POEMS: End-to-end Performance Design of Large Parallel Adaptive Computational Systems

Ewa Deelman* Aditya Dube** Adolfy Hoisie†† Yong Luo†† Richard L. Oliver[o]
David Sundaram-Stukel[oo] Harvey Wasserman††
Vikram S. Adve‡ Rajive Bagrodia* James C. Browne** Elias Houstis†
Olaf Lubeck†† John Rice† Patricia J. Teller[o] Mary K. Vernon[oo]

†Purdue University, ††Los Alamos National Laboratory, ‡Rice University,
**University of California at Los Angeles, **University of Texas at Austin,*
[o]University of Texas at El Paso, [oo]University of Wisconsin Madison

http://www.cs.utexas.edu/users/poems

Abstract

The POEMS project is creating an environment for end-to-end performance modeling of complex parallel and distributed systems, spanning the domains of application software, runtime and operating system software, and hardware architecture. To enable end-to-end modeling of large-scale applications and systems, the POEMS framework is designed to compose models of system components from these different domains, to integrate multiple modeling paradigms (analytical modeling, simulation, and actual system execution), and to allow different components to be modeled at multiple levels of detail. The key components of the POEMS framework include a generalized task graph model for describing parallel computations, automatic generation of the task graph by a parallelizing compiler, a specification language for mapping the computation on models for operating system and hardware components, a library of analytical and simulation models for components from the different domains, and a knowledge base describing the performance properties of widely-used algorithms. This paper provides an overview of the POEMS methodology and illustrates several key components of the POEMS framework. We illustrate the POEMS modeling capabilities for performance prediction of Sweep3D, a complex benchmark for evaluating wavefront application technologies and high performance parallel architectures.

1. Introduction

The goal of the Performance Oriented End-to-end Modeling System (POEMS) project is to create and experimentally evaluate a problem solving environment for end-to-end performance modeling of complex parallel/distributed systems, spanning application software, runtime and operating system software, and hardware architecture. The POEMS project combines innovations from communication models, data mediation, parallel programming, performance modeling, software engineering, and CAD/CAE to realize this goal.

The motivation for POEMS is that determining the end-to-end performance of large-scale computational systems across all stages of design enables more effective development of these complex software/hardware systems. POEMS, to the extent that we are successful, will enable a new generation of performance modeling technology. Multi-scale models allow different components of a system to be modeled at varying levels of detail via the use of associative-object-based module interfaces. Multi-domain models integrate components from three different semantic domains, namely application software, runtime/OS software, and hardware. Multi-paradigm models allow an analyst to use the model evaluation paradigm—analytical, simulation, or the software or hardware system itself —that is most appropriate for the goals of the performance study.

The POEMS project is building an initial library of models, at multiple levels of granularity, for modeling scalable architectures like those envisaged under the DOE ASCI program, and for modeling complex adaptive systems like those envisaged under the DARPA GloMo and Quorum programs. The project supports evaluation of component behaviors through analytical models and through application of discrete-event simulation at multiple levels of detail. The analytical models include deterministic task graph analysis [1], and the LogP [8] and LoPC models [9]. POEMS supports detailed simulation of a variety of state-of-the-art processors and memory hierarchies and incorporates parallel evaluation of discrete-event simulation models through application of the Maisie simulator [4]. The project is also building a knowledge base of performance data that can be used to estimate the performance properties of widely-used algorithms as a function of architectural characteristics.

POEMS development is being driven by modeling of the behavior of large-scale complex applications on parallel architectures. The first driver application is the Sweep3D program that is being used to evaluate advanced parallel architectures at Los Alamos National Laboratories.

This paper describes the POEMS conceptual framework and methodology (section 2), the Sweep3D application (section 3), the performance tools and initial library of models that are under development for Sweep3D (section 4), and some preliminary results from our multi-paradigm evaluation of Sweep3D (section 5). Section 6 discusses related work and the conclusions of the work are given in section 7.

2. POEMS Methodology

The major elements of the conceptual framework of POEMS are a general model of parallel computation and the use of associative objects as the representation basis for components. The models developed in POEMS span three domains: *application, operating system,* and *hardware*. The application domain represents parallel computation as a dynamic task graph where nodes represent sequential computation units and edges represent dependencies. The operating system domain provides the models for process and memory management, inter-process communication, and parallel file systems. The hardware domain provides models for the processor and memory components, where the latter includes models for cache memory as well as shared memory hierarchies.

In this section we describe the general model of parallel computation used in POEMS (section 2.1), automatic generation of the task graph models for parallel applications (section 2.2), the task execution descriptions that characterize the execution of components of the task graph on particular processors and memory hierarchies (section 2.3), the POEMS framework for compositional modeling (section 2.4), and the POEMS performance recommender (section 2.5).

2.1 General Model of Parallel Computation

The POEMS representation of parallel computations is based upon a generalized dependence graph model of parallel computation. A task level data flow representation of this model of computation is obtained by defining the nodes in the data flow system to have extended dependence relationships. Synchronous parallel execution is obtained by extending the data dependence relationships (and specifications of the firing rules for the nodes) to regard a clock as a unit of computation.

A concrete realization of this model of parallel computation is a dynamic hierarchical dependence graph where the nodes are instances of associative objects and the arcs represent flow of information from node to node. The graph is dynamic in that nodes and arcs can be instantiated during execution. The graph is hierarchical in that each node can itself be an interface to another generalized dependence graph and that a node in one domain may invoke nodes in both its own and implementing domains. The nodes may be defined in different domains as shown in figure 1.

A node executes when the computation reaches a state when its associated "firing rule" evaluates to true. Each arc has associated with it a data type specification, which is called a transaction.

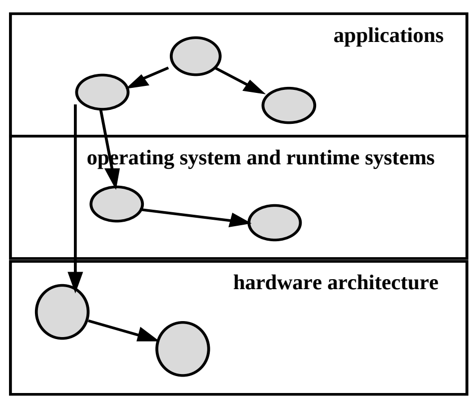

Figure 1. Multi-Domain Dependence Graph.

2.2 Automatic Task Graph Construction

The POEMS generalized task graph representation provides workload information at various levels of abstraction for both analytical and simulation models. Furthermore, the representation is designed to be computable *automatically* by a parallelizing compiler, but may be computed manually for explicitly parallel programs.

The application representation in POEMS is based on a combination of *static* and *dynamic* task graphs. The static task graph is a compact, symbolic graph representation of the parallel structure of the program, including the parallel tasks, precedence edges between tasks, and a subset of the control flow (including loops). It is independent of program input values and computed results. The dynamic task graph is a directed *acyclic* graph that captures the precise parallel structure of an execution of the application for a given input [1]. The dynamic task graph is important for supporting detailed and precise modeling of parallel program behavior. For programs where the tasks are statically scheduled on to the processes (as in most message-passing programs), the dynamic task graph can be represented very concisely by combining all the tasks allocated to a particular process between synchronization points into a single "task." For other cases, the dynamic graph is computed on demand (from the static task graph) during model evaluation.

Data-parallel compiler technology from the Rice University dHPF compiler project [3] will be used to compute the static task graph representation for key example programs, including High Performance Fortran (HPF) as well as manually parallelized programs. The key steps required to construct the static graph are selecting the units that will be represented as tasks, and identifying the precedences, communication and synchronization patterns among the tasks. There are some significant challenges to be faced in constructing the graph. First, computing a symbolic representation of the synchronization and communication patterns requires a powerful symbolic analysis capability, potentially greater than that required for parallelization alone. As one example, in the Sweep3D application described in section 3, eight different communication patterns (for the eight octants) are encoded in a single piece of code using three symbolic "direction" parameters, and these direction values are computed outside the sweep function. Second, the compiler must recognize and capture non-static behavior where the task graph may depend on intermediate computational results. For example, in Sweep3D, a different flux computation is used after the simulation has proceeded for a certain number of time steps. This requires alternative sets of tasks and precedences to be captured in the graph, along with the governing condition. Finally, in manually parallelized codes, the key additional challenge is to extract as much static information as possible about the communication and synchronization patterns in the program. In general, these might be obscured by SPMD control flow, procedure boundaries, and non-standard communication libraries. The dHPF compiler infrastructure provides a combination of symbolic analysis, interprocedural analysis, and communication analysis capabilities that will enable us to address these challenges.

Static and dynamic task graphs for the Sweep3D program are given in section 3.2. These graphs, constructed by hand, have aided in identifying the issues that need to be addressed in automating the task graph generation, as described above.

2.3 Task Execution Descriptions (TEDs)

The size of the multiprocessor systems and applications that will be modeled by POEMS requires that analytical modeling be used as much as possible and simulation be focused on the most important regions of the design space, and used to validate, support, and refine the analytical models.

Analytical modeling requires parameter values that are used to evaluate task execution time. Simulation of a single task's execution time (which will be used both for exploring detailed memory hierarchy performance and for computing parameter values for the analytical models) requires either an executable representation of the task or its memory address trace.

To meet these goals, POEMS will use a Task Execution Description (TED) to describe the modeled execution of a task. A TED is associated with each node of an application's task dependency graph. A TED contains the parametric definition of a hardware subsystem on which the task is to execute, which is comprised of processor, memory, and transport components as described in section 2.4. In addition, a TED contains the attributes required to define the method used to model single-task execution. The methods that are being evaluated for simulating individual tasks are instruction-driven, execution-driven, and trace-driven simulation. For enhanced throughput, performance database queries will be used to compute task execution times for specific configurations of hardware subsystems. The performance database will represent a sparse matrix of recorded task execution times computed from simulations and/or system measurements. These methodologies are described further in section 4.4.

2.4 Compositional Modeling Using Associative Objects

POEMS is developing a specification language and a programming environment for the general dependence graph (task graph) model of parallel computation. In this environment, the nodes of the graph are instances of associative objects which can be specialized to workload, software, and hardware components. Data mediation methods [24] are used to integrate the associative objects, thus enabling composition of multi-scale, multi-domain and multi-paradigm components into a coherent system model.

An associative object is a "standard object" encapsulated with an associative interface [19, 17, 7]. The key concepts are that objects are defined in the context of a semantic domain and the properties of objects are defined in terms of the attribute set of a semantic domain.

As an example, the POEMS hardware domain is comprised of three classes of component models. The processor component model utilizes processor component attributes to define the processor(s) to be simulated. The memory component model permits the specification of all types of memory components, such as cache, translation lookaside buffer (TLB), and main memory. The transport component model allows the specification of all the interfaces between or among multiple components in the system that send/receive data, such as interfaces from a processor to a level-one cache, from a level-one cache to a level-two cache, and from a processor to an interconnection network. Each component of a class (*e.g.*, each transport component) is semantically identical; there is one basic model with a set of associated attributes. A component is an instantiation of this basic model, having identical attributes, but possibly different values paired with these attributes.

An associative interface specifies all of the interactions in which a component can participate via two elements: an "accepts" interface for the services that it implements and a "requests" interface that specifies the services it requires. Interfaces are specified in terms of the attributes that define the behavior and the states of the standard objects. An object that has associative interfaces is said to be an associative object and an object that interacts through associative interfaces is said to have associative interactions. An associative interface is an extension of the associative model of communication [6] to define complex dynamic interactions among object instances.

An *accepts* interface consists of a *profile*, a transaction, and a protocol. A *profile* is a set of name/value pairs over the attributes of the domains. An object may change its *profile* during execution. A transaction is a type definition for a parameterized unit of work to be executed. A protocol specifies a sequence of elementary interactions. A *requests* interface consists of a *selector* expression, a transaction, and a protocol. A *selector* expression is a conditional expression over the attributes of the domains. The *selector* may reference attributes from the profiles of other objects. A selector is said to match a profile whenever it evaluates to "true" when evaluated using the attributes of that profile. A match that causes the *selector* to evaluate to "true" selects an object to receive a message or an invocation. The parameters in a profile and selector that match should either conform or the match should include a mechanism to translate the transaction in the requests interface instance to the transaction in the matched accepts interface and vice versa.

The firing rule for each node in the data flow graph includes one or more *selector*s that must evaluate to true for the rule to fire. The matching of *selectors* in *requests* to *profiles* in *accepts* thus both composes a dynamic data flow graph and controls the traversal of the graph that models the execution behavior of the system. The integration of associative objects into the generalized dependence graph model of parallel computation is illustrated in figure 2.

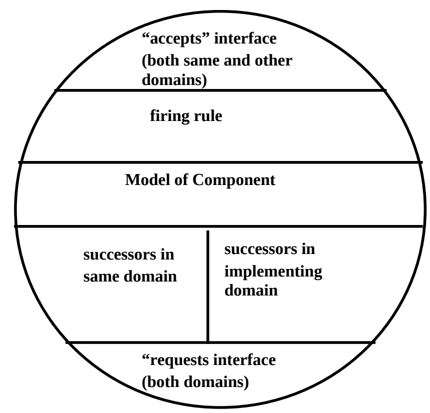

Figure 2. Schematic of a Node of a Dependence Graph as an Associative Object.

2.5 Performance Recommender

The POEMS Performance Recommender system facilitates the selection of computational parameters for widely used algorithms, to achieve specified performance goals. In the Sweep3D context, for example, it is used to obtain the parameters of the algorithm (*e.g.*, grid size, spacing, scattering order, angles, k-blocking factor, convergence test), system (*e.g.*, I/O switches), and machine (*e.g.*, number and configuration of the processors). This facility captures the results of system measurement as well as modeling studies (discussed in section 4), providing insight into how inter-relationships among variables and problem features affect the performance of applications. It functions at several levels ranging from the capture of analytical and simulation model results to those of the measured application.

We are using a kernel (IFESTOS), developed at Purdue [16], that supports the rapid prototyping of recommender systems. IFESTOS abstracts the architecture of a recommender system as a layered system with clearly defined subsystems for problem formulation, knowledge acquisition, performance modeling, and knowledge discovery. The designer of the recommender system first defines a database of application classes (problems) and computation class instances (methods). The data acquisition subsystem generates performance data by invoking the appropriate applications (*e.g.*, Sweep3D). The performance data management subsystem provides facilities for the selective editing, viewing, and manipulation of the information so generated. Performance analysis is performed by traditional attribute-value statistical techniques and "mining" this information produces useful rules that can be used to drive the actual recommender system. This approach has been demonstrated successfully for problem domains in numerical quadrature and elliptic partial differential equations [16]. Currently it is being applied to the Sweep3D application, described next.

3. POEMS Driver Application: Sweep3D

We consider the problem of predicting the performance of an ASCI kernel application called Sweep3D on large-scale parallel architectures such as the IBM SP/2 or the SGI Origin2000. This section contains a brief description of this application; section 4 discusses how the performance analysis tools in POEMS are being applied to Sweep3D.

3.1 Overview of Sweep3D

The Sweep3D kernel is a solver for the three-dimensional, time-independent, neutral particle transport equation on an orthogonal mesh [11]. The main part of the computation consists of a *balance* loop in which particle flux out of a cell in three Cartesian directions is updated based on the fluxes into that cell and other quantities such as local sources, cross section data, and geometric factors

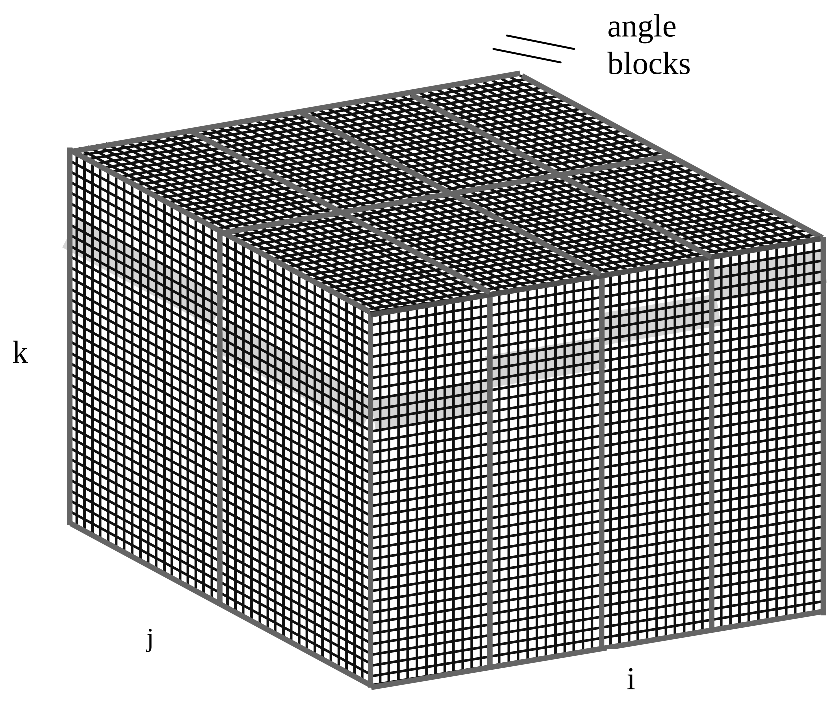

Figure 3. Mapping the Wavefronts of Sweep3D on a 2D Processor Grid.

The flux of a given cell cannot be computed until all of its upstream neighbors have been computed, implying a recursive or wavefront stru2cture. This structure is repeated along the eight diagonal directions through the cube (referred to as octants). Figure 3 shows how this 3D algorithm is partitioned on a 2D processor grid (i and j dimensions). A wavefront consists of a block of ‘k’ cells and a block of angle values for each octant. The wavefront in the figure is originating from the lower left corner of the processor grid (octant 7) and has three cells per k-block. The sweep in octant 7 consists of additional wavefronts for each of the angle blocks until the last k-block has been computed on the processor in the upper right.

3.2 Sweep3D Task Graph

The static and dynamic task graphs for the sweep phase of Sweep3D on a 3x3 processor grid are illustrated in figures 4 and 5. Figure 4 shows a hierarchical view of the static task graph. On the right, two of the eight octants in each time step of Sweep3D are displayed, and the computation and communication tasks performed by each processor for each octant are shown on the left. Figure 5 shows the dynamic task graph for two wavefronts *within* one octant. Communication tasks are circles numbered with a send/receive processor pair, and computation tasks are squares numbered by a computing processor. The critical path for the two wavefronts are the two long vertical sequences of tasks.

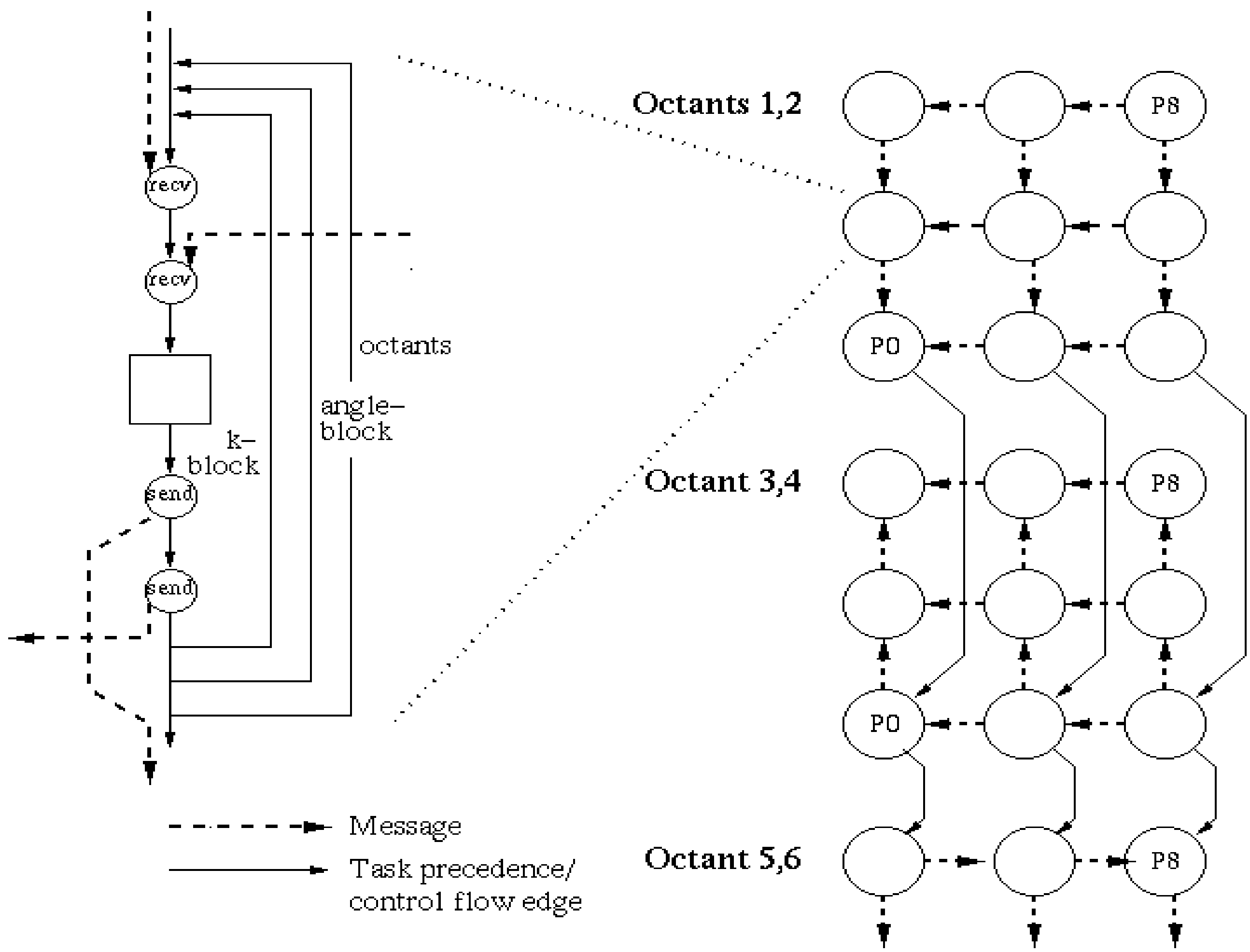

Figure 4. Static Task Graph for Sweep3D on a 3x3 Processor Grid.

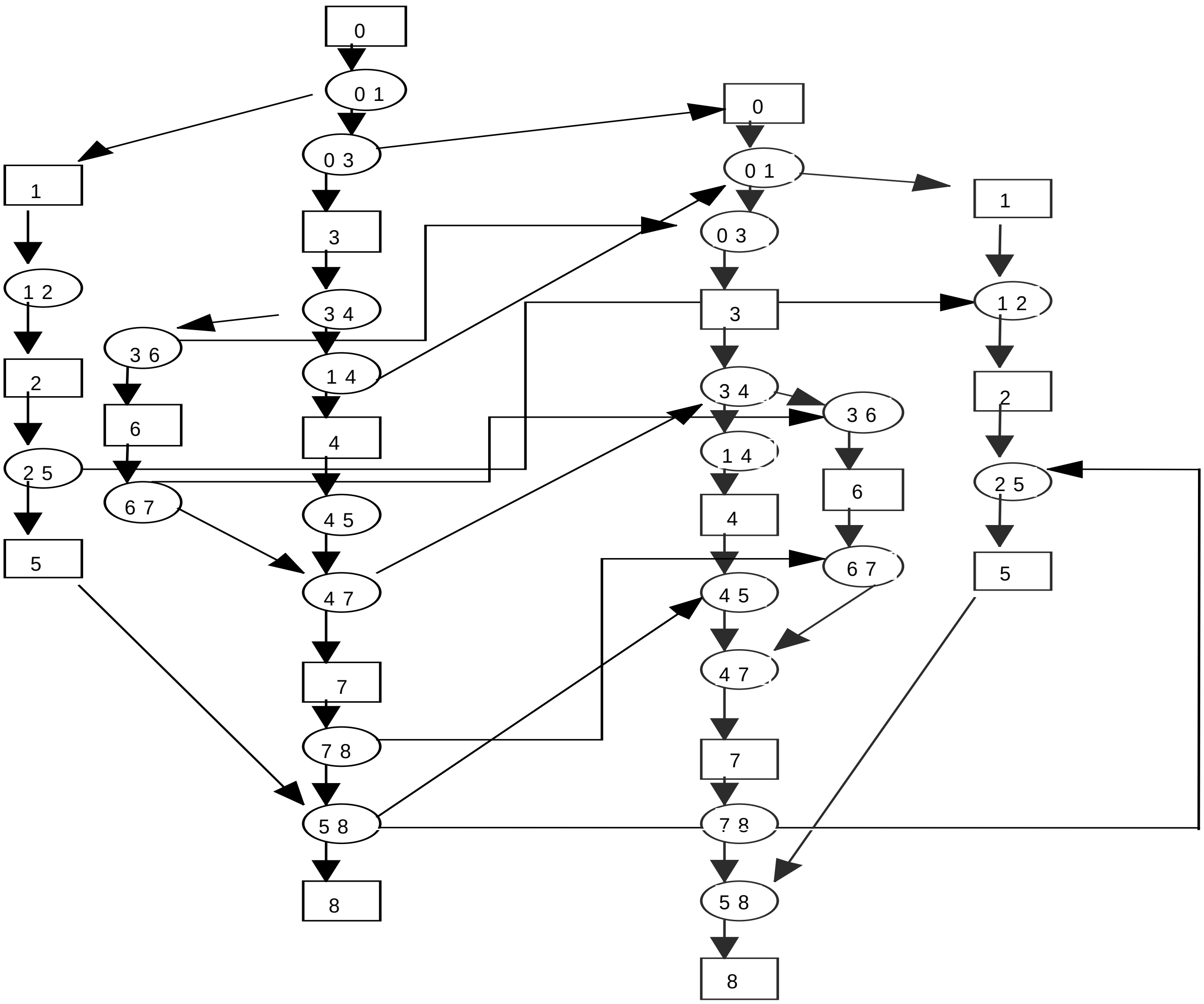

Figure 5. Dynamic Task Graph for Sweep3D on a 3x3 Processor Grid.

4. POEMS Performance Analysis Tools

In this section we describe the initial POEMS analytical and simulation tools for end-end performance prediction, illustrating how these tools are applied to Sweep3D.

4.1 LogP and LoPC

Two of the analytical modeling capabilities in POEMS are the LogP model [8] and an extension of LogP known as LoPC [9]. The task graph for a given application elucidates the principal structure of the code, including the communication events, from which it is relatively easy to generate the LogP or LoPC equations. We illustrate this approach using the Sweep3D application.

As explained above, the Sweep3D application has a pipelined wavefront structure in which each of the m×n processors is mapped to a rectangle in the I,J plane, denoted by (i,j), and alternately performs a block of computation followed by sending data to two nearest neighbor processors. The time to compute one block of wavefront data is modeled in LogP or LoPC as:

$$\mathbf{W_{i,j} = W_g \times MMI \times mk \times it \times jt}$$

where $\mathbf{W_g}$ is the measured time to compute one grid point, and **MMI**, **mk**, **it**, and **jt** are the input parameters that specify the number of grid points per block per processor. The LoPC model includes an inflation factor for $W_{i,j}$ that accounts for processing incoming messages during the computation; however, since the SP/2 has a message co-processor this inflation factor is one for the SP/2.

To account for the pipelining of the wavefronts across the processors, the LogP/LoPC models use the following method for computing the start time for processor (i,j) during a right downward sweep:

$$\mathbf{startP_{i,j} = max\,(startP_{i-1,j} + W_{i-1,j} + o + L,\ \ startP_{i,j-1} + W_{i,j-1} + 2o + 3L)}$$

where **o** is the processing time for an incoming message and **L** is the time for the message to be sent across the interconnect. LoPC adds a term for contention at the message processing resources; however, estimates of this term are negligible for the SP/2 and the application parameters of current interest. The sweeps in other directions have a recursive equation with a similar form to the above.

The Sweep3D application makes a sweep across the processors for each octant pair. We compute the critical path time for the first two right downward sweeps as:

$$\mathbf{time_{1,2} = startP_{m,1} + 2(W_{m,1} + o + 2L) \times \#k\text{-}blocks \times \#angle\text{-}groups}$$

This is the time until the processor (m,1) has finished communicating the results from the last block of the second sweep. At this point, the third sweep can start at processor (m,1). We compute the critical path time for the next two upward right sweeps as follows:

$$\mathbf{time_{3,4} = startP_{1,n-1} + 2(W_{1,n-1} + o + 2L) \times \#k\text{-}blocks \times \#angle\text{-}groups + W_{1,n}}$$

This equation represents the time until processor (1,n) has finished with the fourth sweep. Since the final four sweeps will have the same total time as the first four, we compute the total execution time per iteration of Sweep3D as follows:

$$\mathbf{T = 2\ (\,time_{1,2} + time_{3,4}\,)}$$

The model estimates of execution time are shown in section 5 to be quite accurate when compared with measured Sweep3D execution time for the Fortran implementation on the SP/2. If other pipelined wavefront applications exhibit contention for message processing resources, this contention can be included in the LoPC model as noted above.

4.2 Deterministic Task Graph Analysis

Another analytical model in POEMS is parallel program performance prediction based on deterministic task graph analysis [1]. The inputs to the model are a task graph that describes the tasks and synchronization behavior of an application, a description of the task scheduling method used to allocate tasks to processors, and parameters describing the computational cost and average communication rate of each task.

The model is a two-level hierarchical model in which the application is modeled as a task graph with deterministic total task times, and contention in the underlying parallel system is modeled using a stochastic queueing network. The queueing network model is solved using AMVA to compute the mean values of the task residence times, and these are used as deterministic task execution times in analyzing the task graph model [2]. A modified critical path analysis is used to analyze the task graph, modified to account for task scheduling when the number of ready tasks exceeds the number of available processors. The assumption of deterministic task times is essential to enable a very efficient solution of the task graph, and to make it practical to solve the queueing network for every combination of tasks in execution (viz., O(N) times for a task graph with N tasks).

The deterministic task graph analysis has been shown to be efficient and consistently accurate for several shared-memory programs (including three from the Splash benchmarks [20]). It also has been shown to be useful for *predicting* the impact of program design changes, including the benefits of sophisticated dynamic and semi-static scheduling algorithms that improve communication locality as well as load-balancing [1]. Task graph analysis of the Sweep3D application is in progress.

4.3 Execution-driven System Simulation

POEMS includes a modular, execution-driven, parallel program simulator called MPI-SIM that has been developed at UCLA [15]. MPI-SIM can evaluate the performance of existing MPI programs as a function of various hardware and system software characteristics that include the number of processors, interconnection network characteristics, or message-passing library implementations. The simulator can also be used to evaluate the performance of parallel file systems and I/O systems [5]. Supported capabilities include a number of different disk caching algorithms, collective I/O techniques, disk cache replacement algorithms, and I/O device models.

MPI-SIM simulates the application program by using direct execution to simulate local code fragments and parallel discrete-event simulation to simulate communication (point-point and collective operations) and I/O operations. The parallel simulation uses a set of parallel conservative synchronization protocols together with a number of optimizations to reduce the time to execute the simulation models.

MPI programs execute as a collection of single threaded processes, and, in general, the host machine will have fewer processors than the target machine (for sequential simulation, the host machine has only one processor). This requires that the simulator support multithreaded execution of MPI programs. MPI-LITE, a portable library for multithreaded execution of MPI programs, has been developed for this purpose. A preprocessor replaces all MPI calls by equivalent calls to corresponding routines in the simulator. The simulator does not directly simulate every MPI call. Rather, all collective communication functions first are translated by the simulator in terms of point-to-point communication functions, and all point-to-point communication functions are implemented using a set of core non-blocking MPI functions. Note that the translation of collective communication functions in the simulator is identical to how they are implemented on the target architecture. MPI-SIM has been validated against the NAS MPI benchmarks and has demonstrated excellent performance improvement with parallel execution against these benchmarks [15].

Section 5 presents results on the scalability and performance of Sweep3D as predicted by MPI-SIM. A facility to simulate the task graph notation expressed as an MPI program is currently being developed. Although the use of direct execution speeds up the simulator, it also restricts its ability to evaluate the performance of the application as a function of alternative processor and memory subsystem designs. As described in the next section, POEMS is also developing a simulator for the processor and memory subsystems as part of its hardware domain models. These models will be integrated with MPI-SIM to predict the performance of actual programs or program designs as a function of proposed next-generation processor and memory designs.

4.4 Instruction-level Analysis of the Memory Hierarchy

The POEMS modeling environment is being designed to allow a user to define the computer architecture to be modeled and the modeling methodology to be used. To do this, the user parametrically defines the hardware subsystem on which a task is to execute and the attributes required to define the methodology to be used to compute the task execution time. A task is defined to be a computational unit that does not include interprocessor communication. Task execution time is evaluated by instruction-level simulation of the CPU and access-level simulation of the memory hierarchy. For enhanced throughput, performance database query lookup will be used to compute task execution times for specific configurations of hardware subsystems.

Three types of instruction-level CPU simulation are being considered: instruction-driven, execution-driven, and trace-driven. Instruction-driven simulation (IDS) of the CPU can be enhanced to include simulation of instruction-level parallelism (ILP). This IDS-ILP modeling has been shown to be essential for accurate evaluation of task execution times [14] and validation of analytical models. Modeling of the micro-architecture guarantees an accurate delivery sequence of memory addresses to the memory hierarchy simulator. Because a computational unit is executed each time a simulation is run, IDS-ILP captures both the dynamics of the application code and the memory reference behavior.

Execution-driven simulation (EDS) also captures the dynamics of the application code. However, EDS provides task execution time evaluation only for processors with instruction sets identical to that of the host architecture. EDS platforms commonly described in the literature do not model the micro-architecture. Even with its limitations, EDS of a processor has proven to be helpful in analyzing discrepancies between measured and simulated execution times of Sweep3D tasks.

Trace-driven simulation (TDS) is independent of the host architecture and is attractive because of the speed with which addresses can be delivered to the memory hierarchy simulator. However, for processors with ILP, a simple playback of the recorded trace does not provide an accurate delivery sequence of memory address to the memory hierarchy simulator [21]. In addition, since TDS is driven by a memory address trace that is recorded from a single execution of a program it may not capture the dynamics of the application code [22] and the recorded trace may have large storage requirements, even when compressed.

For the large applications targeted for the POEMS modeling system, simulation throughput will be enhanced with a performance database that can be used to compute task execution times for specific hardware subsystems. These task execution times and associated execution characteristics, such as cache and TLB hit rates, will be obtained using system measurements and subsystem simulation.

As described in section 2.2, this multi-paradigm modeling capability will be supported in POEMS by using a Task Execution Description (TED) associated with each node in an application's task dependency graph. If task execution time is to be evaluated by IDS, IDS-ILP, or EDS, the TED includes the name of the simulator and pointers to the executable image of the application code stored in the simulated memory of each simulated processor. If task execution time is to be evaluated by TDS, the TED includes the names of both the simulator and the file containing the memory address trace of the task. If the task execution time is to be evaluated by a database query, the task identifier and the hardware configuration attributes in the TED are used to perform the lookup.

5. Preliminary Results for Sweep3D

This section summarizes results from the application of a subset of the tools described in the previous section to study the performance of the Sweep3D application. The MPI code previously developed for this application was the point of departure for the study. The original application was written using Fortran. A version was generated in MPI with C to allow the application to be evaluated with the simulator. The goal of the initial study was to investigate how well Sweep3D performance scales with the number of processors on the IBM SP/2 and the SGI Origin2000. Availability of both Fortran and C versions of the program allow us to investigate this question for both implementations.

Clearly, measurement is restricted by both the size and the IPC characteristics of the available hardware. By using the LogP/LoPC model and the MPI-SIM simulator performance scalability for larger numbers of processors and improved communication primitives can be evaluated.

The Sweep3D task graph aided in developing the LogP/LoPC model as well as in measuring the parameter values (*e.g.*, computation granularity of the code) for all of the models. Preliminary results of the analytical and simulation studies are discussed below.

Figure 6 shows the validation of the LogP/LoPC model against SGI Origin2000 measurements. The problem size per processor is fixed in this experiment (*i.e.*, the total problem size increases with the number of processors). Since the length of the critical path is equal to the sum of the dimensions of the grid, the runtime should increase linearly in $P_i + P_j$, as is observed. Furthermore, the breakdown of the total computation time and communication time predicted by the model shows that Sweep3D is heavily cpu-bound. Details of a scalability study of Sweep3D using the LogP model can be found in [10].

Validations of the LogP/LoPC model and the MPI-SIM

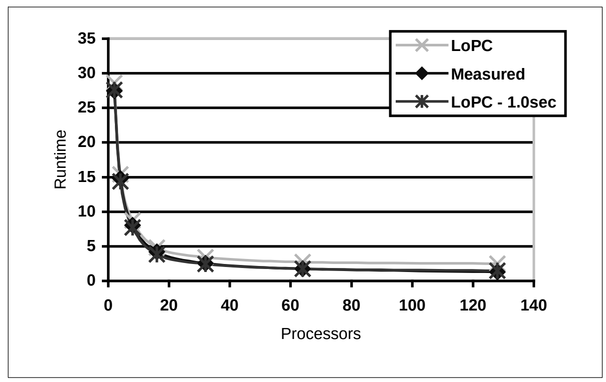

Figure 7. LogP/LoPC Model Validation on an IBM SP/2. (grid size: 50x50x50, k-blocking factor=10, angle-blocking factor=3)

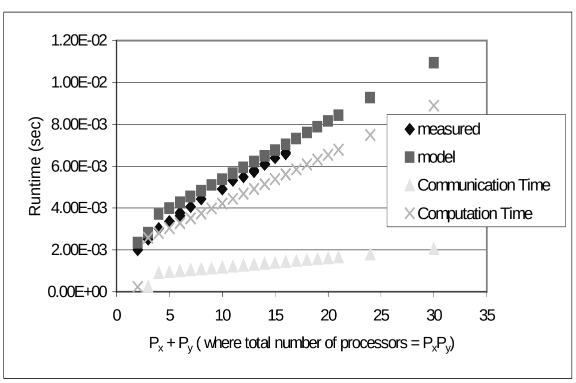

Figure 6. LogP/LoPC Model Validation on SGI Origin2000. (subgrid size: 5x5x10, k-blocking factor=1, single octant execution time)

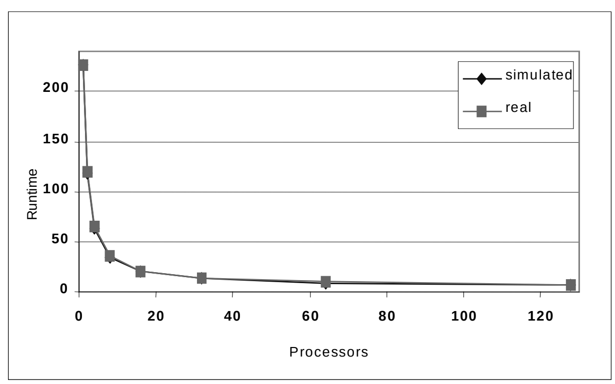

Figure 8. MPI-SIM Model Validation and Scalability Results on an IBM SP/2. (grid size: 50x50x50, k-blocking factor=10, angle-blocking factor=3)

model against the SP/2 measurements, for fixed total problem size, are shown in figures 7 and 8, respectively. We are currently investigating why the model estimates for the SP/2 architecture appear to be high by a nearly constant factor of 1 second. In any case, the simple analytical model appears to predict the performance behavior to within this constant factor extremely well, and the MPI-SIM model produces extremely accurate results

As shown in figures 9 and 10, the performance of Sweep3D as a function of message latencies has been investigated. Five different latencies were used for these studies: 0, *sp*, 10*sp*, 50*sp*, and 100*sp*, where *sp* is the current latency on the SP/2. Surprisingly, the actual SP/2 latency appears to have very small impact on the execution time of the Sweep3D kernel for configurations with up to 64 processors (as shown by both the MPI-SIM and LoPC predictions). For the MPI-SIM experiment based on the C version of Sweep3D, the differences in performance only became apparent when the latency was increased to 50 times that of an SP/2. The LoPC results

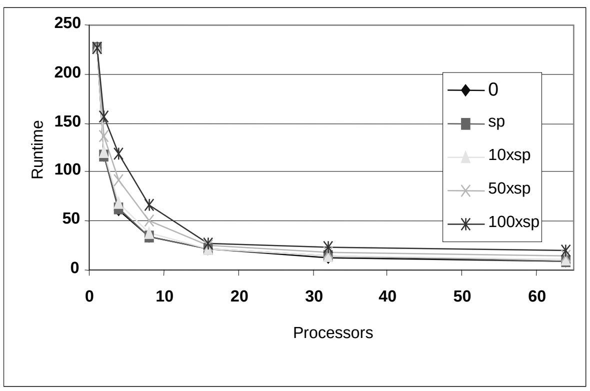

Figure 9. Predicted Impact of Communication Latency on Sweep3D (C code)

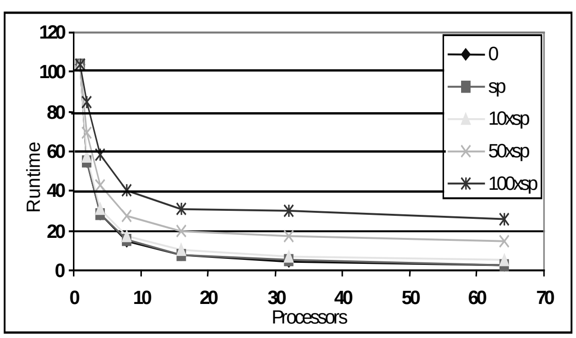

Figure 10. Predicted Impact of Communication Latency on Sweep3D (Fortran code)

based on the Fortran version of Sweep3D show that the execution time is a bit more sensitive to message latency, which makes sense given the smaller overall execution time of the Fortran code.

6. Related Work

The conceptual breadth of the POEMS project makes the list of related research topics far to long to permit an extensive review in this paper. A more extensive but still far from comprehensive survey of related work and a list of references can be found on the POEMS project web page: (http://www.cs.utexas.edu/users/poems). The references cited in this section are the most pertinent sources of concepts.

The conceptual framework for POEMS is a synthesis from models of naming and communication [6, Bayerdorffer 95], Computer-Aided Design (CAD), software frameworks, parallel computation [12], object-oriented analysis [19], data mediation [24] and intelligent agents. There are many research projects focusing on performance modeling. The most closely related projects to POEMS are probably the Maisie parallel discrete-event simulation framework and its use in parallel program simulation [4, 15], SimOS [18], and RSIM [13] and the earlier work in program simulators, direct execution simulators, and parallel discrete-event simulation.

The conceptual extensions offered by POEMS are a formal specification language for composition of system models and integration of multiple paradigms for model evaluation. The alternative modeling paradigms support validation and allow different levels of analyses of existing and future application programs within a common framework.

There are many effective commercial products for simulation modeling of computer and communication systems. The *March 1994 IEEE Communications Magazine* presents a survey of commercial products.

7. Conclusions

The POEMS project is creating a problem solving environment for end-to-end performance models of complex parallel and distributed systems and applying this environment for performance prediction of application software to current and future generations of such systems. This paper has described the key components of the POEMS framework: a generalized task graph model for describing parallel computations, automatic generation of the task graph by a parallelizing compiler, a specification language for mapping the computation on component models from the operating system and hardware domain, compositional modeling of multi-paradigm, multi-scale, multi-domain models, integration of a Performance Recommender for selecting the computational parameters for a given target performance, and a wide set of modeling tools ranging from analytical models to parallel discrete-event simulation tools.

The paper illustrated the POEMS modeling methodology and approach, by using a number of the POEMS tools for performance prediction of the Sweep3D application kernel selected by Los Alamos National Laboratory for evaluation of ASCI architectures. The paper validated the performance predicted by the analytical and simulation models against the measured application performance. The Sweep3D kernel used for this study is an example of a *regular* cpu-bound application. Reusable versions of the analytical and simulation models, parameterized for three-dimensional wavefront applications, will form the initial component model library for POEMS.

In addition to continuing efforts on the preceding topics, a number of interesting research directions are being pursued by the POETS. First, the performance tools described in this paper will be extended to directly support the evaluation of parallel programs expressed using the task graph notation. Second, in the study reported in this paper, there was considerable synergy among the development of the analytical and simulation models, enabling validations to occur more rapidly than if each model had been developed in isolation; however, thus far the models have been applied independently. We are engaged in developing a multi-paradigm modeling environment where the analytical models will be used by the execution-driven simulator, *e.g.*, to estimate communication delays and/or task execution times, and simulation models will be automatically invoked to derive analytical model input parameters. Third, the memory and processor subsystem models that are being developed will be integrated in the execution-driven simulator to permit us to evaluate the impact of alternate hardware designs (*e.g.*, deep memory hierarchies) on existing applications that use popular communication mechanisms like MPI. We will continue to investigate innovations in parallel discrete-event simulation technology to reduce the execution time for the integrated models. The integration of compiler support with analytical and simulation support will enable (to our knowledge) the first fully-automatic end-to-end performance prediction capability for parallel applications and systems.

Acknowledgments

A number of people from the member institutions represented by the POEMS team contributed to the work: Thanks to Thomas Phan and Steven Docy for their help with the use of MPI-SIM to predict the Sweep3D performance on the SP/2. Thanks to the Office of Academic Computing at UCLA and to Paul Hoffman for help with the IBM SP/2 on which many of these experiments were executed. This work was supported by DARPA/ITO under Contract N66001-97-C-8533, "End-to-End Performance Modeling of Large Heterogenous Adaptive Parallel/Distributed Computer/Communication Systems," 10/01/97 - 09/30/00 and by a NSF grant titled "Design of Parallel Algorithms, Language, and Simulation Tools Award ASC9157610", 08/15/91 - 7/31/98. Thanks to Frederica Darema for her support of this research.

References

[1] Adve, V. S., "Analyzing the Behavior and Performance of Parallel Programs", Univ. of Wisconsin-Madison, UW CS Tech. Rep. #1201, Oct. 1993.

[2] Adve, V. S., and M. K. Vernon, "Influence of Random Delays on Parallel Execution Times", *Proc. 1993 ACM SIGMETRICS Conf. on Measurement and Modeling of Computer Systems*, May 1993, pp. 61-73.

[3] Adve, V. S., and J. Mellor-Crummey, "Using Integer Sets for Data-Parallel Program Analysis and Optimization", *SIGPLAN98*, Montreal, June 1998.

[4] Bagrodia, R., and W. Liao, "Maisie: A Language for Design of Efficient Discrete-event Simulations", *IEEE Transactions on Software Engineering*, Apr. 1994.

[5] Bagrodia, R., S. Docy, and A. Kahn, "Parallel Simulation of Parallel File Systems and I/O Programs", *Proc. Supercomputing '97*, San Jose, 1997.

[6] Bayerdorffer, B., *Associative Broadcast and the Communication Semantics of Naming in Concurrent Systems*, Ph.D. Dissertation, Dept. of Computer Sciences, Univ. of Texas at Austin, Dec. 1993.

[7] Booch, G., J. Rumbaugh, and I. Jacobson, *Unified Modeling Language User Guide*, Addison-Wesley, Englewood Cliffs, NJ, 1997.

[8] Culler, D., R. Karp, D. Patterson, A. Sahay, K. E. Schauser, E. Santos, R. Subramonian, and T. VonEiken, "LogP: Towards a Realistic Model of Parallel Computation", *Proc. 4th ACM SIGPLAN Symp. on Principles and Practices of Parallel Programming (PpoPP '93)*, San Diego, May 1993, pp. 1-12.

[9] Frank, M. I., A. Agarwal, and M. K. Vernon, "LoPC: Modeling Contention in Parallel Algorithms", *Proc. 6th ACM SIGPLAN Symp. on Principles and Practices of Parallel Programming (PpoPP '97)*, Las Vegas, June 1997, pp. 62-73.

[10] Hoisie, A., O. M. Lubeck, and H. J. Wasserman. "Performance Analysis of Multidimensional Wavefront Algorithms with Applications to Sn Transport", *in preparation*, Apr. 1998.

[11] Koch, K. R., R. S. Baker, and R.E. Alcouffe, "Solution of the First-Order Form of the 3-D Discrete Ordinates Equation on a Massively Parallel Processor", *Trans. of the Amer. Nuc. Soc.*, **65**(198), 1992.

[12] Newton, P., and J. C. Browne, "The CODE 2.0 Graphical Parallel Programming Language", *Proc. ACM Int. Conf. on Supercomputing*, July 1992.

[13] Pai, V. S., P. Ranganathan, and S. V. Adve. "RSIM Reference Manual version 1.0", Technical Report #9705, Dept. of Electrical and Computer Engineering, Rice Univ., Aug. 1997.

[14] Pai, V. S., P. Ranganathan, and S. V. Adve. "The Impact of Instruction Level Parallelism on Multiprocessor Performance and Simulation Methodology", *Proc. 3rd Int. Conf. on High Performance Computer Architecture*, San Antonio, March 1997, pp. 72-83.

[15] Prakash, S., R. Bagrodia Using Parallel Simulation to Evaluate MPI Programs. To appear in Winter Simulation Conference 1998.

[16] Ramakrishnan, N., *Recommender Systems for Problem Solving Environments*, Ph.D. Dissertation, Dept. of Computer Sciences, Purdue Univ., 1997.

[17] Rumbaugh, J., *et al. Object-Oriented Modeling and Design*, Prentice-Hall, Englewood Cliffs, NJ, 1991.

[18] Rosenblum, M., S.A. Herrod, E. Witchel, and A. Gupta, "Complete Computer System Simulation: The SimOS Approach", *IEEE Parallel and Distributed Technology*, Winter 1995, pp. 34-43.

[19] Shlaer, S., and S. Mellor, *Object Lifecycles: Modeling the World in States*, Yourdon Press, NY, 1992.

[20] Singh, J. P., W.-D. Weber and A. Gupta, "SPLASH: Stanford Parallel Applications for Shared-Memory", *Computer Architecture News*, **20**(1), Mar. 1992, pp. 5-44.

[21] Sorin, D. J., V. S. Pai, S. V. Adve, M. K. Vernon, and D. A. Wood, "Analytic Evaluation of Shared-Memory with ILP Processors", to appear in *the 25th Annual Int. Symp on Computer Architecture*, May 1998.

[22] Teller, P. J., "MP Simulations Via Unscheduled Traces", *Calculateurs paralleles*, **7**(1), Apr. 1995, pp. 9-25.

[23] Wasserman, W. J., O. M. Lubeck, and Y. Luo, "Performance Evaluation of the SGI Origin2000: a Memory-centric Characterization of LANL ASCI Applications", *Proc. Supercomputing '97*, San Jose, 1997.

[24] Wiederhold, G., "Mediation in Information Systems: Research Directions in Software Engineering", *ACM Computing Surveys*, June 1995.