# PHYSICS AWARE PROGRAMMING PARADIGM AND RUNTIME SYSTEM

By

Yeliang Zhang

---

A Dissertation Submitted to the Faculty of the

DEPARTMENT OF ELECTRICAL AND COMPUTER ENGINEERING

In Partial Fulfillment of the Requirements
For the Degree of

DOCTOR OF PHILOSOPHY

In the Graduate College

THE UNIVERSITY OF ARIOZNA

2007

THE UNIVERSITY OF ARIZONA
GRADUATE COLLEGE

As members of the Dissertation Committee, we certify that we have read the dissertation prepared by **Yeliang Zhang** entitled **Physics Aware Programming Paradigm and Runtime System** and recommend that it be accepted as fulfilling the dissertation requirement for The Degree of Doctor of Philosophy.

_____ Date: July 27, 2007
Salim Hariri, Ph.D.

_____ Date: July 27, 2007
Marwan Krunz, Ph.D.

_____ Date: July 27, 2007
Jerzy W. Rozenblit, Ph.D.

Final approval and acceptance of this dissertation is contingent upon the candidate's submission of the final copies of the dissertation to the Graduate College.

I hereby certify that I have read this dissertation prepared under my direction and recommend that it be accepted as fulfilling the dissertation requirement.

_____ Date: July 27, 2007
Salim Hariri, Ph.D.

## STATEMENT BY AUTHOR

This dissertation has been submitted in partial fulfillment of requirements for an advanced degree at the University of Arizona and is deposited in the University Library to be made available to borrowers under rules of the Library.

Brief quotations from this dissertation are allowable without special permission, provided that accurate acknowledgment of source is made. Requests for permission for extended quotation from or reproduction of this manuscript in whole or in part may be granted by the head of the major department or the Dean of the Graduate College when in his or her judgment the proposed use of the material is in the interests of scholarship. In all other instances, however, permission must be obtained from the author.

SIGNED: Yeliang Zhang

TABLE OF CONTENTS

TABLE OF CONTENTS – *Continued*

TABLE OF CONTENTS – *Continued*

# LIST OF FIGURES

LIST OF FIGURES - *Continued*

# LIST OF TABLES

# ABSTRACT

The overarching goal of this dissertation research is to realize a virtual collaboratory for the investigation of large-scale scientific computing applications which generally experience different execution phases at runtime and each phase has different computational, communication and storage requirements as well as different physical characteristics. Consequently, an optimal solution or numerical scheme for one execution phase might not be appropriate for the next phase of the application execution. Choosing the ideal numerical algorithms and solutions for all application runtime phases remains an active research area. In this dissertation, we present Physics Aware Programming (PAP) paradigm that enables programmers to identify the appropriate solution methods to exploit the heterogeneity and the dynamism of the application execution states. We implement a Physics Aware Runtime Manager (PARM) to exploit the PAP paradigm. PARM periodically monitors and analyzes the runtime characteristics of the application to identify its current execution phase (state). For each change in the application execution phase, PARM will adaptively exploit the spatial and temporal attributes of the application in the current state to identify the ideal numerical algorithms/solvers that optimize its performance. We have evaluated our approach using a real world application (Variable Saturated Aquifer Flow and Transport (VSAFT2D)) commonly used in subsurface modeling, diffusion problem

kernel and seismic problem kernel. We evaluated the performance gain of the PAP paradigm with up to 2,000,000 nodes in the computation domain implemented on 32 processors. Our experimental results show that by exploiting the application physics characteristics at runtime and applying the appropriate numerical scheme with adapted spatial and temporal attributes, a significant speedup can be achieved (around 80%) and the overhead injected by PAP is negligible (less than 2%). We also show that the results using PAP is as accurate as the numerical solutions that use fine grid resolution.

# Chapter 1 INTRODUCTION

## 1.1 Introduction

The development of computing and information technology has improved dramatically over the past decade and provides scientists and engineers an easier and more powerful environment to solve real world problems, especially in a Grid environment. However, large-scale distributed scientific applications are inherently large, complex, highly adaptive and heterogeneous. The computational complexity associated with each computational region or domain varies continuously and dramatically both in space and time throughout the whole life cycle of the application execution. To exacerbate the problem, such changes are only available at runtime and cannot be predicted. In addition, the underlying distributed computing environment is similarly complex and dynamic in the availabilities and capacities of the computing resources. This raises an important question that given the dynamism and the complexity of scientific applications and underlying computer architectures, can we propose an efficient programming paradigm to exploit heterogeneity of the application execution states as well as the physical properties associated with each execution phase at runtime and use such information for the benefit of the execution optimization. Apparently, the current paradigms which are based on passive components and static compositions are not suitable for such emerging heterogeneous and dynamic Grid environments. In this dissertation, we have developed an autonomic computing

paradigm named Physics Aware Programming (PAP) Paradigm to address the complexity and dynamism of large scale scientific applications and achieve the self-management and self-optimization of large-scale scientific and engineering applications.

## 1.2 Importance of the Research Problem

Scientific computing applications models real world phenomena which are inherently large, complex, multi-phased, multi-scale, dynamic and heterogeneous (in time, space and state). Furthermore, the platforms these applications are executing on are similarly large, complex, heterogeneous and dynamic. The combination of the two results in application development, configuration and management complexities that break current paradigms based on passive components and static compositions. For example, the core collapse problem in a 3D supernovae simulation with reasonable resolution ($500^3$) would require ~$10^{20}$ floating-point operations or ~10-20 teraflops for 1.5 months running on ~$10^8$ CPU hours with 100 millions CPUs! The physics being modeled in supernovae is multi-phased and heterogeneous. It combines fluid-dynamics, nuclear fusion, heat transferring and transport phases and each simulation phase requires different computational models and different computational resources. The transition from one phase to the next and the computational models applicable at each phase and their computational requirements are determined by complex criteria based on local states that are known only at runtime and make it difficult to optimize such

multi-phase and dynamic simulations. Another example is a wildfire simulation. Many fires that cause significant loss of life (such as the 1994 Storm King and 1991 Oakland Hills Fire) and loss of property (such as the 2003 Southern California fires) occur in conditions with rugged terrain, heterogeneous vegetation fuel beds and time varying atmospheric conditions. To simulate wildfire in these dynamically interacting factors, the application will experience different phases on fluid dynamics, heat transferring on various conductivity materials and boundary conditions. The conventional static methods do not account for the many important, nonlinear, interacting physical processes that determine the behavior of wildfires in complex environments due to the exponential growth in computational requirements.

In summary, the existing optimizing techniques to these large-scale problems focus on static solutions and off-line component compositions which are not capable of handling the complexity and dynamism of the applications.

## 1.3 Research Challenges

The research described in this dissertation developed the expertise, technology, and infrastructure to optimize very large multi-dimensional multi-phase numerical simulations. Realizing these realistic simulations by best utilizing the computing power in a most efficient way without jeopardizing the accuracy of the simulation presents grand research challenges. These challenges are due to

(1) Unprecedented scales in domain size and resolution of the applications.

(2) Unprecedented heterogeneity and dynamics in time, space and state of applications and systems.

(3) Unprecedented complexity in physical models, numerical formulations and software implementations.

(4) Unprecedented programmability, manageability and usability requirements in allocating, programming and managing very large numbers of resources.

In fact, we have reached a level of complexity, heterogeneity and dynamism that our programming environments and infrastructure are becoming brittle, unmanageable and insecure. There is a need for a fundamental change in how these applications are formulated, composed, managed and optimized.

## 1.4 Research Contribution

The overall contribution of this dissertation is that we have developed a novel Physics Aware Programming (PAP) Paradigm that takes current application's physical properties into consideration and helps the programmers exploit the heterogeneity and dynamism of the scientific simulations as well as the physical properties associated with each execution phase. Our method is a general programming methodology that can be applied to many scientific and engineering applications such as hydrodynamics, aeromechanics, astronomy, computational fluid dynamics and wildfire simulation. A

runtime manager is also developed to implement such programming paradigm to achieve (a) online monitoring, and analyzing application physics properties; (b) learning and automatically identifying critical property changes which will affect the execution performance; and (c) self-optimizing the spatial and temporal characteristics of the application and the related numerical algorithms. To achieve these goals, our approach is highlighted as follows:

The first research issue is developing a Physics Aware Programming (PAP) Paradigm which provides a methodology for domain scientists to take advantage of the problem's internal properties which involves 1) developing the theoretical framework to analyze the spatial and temporal characteristics of applications; and 2) analyzing the effect of time and space changes on the application performance.

The second research issue is developing a Physics Aware Runtime Manager (PARM) to implement PAP. PARM determines the application execution phase by monitoring the application execution, identifies the application phase changes by exploiting the knowledge about the application physics and how it behaves at runtime and then uses the appropriate numerical algorithms for each application phase. For each execution phase of a scientific application, different numerical schemes and solvers that can best exploit its physics characteristics and its state were stored in a Knowledge Base that will be used by PARM at runtime. For example, in a wildfire simulation [113], we use PAP to decompose the computational domain into several regions (e.g., burning,

unburned and burned) at runtime. The number of burning, unburned and burned cells determines the current state of the fire simulation and can then be used to accurately predict the computational power required for each region. We can achieve significant performance gains if the physics properties of the application and its current state are exploited (e.g., we can achieve 43% speedup on a 65536 cells wildfire simulation using PAP) [113]. In a similar methodology, PAP can be applied to many other applications such as supernovae simulation, nuclear fusion and hydrodynamics, just to name a few.

The third research issue is to map the application execution to a heterogeneous environment such that its performance is optimized. Our runtime system will maintain performance and utilization by monitoring the available resources and responding to the application's computational requirements by appropriately and dynamically scheduling components from the set of available resources. The key components of this research include:

- Formulation of predictive performance functions that hierarchically combine analytical and empirical performance models for individual elements, and use these functions along with current system/network state information to estimate the performance of applications for a given workload and system configuration.

- Development of mechanisms for monitoring and characterizing the state of adaptive applications, and abstracting the current computational, communication and storage requirements.

The main contributions of this dissertation are:

- Advanced programming and execution models, tools and environments that are based on autonomous, self configuring, adapting and optimizing components and are capable of effectively managing and exploiting the heterogeneity and dynamics of the applications. A key contribution is the development of a Physics Aware Programming (PAP) paradigm that will enable the application developer to fully exploit the heterogeneity and dynamism of the physics properties of the application and thus improve performance.
- Implementation of Physics Aware Runtime Manager (PARM) which is capable of reactively and proactively managing and optimizing application execution using current system and application state, predictive models for system behavior and application performance.
- Applying PAP to various numerical applications such as hydrology, astronomy and earth physics. Our results show that PAP paradigm produces superior execution results than conventional solutions. For example, a 81% speedup is observed in a 2 million node simulation of a

hydrology application using PAP methodology when compared with conventional methods.

- Developed proactive runtime partitioning strategies based on performance prediction functions that are formulated in terms of system states such as CPU load and available bandwidth.

## 1.5 Dissertation Organization

The rest of this dissertation is organized as follows.

In Chapter 2, we introduce related work on autonomic system designs and focus on self-optimization. We categorize the self-optimization techniques into three major categories based on the knowledge exploited by the optimization techniques: architecture aware, error aware and application aware. We introduce the major research activities associated with each type of optimization techniques and highlight the difference between them.

In chapter 3, we discuss the related work for distributed load balancing techniques. Based on the load balancing information exchange type, we classify the load balancing techniques into synchronous and asynchronous ones. Specifically, we also introduce the load balancing techniques that are applied to Structure Adaptive Mesh Refinement (SAMR) applications. Firstly, we will introduce SAMR applications briefly and the problems imposed on load balancing from SAMR simulations. Secondly, we discuss

the main SAMR load balancing techniques and their implementations.

In Chapter 4, we present our PAP paradigm and introduce the theoretical background of this methodology and some examples on how this methodology will benefit the developers of scientific applications without sacrificing the accuracy and convergence of the application. We also discuss the Physics Aware Runtime Manager (PARM), a runtime manager we developed to implement the PAP paradigm. We explain the functionalities of the main components of PARM including Knowledge Base, Policy Engine, Configuration Engine and Monitoring Engine and how they are interacting with each other.

In Chapter 5, we discuss our experimental results of applying the PAP paradigm to a real world application VSAFT2D developed by The Hydrology Department at The University of Arizona. We also analyze the PAP overhead, the error rate and the speedup compared to the original implementation method. Our result shows superior performance gain can be achieved using our approach. For example, in a 2 million node simulation, PAP will have 80% speedup, less than 2% error rate and less than 2% overhead, respectively, compared to the original implementation on 32 processors.

In Chapter 6 we present a case study of an adaptive runtime infrastructure reactively manages the application execution using current system and application state and predictive models for system behavior and application performance. Our runtime

partitioning strategies based on performance prediction functions that are experimentally formulated in terms of system parameters such as CPU load and available networking bandwidth. The experiment of this case study shows that our adaptive runtime infrastructure will make SAMR application well balanced and outperform the original methods without considering system and application states. For example, for a $128 \times 32 \times 32$ problem executed on 32 processors, our methodology will outperform the conventional methods with 40.9%, 31.35% and 20.45% respectively on lightly loaded, moderately loaded and heavily loaded CPU scenarios.

Chapter 7 concludes the works in this dissertation and gives future research directions.

# CHAPTER 2 RELATED WORK ON DISTRIBUTED APPLICATION OPTIMIZATION

## 2.1 Introduction

Runtime optimization of parallel and distributed applications has been an active research area and has been approached using different techniques. These techniques can be classified based on the knowledge exploited by the optimization techniques into three major categories: Error Aware, Application Aware, and Architecture Aware (see Figure 2.1).

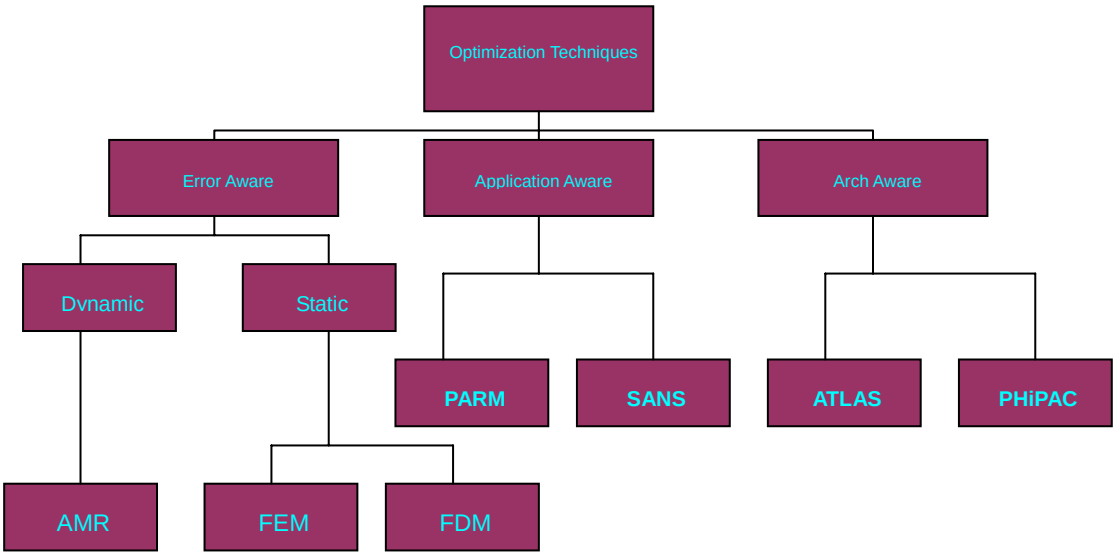


**Figure 2.1 a Hierarchical View of Different Optimization Techniques Adopted in Science**

a. *Error Aware Solution.* These kinds of optimization techniques use the error rate of the application's execution as the optimization metrics. The optimization goal is to

reduce the error rate to a minimal level. Though the optimized codes will deliver an accurate result, the execution time might be inefficiently lengthened due to unnecessary error correction.

*b. Architecture Aware Solution.* These kinds of optimization techniques utilize the architecture information of the underlying computing systems in order to optimize the application performance.

*c. Application Aware Solution.* These kinds of optimization techniques use application information to drive their optimization.

## 2.2 Error Aware Optimization

The representatives of this approach include Adaptive Mesh Refinement (AMR) [8] and Finite Difference Method (FDM) [9]. They can also be classified into dynamic and static methods based on how the error rate is used. AMR is a dynamic method and FDM is a static method.

AMR focuses on partial differential equation (PDE) solving [114]. The solution of PDE equations are often accurate and smooth for large portions of the computation domain but some locally isolated internal regions with steep gradients, shocks, or discontinuities. In AMR application, the grid hierarchy is represented as a tree of grids

at any instant of time. The number of levels, the number of grids and the locations of the grids change at each mesh adaptation. Initially, a uniform coarse mesh covers the entire computational domain. At runtime, local regions that have larger error rate will need higher resolution, and then finer subgrid will be added on these local regions. However, if the error rate decreases, the local regions will combine finer subgrid into coarser grids. This process repeats recursively with each adaptation until the final solution is obtained. Each computation iteration starts with a time advance at coarsest level. For those regions that require higher resolutions, finer grids are constructed and then are time advanced with a smaller timestep.

Different from AMR technique that allows adaptive grid refinement at runtime based on the error rate in certain computation region, FDM statically shrinks the grid size at all the computation domains if the error rate exceeds a predefined threshold.

The main idea of FDM approach is to replace the continuous problem domain by a finite difference mesh or grid and the numerical equations derived from FDM is called Finite Difference Equation (FDE). Suppose we need to solve a PDE with dependent variable ***u(x,y)*** in a square domain $0 \le x \le 1, 0 \le y \le 1$. Grids are established on the domain by replacing ***u(x,y)*** with ***u(i$\Delta$x,j$\Delta$y)***. An example of grids is given in Figure 2.2.

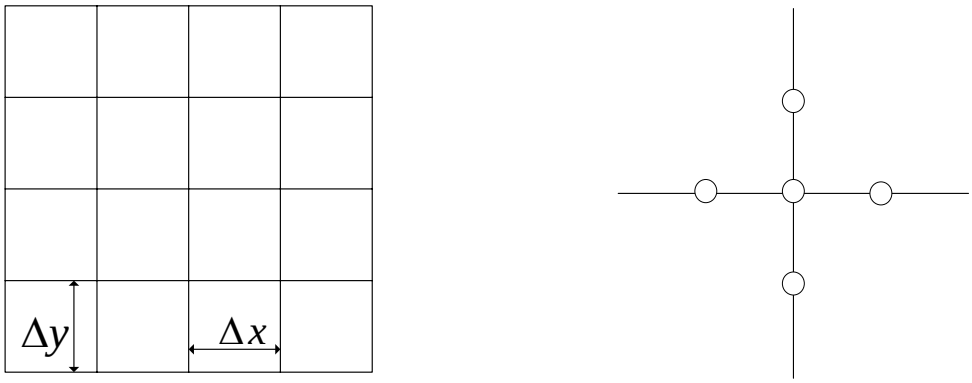


**Figure 2.2 a Typical Finite Difference Grid**

If we treat $u_{i,j}$ as $u(x_0,y_0)$, then we will have:

$$u_{i+1,j} = u(x_0 + \Delta x, y_0) \qquad u_{i-1,j} = u(x_0 - \Delta x, y_0)$$

$$u_{i,j+1} = u(x_0, y_0 + \Delta y) \qquad u_{i,j-1} = u(x_0, y_0 - \Delta y)$$

For convenience, we often write the $i$ as a superscript. Then $u_{i,j}$ will be indicated as $u_j^i$. The idea of a finite difference representation for a derivative can be derived from the definition of derivative for any function $u(x,y)$ at $x = x_0$ and $y = y_0$ as Equation 2.1 shows.

$$\frac{\partial u}{\partial x} = \lim_{\Delta x \to 0} \frac{u(x_0 + \Delta x, y_0) - u(x_0, y_0)}{\Delta x} = \lim_{\Delta x \to 0} \frac{u_{i+1,j} - u_{i,j}}{\Delta x} \tag{2.1}$$

Here, if $u$ is continuous at point $(x_0,y_0)$, then the limit $[u(x_0+\Delta x,y_0) - u(x_0,y_0)] / \Delta x$ will exist and it is an approximation to $\frac{\partial u}{\partial x}$ for a small $\Delta x$.

The derivative can be explained with Taylor expansion [33] in a formal way. For $u(x_0 + \Delta x, y_0)$, if we expand $u$ with regarding to $(x_0,y_0)$, the Taylor expansion gives

$$u(x_0+\Delta x, y_0) = u(x_0, y_0) + u_x(x_0)\Delta x + u_x{}^2(x_0)\frac{\Delta x^2}{2!} + \ldots + u_x{}^{n-1}(x_0)\frac{\Delta x^{n-1}}{(n-1)!} + u_x{}^n(\xi)\frac{\Delta x^n}{n!}$$

$$\text{where} \quad x_0 \le \xi \le (x_0+\Delta x) \qquad (2.2)$$

The last term can be identified as the remainder. Thus from Equation (2.2), it's easy to derive that

$$\left.\frac{\partial u}{\partial x}\right|_{x_0,y_0} = \frac{u(x_0+\Delta x, y_0) - u(x_0, y_0)}{\Delta x} + T.E. = \frac{u_{i+1,j} - u_{i,j}}{\Delta x} + T.E. \qquad (2.3)$$

Equation (2.3) is essentially the same as Equation (2.2) except the Truncation Error (T.E.) term. Numerically speaking, T.E. is the difference between the partial derivative and its finite difference representation. Since Equation (2.3) uses the information at grid point ***i*** to calculate the value at grid point ***i+1***, Equation (2.3) is called "forward" difference method. If we change the sign in Equation (2.2),

$$u(x_0-\Delta x, y_0) = u(x_0, y_0) - u_x(x_0)\Delta x + u_x{}^2(x_0)\frac{\Delta x^2}{2!} - u_x{}^3\frac{\Delta x^3}{6} + \ldots \qquad (2.4)$$

Rearrange (2.4), we obtain

$$\left.\frac{\partial u}{\partial x}\right|_{x_0,y_0} = \frac{u(x_0, y_0) - u(x_0-\Delta x, y_0)}{\Delta x} + T.E. = \frac{u_{i,j} - u_{i-1,j}}{\Delta x} + T.E. \qquad (2.5)$$

This method is called "backward" because we are using information on grid point ***i-1*** to calculate values on grid point ***i***. We can subtract Equation (2.4) from Equation (2.2) and rearrange to obtain an approximation on "central" difference

$$\left.\frac{\partial u}{\partial x}\right|_{x_0,y_0} = \frac{u_{i+1,j} - u_{i-1,j}}{2\Delta x} + T.E. \qquad (2.6)$$

We can also add (2.4) and (2.2) to obtain an approximation to the second derivative:

$$\left.\frac{\partial^2 u}{\partial x^2}\right|_{x_0, y_0} = \frac{u_{i+1,j} - 2u_{i,j} + u_{i-1,j}}{\Delta x^2} + T.E. \tag{2.7}$$

Generally, in FDM, the spatial spacing on ***x, y, z*** coordinate will be represented as ***Δx, Δy, Δz*** while some other reference books use ***h***. And the temporal spacing will be denoted as ***Δt*** or ***k***.

The solution of the FDE will approximate the solution of the PDE at the grid points. If the discretization is a good approximation, the solution of the FDE should satisfy the analytical solution of the PDE when the grid spacing ***h*** and ***k*** are taken sufficiently small. The difference between the solution of PDE and FDE is the T.E. as shown in Equation (2.3). If ***h*** and ***k*** are sufficiently small, the T.E. vanishes, the FDE is said to be consistent with the PDE. Besides consistency, numerical stability is another important issue for FDM. A stable numerical scheme is a scheme where round-off error and truncation error are not growing in an expanding way during the computation. We use an example to introduce an important stability analysis theorem: von Neumann Stability Analysis [33]. Considering a forward time centered equation we obtained by discretizing function $\frac{\partial u}{\partial t} = -v\frac{\partial u}{\partial x}$.

$$\frac{u_j^{n+1} - u_j^n}{\Delta t} = -v\left(\frac{u_{j+1}^n - u_{j-1}^n}{2\Delta x}\right) \tag{2.8}$$

Imagine that the coefficients of the difference equations are slowly varying that we can treat them as constant. Then the general solutions of Equation (2.8) are in the

form of

$$u_n^j = \xi^n e^{ikj\Delta x} \tag{2.9}$$

where $\boldsymbol{k}$ is a real spatial wave number and $\xi = \xi(k)$ is a complex number that depends on $\boldsymbol{k}$ [32, 33]. The difference equation is not stable if $|\xi(k)| > 1$ for some $\boldsymbol{k}$ because the solution has exponential growing modes. Therefore the number $\boldsymbol{\xi}$ is called the amplification factor at a given wave number $\boldsymbol{k}$. Using flux conservative equation with constant velocity of propagation [33], we can write:

$$\xi = \cos k\Delta x - i\frac{v\Delta t}{\Delta x}\sin k\Delta x \tag{2.10}$$

The stability condition $|\xi|^2 \le 1$ leads to the requirement

$$\frac{|v|\Delta t}{\Delta x} \le 1 \tag{2.11}$$

Equation (2.11) is called Courant-Friedrichs-Lewy stability criterion, or Courant condition [33]. When solving physics problems, FDM first needs to satisfy this stability criterion to provide an accurate solution. If in the middle of execution, a large error occurs, FDM program has to halt and restarted with a different $\boldsymbol{\Delta t}$ or $\boldsymbol{\Delta x}$ on the entire computation domain.

To sum up, AMR allows adaptive grid refinement at runtime based on the error rate in certain computation region while FDM statically shrinks the grid size at all the computation domains if error occurs. All these two techniques refine a computational mesh based on the acceptable error rate. But AMR and FDM cannot guarantee a grid

size leading to a small error rate until the application has been executed and the error rate is calculated. A common programming paradigm for this solution approach is keeping on halving the grid size in each time step if the error is larger than a threshold until a converged result is calculated.

## 2.3 Architecture Aware Optimization

Some optimization techniques exploit the targeted architectures where the scientific applications suppose to be executed to achieve better application performance. The representatives of these approaches include PHiPAC [1], and ATLAS [2]. PHiPAC is a portable and high-performance ANSI C coded scheme in which the authors provide search scripts to find the best parameters for a given system executing the code. PHiPAC provides a static optimization technique to achieve machine-specific near-peak performance. The PHiPAC methodology contains three components: (1) A generic model of C compilers and microprocessors that provide guidelines for producing portable optimized ANSI C code; (2) parameterized generators that produce code according to the guidelines; and (3) scripts to automatically optimize code for a particular architecture by varying the generators' parameters and benchmarking the resulting routines.

The PHiPAC coding guidelines are mainly suggestions derived from compiler design and hand tuning experience. For example, the guidelines suggest that the

programmer use local variables to explicitly remove false dependencies between memory accesses to variables, take advantage of multiple integer and floating-point registers to accelerate memory access, minimize pointer updates by striding with constant offsets to increase cache hit ratio, hide multiple instruction Floating Point Unit (FPU) latency with independent operations, increase locality to improve cache performance, minimize branches to sustain cache coherence and loop unrolling to expose optimization opportunities.

The programmer can use these guidelines without using PHiPAC to improve performance but it might lead to a less maintainable code. Using parameterized code generators of PHiPAC can help in addressing this problem. The code generators will produce a cache-blocked matrix multiply code with reduced cache misses and unnecessary loads and stores. With the control of command line parameters, the code generators can produce blocking code for any number of levels of memory hierarchy, including register, L1 cache, TLB, L2 cache and so on.

Command line parameters are derived from search scripts in which parameters of targeted machine architecture such as the number of integer and floating-point registers and the size of each level of cache are inputted. For each combination of generator parameters and compilation options, the search scripts calls the generator, compiles the resulting routine, links it with the timing subroutine and benchmarking the resulting

executable code. The search scripts will first find the best register (or L0) parameters, and then find the best L1 parameters such as blocking parameters.

In [1], the authors use PHiPAC to produce a portable, Basic Linear Algebra Subprograms (BLAS) [5] matrix multiply generator and the resulting code can achieve over 90% of peak performance on a variety of current workstations, and sometimes even faster than the vendor optimized numerical libraries.

PHiPAC relies on architecture information to generate optimized code and uses search script to find out the best system related parameters for the code. This makes the generated code less maintainable and system dependent. In addition, the programming guideline focuses too much on compiler optimization and most of them are difficult for the programmer to follow to write clean and easy-to-understand applications. Actually most of the programming guideline can be achieved by modern compilers. The search script is very time consuming since the combination of system parameters is enormous and if the application is migrated onto a new platform, previous search has to be discarded and a new round of search is needed.

Automatically Tuned Linear Algebra Software (ATLAS) [2] provides a code generator coupled with a timer routine which takes parameters such as cache size, length of floating point and fetch pipelines etc., and then evaluates different strategies

for loop unrolling and latency based on environment states. The combination of these parameters which deliver the best performance will be selected for future application execution.

The first step of the timing tries to figure out the size of L1 cache. This is achieved by performing a fixed number of memory references, while successively reducing the amount memory addresses. The most significant difference between timings for successive memory sizes is the L1 cache boundary.

Next, ATLAS probes the system to obtain the floating point units' information. First, ATLAS needs to find out whether the architecture possesses a mul-add unit, or if independent multiply and add pipes are required. To obtain this information, ATLAS generates simple register-to-register codes which performs the required multiply-add using a combined mul-add and separate multiply and add pipes. Both codes are tried with various pipeline lengths. ATLAS then replicates the best of these codes in such a way that increasing numbers of independent registers are required until performance degrades sufficiently to demonstrate that the available floating point units have been exceeded. Since ATLAS has obtained the L1 cache size in the first step, ATLAS is then able to choose the optimal loop unrolling and the relevant range of blocking factors to examine. Once an optimal unrolling has been found, ATLAS again tries all blocking factors and picks the one with minimal latency.

In ATLAS, L1 and L2 level caches play an important role to determine loop dimension. But application performance also depends on available compiler in the environment. ATLAS optimizes the application in a fine-grained manner by fitting a loop block into the cache. Some systems such as Cray T3E or SGI R8000 without an adequate compiler or no L1 cache accessible by the floating point unit, the application performance will be deteriorated. Since ATLAS uses code generation to perform many optimizations typically done by compilers, an aggressive compiler can transform already optimal code into suboptimal code. Also, ATLAS assumes a hierarchical memory and an adequate C compiler are present. Lack of hierarchical memory would turn some of ATLAS's blocking and register usage into overheads. For most of scientific computing legacy codes, they were written in Fortran 77 which will impose difficulties using ATLAS to tune these legacy codes.

### 2.4 Application Aware Optimization

As its name indicates, application aware optimization uses application information to optimize the performance of distributed applications. The representatives of this approach include Self-Adapting Numerical Software (SANS) [24] and our approach, PAP. We will discuss PAP in detail in Chapter 4. Self-Adapting Numerical Software is an effort to let domain scientists manage the complex computational environment while at the same time, take advantage of the flexible composition of the available algorithmic alternatives. SANS tries to reduce the gap between the software and

hardware by using the following methods:

- code optimization based on hardware parameters
- picking optimal algorithms based on statistical analysis of the user problem

SANS uses ATLAS and PHiPAC to achieve code optimization based on hardware parameters and the details of ATLAS and PHiPAC have been discussed above. The optimal algorithm selection is a difficult task because for any of the modern scientific applications, the input space is of a very high dimension. Consequently, it is not feasible and possible to search this large input space exhaustively to find the best algorithm. Two issues are important for optimal algorithm selection: First, there might be various algorithms which all can solve the problem; second, these algorithms often have one or more parameters. Taking these two issues into consideration, in [24], the authors propose a strategy to determine the appropriate numerical algorithm by applying statistical techniques.

- Solve a large collection of training test problems using available methods.
- Each problem is assigned to a class corresponding to the method that gives the fastest solution.
- Retrieve a list of characteristics for each problem.
- Compute a probability density function for each class.

As a result of these four steps, the density function $p_i(\bar{x})$ where ***i*** ranges over all the classes is found. $\bar{x}$ denotes the vectors of features of the input problems. Given a new problem with its vector $\bar{x}$, we could find the optimal algorithm by finding a method ***i*** which will maximize $p_i(\bar{x})$. The basic problem features adopted by SANS are structural features pertaining to the matrix nonzero structure, spectral properties that have to be approximated and measures of the variance of the matrix. SANS uses a multivariate Bayesian decision rule in numerical decision making. Each method class has a multivariate density function:

$$p_j(\bar{x}) = \frac{1}{2\pi|\sum|^{1/2}} e^{-(1/2)(\bar{x}-\bar{\mu})\sum^{-1}(\bar{x}-\bar{\mu})} \tag{2.12}$$

where ***j*** denotes a class, $\bar{x}$ are the features, $\bar{u}$ is the means and $\sum$ is the covariance matrix. Having computed these density functions, the authors compute *a posteriori* probability of a class (i.e., given a problem with features $\bar{x}$, what is the probability that it belongs to class ***j***?) as

$$P(w_i \mid \bar{x}) = \frac{p(\bar{x} \mid w_j)P(w_j)}{p(\bar{x})} \tag{2.13}$$

Then the numerical method with the maximum quantity of Equation (2.13) will be used as the optimal algorithm for the application.

SANS depends on ATLAS and PHiPAC to generate optimized code. Since ATLAS and PHiPAC are written in C and platform related, SANS has the same problems as ATLAS and PHiPAC such as compiler dependent, long time architecture

parameters search and difficult to be used on legacy codes written in Fortran.

# CHAPTER 3 RELATED WORK ON LOAD BALANCING

One approach to improve performance of applications and services is through concurrency and by exploiting the available computational resources (processor, memory and network capacities). For example, if a computational domain is divided into sub-domains and each sub-domain represents its own computational requirement, a good load balancing scheme will map each sub-domain onto one or multiple processors to achieve a balanced execution with minimum communication overhead. In some cases, load balancing may involve moving partially completed work as the loads on the hosts change [35].

A wide variety of both general-purpose and application-specific load balancing techniques have been proposed [8, 21, 41, 42, 48, 49, 50, 51]. Since in our research, we are more interested in using application related information to optimize performance, we will focus on application-specific load balancing algorithms. The major steps in load balancing processes are: (1) load information collection; (2) decision making; and (3) data and/or computation migration [36]. From the application's perspective and how the load balancing scheme implements the first two aspects, we can classify load balancing schemes into loosely *synchronous* and *asynchronous* ones. For loosely synchronous load balancing methods, all the load information from all processors must be obtained in order to reconstruct the load

balancing decision. Asynchronous methods only exchanges information between “neighbor” processors. Load balancing occurs only within neighborhoods which means that unaffected processors can continue with their computations and thus the communication overhead is less than synchronous methods. Since synchronous load balancing schemes are using global information among the processors, they can achieve load balancing with high quality but the cost associated with synchronization and data communication grows as we increase the number of processors and the size of the problem. In contrast, asynchronous methods have less data communications and synchronization overhead but with less efficient load balancing [36].

## 3.1 Asynchronous Load Balancing

In scientific computing applications, computation domain can be divided into meshes. If we divide the meshes into blocks and assign blocks to different processors, we need to consider two challenging problems. First, how can we distribute the blocks among the processors while balancing their loads? Second, since the boundaries between different blocks represent the communication between different processors, how can we partition the graph by just migrating within adjacent processors in order to minimize the inter-processor communications?

In [77], the authors developed the Zoltan Parallel Data Services Toolkit [78, 79]

to provide dynamic load balancing to a wide range of applications. The application developers can use Zoltan to switch partitioners by changing the run-time parameters. First Zoltan asks the user to decide the graph partition quality factors [80] such as computational balance and surface index. Then Zoltan uses these factors to create hierarchical balancing schemes using a dynamic resource utilization model (DRUM). DRUM uses a tree structure to model the computing environment including the capacities and communication. The tree is annotated with performance data and used to guide the hierarchical partitioning and dynamic load balancing. The performance data will be input to some callback function which will be used to adjust the hierarchical load distribution based on the function values.

ParMETIS [81, 82] is one of the most frequently used dynamic load balancing tools found in the scientific computing literature. Upon imbalance, ParMETIS uses scratch-remap or diffusive scheme to adjust the existing partition to achieve a better load balance. ParMETIS calculates Relative Cost Factor ($\beta$) which is application-specific and describes the relative costs required for performing the inter-processor communication during parallel computations and performing the required data redistribution to maintain load balancing. This means ParMETIS needs to minimizes Equation (3.1),

$$|E_{cut}|+\beta\ |V_{move}| \tag{3.1}$$

here $|E_{cut}|$ is the edge-cut of the partitioning and $|V_{move}|$ is the total cost of data

redistribution. The partitioning has three phases. First, the graph is coarsened to minimize both the number of edge-cuts and the data redistribution cost. Second, use the partitioning method to create an initial partition. Then the cost function of Equation (3.1) is calculated. Finally, a multilevel refinement algorithm is applied to this initial partition to minimize Equation (3.1).

In DRAMA project [84], it uses a cost model [85] for a load balancing graph partitioning algorithm. The cost model measures the quality of the current distribution of load and predicts the possible effect on the computation if some parts of the graph are migrated to other processors. By taking into consideration the heterogeneity of the modern cluster architecture, DRAMA uses hardware specific parameters to predict the computation and communication speeds of the processors. The cost model will consider element-element, node-node and element-node data dependencies in order to calculate the communication costs. DRAMA defines application into different phases according to the communication synchronization points. The cost function evaluates the cost at each phase ***h*** and per processor ***i*** which will include accumulated calculation contributions $w_{hi}$ and communication costs $c_{hi}$. Thus the costs $F_h$ at each phase and the total costs ***F*** at all the phases are given by

$$F_h = \max_{i \in \{0 \ldots p-1\}} (w_{hi} + c_{hi}) \tag{3.2}$$

$$F = \sum_{h=1 \ldots \# \, phases} F_h \tag{3.3}$$

where ***p*** is the total number of processors used in the application. Then DRAMA uses

Equation (3.2) and (3.3) to calculate the load imbalance ratio,

$$\lambda = \frac{\max\limits_{i=0...p-1}(F_h)}{\operatorname{avg}\limits_{i=0...p-1}(F_h)} \tag{3.4}$$

If $\lambda$ is larger than certain threshold, the partitioner will use DRAMA to calculate a new load balance ratio and the costs using its cost model to re-balance the computation and keep Equations (3.2 – 3.4) minimized. DRAMA provides an efficient way to calculate the costs related to FEM applications and how the computation and communication will vary if the meshes are repartitioned.

Most of the scientific applications (simulations) are computed at discrete time steps. It is advisable to perform load balancing at strategic point instead of random computation point. CHARM++ is a toolkit providing a runtime framework in which load balancing policies can be switch in and out during the application's execution [83]. CHARM++ divides the application data domain into many chunks which is called "chare" object. First CHARM++ sends messages to physical processors. Each processor receives the message then invokes the computation specified by the entry point of the program contained in the message. Load balancing is achieved by mapping chares to available processors in a balanced fashion. Different mapping strategies can be used here. The simplest one is Greedy Strategy, which sorts both chare workloads and processor load levels then assigns the most time consuming chare to the most lightly loaded processor. Refinement strategy minimizes the number of chare

migrations if an imbalance occurs. For each overloaded processor, heavy chares are migrated to under-loaded processors until the load falls below a pre-specified threshold.

Asynchronous methods are attractive because they do not need the global system and application information on system and application which requires a great deal of communication overhead to obtain. But the load balance across the entire system using asynchronous system is often in a lower quality comparing to synchronous methods.

## 3.2 Synchronous Load Balancing

Synchronous load balancing methods need to gather all the load information across the entire system before they can make a balancing decision.

In large parallel distributed systems, due to the complexity of the system topology and communication, the global information exchange and synchronization phase becomes the performance bottleneck. In scientific computing, such global information is the local mesh computation results needed by other processors. In order to reduce the global information exchange and synchronization overhead, it is recommended to map multidimensional space to one dimension to minimize communications costs. Space-Curve Filling (SCF) [43, 44] is one method to decompose a multi-dimensional problem into one dimension but keep the data locality. In this method, SFC partitioning scheme is based on a recursive linear representation of a multi-dimensional grid hierarchy generated using space-filling mappings [43, 45].

These mappings are computationally efficient and provide recursive mappings from $N$ dimensional space to 1-dimensional space. The resulting space is first partitioned into blocks of a minimum acceptable granularity. The SFC then passes through these blocks. SFC mappings maintain the application domain locality and maintain this locality when the meshes are restricted or prolonged. Using SFC linear representation, the problem of partitioning the multi-dimensional multi-level grid hierarchy to balance load across the processors is then converted to decomposing the 1-dimension representation to balance load across the processors. Computational load is determined by the length of the SFC segment and the levels of recursion it contains. The former corresponds to block sizes and the latter corresponds to the levels of refinement [46]. The SFC partitioning scheme is illustrated in Figure 3.1.

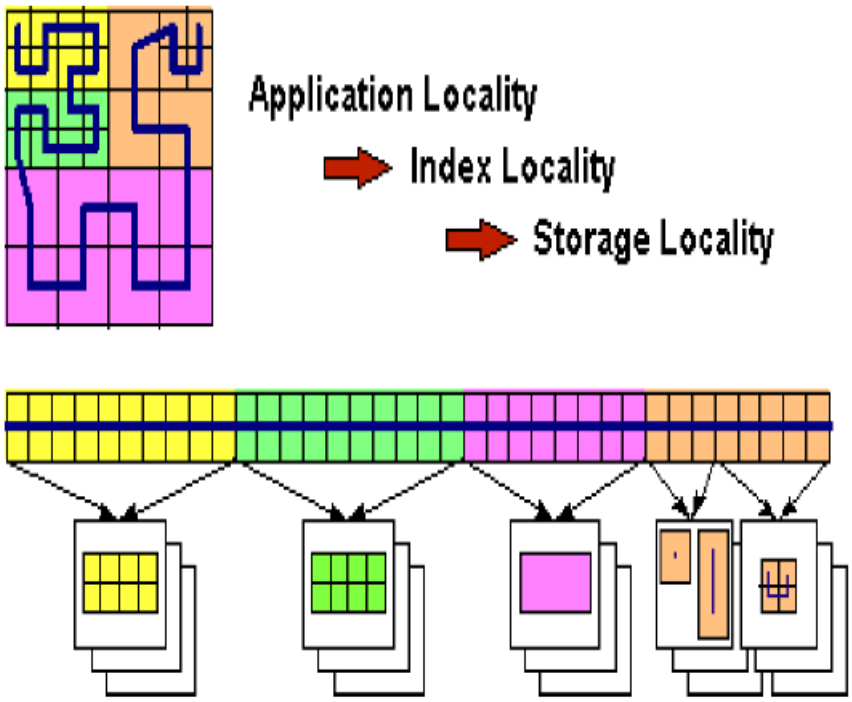


**Figure 3.1 Space-Filling Curve Mappings Encode Application Domain Locality and Maintain Locality through Expansion and Contraction**

Among mesh-based numerical algorithms, Structure Adaptive Mesh Refinement (SAMR) is a type of multi-scale algorithm that achieves high resolution in error-prone regions [47]. The dynamic nature of SAMR requires a dynamic load balancing (DLB) to solve the load balance problem. Dynamic load balancing on SAMR application is a large topic and has been intensively studied for more than a decade and a large number of schemes have been presented [48, 49, 50, 51, 52, 53, 54, 55].

As a synchronous method, the major issue of any DLB scheme is to address the identification of overloaded versus underloaded processors, the quantity of data to be

transferred from the overloaded processors to the underloaded processors and the total cost of applying the DLB scheme.

Structure Adaptive Mesh Refinement (SAMR) method represents the grid hierarchy in a tree structure. In each prolongation or restriction time (Grid changes from fine to coarse is called prolongation and the contradiction is called restriction), the number of levels, the number of grids and the locations of the grids change. Initially, a uniform mesh covers the entire computational domain and in the regions where higher resolution is required, a finer grid is added (restriction) over this region. When higher resolution area is no longer needed, the finer grid will be merged into coarser grid (prolongation). This process repeats recursively with each prolongation and restriction resulting in a tree of grids which is shown in Figure 3.2 [56, 57].

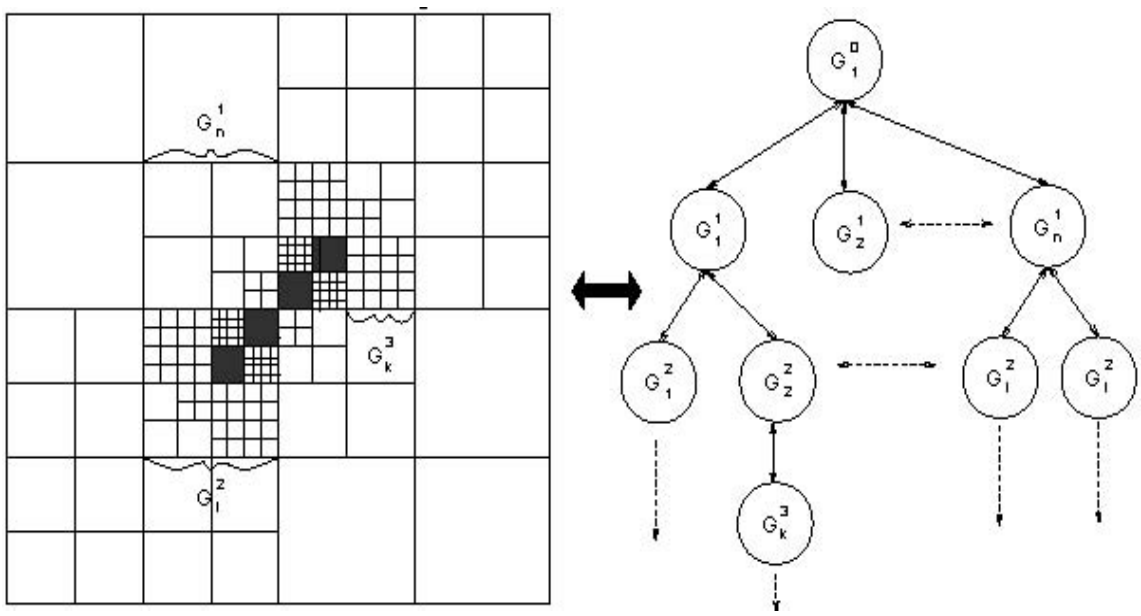


**Figure 3.2 Berger-Oliger Formulation of Adaptive Grid Hierarchies for SAMR Applications**

In the middle of the left graph of Figure 3.2, we can see that numerous restrictions are taken. This represents a highly active part in the computations such as the location of a shock wave in a CFD simulation or the front of the fire in a wild fire simulation or largest acceleration of water flow in a hydraulic code. The tree hierarchy of SAMR representation is shown in the right figure. In this example, there are four levels of grids going from level 0 to level 3. Throughout the execution of SAMR application, the tree hierarchy will change according to the grid adaptation.

SAMR also imposes some requirement on new subgrids generation. A subgrid must be uniform, rectangular and aligned with its parent grid and be contained in the

range of its parent grid. All parent grids are totally refined or completely unrefined in the restrict or prolong phases and the refinement factor has to be an integer [58].

The PDE integration in SAMR goes through the various adaptation levels. In each level, it has its own time step advance; finer level has a finer time step. The time step will be advanced recursively until the same physical time as that of the current level is reached. Figure 3.3 illustrates the execution sequence for an application with four levels and a refinement factor of 2 [59].

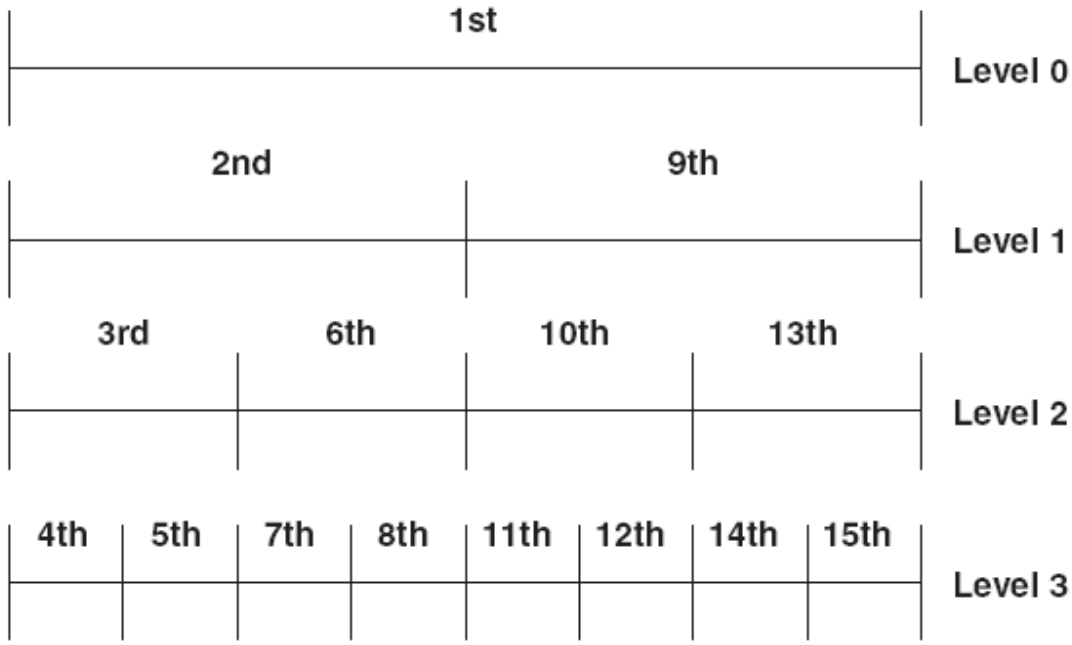


**Figure 3.3 Integrated Execution Order (Refinement Factor = 2)**

First we start with the coarsest grid at level 0 with time step ***dt***. Then the integration continues with one of the finer grid on level 1 with a time step of ***dt/2***. Next, the integration continues with one of the subgrid on level 2 with a further

smaller time step ***dt/4***, followed by the finest grid on level 3 with time step ***dt/8***.

Various load balancing experiments have been done on SAMR applications [59, 60]. The experiments are focusing on four aspects: granularity, dynamism, imbalance and dispersion. Granularity is the basic data migration unit in a synchronous load balancing scheme. A too large granularity will cause new imbalance and large data movement overhead. A too small granularity will take longer time to achieve new balance. Dynamism means the frequency of load changes. After each time step of each level in SAMR, the adaptation process is invoked based on the error rate calculated for each level and thus change the load for each different regions. If the load changes very frequently, the load balancing scheme has to be executed fast to accommodate this change. Load imbalance is a metric to test the imbalance ratio on each processor. Dispersion defines how the load is distributed among all the processors. There could be certain group of processors who have dramatic changes in loads while other groups encounter only minor load changes. Therefore, the DLB should maintain a high quality load balancing for all these groups.

In order to tackle the load balancing problem in SAMR applications, DistDLB [61, 62] takes a two-level method. It first divides a distributed system into “groups”. A “group” is defined as a set of homogeneous processors connected with a dedicated network. The communication within a group is referred as the local communication

(i.e. low overhead communication) and between different groups is denoted as remote communication (i.e. high overhead communication). The DistDLB explores the two level approaches in two load balancing phases: global balancing phase and local balancing phase. The global balancing phase is performed on grids at level 0 by appropriately partitioning the workload among groups based on their relative capacity. This phase is a coarse grain load balancing. The local balancing phase is performed on grids at finer levels with the objective to equally partitioning the workload among processors within a group [59]. This two-level approach of DistDLB has several advantages. First, it addresses the heterogeneity of the networks between the processors and reduces the number of remote communications which is more expensive than local communications. Second, by separating the two phases, we can explore different load balancing schemes for each phase. For example, in the local balancing phase, paraDLB [63] or other scheme can be used. paraDLB divides each load balancing step into one or more iterations of moving-grid phase and splitting-grid phase. The moving-grid phase moves grids from overloaded processors to underloaded processors. The splitting-grid phase splits a grid into smaller grids along the longest dimension in order to reduce the communication caused by splitting. These two phases are interleaved and executed in parallel in paraDLB scheme.

Instead of focusing on the difference between global and local data communications, [87] proposed a DLB incorporating a heuristic remapping algorithm.

The idea is to measure the processor workloads and the amount of data movement as precise as possible. Initially a mesh is partitioned and mapped among the available processors. Then a few iterations are computed and the solution variables are updated. Based on the errors found in the solution variables, an evaluation step will determine whether a new mesh is needed and if it is sufficiently unbalanced to grant such a new partition. If the current partition is still balanced under the new mesh, the control is returned back to the code. Otherwise, a new partition procedure is invoked and the processors are reassigned. If the cost of remapping the data is bigger than the gain of the new load balance, the suggested partition will be discarded. The flow of this algorithm is shown in Figure 3.4.

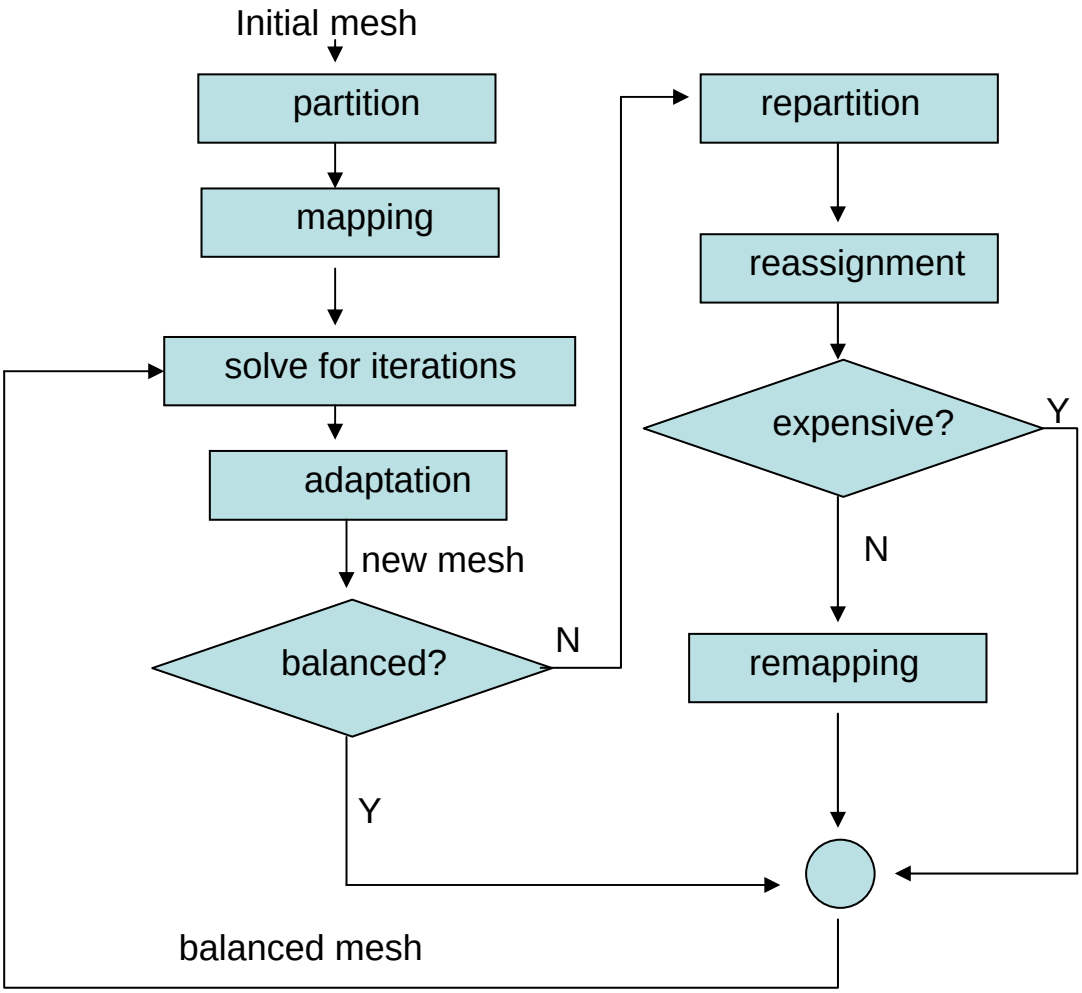


**Figure 3.4 Overview of Adaptive Mesh Refinement Computation**

In this chapter, we reviewed asynchronous load balancing and synchronous load balancing schemes. In Chapter 4, we present a novel Physics Aware Programming (PAP) paradigm which explores physical properties to develop efficient and scalable parallel algorithms and programs.

# CHAPTER 4 PHYSICS AWARE PROGRAMMING (PAP) PARADIGM AND RUNTIME MANAGER

## 4.1 Introduction

The advances in computing and communication technologies and software tools have resulted in an explosive growth in the number of scientific computations that are running on a large scale computing facility. Numerous researches have been done on how to effectively use the hardware resources and focus on the hardware states to steering the application execution. For example, in [64], computing facilities are grouped into domains and an application will be executed based on the capability of accessing the datasets and storage devices. Other examples include utilize the hardware effectively with load balancing [60, 61, 62, 63], using mobile agent to migrate jobs between nodes for unexpected node errors [65]. Yet, there are not only hardware states that can be exploited, application states are more important for efficient application execution. We developed a runtime manager system that has self-optimizing capabilities to exploit the application's physics properties and runtime states. The runtime manager will create a computational framework that is capable of exploiting the physics complexity of applications and the numerical techniques used to optimize the execution of a given application.

In the domain of scientific computations, multi-phase problems are usually encountered in a large class of problems such as groundwater modeling, oil reservoir simulations, computational fluid dynamics and thermoelastic displacement analyses [17, 18, 19, 20, 21, 22]. However, these applications are generally time dependent and their volatile nature make them hard to implement efficiently. As time evolves, these problems will evolve to different phases with different physical characteristics. Most of the current research uses one algorithm to implement all the phases of the application execution that might experience at runtime, but a static solution or a numerical scheme for all the application phases might not be ideal and consequently results in a poor application performance.

Some runtime optimization techniques such as Adaptive Mesh Refinement (AMR) refine the mesh size if the error in the current state exceeds some predefined threshold [8]. In our approach, we take a more general runtime optimization methodology that is not based on error as in AMR (the error depends on the numerical algorithm used and its stability) but rather it is based on the application physics temporal and spatial characteristics. Our method is general and it can be applied to Finite Element Method, Domain Decomposition Method, Finite Volume approximations [9], just to name a few.

We determine the application execution phase by exploiting the knowledge about the application physics and how it behaves at runtime, and then use the appropriate

numerical solution/algorithm for each detected phase during the application execution. We have developed and implemented an autonomic runtime manager that we refer to as Physics Aware Runtime Manager (PARM) to exploit Physics Aware Programming (PAP) methodology. We have evaluated our PAP approach on several applications such as Variable Saturated Aquifer Flow and Transport (VSAFT2D) [15] and wildfire simulation [113]. For example, for each phase of the VSAFT2D application, different numerical schemes and solvers that can best exploit its physics characteristics were identified and used to implement that phase at runtime. Our experimental results show that PAP has very little overhead and its solution is as accurate as the one generated by the finest grid size. In our experiment on 2,000,000 elements implemented on 32 processors, PAP results in an improvement of 80% in performance when compared to the finest grid size points.

## 4.2 Physics Aware Programming Paradigm—Methodology

To explain the PAP paradigm, we use a diffusion application that contains routines that are used in many different real-world applications (e.g. sideway heat equation [96], boundary-layer problems [95]). Let us consider the heat diffusion problem shown in Figure 4.1.

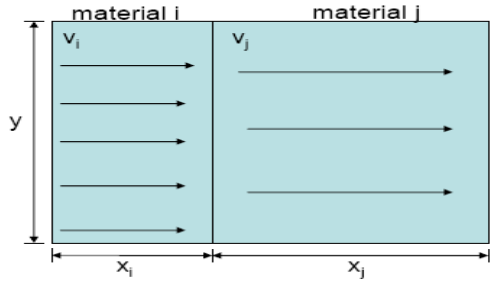


**Figure 4.1 an Example of Diffusion Problems**

The heat propagation can be modeled as a diffusion-type problem and it is governed by Equation (4.1) [97]

$$\alpha_x \frac{\partial^2 u}{\partial x^2} + \alpha_y \frac{\partial^2 u}{\partial y^2} + \alpha_z \frac{\partial^2 u}{\partial z^2} = \frac{\partial u}{\partial t} \tag{4.1}$$

where $u$ is the temperature, $\alpha_x$, $\alpha_y$ and $\alpha_z$ are the thermal conductivity in the $x$, $y$ and $z$ coordinate directions. Let us also assume the heat is propagating through two different media $i$ and $j$ as shown in Figure 4.1.（In reality, more heterogeneous materials can be encountered, but for simplicity, we will only consider two materials in this illustrative example.） The temperature change in these two materials is determined by the thermal conductivity $\alpha$. The Partial Differential Equations (PDE) shown in Equation (4.1) are usually solved by applying Taylor's theorem [98],

$$\begin{aligned} u_t(x,t) &= \frac{u(x,t+k)-u(x,t)}{k} + TE \\ TE &= -\frac{k}{2} u_{tt}(x,\tau), t < \tau < t+k \end{aligned} \tag{4.2}$$

where $TE$ is a truncation error, i.e., an analytical PDE solution equals numerical solution plus the truncation error ($TE$).

In what follows, we show how to apply the PAP approach to choose the spatial and temporal characteristics of the solution based on the thermal conductivity $\boldsymbol{\alpha}$ associated with media type in order to improve the performance of Equation 4.2 solving.

### 4.2.1 Determining the Solution Temporal Characteristics ($\Delta t$)

The temporal characteristics of the solution depend primarily on the required precision. For simplicity, we only discuss the temporal characteristics with respect to one dimension; other dimensions can be treated in a similar way. Assuming we discretize the problem domain into small grids. On each grid point, $u_n^j$ represents the exact solution. Suppose that $v_n^j$ represents a measured or computed value of $u_n^j$ that differs from the exact $u_n^j$ value by an error amount $e_n^j$ that is,

$$u_n^j = v_n^j + e_n^j \tag{4.3}$$

Usually, the user will specify a small constant $\boldsymbol{E}$ as the upper bound on the acceptable error for the solution, thus for all grid points, the errors $e_n^j$ must be less than $\boldsymbol{E}$, that means

$$|e_n^j| \leq E \qquad \forall\ \boldsymbol{n}, \boldsymbol{j} \tag{4.4}$$

Finally, assume that $\boldsymbol{M}$ is an upper bound for $u_{tt}(x,t)$. Notice that $\boldsymbol{u_{tt}}$ is the acceleration of the heat transmission in the problem, this value can be obtained by calculating the problem for one time step and then use it to estimate the maximum acceleration of the heat as shown in Equation (4.5) [98].

$$|u_{tt}(x,t)| \leq M\ \forall\ \ \boldsymbol{x},\ \boldsymbol{t} \tag{4.5}$$

Now, if we apply the forward difference formula [98] to estimate $u_t(x_n, t_j)$, we get

$$u_t(x_n, t_j) = \frac{u_n^{j+1} - u_n^j}{\Delta t} - \frac{\Delta t}{2} u_{tt}(x_n, \tau_j), t_j < \tau_j < t_{j+1} \tag{4.6}$$

which is the same as Equation (4.2). According to Equation (4.3), Equation (4.6) can be rewritten as shown in Equation (4.7)

$$u_t(x_n, t_j) = \frac{v_n^{j+1} - v_n^j}{\Delta t} + \frac{e_n^{j+1} - e_n^j}{\Delta t} - \frac{k}{2} u_{tt}(x_n, \tau_j) \tag{4.7}$$

From Equation (4.7), the $u_t$ consists of three parts: a computed difference quotient, a rounding-measurement error and a truncation error. Hence the total error incurred in using the difference quotient $(v_n^{j+1} - v_n^j)/\Delta t$ to approximate $u_t(x_n, t_j)$ can be bounded as follows:

$$\left| \frac{e_n^{j+1} - e_n^j}{\Delta t} - \frac{\Delta t}{2} u_{tt}(x_n, \tau_j) \right| \le \left| \frac{e_n^{j+1} - e_n^j}{\Delta t} \right| + \left| \frac{\Delta t}{2} u_{tt}(x_n, \tau_j) \right| \le \frac{2E}{\Delta t} + \frac{M \Delta t}{2} \tag{4.8}$$

In order to satisfy the desired accuracy, we choose $\Delta t$ that minimizes the maximum total error. In this example, $\Delta t_{opt} = 2\sqrt{E/M}$ where $M$ is an upper bound on the heat transmission acceleration. The $M$ value depends on the physics properties of each material. For a fixed error $E$, $\Delta t$ can be optimally chosen to improve performance by increasing $\Delta t$ (solution uses less number of time steps) without compromising the accuracy of the solution. For example, $\Delta t$ for the rubber material can be made much larger than the $\Delta t$ for the aluminum material. If physics properties are not exploited, which is normally the case, the domain scientists will choose the smallest $\Delta t$ for all the materials shown in Equation (4.1). In PAP approach, the thermal conductivity of each

material is utilized to determine the optimal time step of the solution method associated with each sub-domain that minimizes the maximum total error. For example, assuming we have a computation domain composed by two different materials with a constant heat boundary, if we found the heat transmission acceleration in each material is 0.2 and 7.2 respectively through our calculation, then the time step in the slow transmission material could be set 6 times faster than the other one. This can be justified because in a faster heat transmission; we need a smaller time step to capture the heat variation between each time step.

### 4.2.2 Determining the Solution Spatial Characteristics ($\Delta x$, $\Delta y$, $\Delta z$)

The appropriate spatial characteristics $\Delta x$, $\Delta y$, $\Delta z$ of the solution depend on the physical properties of the material and the numerical method being used. If implicit method [33] is used, in the illustrative example, the stability criterion [33] for the $x$ direction can be determined by:

$$\frac{2\alpha\Delta t}{(\Delta x)^2} \leq 1 \tag{4.9}$$

where $\alpha$ is the heat conductivity along the $x$ direction. The same criteria can be applied for $y$ and $z$ directions. The intuition behind equation (4.9) is that as long as within one time step, the heat transmission distance is within $\Delta x$, the computation should be stable, i.e., $\Delta x$ captures the heat transmission distance during one time step. Traditionally, domain scientists will apply the smallest spatial characteristics $h$ =

***min(Δx, Δy, Δz)*** for the entire domain to achieve stability. As discussed in the previous section, different thermal conductivity materials will have different time steps and thus their spatial characteristics could be chosen such that the computational complexity is reduced while maintaining stability. In the example shown in Figure 4.1, PAP-based solution will use larger spatial characteristics for material ***i*** since it has a higher thermal conductivity (i.e., it has slow heat transmission rate). For example, if we keep ***Δt*** as a constant, the material with larger heat conductivity will have a larger spatial characteristic.

## 4.3 Seismic Application – Wave Equation

We apply the PAP paradigm to a seismic application [99, 100] which is governed by the wave equation shown in Equation (4.10).

$$u_{tt} = c^2 \nabla^2 u \tag{4.10}$$

where ***c*** is a physical parameter represents the phase velocity. $\boldsymbol{u_{tt}}$ can be viewed as the vertical acceleration at the point ***(x, y, z)*** in the coordinator. Intuitively, if the point is far from the equilibrium point, $\boldsymbol{u_{tt}}$ will be larger and vice versa. The ***c*** parameter needs to be considered because it models the physical properties of the application. If the earthquake wave is passing through different geological structures, say the wave is propagating from the ocean to the coast, the phase velocity will be different for each structure and as a result, we can use different solutions that have different temporal and spatial characterisitics that maximize performance without compromising the

accuracy of the solution. The performance gain that can be achieved using our PAP paradigm is detailed in Chapter 5.

## 4.4 A Real World Application --- VSAFT2D

We evaluated the performance gain of the PAP paradigm by using it to implement a Variable Saturated Aquifer Flow and Transport (VSAFT2D) application kernel developed at The University of Arizona [15]. The discussion here focuses on the two dimensional case, but the solution can be easily extended to a 3D case because of the similarity of the governing equation and application implementation. The VSAFT2D flow in variably porous media is computed as in Equation (4.11):

$$\frac{\partial}{\partial x_i}\left(K_{ij}\frac{\partial}{\partial x_j}(\psi+\beta_g x_2)\right)=(C+\beta_s S_s)\frac{\partial \psi}{\partial t}-q_s \qquad \text{in } \Omega \qquad (4.11)$$

where $x_i$ is a spatial coordinate (i = 1, 2 in 2D case); $K_{ij}$ is the hydraulic conductivity tensor; $\psi$ is the pressure head; $C$ is the specific moisture capacity; $\beta_g$ is the index for gravity; $\beta_s$ is the index for saturation; $S_s$ is the specific storage; $t$ is time; $q_s$ is the source/sink term and $\Omega$ is the solution domain.

Equation (4.11) can be solved using the discretization method such as Finite Difference or Finite Element method [18, 19]. The whole solution domain $\Omega$ is divided into uniform small elements. The calculation is done on each element iteratively until a converged result is achieved. (This is how the original code is implemented and we denote this approach as the “finest grid” implementation. The same notation is used in

the PAP approach.) Yet, considering different elements will have different $K_{ij}$ (which means the computation domain is composed by different materials). According to PAP paradigm, we can use different grid size and timestep for each different $K_{ij}$ at runtime to improve performance without violating the numerical accuracy as discussed in Sections 4.2.1 and 4.2.2. More interestingly, for transient cases, $K_{ij}$ is a variable for two different materials (e.g., sand and silt) as shown in Figure 4.3.

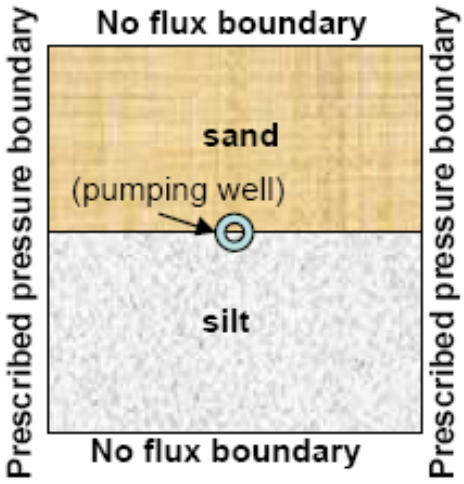


**Figure 4.2 VSAFT2D Domain with 2 Different Materials**

At the intersection of curves $K_{sand}(h)$ and $K_{silt}(h)$, $K_{sand}(h')=K_{silt}(h')$. When $h < h'$, $K_{silt} < K_{sand}$, according to our analysis in Sections 4.2.1 and 4.2.2, we can use a finer grid for the silt sub-domain, i.e., a finer $\Delta x$, $\Delta y$ and $\Delta z$ and a coarser grid for the sand domain. However, when $h > h'$, $K_{silt} > K_{sand}$, we need to apply a coarser grid for the silt region but a finer grid for the sand region. The exact value of spatial and

temporal characteristics for each sub-domain silt and sand are computed as discussed in Sections 4.2.1 and 4.2.2.

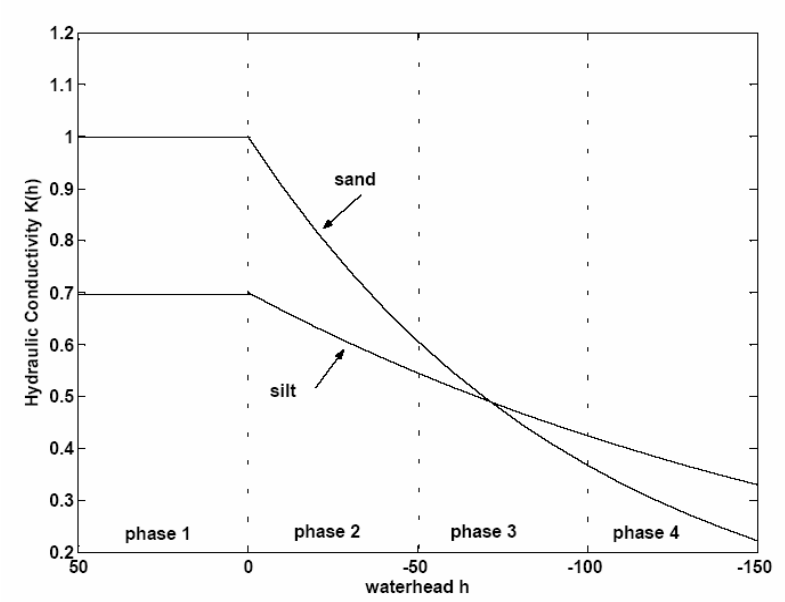


**Figure 4.3 Functional Relationships between Hydraulic Conductivity and Waterhead with Sand and Silt**

At last, two PAP features are noteworthy. First, since we are using different spatial and temporal characteristics at different sub-domains, at the boundary of different sub-domains, we need to use some weighing interpolation. Since this operation is done once for each time step, it is negligible compared to the other operations needed to solve the linear system. Second, in our illustrative example, the physics property being exploited is the thermal conductivity which is assumed to be constant. But in real world applications, the thermal conductivity is variable and it depends on the orientation and temperature. The PAP solution determines the thermal conductivity changes at runtime

and consequently adjusts the spatial and temporal characteristics of the solution. The traditional solution methods are static and do not exploit the physical properties of application domains which lead to poor performance.

## 4.5 The Autonomic Computing System

Due to the increase in complexity, dynamism, and heterogeneity of resources in computing systems and applications, the control and management of such systems and their services become a challenging research problem. Current state of the art control and management techniques are insufficient in managing such systems and services. Therefore, new approaches must be developed such as Autonomic Computing [103] that is analogous to the biological nervous systems. Autonomic Computing is a promising paradigm to efficiently address the control and management challenges of the next generation of large scale distributed computing systems (e.g. Grid). An autonomic computing system is a system that can self-configure, self-protect, self-heal and self-optimize its operations automatically at runtime to meets its overall system and service requirements [108]. In this dissertation, we focus on self-optimizing attribute of autonomic computing systems.

### 4.5.1 Physics Aware Runtime Manager Architecture

The Physics Aware Runtime Manager (PARM) is an autonomic computing implementation of PAP paradigm, responsible for setting up the execution environment

for an application and then control and manage the application execution to optimize its performance.

Figure 4.4 shows a Physics Aware Runtime Manager (PARM) to implement the PAP methodology.

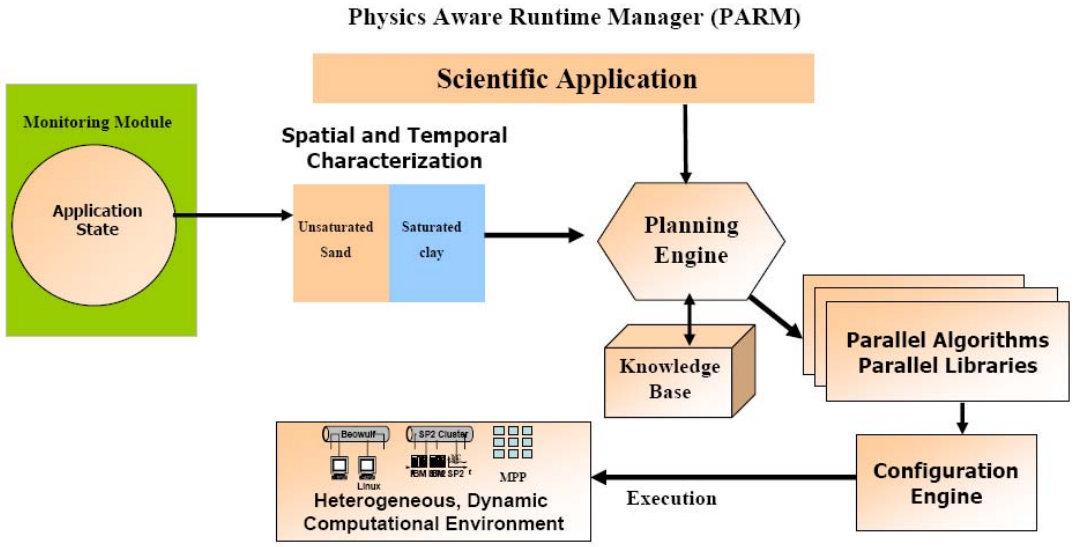


**Figure 4.4 Physics Aware Runtime Manager Architecture**

At runtime, the application execution can be affected in different ways, for example, it can encounter a state change during execution or it may slow down due to some hardware overloading. It is the job of PARM to manage such situations as they occur at runtime. PARM has the following main components as shown in Figure 4.4: 1) Monitor Engine, 2) Planning Engine, 3) Configuration Engine, 4) Knowledge Base and 5) Parallel Algorithms.

The Monitor Engine monitors the application execution; identifies different phases based on the physical properties of the application. Monitor Engine will focus the variables that can be used to characterize the computational domain state. In the example discussed in Section 4.2, the monitor engine will monitor the heat diffusion acceleration $u_{tt}$ change and the heat conductivity $\alpha$ change and use these variables to define the phase switch. The monitored physical property changes are then fed into the Planning Engine.

The Planning Engine selects alternate execution strategies to optimize the behaviors and operations of the application. Our approach is to use runtime performance models to predict the behavior of complex applications at runtime then the Planning Engine will determine the optimal spatial ($\Delta x$, $\Delta y$, $\Delta z$) and the temporal characterization ($\Delta t$) for the application solution while maintaining the desired accuracy of the solution.

In addition to exploiting the physics properties of the application, the Planning Engine will search for a better solver for the linear system. Since most scientific computations involve solving sparse matrices in general, we focus on linear solvers for sparse matrices. Generally, a sparse matrix can be categorized into three types: diagonized, tri-diagonized and band diagonized. Numerical libraries such as LAPACK [3, 5, 6, 7] and PETSc [12] provide various solvers for matrices with different structures.

It is a daunting task to select an optimized solver for a particular application from these library routines. In addition, a particular solver will be best for a particular matrix format but performs poorly with other matrix formats. Even for the same matrix format, the same solver will perform differently on various computer platforms since memory hierarchy plays an important role in linear algebra computations [109]. Our experimental results show that for a particular hardware platform, the performance of a solver is unrelated to the matrix size but only to the matrix type (e.g. sparse, dense, diagonal, tridiagonal) as long as the matrix size passed the point where memory hierarchy will impose significant effect on matrix operations [109]. We use a Variable Saturated Aquifer Flow and Transport (VSAFT2D) application kernel [15] as shown in Section 4.4 to explain the impact of linear solver on an application execution.

Figure 4.5 shows the execution time of VSAFT2D for saturated steady cases with a homogeneous grid size for the entire simulation domain with different problem sizes and solver types. The y-axis is on a logarithm scale. When the problem size exceeds 288 elements, the conjugate gradient solver will always give the best performance for a saturated problem. The selection of a linear solver is very important because it will be called many times in each time step. Consequently, significant performance differences will be observed for various solvers even for the same problem size. Considering that transient problems like VSAFT2D needs large number of time steps to have an accurate approximation of the real world problem, and hence, by choosing the optimal linear

solver, a significant performance gain can be achieved. For example, as shown in Figure 4.5, for a problem size as small as 490,000 elements, the conjugate gradient will save 38% of the time spent solving the linear system when compared with a transpose free QMR (tfqmr) solver.

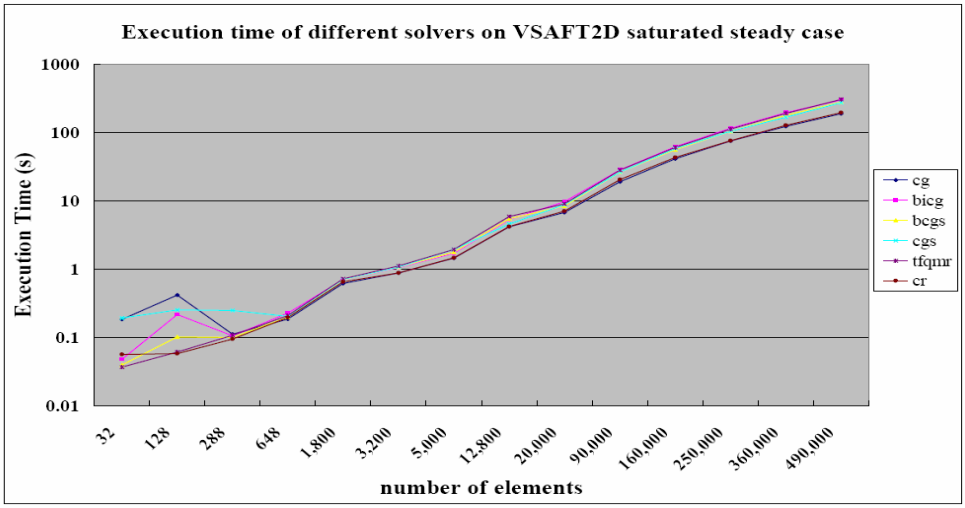


**Figure 4.5 Different Solvers' Performance on Various Problem Sizes**

In order to develop online planning that can be carried out efficiently at runtime, we have developed Knowledge Base to exploit the knowledge acquired from previous successful execution strategies and the Knowledge Base provides the decision support to the Planning Engine to pick up the appropriate rule from a set of rules to improve the performance. For example, the knowledge base could have a rule as: if the application is using Finite Difference implicit Method, then choose Conjugate Gradient with block Jacobi preconditioner. From our experimental results, this approach has been the best linear solver.

After the optimal algorithm is selected, the Configuration Engine generates the data needed by the new algorithm based on the previous phase execution results and the temporal and spatial characteristics of the new solution. For example, regenerate grid values using interpolation and extrapolation after we change the grid size. After the Configuration Engine finishes its job, the application resumes its execution with the new configuration.

# CHAPTER 5 EXPERIMENTAL RESULTS OF PHYSICS AWARE PROGRAMMING PARADIGM

We evaluate PAP methodology with two application kernels we implemented, 2-D diffusion kernel and 2-D seismic application kernel together with VSAFT2D, a 2D hydrodynamics application developed by researchers at the Hydrology and Water Resources Department of The University of Arizona. We use the first two kernels to evaluate the PAP performance gain and the total overhead of the PAP paradigm. We use VSAFT2D to evaluate the PAP performance, overhead and the error rate of the final solution. For simplicity, we focus PAP on the spatial characteristics of the application such as the grid size for each application execution phase. However, we do not change the temporal characteristics of the application, although our approach can support both types of adaptations (spatial and temporal).

We integrated the PETSc [12] with our runtime optimization approach. PETSc is a widely used library that provides a uniform interface for many solvers. Other linear solvers such as SuperLU [14] can be easily integrated with our approach. We have evaluated the performance of our approach on The Ohio Supercomputing Center Pentium 4 Cluster, a distributed/shared memory hybrid system constructed from commodity PC components running the Linux operating system. The cluster has 112 compute nodes for parallel jobs; approximately half of them are connected using

Infiniband, a switched 8 Gbit/s network [16].

Each compute node is configured with the following features:

- Four gigabytes of RAM
- Two, 2.4 GHz Intel P4 Xeon processors, each with 512kB of secondary cache
- One 80 gigabyte ATA100 hard drive
- One Infiniband 10Gb interface
- One 100Base-T Ethernet interface and one 1000Base-T Ethernet interface

## 5.1 2-D diffusion Application Kernel

We evaluated the performance gain of PAP using a 2-D diffusion application kernel. The same approach can be easily extended to a 3D case. We solve the diffusion equation using the Finite Difference Method. The application is composed of two different materials and each occupies 50% of the total computing domain area. In reality, this distribution varies depending on the area being modeled and our objective here is to show how PAP approach can exploit the physics properties to reduce the computational complexity of the solution. The application execution changes its phase every 100 timesteps.

We execute the code without PAP with finest grid size and then with PAP approach that determines the optimal grid size for each execution phase.

The execution time of the application kernel on different number of nodes is shown in Figure 5.1.

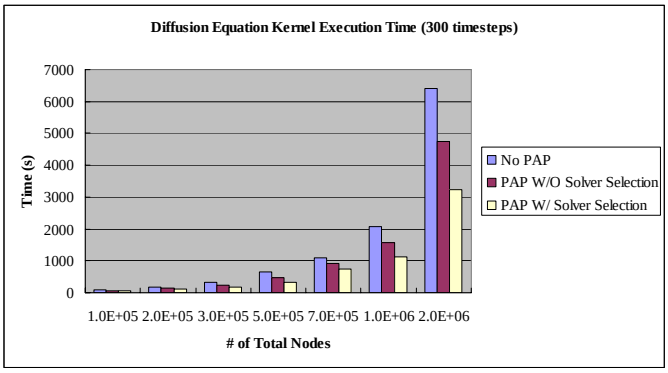


**Figure 5.1 2D-Diffusion Kernel Execution Time on 32 Processors**

In Figure 5.1, we evaluated the PAP performance for two cases: with and without intelligent solver selection. The PETSc default solver for this application is the “least square” and PARM selects the “conjugate gradient” for this particular application. We can see that by using the intelligent solver, we can obtain further speedup. Both cases exceed the original “finest grid” execution. For example, for one million (1M) node case, without and with solver selection we can achieve a performance gain of 31% and 84%, respectively, when compared to the original code. For two million (2M) node case, the performance gains are 84% and 97% respectively.

We also evaluated the implementation overhead of the PAP paradigm. The

overhead is incurred due to the interpolation or the extrapolation of the data points generated from the changes in the spatial/temporal characteristics. In Table 5.1, we can see that as the problem size increases, the overhead decreases because PAP approach can save more execution time due to reducing the linear system size for large size problems.

**Table 5.1 PAP Overhead on 2-D Diffusion Kernel**

| Problem Size (K) | 200 | 500 | 700 | 1000 | 2000 |
|---|---|---|---|---|---|
| PAP Exec. Time (s) | 106 | 338 | 736 | 1128 | 3243 |
| Overhead (s) | 1.44 | 3.26 | 6.81 | 10.67 | 29.6021 |
| Percentage (%) | 1.3 | 1.3 | 0.96 | 0.94 | 0.91 |

## 5.2 Seismic Application Kernel

We implemented the seismic application kernel using the wave Equation (4.10). The problem setting contains two different geological structures with equal proportion. We dynamically change the phase velocity ***c*** in every 100 time steps. PARM monitors such a change and adjusts the spatial characteristics to exploit the physics property of the kernel at each phase change.

The execution time using PAP versus without PAP on different number of nodes is shown in Figure 5.2. In this case, PARM selects the appropriate solver for this kernel. For this kernel, PETSc default solver is the least square and PARM selects the

BiConjugate Gradient with a block Jacobi preconditioner. For 1M node case, without and with solver selection, we have a performance gain of 50% and 115%, respectively, when compared to the original code. For 2M node case, the performance gains are 69% and 191%, respectively.

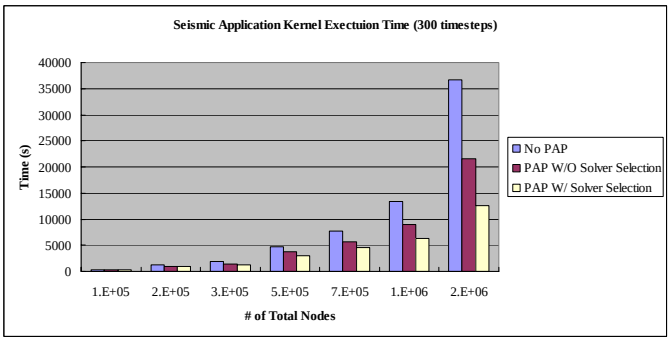


**Figure 5.2 2D-Seismic Kernel Execution Time on 32 Processors**

We also evaluated the overhead of PAP on this kernel as shown in Table 5.2. Similar results as shown in Table 5.1 that shows the overhead decreases as the problem size increases. It is because PAP saves execution time due to reducing the linear system size of the large applications.

**Table 5.2 PAP Overhead on 2-D Seismic Kernel**

| Problem Size (K) | 200 | 500 | 700 | 1000 | 2000 |
|---|---|---|---|---|---|
| PAP Exec. Time (s) | 895 | 3002 | 4588 | 6238 | 12568 |
| Overhead (s) | 11.625 | 28.83 | 41.05 | 58.66 | 118.83 |
| Percentage (%) | 1.3 | 0.9 | 0.8 | 0.9 | 0.9 |

## 5.3 VSAFT2D Application Kernel

We evaluated the performance of the PAP implementation of the VSAFT2D application, error rate and overhead. The convergence of the VSAFT2D application is determined by comparing the maximum pressure head difference between two successive iterations and comparing that difference to a prescribed tolerance. We evaluated the performance of the PAP approach focusing on the transient problem setting as shown in Figure 4.3. For simplicity, we only consider the spatial characteristics of the application such as the grid size for each phase switch. However, we do not change the temporal characteristics of the application, although our approach can support both types of adaptations (spatial and temporal). Furthermore, we assume the area is divided evenly between silt and sand (each represents 50% of the computational domain). In reality, this distribution varies depending on the area being modeled. We compare the performance of PAP implementation with the algorithm that uses the finest grid resolution.

In our test case, the computation domain contains at least one node having a negative waterhead. The hydraulic conductivity is homogeneous at each sub-domain but it will change when the waterhead $\boldsymbol{h}$ changes. In the middle of the computation domain, there is a well that is injecting water at a rate of $\boldsymbol{2.5m^3/day}$ and the initial boundary condition is $\boldsymbol{-75.5}$ m at $\boldsymbol{X}$ direction and $\boldsymbol{0}$ at the $\boldsymbol{Y}$ direction. With the water injection, the water head of the nodes around the well will increase which implies that

the application execution goes through rapid phase changes between phases 4, 3, 2 as shown in Figure 4.3. Phase 3 is used to avoid abrupt change in spatial characteristics; therefore, it shall not consume a lot of time steps. In this case, Phase 3 takes 100 timesteps in order to show perceptible water head change. In real simulation, the time steps for this phase can be set to 2 or 4. Furthermore, because this phase is within a relative small interval around ***h'***, we can assume that silt and sand have the same hydraulic conductivity in phase 3, therefore we use 1:1 grid ratio for these two materials.

We execute the code without PAP with finest grid size and with PAP approach in which PARM chooses the optimal grid size. We execute the application for a total simulation period of 0.3 day.

The execution time of this application on different number of nodes is shown in Figure 5.3.

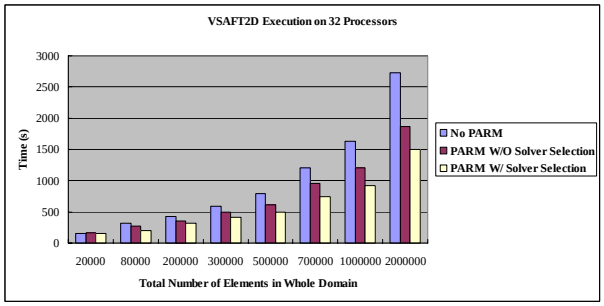


**Figure 5.3 VSAFT2D Execution Time on 32 Processors**

The error of using PAP versus the finest grid resolution through all the application phases is shown in Table 5.3. The error is calculated as the absolute error and the relative error as shown in Equations (5.1) and (5.2).

$$\textit{Absolute error} = \max(|h_i' - h_i|) \quad (5.1)$$

$$\textit{Relative error} = \max(|h_i' - h_i|/h_i) \quad (5.2)$$

where $h'_i$ is the PAP calculated value at node $i$ and $h_i$ is the finest grid calculation at node $i$. $i = 0 \ldots N$; ($N$ is the total number of nodes)

**Table 5.3 Absolute and Relative Error on Diffusion and Seismic Case at Different Time Steps for Problem Size 20,000**

| Saturated | $200\Delta t$ | $400\Delta t$ |
|---|---|---|
| Absolute error | 0.1925 | 0.1928 |
| Relative error(%) | 1.68 | 1.79 |

| Unsaturated | $200\Delta t$ | $300\Delta t$ | $500\Delta t$ | $600\Delta t$ |
|---|---|---|---|---|
| Absolute error | 0.0139 | 0.0652 | 0.1659 | 0.1843 |
| Relative error (%) | 0.019 | 0.091 | 0.23 | 0.26 |

Though we reduce the execution time effectively, we also introduce additional overhead into the code which is the extra time spent to reform the grid. Tables 5.4 shows the overheads for different problem sizes versus the total execution with 1 processor for that problem size with only ten time steps. (It's not very likely a phase

will change within 10 time steps.)

**Table 5.4 PAP Overhead**

**(Unit: Time in seconds, Problem Size in Million Nodes)**

| Problem size | 0.32 | 0.72 | 1.28 | 2.0 | 2.88 |
|---|---|---|---|---|---|
| Overhead | 6.03 | 7.87 | 11.02 | 15.02 | 20.62 |
| Total Execution Time | 155.2 | 360.0 | 615.1 | 1003.8 | 1445.7 |
| Percentile | 3.89% | 2.19% | 1.79% | 1.49% | 1.42% |

We can see that as the problem size increases, the overhead introduced by PAP approach will become insignificant as compared to the execution time. The total overhead introduced by PAP is small enough and can be neglected.

# CHAPTER 6 AUTONOMIC RUNTIME PARTITIONING STRATEGIES FOR STRUCTURED ADAPTIVE MESH REFINEMENT APPLICATIONS

## 6.1 Introduction

Dynamically adaptive simulations based on structured adaptive mesh refinement (SAMR) techniques can yield highly advantageous ratios for cost/accuracy when compared to methods based on static uniform approximations, and can be effectively used to enable large-scale, physically realistic scientific and engineering simulations. SAMR techniques start with a coarse base grid with minimum acceptable resolution that covers the entire computational domain. As the simulation progresses, regions in the domain requiring additional resolution are identified and are dynamically refined. Parallel/distributed implementations of SAMR applications lead to interesting research problems in dynamic resource allocation, data-distribution and load balancing, and communication and coordination. Furthermore, the underlying computing environment is dynamic and heterogeneous in nature. As a result, configuring, managing, and optimizing the execution of dynamic SAMR applications to exploit the underlying computational power of the heterogeneous computing environment remains a significant challenge.

In this chapter, we present a runtime partitioning strategies based on performance prediction functions to optimize the performance of SAMR applications in distributed and dynamic computing environments. In these environments, application performance may be severely degraded due to changes in the available CPU, network load, and memory resources. The performance prediction functions presented in this chapter are experimentally formulated [89] in terms of the current system state parameters and estimate the expected performance of a particular application distribution, given the current system state. Though CPU load, available memory, and bandwidth are the primary system parameters considered here, the same strategies can apply to other parameters such as cache size etc., once the performance function is obtained. The partitioning strategies then use the performance functions to identify a redistribution of the application domain that addresses changes in system state and available resources, and maximizes performance. These partitioning strategies form a part of the Physics Aware Runtime Manager (PARM) which enables self-managing, self-adapting, and self-optimizing SAMR applications. Experimental evaluation of the partitioning schemes using the 3-D adaptive Richtmyer-Meshkov compressible fluid dynamics kernel (RM3D) for different system configurations and workloads demonstrates the improvement in overall runtime performance.

We have presented our PARM system in Section 4.5. PARM is able to reactively manage and optimize scientific application execution using current system and

application state, online predictive models for system behavior and application performance. Recall that PARM includes Monitor Engine, Planning Engine, Configuration Engine and Parallel Algorithm Libraries as shown in Figure 6.1.

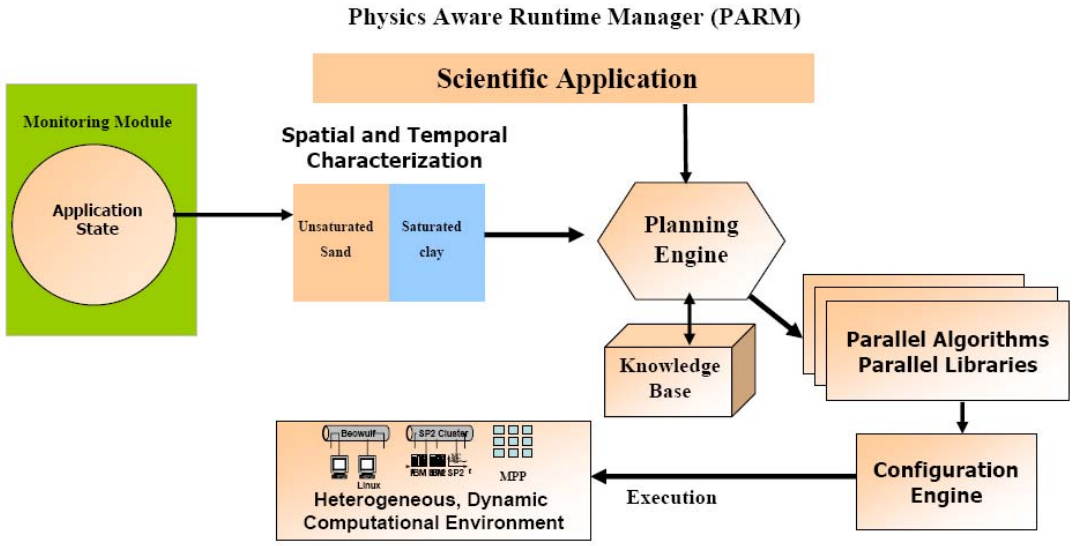


**Figure 6.1 Physics Aware Runtime Manager Architecture**

In addition to identifying the computational characteristics and the physics properties of the current application execution phase, the Monitor Engine component within the PARM is also responsible for detecting conditions under which the parameters affecting the application execution deviate from their acceptable behavior or operation. For example, application performance may degrade severely due to increased computational and/or network load, low available memory, or due to software or hardware failures. The characterization of current state is then used to drive the predictive performance functions and models that can estimate its performance in

the near future. The Planning Engine is responsible for describing the behavior of a system component, subsystem or compound system through Performance Functions, developed in previous research [89]. The Planning Engine determines the appropriate application reconfiguration strategy and the resources required to repartition work and optimize SAMR performance. The Configuration Engine will enforce the reconfiguration strategy and restart the SAMR computation with new partitioning scheme.

The work presented here is built on our earlier research efforts on application-sensitive partitioning [90, 92], system-sensitive partitioning [94], Pragma infrastructure [93], and adaptive runtime management [89, 91].

## 6.2 SAMR Partitioning Strategies

Distributed SAMR applications are CPU-intensive, memory-intensive, and bandwidth-intensive programs. The application performance may degrade severely due to increased CPU and/or network loads, and with reduced available memory. To optimize application performance, the Planning Engines uses current system parameters obtained using Monitoring Engine, current application state, and performance functions to repartition the entire application domain among processors during runtime. The overall model is as follows. Suppose that the work is to be distributed among $\boldsymbol{K}$ processors.

$$\sum_{i=0}^{K} CR_i = 1 \tag{6.1}$$

where $\boldsymbol{CR_i}$ represents the combined work partitioning ratio of processor $\boldsymbol{i}$ defined as follows. The work $\boldsymbol{W_i}$ assigned to the $\boldsymbol{i}$th processor can be computed as

$$W_i = CR_i \times W \tag{6.2}$$

where $\boldsymbol{W}$ is the total work. The combined work partitioning ratio can now be computed using system information, i.e. CPU load, available memory and network load.

$$CR_i = w_c R_i^C + w_m R_i^M + w_b R_i^B \tag{6.3}$$

where $R_i^C$, $R_i^M$ and $R_i^B$ are work partitioning ratio based on CPU load, available memory, and link bandwidth, respectively. $\boldsymbol{w_c}$, $\boldsymbol{w_m}$ and $\boldsymbol{w_b}$ describe the weights reflecting how important CPU, memory, and bandwidth are and $\boldsymbol{w_c} + \boldsymbol{w_m} + \boldsymbol{w_b} = 1$. Note that

$$\sum_{i=0}^{K} R_i^C = \sum_{i=0}^{K} R_i^M = \sum_{i=0}^{K} R_i^B = 1 \tag{6.4}$$

The application and system states on each processor are monitored at runtime. In the case of SAMR applications, application state is defined in terms of the current levels of refinement, the number, shape, and aspect ratio of the re-fined patches and the dynamism of the application [90, 92]. System state includes CPU availability, available memory, and link bandwidth. The Planning Engines uses application configuration, state information, and the appropriate performance function to calculate the combined work partitioning ratio $\boldsymbol{CR_i}$ for each processor at runtime.

These ratios are then used to redistribute the application domain among processors in subsequent time-steps.

The frequency for monitoring application and system state, recomputing system capacity, and repartitioning the application depends on the rate of system/application dynamics and SAMR adaptation. There are three extreme cases in equation (6.2), which lead to three different strategies for work partitioning.

- *CPU-based runtime partitioning:* When $\boldsymbol{w_c}$ is 1 and $\boldsymbol{w_m}$, $\boldsymbol{w_b}$ are 0, $\boldsymbol{CR_i}$ becomes $R_i^C$. This strategy partitions the work among processors based on their CPU load status.
- *Memory-based runtime partitioning:* When $\boldsymbol{w_m}$ is 1 and $\boldsymbol{w_c}$, $\boldsymbol{w_b}$ are 0, $\boldsymbol{CR_i}$ becomes $R_i^M$. This strategy partitions the work among processors based on their available memory.
- *Bandwidth-based runtime partitioning:* When $\boldsymbol{w_b}$ is 1 and $\boldsymbol{w_c}$, $\boldsymbol{w_m}$ are 0, $\boldsymbol{CR_i}$ becomes $R_i^B$. This strategy partitions the work among processor based on their link bandwidth.

In this dissertation, we will focus on CPU-based runtime partitioning and Bandwidth-based runtime partitioning schemes. The Memory-based runtime partitioning scheme is discussed in detail in [115].

### 6.2.1 CPU-based Runtime Partitioning

The performance of parallel SAMR applications in timeshared systems may degrade due to multi-programming. It has been observed that the execution time of a computation intensive program linearly increases with the number of jobs sharing the same processor. To optimize the SAMR application performance, our CPU-based partitioning strategy uses the CPU current load and application state to repartition the work among processors.

#### 6.2.1.1 CPU Performance Function Model

Performance Functions (PF) describes the behavior of system or application in terms of changes in one or more of its attributes. We can characterize the relationship between the execution time of an SAMR application and its attributes-application work and refinement level as a time performance function $\boldsymbol{T = PF_c(W, LV)}$. By using the performance function, we can estimate the time to finish work $\boldsymbol{W}$ at refinement level $\boldsymbol{LV}$ for an SAMR application on one processor if this processor is dedicated to it. If this processor is shared with other applications, the SAMR application would experience longer delay. Let $\boldsymbol{L_i}$ be the load index for processor $\boldsymbol{i}$, which is represented by the length of the CPU waiting queue. In such a multiprogramming case, the execution time of the SAMR application can be estimated as a function of the CPU load, application work and refinement level as follows:

$$T_i = T \times L_i = PF_c(W_i, LV_i) \times L_i \qquad (6.5)$$

The time performance function for RM3D is empirically defined as follows:

$$\begin{aligned} T &= PF_c(W, LV) \\ &= a_0 + a_1 W + a_2 LV + a_3 W \times LV + a_4 W^2 \\ &+ a_5 LV^2 + a_6 W^2 \times LV + a_7 W \times LV^2 + a_8 W^2 \times LV^2 \end{aligned} \tag{6.6}$$

Where $a_i$ is heuristic coefficient derived from our previous research [91].

### 6.2.1.2 Work Partitioning Algorithm

The average execution times of an SAMR application on all processors can be estimated as follows.

$$T_{avg} = \frac{\sum_{i=0}^{K} T_i}{K} \tag{6.7}$$

where $T_i = PF_c(W_i, LV_i) \times L_i$.

To improve the application performance, the execution time should be as equal as possible on all processors. In doing so, the work partitioning ratio of each processor is adjusted at runtime such that their execution time during the next time steps will be identical within an acceptable tolerance. The adjustment factor associated with each processor is defined as,

$$f_i(t) = \frac{T_{avg}(t)}{T_i(t)} \tag{6.8}$$

where $T_i(t)$ is the estimated execution time for processor $i$ at time step $t$; $T_{avg}(t)$ is the average estimated execution time for all processors at time step $t$.

Once the adjustment factor is determined, we can compute CPU-based work partitioning ratio of processor ***i*** for the next time step as follows:

$$R_i^C(t+1) = R_i^C(t) \times f_i(t) \tag{6.9}$$

To make sure that the sum of partitioning ratios on all processors is equal to 1, the new ratio is normalized as follows:

$$R_i^C(t+1)' = \frac{R_i^C(t+1)}{\sum_{i=0}^{K} R_i^C(t+1)} \tag{6.10}$$

## 6.2.2 Bandwidth-based Runtime Partitioning

When the network is heavily loaded, the application performance will be degraded when the processor interconnection experiences different speed. Evenly distributed workload among processors as we discussed in previous section might not be the optimal solution because the slowest processor will delay all the other processors in communication phase. For RM3D application, every processor needs to exchange the boundary information with other processors. One observation is that the number of messages generated by one processor is proportional to its workload. For the processor with a slow link, it should be assigned a light load to reduce the messages it generates. The processor with a faster link is capable of handling more message exchanges and thus it can be assigned more computation load. We present a new algorithm that considers this scenario.

Our methodology is based on a Master-Slave paradigm. Processor ***0*** is the master and all the other processors are slaves. Processor ***0*** is responsible to collect all the real-time system and application information from all the slaves before every regridding phase. Processor ***0*** also is responsible for generating work partitioning ratio. Every processor *k* reads system file from Linux operation system, calculate the packet transferring time, and send this value to the master processor $\boldsymbol{P_0}$. For every processor, this time is denoted as $t_k$ . The average packet transferred time is:

$$t_{avg} = \frac{\sum_{k=0}^{K} t_k}{K} \tag{6.11}$$

If the packet transferring time $t_k$ of processor ***k*** is larger than $t_{avg}$, it means the processor ***k*** has a busier link to other processors at the current iteration. We need to reduce the number of messages generated by ***k*** to alleviate its communication cost in the next iteration. On the other hand, if packet transferring time $t_j$ of processor ***j*** is smaller than $t_{avg}$, it means processor ***j*** does not have a busy link to other processors, and then it can handle more messages in the next iteration. As we mentioned above, more message generation is proportional to the workload assigned to the processor, by changing the workload, we can change the number of messages generated by processors.

Base on this observation of RM3D application, we obtain a performance function that describes the relationship between local workload ***W*** of every processor and the

number of message generated.

$$num\_msg_k = PF_{load}(W_k) = a + b \times W_k \quad (k = 0\ldots K) \tag{6.12}$$

$a = 15172.918$
$b = 0.51997046$

This performance function is obtained based on numerous experimental data sets generated by the RM3D on our test bed.

In order to predict the optimal number of messages generated for the next iteration, we calculate it based on each processor's packet transferring time.

$$num\_msg_k' = \frac{t_{avg}}{t_k} \times num\_msg_k \quad (k = 0\ldots K) \tag{6.13}$$

- if $t_k < t_{avg}$, i.e. processor ***k***'s network is lightly loaded; in the next iteration, it can handle more messages.
- if $t_k > t_{avg}$, i.e. processor ***k***'s network is heavily loaded; in the next iteration, we need to reduce the number of messages generated by processor ***k***.

We use the same data sets used in CPU-based runtime partitioning to predict the local workload $W_k'$ on processor ***k*** in the next iteration based on the predicted number of messages.

$$W_k' = PF_{msg}(num\_msg_k') = a + b \times num\_msg_k' \tag{6.14}$$

$a = 13353.751$

$b = 0.90871189$

One naïve way to get the $PF_{msg}$ is to inverse the $PF_{load}$. But because $PF_{load}$ has

some data that is not entirely fitting with the curve, the inversion will introduce more random error into $PF_{msg}$. This is why we are using the same data set to re-generate the performance function instead of inversing the $PF_{load}$.

Once we calculate the predicted local workload for every processor, the master processor ***0*** can generate the work partition ratio $R_k$. In order to ensure the total of work partition ratio is 1, we use the formula below:

$$R_k = \frac{W_k'}{\sum_{k=1}^{K} W_k'} \tag{6.15}$$

We can describe the algorithm using pseudo-code as:

```
while(not finished){
  if (regridding time){
    if (myid == 0) {
      collect  t_k from slaves;
      calculate  t_avg ;
      collect  num_msg_k from slaves;
      num_msg_k' = (t_avg / t_k) × num_msg_k ;
      W_k' = PF_msg(num_msg_k') = a + b × num_msg_k' ;
      R_k = W_k' / (sum_{k=1}^{K} W_k') ;
      write  R_k into a file CAP;
    }
    else{
      collect system information to get t_k ;
      send  t_k  to processor 0;
      collect application information to get  W_k ;
      num_msg_k = PF_load(W_k) = a + b × W_k ;
      send  num_msg_k to processor 0;
    }
      every processor reads from CAP to get its work partitioning
      ratio for next iteration;
  }
}
```

## 6.3 Experimental Results

The autonomic runtime partitioning system has been integrated into the GrACE (Grid Adaptive Computational Engine) data management framework for parallel/distributed SAMR applications. The autonomic runtime partitioning strategies are evaluated

using the RM3D CFD kernel on Beowulf clusters at Rutgers University and University of Arizona. We establish three scenarios to evaluate our runtime partitioning approaches.

- *Lightly loaded scenario*: Among the processors executing the RM3D application, 75% processors are lightly loaded and the other 25% are heavily loaded.
- *Moderately loaded scenario*: Among the processors executing the RM3D application, 50% processors are lightly loaded and the other 50% are heavily loaded.
- *Heavily loaded scenario*: Among the processors executing the RM3D application, 25% processors are lightly loaded and the other 75% are heavily loaded.

### 6.3.1 CPU-based Partitioning

In this subsection, we quantify the performance gain that can be achieved by using the CPU-based partitioning approach to adapt to CPU load dynamics. A synthetic program is used to control the CPU load dynamics among the processors and establish the three different load scenarios discussed previously. We compare the performance of the RM3D application with and without the CPU-based runtime partitioning approach.

Figure 6.2 presents CPU load situation on 16 processors and the corresponding work assignment of the RM3D application with and without CPU load adaptation under the moderately loaded scenario. The CPU load is measured as the length of the CPU waiting queue and is monitored by system monitoring tool at runtime. The work is the size of computation point set of the RM3D application. Without CPU load adaptation, the work is assigned almost evenly among the processors. However, by using CPU partitioning algorithm, the work is assigned to processors based on its CPU load status. Tables 6.1, 6.2, 6.3, and 6.4 present the performance gain of RM3D with CPU-based partitioning strategy for different base sizes and different number of processors under different load scenarios.

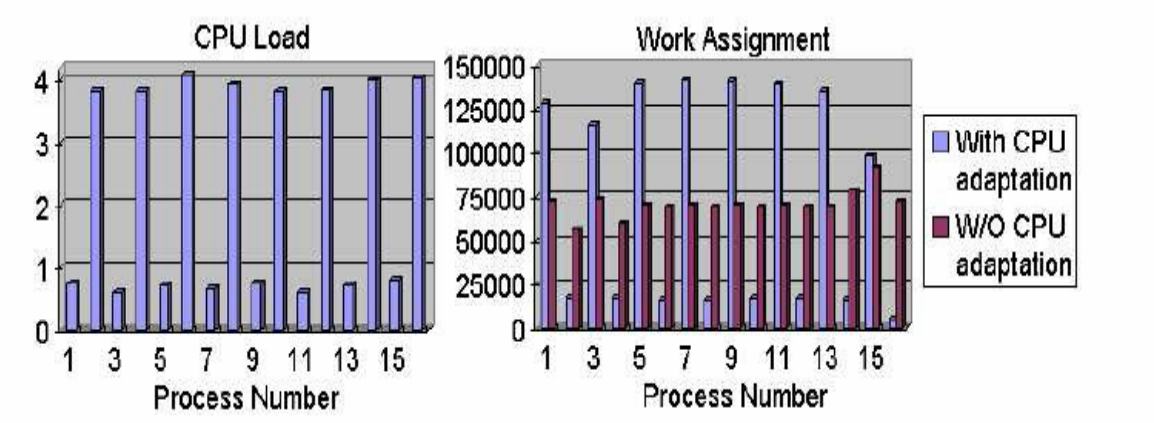


**Figure 6.2 Moderately Loaded Scenario on 16 Processors: CPU Load Distribution (left) and Work Assignment for 16 Processors (right)**

**Table 6.1. CPU-based Partitioning Performance Gain on 8 Processors**

**(Problem Size: 64×16×16)**

| Scenarios | Execution time W/O CPU adaptation (seconds) | Execution time W/ CPU adaptation (seconds) | Percentage Improvement |
|---|---|---|---|
| Lightly loaded | 2618.2 | 1560.87 | 40.38% |
| Moderately loaded | 2706.84 | 1964.87 | 27.41% |
| Heavily loaded | 2727.51 | 2127.32 | 22% |

**Table 6.2. CPU-based Partitioning Performance Gain on 16 Processors**

**(Problem Size: 64×16×16)**

| Scenarios | Execution time W/O CPU adaptation (seconds) | Execution time W/ CPU adaptation (seconds) | Percentage Improvement |
|---|---|---|---|
| Lightly loaded | 2126.06 | 727.17 | 65.8% |
| Moderately loaded | 2301.15 | 1641.73 | 28.66% |
| Heavily loaded | 2378.25 | 1624.15 | 31.71% |

**Table 6.3. CPU-based Partitioning Performance Gain on 32 Processors**

**(Problem Size: 128×32×32)**

| Scenarios | Execution time W/O CPU adaptation (seconds) | Execution time W/ CPU adaptation (seconds) | Percentage Improvement |
|---|---|---|---|
| Lightly loaded | 4908.79 | 2901.1 | 40.9% |
| Moderately loaded | 4976.78 | 3378.65 | 31.35% |
| Heavily loaded | 5170.52 | 4140.56 | 20.45% |

**Table 6.4. CPU-based Partitioning Performance Gain on 64 Processors**

**(Problem Size: 128×32×32)**

| Scenarios | Execution time W/O CPU adaptation (seconds) | Execution time W/ CPU adaptation (seconds) | Percentage Improvement |
|---|---|---|---|
| Lightly loaded | 4904.78 | 2827.29 | 42.36% |
| Moderately loaded | 5049.72 | 3281.31 | 31.35% |
| Heavily loaded | 5119.89 | 3587.89 | 29.92% |

The above results show that RM3D experiences longer delays when some processors are heavily loaded. Without the CPU-based partitioning algorithm, the application work is partitioned equally among the processors regardless of their CPU load status that leads to a longer application execution time. However with our CPU-based runtime partitioning strategy the application work is partitioned according to the processors' CPU load status and the performance of RM3D can be significantly improved. For example, in Table 6.1, for lightly loaded scenario we can obtain 40% percentage improvement. The results also demonstrate that better performance can be achieved under lightly and moderately loaded scenarios. The reason is that under heavily loaded scenario most processors are heavily loaded and there are not enough lightly loaded processors to accept more work. Furthermore, better performance can be obtained with large number of processors. For example, in Table 6.2, 65.8% percentage improvement is obtained on 16 processors under lightly loaded scenario with the same base grid size of 64×16×16 as shown in Table 6.1. With more processors, the work

assigned to each processor would be reduced.

### 6.3.2 Bandwidth-based Partitioning

In this subsection, we evaluate experimentally the performance gain by taking processor bandwidth into consideration. The following scenarios are used to evaluate our approach.

- Lightly busy scenario: Among the processors on which RM3D application is running, 75% processors are lightly busy, i.e. they don't have intensive network activities and 25% are busy, i.e. they have intensive network activities.

- Moderately busy scenario: Among the processors on which RM3D application is running, 50% processors are lightly busy, i.e. they don't have intensive network activities and 50% are busy, i.e. they have intensive network activities.

- Heavily busy scenario: Among the processors on which RM3D application is running, 25% processors are lightly busy, i.e. they don't have intensive network activities and 75% are busy, i.e. they have intensive network activities.

Our test is based on Linux cluster with two different problem sizes. For RM3D without our approach, every processor is assigned an equal work partition ratio in every regridding time. While for RM3D with our algorithm, every processor initially is assigned equal work partition ratio but this value is changed before every regridding phase according to its communication status based on our algorithm. Tables 6.5 and 6.6 show the results of RM3D execution with and without considering the processor link status on 8 processors.

**Table 6.5: Bandwidth-based Partitioning Performance Gain on 8 Processors**

**(Problem Size: 64×16×16)**

| Scenarios | Execution time W/O bandwidth adaptation (seconds) | Execution time W/ bandwidth adaptation (seconds) | Percentage Improvement |
|---|---|---|---|
| Lightly busy | 1339.58 | 858.817 | 35.89% |
| Moderately busy | 1578.9 | 907.048 | 42.55% |
| Heavily busy | 1653.85 | 1107.78 | 33.02% |

**Table 6.6: Bandwidth-based Partitioning Performance Gain on 8 Processors**

**(Problem Size: 128×32×32)**

| Scenarios | Execution time W/O bandwidth adaptation (seconds) | Execution time W/ bandwidth adaptation (seconds) | Percentage Improvement |
|---|---|---|---|
| Lightly busy | 6096.64 | 3441.57 | 43.55% |
| Moderately busy | 6590.13 | 3972.17 | 39.73% |
| Heavily busy | 7436.6 | 6216.25 | 16.4% |

From above data, we can see that the application will experience slower execution time when the network is busy that lead to more communication and synchronization delay. If we didn't change the work partitioning ratio, the processor that is experiencing heavy loaded network will be the bottleneck. By using our algorithm, we improve the performance of the application execution by adaptively changing the work partitioning ratio to every processor to make it compatible with current load on its communication links.

We can also observe from the above results that the heavily loaded network link gives the worst improvement ratio and the performance of the problem size of 64×16×16 is better than the performance of the problem size of 128×32×32. This is because the problem size of 64×16×16 has less workload than 128×32×32, then the former problem size has less boundary areas to exchange between processors compared to the latter one. Therefore 64×16×16 will experience less impact from the busy network link. Lightly loaded and moderately load network link have better improvement than heavily loaded ones; because in our algorithm, if 25% or 50% processors are busy, there are still lightly loaded processors that can receive extra workloads. Once the network is heavily loaded, in our case, for example, 75% of the processors are busy, our algorithm will not be effective because there are not enough lightly loaded processors that can be assigned more workload.

Figures 6.3, 6.4 and 6.5 show the work assignment on different processor bandwidth using bandwidth-based partitioning approach for three different scenarios.

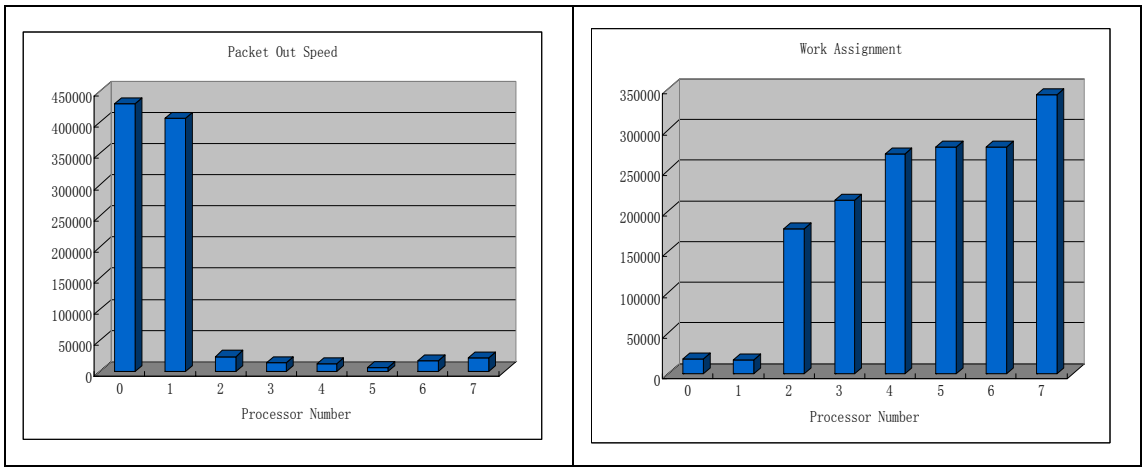


**Figure 6.3 Lightly Busy Scenario on 8 Processors: Bandwidth (left) and Work Assignment for Bandwidth-based Partitioning Algorithm (right)**

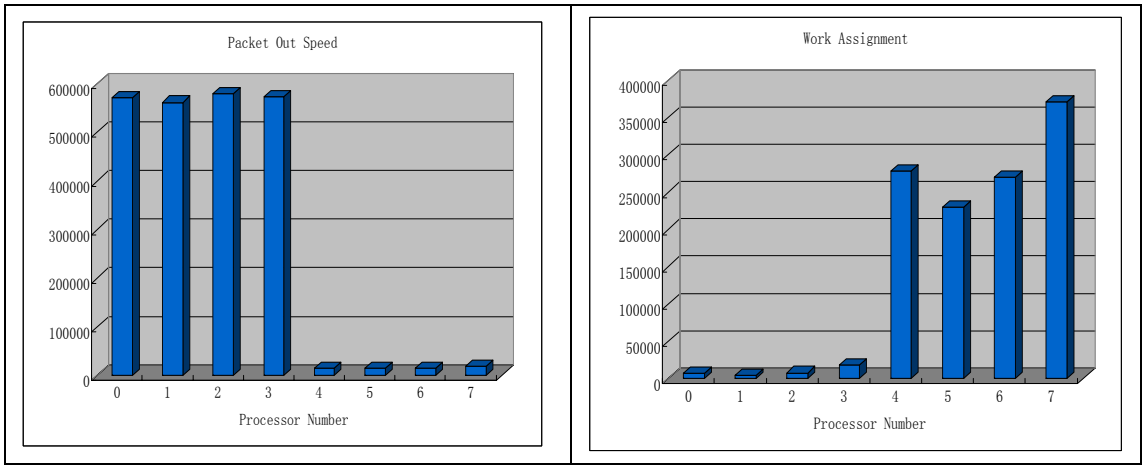


**Figure 6.4 Moderately Busy Scenario on 8 Processors: Bandwidth (left) and Work Assignment for Bandwidth-based Partitioning Algorithm (right)**

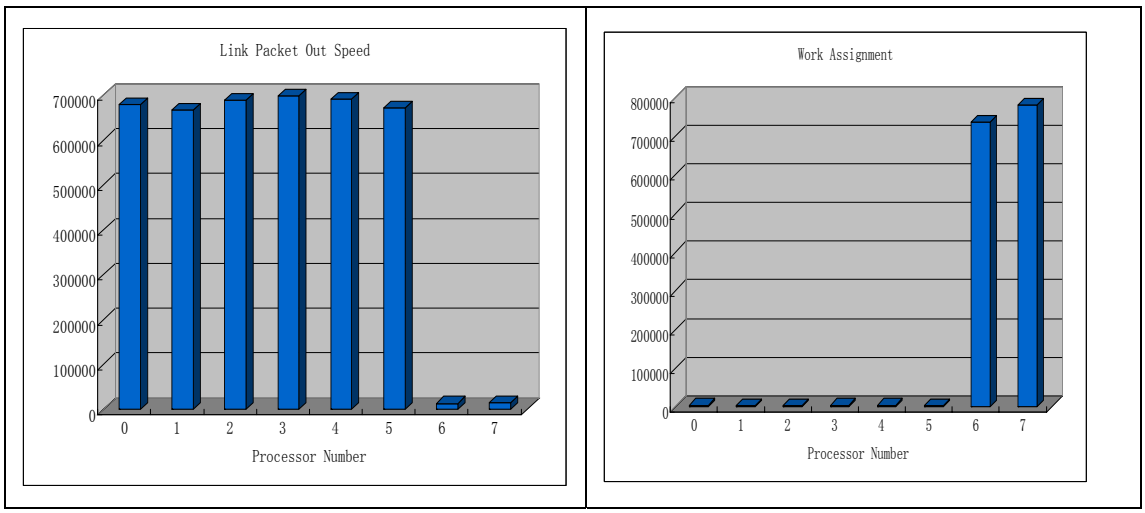


**Figure 6.5 Heavily Busy Scenario on 8 Processors: Bandwidth (left) and Work Assignment for Bandwidth-based Partitioning Algorithm (right)**

## 6.4 Conclusions and Future Work

This chapter aims at showing that autonomic runtime partitioning strategies can improve the overall performance of SAMR applications in a dynamic and heterogeneous computational environment. To ensure the efficient execution of SAMR applications, these strategies use current system states and application states to appropriately partition the application work among processors at runtime. Based on different system parameters, we presented CPU-based and bandwidth-based runtime partitioning strategies.

# CHAPTER 7 CONCLUSION AND FUTURE WORKS

## 7.1 Summary

The research presented in this dissertation is part of a research effort to develop an autonomic control and management environment (AUTONOMIA) that provide self-configuration, self-optimization, self-healing and self-protecting in a distributed computing environment [116].

In this dissertation, we developed a theoretical framework and general analysis methodology for Physics Aware Programming (PAP) paradigm and its runtime system that we refer to as a Physics Aware Runtime Management (PARM) system. PAP enables programmers to identify the appropriate application solution methods via exploiting the dynamism and heterogeneity of the application execution states as well as the physical properties associated with each execution phase. PARM performs the following tasks: (a) analyzing current physics state of the application; (2) identifying application execution state changes and identifying the corresponding optimizing polices for each state; (c) using a policy engine to identify the best numerical algorithm for the application based on its current physics state; and (d) identifying the current system state and using performance functions to improve the overall performance.

The main research activities of this dissertation can be highlighted in the following aspects:

(1) We have studied important numerical methods that are commonly used in scientific computing applications. Using the error analysis and stability analysis, we studied the techniques to change the temporal and spatial characteristics of the application and how they improve performance. The analysis also helps determine the change of the application's numerical algorithm and the system resource consumption.

(2) We have studied the techniques to monitor the system resources and its relationship to the application execution. We studied autonomic runtime partitioning strategies for SAMR applications on a distributed and dynamic execution environment, such as the computational Grid [117]. To optimize the performance of SAMR applications, our autonomic practitioner uses current system parameters (CPU load and bandwidth), current application states and predictive performance functions to partition the whole application work among the processors at runtime. Specifically we studied two types of partitioning strategies: CPU-based and Bandwidth-based partitioning strategy. We compare the performance of RM3D application with and without our autonomic runtime partitioning algorithms under different system load scenarios. Our experimental results demonstrated that our CPU-based strategy achieved

up to 65.8% performance improvement and better performance can be obtained with a larger number of processors. For the bandwidth-based strategy, better performance can be achieved under the lightly and moderately loaded scenarios and up to 43% performance gain can be obtained.

## 7.2 Contributions

The contributions of this dissertation are the following:

1. Introduced a new novel physics aware programming paradigm to characterize the application's temporal and spatial characteristics based on its physical properties and use this knowledge to optimize application performance.
2. Developed a novel Autonomic Runtime System that can efficiently implement applications developed based on our PAP paradigm.
3. Developed a Knowledge Repository that identifies the best matrix solver and preconditioner to solve a given problem.

## 7.3 Future Research

Our future research will focus on the following aspects:

(1) In order to facilitate programming applications with PAP paradigm, the following functions need to be developed:

***pap_init***(struct p_property *pp, char *files) → Initiating the PAP applications. struct p_property is a structure defined to store the physics properties essential on spatial and temporal characteristics. Files are the input files used by the code to determine the initial values of p_property.

***pap_grid***(double **global, double *rhs) → this function generates the computational grid. If it is necessary, before the grid is formed, the code will be executed for a few iterations to decide the order of upper bound of $u_{tt}$.

***pap_solve***(double **global, double *rhs, KSP solver) → this function passes in the global matrix of the linear system and the right hand side of the system. The solver is type of KSP which is Krylov subspace solvers. This function will extract the matrix format and search the knowledge base to select a solver fitting for the matrix type.

***pap_update***(double **global, double *rhs) → this function is used to update knowledge base.

***pap_check***(struct p_property *pp) → this function checks the variation of the physics property pp. A gain will be calculated if the grid will be changed.

***pap_regrid***(double *global, double *rhs) → this function is used to regrid the

application when the physics property has changed. On the coarse to fine grids, interpolation will be used and for the fine to coarse grids, numerical values are copied according to their physical locations.

***pap_finalize***() → this function releases the resources.

(2) More numerical methods and algorithms will be incorporated into the Knowledge Base of PARM. The search space for the optimal solution will become very big. We need to propose a better heuristic method to find an optimal solution for a particular application proposed by the user in a reasonable time limit. A possible solution will be based on Data Mining techniques.

(3) We have shown that by using current system states and application states based on different system parameter strategies such as CPU-based and bandwidth-based partitioning strategies, a significant performance gain can be achieved. A hybrid strategy obtained by combining CPU-based and bandwidth-based partitioning strategies (i.e., both CPU load information and bandwidth information will be used to decide the partitioning strategies) might deliver a better performance than using CPU-based or bandwidth-based runtime partitioning strategies alone. The CPU load and bandwidth information will be weighted in the hybrid strategy and the weights associated with CPU and bandwidth may affect the overall effectiveness of the hybrid strategy. In our future research, we will address the weight issue and how they affect the performance of the hybrid strategy.

(4) Further research will be conducted on phase identification when an unknown application is fed into PARM environment. Since PARM uses Knowledge Base to identify each execution phase of the application as well as to select the best algorithm for the application, PARM will have difficulty to identify the application states and choose the optimal algorithm from the parallel algorithm library if it is an unknown application. One possible solution to this problem is using application's "signature" to identify the application's states. The "signature" of an application is an abstraction of the application's execution. For example, if the application experiences 3 phases: reading I/O, computing, and intensive memory access, its signature will be "I/O intensive", "CPU intensive" and "memory intensive". The Knowledge Base will be built with different "signatures". When an unknown application is fed into PARM, the application will be executed using a default signature. Once the application is running, the PARM can map the current behavior of the application into one of existing signatures on its Knowledge Base. Once the application is mapped into our existing signatures, the approaches (solvers/algorithms) will be chosen for the application execution.

# REFERENCES

[1] Jeff Bilmes, Krste Asanovic, Chee-Whye Chin, Jim Demmel, “Optimizing Matrix Multiply Using PHiPAC: A Portable, High-Performance ACSI C Coding Methodology,” In Proceedings of the International Conference on Supercomputing, Vienna, Austria, July 1997.

[2] R. C. Whaley, A. Petitet, and J. Dongarra, “Automated Empirical Optimizations of Software and the ATLAS Project,” Parallel Computing, 27(1-2): 3-35, January 2001.

[3] Zizhong Chen, Jack Dongarra, Piotr Luszczek, and Kenneth Roche, “Self Adapting Software for Numerical Linear Algebra and LAPACK for Clusters,” Parallel Computing, Volume 29(11-12):pp. 1723–1743.

[4] Jack Dongarra, Victor Eijkhout, “Self-adapting numerical software for next generation applications,” Lapack Working Note 157, ICL-UT-02-07

[5] R. Whaley and J. Dongarra, “Automatically Tuned Linear Algebra Software,” Technical Report UT CS-97-366, LAPACK Working Note No.131, University of Tennessee, 1997

[6] C. L. Lawson, R. J. Hanson, D. Kincaid, and F. T. Krogh, “Basic Linear Algebra Subprograms for FORTRAN usage,” ACM Trans. Mathematical Software, 5 (1979), pp. 308--323.

[7] http://www.netlib.org/lapack/lug/index.html

[8] Gerhard Zumbusch, “Parallel Multilevel Methods: Adaptive Mesh Refinement and

Loadbalancing," Teubner, 2003.

[9] J. H. Ferziger, "Numerical Methods for Engineering Application," John Wiley & Sons, Inc. 1998.

[10] Benjamin A. Allan , "The CCA Core Specification in a Distributed Memory SPMD Framework," Concurrency Computat. 2002;14:1-23.

[11] J. D. Istok, "Groundwater Modeling by the Finite Element Method (Water Resources Monograph)," Amer Geophysical Union, 1989.

[12] Satish Balay, Kris Buschelman, Victor Eijkhout, William D. Gropp, Dinesh Kaushik, Matthew G. Knepley, Lois Curfman McInnes, Barry F. Smith and Hong Zhang, "PETSc Users Manual," ANL-95/11 - Revision 2.1.5, Argonne National Laboratory, 2004

[13] Y. Zhang, X. S. Li and O. Marques, "Towards an Automatic and Application-Based Eigensolver Selection," LACSI Symposium 2005.

[14] Xiaoye S. Li and James W. Demmel, "SuperLU_DIST: A Scalable Distributed-Memory Sparse Direct Solver for Unsymmetric Linear Systems," ACM Trans. Mathematical Software, 2003.

[15] Srivastava, R., and T.-C.J. Yeh, "A three-dimensional numerical model for water flow and transport of chemically reactive solute through porous media under variably saturated conditions," Advances in Water Resources, Vol. 15, p 275-287, 1992.

[16] http://www.osc.edu/hpc/hardware/index.shtml#beo

[17] George Hornberger and Patricia Wiberg, "Numerical Methods in the Hydrological

Sciences", American Geophysical Union, 2005.

[18] O. C. Zienkiewicz, The Finite Element Method, 3rd.Ed. MacGRAW-HILL Book Company (UK) Ltd. 1977.

[19] P. Bochev, M. Gunzburger and R. Lehoucq, "On stabilized finite element methods for transient problems with varying time scales", European Congress on Computational Methods in Applied Sciences and Engineering, ECCOMAS 2004.

[20] S. Liu, M. Rodgers, Q. Wang and L. Keer, "A Fast and Effective Method for Transient Thermoelastic Displacement Analyses", Journal of Tribology -- July 2001 -- Volume 123, Issue 3, pp. 479-485.

[21] Touheed, N; Selwood, P; Jimack, P K; Berzins and P. M. Dew, Parallel dynamic load-balancing for the solution of transient CFD problems using adaptive tetrahedral meshes in: Emerson D R, Ecer, A, Periaux, J, Satofuka, N & Fox, P (editors) Parallel Computational Fluid Dynamics - Recent Developments and Advances Using Parallel Computers, pp. 81-88 Elsevier Science. 1998.

[22] Goodyer, C; Fairlie, R; Berzins, M; Scales, L. An in-depth investigation of the multigrid approach for steady and transient EHL problems in: Dowson, D (editors) Thinning Films and Tribological Interfaces - Proceedings of the 26th Leeds-Lyon Symposium, pp. 95-102 Elsevier Science. 1999.

[23] Amdahl, G. M., "Validity of the single-processor approach to achieving large scale computing capabilities", AFIPS Conference Proceedings, Vol. 30 pp. 483-485, 1967.

[24] Bosilca, G., Chen, Z., Dongarra, J., Eijkhout, V., Fagg, G., Fuentes, E., Langou, J.,

Luszczek, P., Pjesivac-Grbovic, J., Seymour, K., You, H., Vadhiyar, S. "Self Adapting Numerical Software SANS Effort," IBM Journal of Research and Development (to appear), June, 2005.

[25] T. F. CHAN AND T. MATHEW, Domain decomposition algorithms, Acta Numerica, (1994), pp. 61-144.

[26] http://math.nist.gov/MatrixMarket/

[27] R. Barrett, M. Berry, T. F. Chan, J. Demmel, J. M. Donato, J. Dongarra, V. Eijkhout, R. Pozo, C. Romine and H, Van Der Vorst, Templates for the Solution of Linear Systems: Building Blocks for Iterative Methods, Society for Industrial and Applied Mathematics, Philadelphia, 1994.

[28] G. Verdu, D. Ginestar, V. Vidal, and J. L. Munoz Cobo. 3D λ-Modes of the Neutron-Diffusion Equation. Ann. Nucl. Energy, 21:405-421.

[29] F. Y. Cheng. Matrix Analysis of Structural Dynamics: Applications and Earthquake Engineering. Marcel Dekker, New York, N.Y., 2000.

[30] A. K. Chopra. Dynamics of Structures: Theory and Applications to Earthquake Engineering. Prentice Hall, Upper Saddle River, N. J., 2nd. edition, 2000.

[31] M. Turk and A. Pentland. Eigenfaces for Recognition. Technical report, Vision and Modeling Group, The Media Laboratory, MIT, 1990.

[32] Aslak Tveito, Ragnar Winther, Introduction to Partial Differential Equations, A Computational Approach, Springer, 1998.

[33] W. Press, S. Teukolsky, W. Vetterling and B. Flannery, Numerical Recipes in Fortran 77,

Cambridge University Press, 1992.

[34] Frank L. Stasa, Applied Finite Element Analysis for Engineers, Holt, Rinehart and Winston, 1985.

[35] George Coulouris, Jean Dollimore, Tim Kindberg, Distributed Systems Concepts and Design, 4th. Ed. Addison-Wesley Publishing Co., 2005.

[36] Ian Foster, Designing and Building Parallel Programs, Addison-Wesley Publishing Co., 1995.

[37] P. Liu and S. Bhatt, Experiences with Parallel N-body Simulation, Parallel and Distributed Systems, IEEE Transactions on, 2000.

[38] D. Blackston and T. Suel, Highly Portable and Efficient Implementations of Parallel Adaptive N-Body Methods, SC'97, 1997.

[39] M. S. Warren and J. K. Salmon, Astrophysical N-body Simulations Using Hierarchical Tree Data Structures, Conference on High Performance Networking and Computing, Proceedings of the 1992 ACM/IEEE conference on Supercomputing Minneapolis, Minnesota, United States, Pages: 570 - 576.

[40] John Dubinski, A parallel Tree Code, New Astronomy 1 (1996) 133-147.

[41] K. Li, ``Managing divisible load on partitionable networks," in High Performance Computing Systems and Applications, J. Schaeffer ed., pp. 217-228, Kluwer Academic Publishers, Boston, Massachusetts, 1998.

[42] K. Li, X. Lin, Y. Sun, and P.Y.S. Cheung, ``Divisible load distribution on hypercubes," Proceedings of Tenth International Conference on Parallel and Distributed Computing

and Systems, pp. 421-426, Las Vegas, Nevada, October 28-31, 1998.

[43] Hans Sagan, Space-Filling Curves, Springer-Verlag, 1994.

[44] M. Parashar and J. Browne: On Partitioning Dynamic Adaptive Grid Hierarchies, 29th. Annual Hawaii International Conference on System Sciences, pp. 604-613.

[45] H. Samet. The design and analysis of spatial data structure. Addison-Wesley, Reading, Massachusetts, 1989.

[46] Johan Steensland, Michael Thuné, Sumir Chandra, Manish Parashar, Characterization of Domain-Based Partitioners for Parallel SAMR Applications, PDCS, 425-430,Nov. 2000.

[47] M. Berger and P. Colella, Local adaptive mesh refinement for shock hydrodynamics, In Journal of Computational Physics, 82(1): 64-84, May 1989.

[48] G. Cybenko, Dynamic load balancing for distributed memory multi-processors, In IEEE Transactions on Parallel and Distributed computing, 7:279-301, October 1989.

[49] K. Dragon and J. Gustafson, A low-cost hypercube load balance algorithm, In Proc. Fourth Conf. Hypercubes, Concurrent Comput. and Appl. Pages 583-590, 1989.

[50] F. Lin and R. Keller, The gradient model load balancing methods, In IEEE Transactions on Software Engineering, 13(1): 8-12, January 1987.

[51] G. Horton, A multilevel diffusion method for dynamic load balancing. In Parallel Computing, (19): 209-218, 1993.

[52] L. Oliker and R. Biswas. PLUM: parallel load balancing for adaptive refined meshes. In Journal of Parallel and Distributed Computing, 47(2): 109-124, 1997.

[53] A. Sohn and H. Simon, Jove: A dynamic load balancing framework for adaptive computations on an sp-2 distributed multiprocessor, In NJIT CIS Technical Report, New Jersey, 1994.

[54] C. Walshaw, Jostle: partitioning of unstructured meshes for massively parallel machines. In Proc. Parallel CFD'94, 1994.

[55] M. Willebeek-LeMair and A. Reeves, Strategies for dynamic load balancing on highly parallel computers, In IEEE Transactions on Parallel and Distributed Systems, 4(9): 979-993, September 1993.

[56] Z Lan, V Taylor, G Bryan, Dynamic Load Balancing for Structured Adaptive Mesh Refinement Applications, Proc. of ICPP, 2001.

[57] S. Bhavsar, M. Shee and M. Parashar, An application-centric characterization of distribution techniques for dynamic adaptive grid hierarchies, PDPTA'99 Las Vegas, Nevada, U. S. A. June 28 – July 1, 1999.

[58] G. Bryan, Fluid in the universe: Adaptive mesh refinement in cosmology, In Computing in Science and Engineering, 1(2): 46-53, March/April, 1999.

[59] Zhiling Lan, Valerie E. Taylor and Yawei Li, DistDLB: Improving cosmology SAMR simulations on distributed computing systems through hierarchical load balancing, J. Parallel Distrib. Comput. 66 (2006) 716-731.

[60] A. Corradi, L. Leonardi and F. Zambonelli, Diffusive load-balancing policies for dynamic applications, In IEEE Concurrency, pages 22-31, January-March 1999.

[61] J. Chen and V. Taylor, Parapart: parallel mesh partitioning tool for distributed systems,

Concurrency: Practice Experience 12 (2000) 111-123.

[62] J. Chen and V. Taylor, Part: mesh partitioning for efficient use of distributed systems, IEEE Trans. Parallel Distributed Systems 12(2000) 111-123.

[63] Z. Lan, V. Taylor and G. Bryan, A novel dynamic load balancing scheme for parallel systems, J. Parallel Distributed Comput. (2002) 1763—1781.

[64] Jim Basney, Miron Livny, Paolo Mazzanti, Utilizing Widely Distributed Computational Resources Efficiently with Execution Domains, J. Computer Physics Communications 2000.

[65] Salim Hariri, C.S. Raghavendra, Yonhee Kim, Muhamad Djunaedi, Rinda P. Nellipudi, Ashok Rajagopalan, Prasad Vadlamani, Yeliang Zhang. CATALINA: A Smart Application Control and Management. Active Middleware Services Conference, 2000.

[66] H. Versteeg (Author), W. Malalasekra, An Introduction to Computational Fluid Dynamics: The Finite Volume Method, 2nd. Ed. Prentice Hall, 2007.

[67] W. Gropp, E. Lusk, T. Sterling and J. Hall, Beowulf cluster computing with Linux, 2nd. Edition, The MIT Press, 2003.

[68] A. Date, Introduction to computational fluid dynamics, Cambridge University Press, 2005.

[69] T. J. Chung, Computational fluid dynamics, Cambridge University Press, 1st. Edition, 2002.

[70] J. Ferziger and M. Peric, Computational methods for fluid dynamics, Springer, 3rd. Edition, 2001.

[71] R. Wangsness, Electromagnetic fields, 2nd. Edition, Wiley, 1986.

[72] D. Cook, The theory of the electromagnetic field, Dover Publication, 2003.

[73] Niederreiter, H. and Spanier, J. (editor): Monte Carlo and Quasi-Monte Carlo Methods 1998. Springer-Verlag, New York, 2000.

[74] W.L. Briggs, "A multigrid tutorial" , SIAM (1987).

[75] H. D. Simon. Partitioning of unstructured problems for parallel processing. Computing Systems in Engineering, 2:135--148, 1991.

[76] A. Pothen, H. Simon and K.-P. Liou, Partitioning sparse matrices with eigenvectors of graphs. SIAM J. Mat. Anal. Appl., 11(3):430-452, 1990.

[77] James D. Teresco, Jamal Faik, and Joseph E. Flaherty, Hierarchical Partitioning and Dynamic Load Balancing for Scientific Computation Proc. Workshop on State-Of-The-Art in Scientific Computing: 7th International Conference, PARA 2004, Lyngby, Denmark, June 20-23, 2004.

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

[79] K. D. Devine, B. A. Hendrickson, E. Boman, M. St. John and C. Vaughan. Zoltan: A dynamic load balancing library for parallel applications; User's Guide. Sandia National Laboratories, Albuquerque, NM, 1999. Tech. Report SAND 99-1377.

[80] C. L. Bottasso, J. E. Flaherty, C. Ozturan, M. S. Shephard, B. K. Szymanski, J.

D. Teresco and L. H. Ziantz, The quality of partitions produced by an iterative load balancer, Proc. Third Workshop on Languages, Compilers, and Runtime Systems, pages 265-277, Troy, 1996.

[81] Kirk Schloegel, George Karypis, and Vipin Kumar, Parallel static and dynamic multi-constraint graph partitioning.Concurrency and Computation: Practice and Experience. Volume 14, Issue 3, pages 219 - 240, 2002.

[82] Kirk Schloegel, George Karypis, and Vipin Kumar, A Unified Algorithm for Load-balancing Adaptive Scientific Simulations, Supercomputing, 2000.

[83] L. Kale and S. Krishnan, CHARM++: A portable concurrent object oriented system based on C++. In Proceedings of OOPSLA 1993, pages 91-108, 1993.

[84] B. Maerten, D. Roose, A. Basermann, J. Fingberg, G. Lonsdale, DRAMA: A library for parallel dynamic load balancing of finite element applications, Proceedings of the Ninth SIAM Conference on Parallel Processing for Scientific Computing, San Antonio, March 2224, 1999.

[85] B. Maerten, A. Basermann, J. Fingberg, G. Lonsdale and D. Roose, Parallel dynamic mesh re-partitioning in FEM codes, Advances in Computational Mechanics with High Performance Computing (Proceedings of the 2nd. Euro-Conference on parallel and distributed computing for computational mechanics, B.H.V. Topping Ed.), Saxe-Coburg, 1998, pp. 163-167.

[86] E. Luque, A. Ripoll, A. Cortes and T. Margalef, A distributed diffusion method for dynamic load balancing on parallel computers. In IEEE CS Press, editor,

Proc. of EUROMICRO Workshop on Parallel and Distributed Processing, San Remo, Italy, January 1995.

[87] R. Biswas, L. Oliker and A. Sohn, Global load balancing with parallel mesh adaptation on distributed-memory systems, Conference on High Performance Networking and Computing, Proceedings of the 1996 ACM/IEEE conference on Supercomputing, Pittsburgh, Pennsylvania, United States, 1996

[88] K. Yotov, X. Li, G. Ren, M. gibulskis, G. DeJong, M. Garzaran, D. Padua, K. Pingali, P. Stodghill, and P. Wu, A comparison of empirical and model-driven optimization, Proceedings of the ACM SIGPLAN Conference on Programming Language Design and Implementation, 2003, pp. 63-76.

[89] H. Zhu, M. Parashar, J. Yang, Y. Zhang, S. Rao and S. Hariri, Self adapting, self optimizing runtime management of grid applications using PRAGMA, Proc. Of NSF NGS Program Workshop, IEEE/ACM 17th. IPDPS, Nice, France, CDROM, 7 pages, April 2003.

[90] S. Chandra and M. Parashar. “ARMaDA: An Adaptive Application-Sensitive Partitioning Framework for Structured Adaptive Mesh Refinement Applications”, Proc. Of Parallel and Distributed Computing Systems (PDCS 02), Cambridge, MA, pp. 446–451, November 2002.

[91] S. Chandra, S. Sinha, M. Parashar, Y. Zhang, J. Yang, and S. Hariri. “Adaptive Runtime Management of SAMR Applications”, Proc. of High Performance Computing (HiPC 02), LNCS, India, Vol. 2552, pp. 564–574, December 2002.

[92] S. Chandra, J. Steensland, M. Parashar, and J. Cummings. “An Experimental Study of Adaptive Application Sensitive Partitioning Strategies for SAMR Applications”, Proc. Of 2nd LACSI Symposium (best poster SC’01), October 2001.

[93] M. Parashar and S. Hariri. “PRAGMA: An Infrastructure for Runtime Management of Grid Applications”, Proc. of NSF NGS Program Workshop, IEEE/ACM IPDPS, Fort Lauderdale, FL, CDROM, 8 pages, April 2002.

[94] S. Sinha and M. Parashar. “Adaptive Runtime Partitioning of AMR Applications on Heterogeneous Clusters”, Proc. Of Cluster Computing, Newport Beach, CA, IEEE Computer Society Press, pp. 435–442, October 2001.

[95] J. C. Tannehill, D. A. Anderson, R. H. Pletcher. Computational fluid mechanics and heat transfer, 2nd. Edition, Taylor & Francis.

[96] L. Eldén. Numerical Solution of the Sideways Heat Equation. In Inverse Problems in Diffusion Processes. Proceedings in Applied Mathematics, ed. H. Engl and W. Rundell, SIAM, Philadelphia 1995.

[97] Stanley J. Farlow, Partial Differential Equations for Scientists and Engineers, Dover Publications, Inc. 1993.

[98] Paul Duchateau and David Zachmann, Applied Partial Differential Equations, Dover Publications, Inc. 1989.

[99] Phillip Bording and Larry Lines, Seismic Modeling and Imaging - Making Waves, Geophysical Corner column in AAPG Explorer December, 2000.

[100] Dheeraj Bhardwaj, Jeremy Cohen, Stephen McGough, Steven Newhouse, A Componentized Approach to Grid Enabling Seismic Wave Modeling Application, 5th International Conference on Parallel and Distributed Computing, Applications and Technologies (PDCAT04).

[101] Autonomic Nervous System, http://users.rcn.com/jkimball.ma.ultranet/BiologyPages/P/PNS.html#autonomi

[102] W. Ross Ashby, Design for a brain (Second Edition Revised 1960), published by Chapman & Hall Ltd, London.

[103] J.O. Kephart and D.M. Chess, The vision of autonomic computing, IEEE Computer 36(1) (2003) 41–503.

[104] P. Horn. Autonomic Computing: IBM's Perspective on the State of Information Technology. http://www.research.ibm.com/autonomic/, October 2001.

[105] IBM Corporation. Autonomic computing concepts, http://www3.ibm.com/autonomic/library.shtml , 2001.

[106] IBM An architectural blueprint for autonomic computing, April 2003.

[107] Albert Tarantola, Inverse problem theory and methods for model parameter estimation, Society for Industrial and Applied Mathematics, Philadelphia, 2005.

[108] Manish Parashar and Salim Hariri, Autonomic Computing, concepts, infrastructure, and applications, CRC Press 2007.

[109] G. W. Stewart, Matrix Algorithms volume I: Basic decompositions. Society for

Industrial and Applied Mathematics, Philadelphia, 1998.

[110] Alvarado, E; D. Sandberg and S. Pickford 1998. Modeling large forest fires as extreme events. Northern Science, 72: 66-75

[111] Clarke, David A., M. L. Norman, and R. A. Fiedler, Zeus-3D User Manual, LCA Technical Report no. 15, 1994.

[112] Stone, James M., and M. Norman, Astrophysical Journal Supplement Series, vol. 80, pp. 753-818, 1992.

[113] Jingmei Yang, Huoping Chen, Salim Hariri and Manish Parashar, Self-optimization of large scale wildfire simulations, ICCS 2005, LNCS 3514, pp. 615-622, 2005.

[114] Marsha J. Berger and Joseph Oliger, Adaptive mesh refinement for hyperbolic partial differential equations, Journal of computational physics 53, 484-512 (1984).

[115] Yeliang Zhang, Jingmei Yang, Salim Hariri, Sumir Chandra, Manish Parashar. Autonomic Proactive Runtime Partitioning Strategies for SAMR Applications. IPDPS Next Generation Software Program - NSFNGS - PI Workshop 2004

[116] S. Hariri, B. Khargharia, H. Chen, Y. Zhang, B. Kim, H. Liu and M. Parashar, The Autonomic Computing Paradigm, to appear in the Cluster Computing Journal, 2006.

[117] Ian Foster and Carl Kesselman, The Grid 2: blueprint for a new computing infrastructure (The Elsevier Series in Grid Computing), Morgan Kaufmann, 2ed. 2003.