Analysis Approach for High Performance Computing and Communication Applications. Proceedings of the *32nd Hawaii International Conference on System Sciences*, Maui, HW, January 1999.

[6] S. Hariri, H. Xu, and A. Balamash. A Multilevel Modeling and Analysis of Network-Centric Systems. Special Issue of *Microprocessors and Microsystems Journal*, Elsevier Science on Engineering Complex Computer Systems, 1999.

[7] S. Kaur, V. Mann, V. Matossian, R. Muralidhar, and M. Parashar. Engineering a Distributed Computational Collaboratory. Proceedings of the *34th Hawaii International Conference on System Sciences*, Maui, HW, January 2001.

[8] M. Parashar and J. Browne. On Partitioning Dynamic Adaptive Grid Hierarchies. Proceedings of the *29th Hawaii International Conference on System Sciences*, Maui, HW, January 1996.

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

[10] M. Parashar and S. Hariri. Compile-Time Performance Interpretation of HPF/Fortran 90D. *IEEE Parallel and Distributed Technology*, Spring 1996.

[11] S. Sinha and M. Parashar. Adaptive Runtime Partitioning of AMR Applications on Heterogeneous Clusters. Proceedings of the *3rd IEEE International Conference on Cluster Computing*