

A Physics Aware Programming Paradigm

by Salim Hariri and Yeliang Zhang

Large scale scientific applications generally experience different execution phases at runtime and each phase has different computational, communication and storage requirements as well as different physical characteristics. 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. A new programming methodology, A novel new approach to develop and implement applications based on autonomic computing principles is critically needed. Autonomic Programming (AP) paradigm enables application developers to identify the appropriate solution methods to exploit the heterogeneity and the dynamism of the application execution states. Once an application is developed based on AP paradigm, an Autonomic Runtime Manager (ARM) can then 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, ARM will 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 this programming paradigm using a real world application (Variable Saturated Aquifer Flow and Transport (VSAFT2D)) commonly used in subsurface modeling. We evaluated the performance gain of the AP 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 ARM is negligible. We also show that the results using AP is as accurate as the numerical solutions that use fine grid resolution.

Motivation

In the domain of scientific computations, discretization of time and space is usually encountered in a large class of problems such as hydrology underground water study, Stokes problem, thermo-mechanical, Computational Fluid Dynamics (CFD) and Elastohydrodynamic Lubrication problems. Most of these problems are solved by dividing computation domain into small grids (represented by spatial characteristics D_x , D_y and D_z) and advancing the computation in a small time step (represented by temporal characteristics D_t) until the desired results obtained. However, most of the applications are generally time dependent and their volatile nature make them hard to be solved. As time evolves, these problems will evolve into different phases with different physical characteristics. For example, in wildfire simulation, for over fifty years, attempts have been made to understand and predict the behavior (intensity, propagation speed and direction, and modes of spread) of wildfires. However, the factors that determine wildfire behavior are complex; they include fuel characteristics and configurations, chemical reactions, balances between different modes of heat transfer, **topography, and fire/atmosphere interactions. These factors influence a fire's behavior over a wide range of time and spatial scales** while the dynamism of the problem and the complicated interactions between these factors make accurate wildfire simulation difficult. Multiple physical phases not only exist in wildfire simulation, it also exists in forefront astronomical research such as supernovae. The supernovae core-collapse problem being modeled is inherently multi-phased, heterogeneous (in time, space and computational complexity) and dynamic. It involves hydrodynamics, nuclear fusion and transport phases. Each simulation phase requires different computational models with computational resources and the transition from one phase to the next and the computational models applicable at each phase and their computational requirements are determined by criteria based on local state and known only at runtime.

Autonomic Programming (AP) Paradigm

Most of the current execution techniques of applications use one algorithm to implement all the phases of the application execution, but a static solution or numerical scheme for all the execution phase might not be ideal to handle the dynamism of the problem as discussed in Section 1. Some techniques use application errors to refine the solution as in Adaptive Mesh Refinement (SAMR) where the mesh size is refined to achieve a smaller error rate on particular computation domain. In this scheme, the error is used to drive the dynamic changes in the grid point size that might or might not optimize the application performance. In our AP approach, **we apply a novel programming paradigm that takes into consideration the current application's physical properties and derive** spatial and temporal characteristics from such properties at runtime. In this programming paradigm, the appropriate solution that can meet

the desired accuracy and improve performance will be determined for each application phase at runtime. This programming technique is general and can be applied to Finite Element Method, Domain Decomposition Method, and Finite Volume Approximations.

For example, let us consider a Variable Saturated Aquifer Flow and Transport (VSAFT2D) application kernel developed at The University of Arizona. The major computing step in this routine is matrix solving routine. For two different hydraulic media: Slit and Sand, the routine goes through several phases, where in some phases it is possible to enlarge D_x and D_y by 10 and reduce the time step by 10 without affecting the stability of the algorithm and its accuracy. Table 1 shows the performance gains that can be obtained when that is exploited in the AP paradigm. It is clear from this Table that several order of magnitudes can be obtained by just exploiting the physics properties of the applications and identifying the right solution for each phase.

To exploit this programming paradigm, an Autonomic Runtime Manager (ARM) is developed to determine the application execution phase by monitoring the application execution, identify the application phase changes by exploiting the knowledge about the application physics and how it behaves at runtime, and then use the appropriate numerical algorithm/solver for each detected phase during the application execution. For each execution phase of numerical application, different numerical schemes and solvers that can best exploit its physics characteristics and its state were chosen by a knowledge base. In wildfire simulation, we use AP to decompose the computational domain into several natural regions (e.g., burning, unburned and burned) at runtime according to wildfire phase. 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. By regularly monitoring and analyzing the state of the simulation and the phase transition and drive the runtime optimization through this information, we can achieve significant performance gains.

Table 1

	Computational Requirements	Science Driven Solution
Grid point selection	Assume that there are 500,000 Gridpoints along x and y . The computational requirement is $(500,000 \times 500,000) \times 1000 \times 1000 = 25 \times 10^{16}$ floating point operations 10 Gflops processor 25×10^6 seconds = 8 Years!	Since it's a saturated situation, K_s and K_c are constants and $K_c < K_s$. We can enlarge D_x and D_y by 10 and reduce the time step by 10. Then the computational requirement is $(50,000 \times 50,000) \times 100 \times 1000 = 25 \times 10^{13}$ 25×10^3 seconds = 4 Days!

The Monitor Engine monitors the application execution to identify the computational characteristics and the physical properties of the current application execution phase. In the example discussed in Section 2, the monitor engine will identify the heat diffusion acceleration changes and the heat conductivity changes. The application properties are then fed into the Planning Engine.

The Planning Engine will determine the optimal spatial (D_x , D_y , D_z) and temporal characterization (Dt) for the application solution while maintaining the desired accuracy of the solution. Consequently, the format of the linear system will be projected based on the numerical method the application uses. The Knowledge Base identifies the optimal solution for each execution phase based on analytical and historical data; for example, if an application is using Finite Difference implicit Method, the linear system will involve solving a tri-diagonal matrix, then Conjugate Gradient with block Jacobi preconditioner will be stored in the knowledge based as 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.

We evaluate the performance of the PAP approach with transient problem setting as shown in Figure 2. For simplicity, we only consider the spatial characteristics of the application such as determining the ideal grid size for each phase, although our approach can support both of adaptations types (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. In fact, the more heterogeneous the computational domain is, the more performance gain that can be achieved by using the PAP paradigm; traditional programming techniques suffer more degradation in performance because the adopted solution must satisfy all domains by choosing the most conservative solution. We compare the performance of PAP implementation of VSAFT2SD with the implementation that uses the finest grid resolution. We execute the code with finest grid size and with PAP approach in which PARM chooses dynamically the optimal grid size for each phase of the application execution that ran for a total simulation period of 0.3 day.

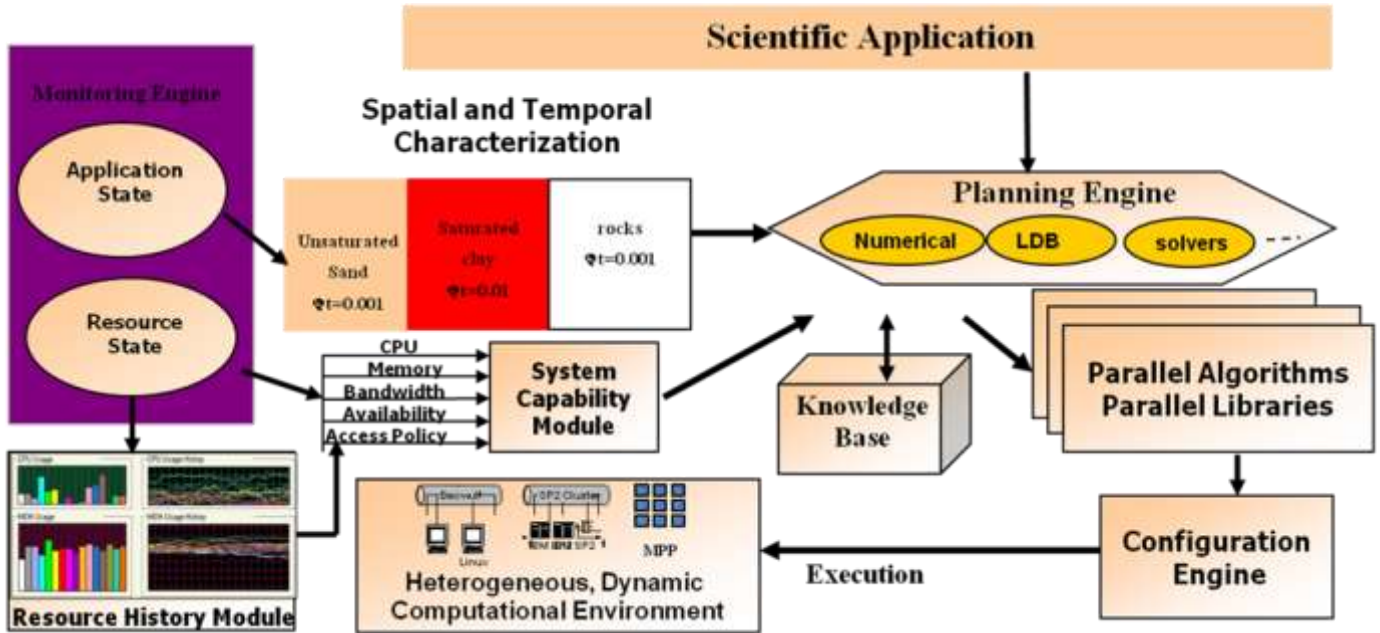


Figure 1: Autonomic Runtime Manager Architecture

The execution time of this application on different number of nodes is shown in Figure 2. For 2M nodes, PAP approach achieved an 81% performance gain when compared with the finest grid implementation.

Importance of the research problem

This research will formulate methodologies and develop infrastructures for building the next generation scientific and engineering simulations of complex physical phenomenon on widely distributed, highly heterogeneous and dynamic, networked computational environment. The simulations targeted by this effort will be built as dynamics compositions of autonomous components that integrate scalable distributed (and heterogeneous) computing with interactive control and computational steering, collaborative analysis, and scientific databases and data archives. Composing, configuring and managing the execution of these applications to exploit the underlying computational power in spite of its heterogeneity and dynamism will present significant challenges. Our programming paradigm, execution model, component frameworks and runtime infrastructures will help researchers better understand the operations and performance issues of applications, limitations of algorithms and architectures. It will also enable the development of “smarter” algorithms and applications that are capable of sensing the state of their environments and reacting to optimize overall execution, utilization and performance. Finally, this research will support the trend toward the next generation of an integrated software development life cycle that allows users to describe the requirements of the application components at each phase of its life cycle. These requirements can then be used by compilers and runtime systems to produce applications that are controllable, observable, and maintainable.

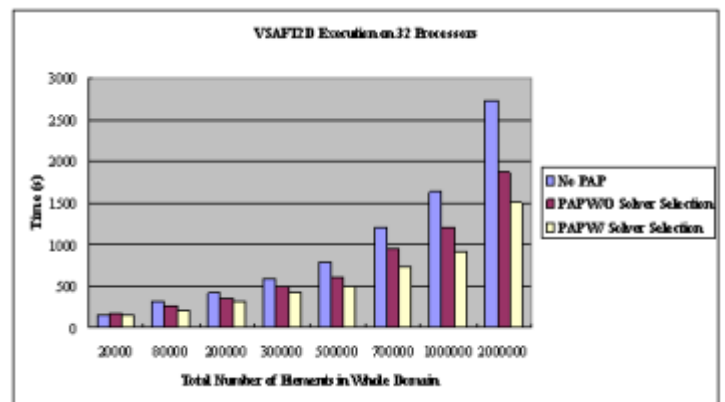


Figure 2: VSAFT2D Execution Time