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

Vikram S. Adve, Rajive Bagrodia, James C. Browne, Ewa Deelman, Aditya Dube,
Elias Houstis, John Rice, Rizos Sakellariou, David Sundaram-Stukel,
Patricia J. Teller and Mary K. Vernon

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. Towards this end, the POEMS framework supports composition of component models from these different domains into an end-to-end system model. This composition can be specified using a generalized graph model of a parallel system, together with interface specifications that carry information about component behaviors and evaluation methods. The POEMS Specification Language compiler, under development, will generate an end-to-end system model automatically from such a specification. The components of the target system may be modeled using different modeling paradigms (analysis, simulation, or direct measurement) and may be modeled at various levels of detail. As a result, evaluation of a POEMS end-to-end system model may require a variety of evaluation tools including specialized equation solvers, queuing network solvers, and discrete-event simulators. A single application representation based on static and dynamic task graphs serves as a common workload representation for all these modeling approaches. Sophisticated parallelizing compiler techniques allow this representation to be generated automatically for a given parallel program. POEMS includes a library of predefined analytical and simulation component models of the different domains, and a knowledge base that describes performance properties of widely-used algorithms. This paper provides an overview of the POEMS methodology and illustrates several of its key components. The methodology and modeling capabilities are demonstrated by predicting the performance of alternative configurations of Sweep3D, a complex benchmark for evaluating wavefront application technologies and high-performance, parallel architectures.

Index Terms— Performance modeling, parallel system, message passing, analytical modeling, parallel simulation, processor simulation, task graph, parallelizing compiler, compositional modeling, recommender system.

- *Vikram Adve is with the Computer Science Department, University of Illinois at Urbana-Champaign, Urbana, IL 61801. E-mail: vadve@cs.uiuc.edu.*
- *Rajive Bagrodia and Ewa Deelman are with the Computer Science Department, University of California at Los Angeles, Los Angeles, CA 90095-1596. E-mail: {rajive,deelman}@cs.ucla.edu.*
- *James C. Browne and Aditya Dube are with the Department of Computer Science, University of Texas at Austin, Austin, TX 78712-1188. E-mail: {browne,dube}@cs.utexas.edu*
- *Elias Houstis and John Rice are with the Computer Science Department, PurdueUniversity, West Lafayette, IN 47907-1398. E-mail: {jrr,enh}@cs.purdue.edu.*
- *Rizos Sakellariou is with the Computer Science Department,. University of Manchester, Manchester M13 9PL, U.K. E-mail: rizos@cs.man.ac.uk.*
- *David Sunderam-Stukel and Mary Vernon are with the Computer Science Department, University of Wisconsin-Madison, Madison, WI 53706. E-mail: {sunderam,vernon}@cs.wisc.edu*
- *Patricia Teller is with the Department of Computer Science, The University of Texas at El Paso, El Paso, TX 79968. E-mail: pteller@cs.utep.edu.*

A preliminary version of this paper appeared in the Proceedings of the *First International Workshop on Software and Performance (WOSP) '98*, October 1998.

1 INTRODUCTION

Determining the performance of large-scale computational systems across all stages of design enables more effective design and development of these complex software/hardware systems. Towards this end, the Performance Oriented End-to-end Modeling System (POEMS) project is creating and experimentally evaluating a problem-solving environment for end-to-end performance analysis of complex parallel/distributed systems, spanning application software, operating system (OS) and runtime system software, and hardware architecture. The POEMS project leverages innovations in communication models, data mediation, parallel programming, performance modeling, software engineering, and CAD/CAE into a comprehensive methodology to realize this goal. This paper presents an overview of the POEMS methodology, illustrates the component models being developed in the POEMS project by applying them to analyze the performance of a highly-scalable application, and finally describes preliminary work being performed in integrating multiple different models for a single application using the POEMS methodology.

The key innovation in POEMS is a methodology that makes it possible to compose multi-domain, multi-paradigm, and multi-resolution component models into a coherent system model. Multi-domain models integrate component models from multiple semantic domains; in the case of POEMS, these domains are application software, OS/runtime software, and hardware. Multi-paradigm models allow the analyst to use multiple evaluation paradigms—analysis, simulation, or direct measurement of the software or hardware system itself—in a single system model. To facilitate the integration of models from different paradigms, the specification and implementation of POEMS models is accomplished through formulating component models as compositional objects with associative interfaces [11,12]. Associative module interfaces extend conventional interfaces to include complete specification of component interactions. A specification language (the POEMS Specification Language or PSL) for specification of multi-domain, multi-paradigm, multi-resolution performance models has been designed and a compiler for this language is under development.

The POEMS project is building an initial library of component models, at multiple levels of granularity, for analyzing both non-adaptive and adaptive applications on highly-scalable architectures. POEMS supports analytical models, discrete-event simulation models at multiple levels of detail, and direct execution. The analytical models include deterministic task graph analysis [1], the LogP[15] family of models including LogGP [5] and LoPC [18], and customized Approximate Mean Value Analysis (AMVA) [37]. Simulation models include validated state-of-the-art processor and memory hierarchy models based on SimpleScalar [14], interconnection network models using the PARSEC parallel simulation language [9], large-scale parallel program simulations using the MPI-Sim simulator [28,10], and parallel I/O system simulators [8]. A unified application representation based on a combination of static and dynamic task graphs has been developed that can serve as a common workload representation for this wide range of performance models.

Ongoing research within POEMS, is developing techniques to integrate subsets of two or more of these component models, as a first step towards automatic integration of models within the POEMS framework. One such effort is integrating a compiler-generated analytical model based on the static task graph with the MPI-Sim simulator. This integrated model has the potential to increase greatly the size of problems and systems that can be simulated in practice. In addition, it can be expanded to include detailed processor simulation (e.g., with the SimpleScalar simulator), ideally for a small subset of the computational tasks. Another effort is examining the integration of MPI-Sim, LogGP, and SimpleScalar models. These ongoing efforts are described briefly in Section 6.

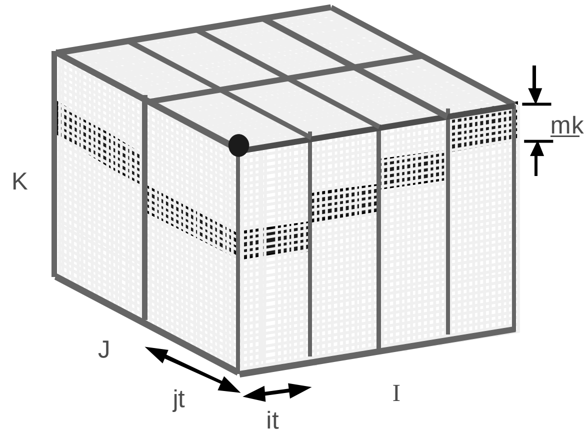


Figure 2.1. One Wavefront of Sweep3D on a 2×4 Processor Grid.

The project also is building a knowledge base of performance data, gathered during the modeling and evaluation process, which can be used to estimate, as a function of architectural characteristics, the performance properties of widely-used algorithms. The knowledge base, along with the component models, application task graphs, and formal task executions descriptions (called TEDs), is stored in the POEMS database, an integral part of the POEMS system.

POEMS development is being driven by the design evaluations of highly-scalable applications executed on parallel architectures. The first driver application is the Sweep3D [22] program that is being used to evaluate advanced and future parallel architectures at Los Alamos National Laboratory.

Section 2 presents an overview of the Sweep3D application, which is used to illustrate the POEMS methodology and performance prediction capabilities. Section 3 describes the conceptual elements of the POEMS methodology, and illustrates it with an example. Section 4 presents the suite of initial POEMS performance tools and the initial library of component models that are under development for Sweep3D. Section 5 presents results of applying these models to provide performance projections and design results for Sweep3D. Section 6 describes ongoing research on the development and evaluation of integrated multi-paradigm models in POEMS. Section 7 discusses related work. Conclusions are presented in Section 8.

2 POEMS DRIVER APPLICATION: SWEEP3D

The initial application driving the development of POEMS is the analysis of an ASCI kernel application called Sweep3D executed on high-performance, parallel architectures such as the IBM SP/2, the SGI Origin 2000,

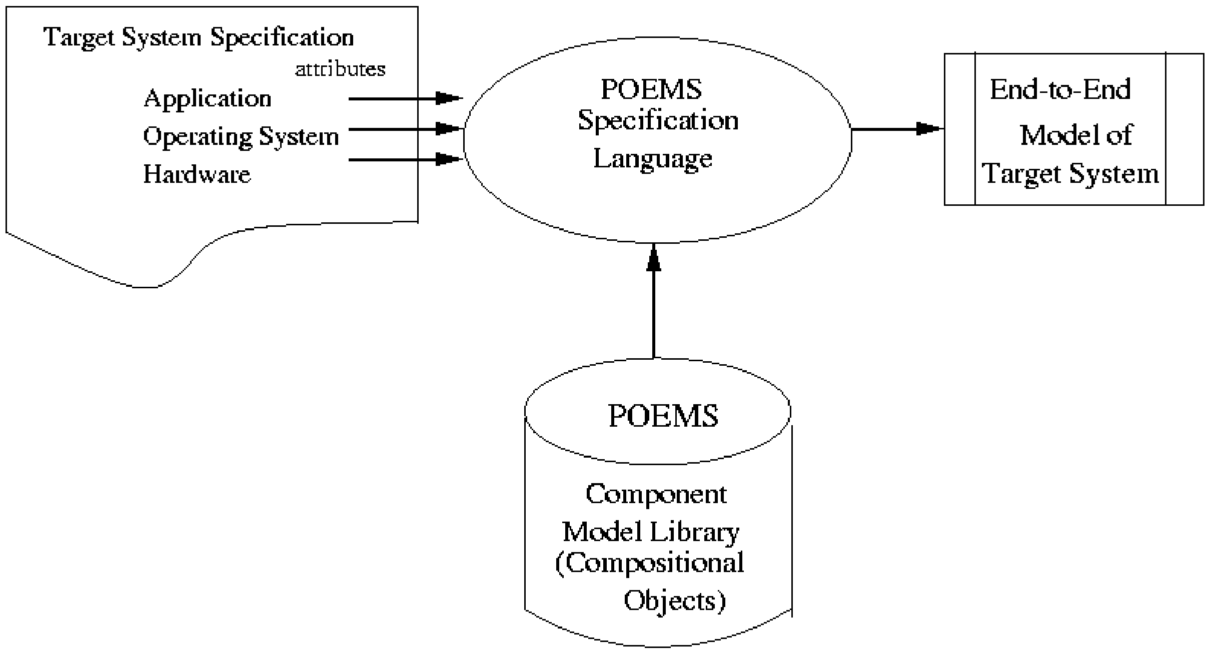


Figure 3.1. Overview of the POEMS Environment

and future architectures. The Sweep3D application is an important benchmark because it is representative of the computation that occupies 50-80% of the execution time of many simulations on the leading edge DOE production systems [19]. Our analysis of this application has three principal goals. One goal is to determine which of the alternative configurations of the Sweep3D application has the lowest total execution time on a given architecture. A related, second goal is to provide quantitative estimates of execution time for larger systems that are expected to be available in the near future. The third goal is to predict the quantitative impact of various possible architectural and/or OS/runtime system improvements in reducing the required execution time. This section contains a brief description of this application; Sections 4 and 5 discuss how the POEMS methodology is being applied to obtain the desired performance projections for Sweep3D.

The Sweep3D kernel is a solver for the three-dimensional, time-independent, neutron particle transport equation on an orthogonal mesh [22]. 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 2.1 illustrates how the three-dimensional computation is partitioned on a 2×4 two-dimensional processor grid. That is, each processor performs the calculations for a column of cells containing $J/2 \times I/4 \times K$ cells. The output fluxes are computed along a number of directions (called angles) through the cube. The angles are grouped into eight *octants*, corresponding to the eight diagonals of the cube (i.e., there is an outer loop over octants and an inner loop over angles within each octant). Along any angle, the flux out of a given cell cannot be computed until each of its upstream neighbors has been computed, implying a *wavefront* structure for each octant. The dark points in the figure illustrate one wavefront that originates from the corner marked by the circle. The processor containing the marked corner cell is the first to compute output fluxes, which are forwarded to the neighboring processors in the i and j dimensions, and so on. In order to increase the available parallelism, the wavefront is pipelined in the k dimension. That is, a processor computes the output fluxes for a partial column of cells of height mk, as shown in the figure, and then forwards the output fluxes for the partial column to its neighboring processors before computing the next partial block of results. (Since the octant starts in the marked upper corner of the cube, the next partial block of cells to be computed is below the dark shaded partial block in each column.) The computation is pipelined by groups of mmi angles, not shown in the figure. The amount of computation for each "pipelined block" is therefore proportional to $it\times jt\times mk\times mmi$, where it and jt are the number of points mapped to a processor in the i and j dimensions, mk is the number of points in the k dimension per pipeline stage, and mmi is the number of angles per pipeline stage.

The inputs to Sweep3D include the total problem size (I, J, K), the size of the processor grid, the k-blocking factor (*mk*), and the angle-blocking factor (*mmi*). Two aggregate problem sizes of particular interest for the DOE ASCI program are one billion cells (i.e., I=1000, J=1000, K=1000) and 20 million cells (i.e., I=255, J=255, J=255). Key configuration questions include how many processors should be used for these problem sizes, and what are the optimal values of *mk* and *mmi*.

3 POEMS METHODOLOGY

Using POEMS, a performance analyst specifies the workload, operating system, and hardware architecture for a system under study, henceforth referred to as the target system. In response, as depicted in Figure 3.1, the completed POEMS system will generate and run an end-to-end model of the specified software/hardware system. The target system is defined via an integrated graphical/textual specification language (the POEMS Specification Language or PSL) for a generalized dependence graph model of parallel computation. Section 3.1 describes the process of component model composition and evaluation tool interfacing. The generalized dependence graph model of parallel computation is described in Section 3.2. The PSL is briefly introduced and illustrated in Section 3.3.

The nodes of the dependence graph are models of system components, i.e., instances of component models in the application, OS/runtime, and hardware domains. In specifying the target system, the analyst defines the properties of each system component in the context of and in terms of the attribute set of its semantic domain. For example, for a hardware component, the analyst defines the design parameters, the modeling paradigm, and the level of detail of the component model. As discussed in Section 3.3, the component models are implemented as compositional objects, i.e., "standard objects" encapsulated with associative interfaces [11, 12, 16] specified in the PSL. These component models, together with application task graphs, and performance results, are stored in the POEMS database, described in Section 3.6. POEMS also includes a knowledge-based system called Performance Recommender to assist analysts in choosing component models. The Performance Recommender is sketched in Section 3.5. As described in Section 3.2, the application domain represents a parallel computation by a combination of static and dynamic task graphs, which are specialized forms of generalized dependence graphs. Generalized dependence graphs can be used in the operating system domain to model process and memory management, interprocess communication, and parallel file systems. In the hardware domain, the nodes of a graph are associated with models of hardware components. Figure 3.2 is a schematic of a generalized dependence graph model spanning multiple domains.

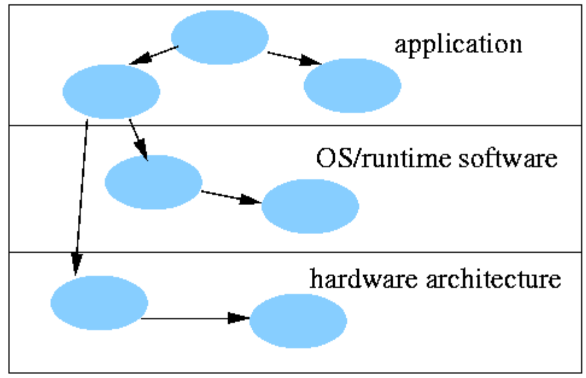


Figure 3.2. Multi-Domain Dependence Graph

When specifying the target system, the analyst selects the components that are most appropriate for the goals of her/his performance study. As discussed in Section 3.4, model composition is facilitated by task execution descriptions (TEDs), which characterize the execution of components of the task graph on particular hardware domain components, e.g., processors, memory hierarchies, and interconnection networks. It also relies on the models being implemented as compositional objects and on data mediation methods. The performance knowledge base of the POEMS database may provide guidance to the analyst in the selection of components. The compiler for the POEMS specification language will access the specified component models from the POEMS database, when appropriate component models are present, and will incorporate them into the system model.

3.1 Component Model Composition and Evaluation Tool Integration

As noted earlier, POEMS composes its system models from component models of different types ranging from simple analytical models such as LogGP to detailed simulation models of instructions executing on modern processor/cache architectures. Each component model can be characterized as a "document" that carries an *external interface* defining its properties and behavior. Successful composition requires that this interface specifies the properties and behavior of the component models with which the given component model can be composed successfully.

Evaluation of the multi-paradigm system models requires interfacing and integration of evaluation tools for each component model type. Each evaluation tool also can be characterized as a processor that "understands" and acts upon information supplied for the component model it is evaluating. Each tool evaluates the behavior of the component model types it "understands". The input to each tool must be in the types set it "understands." Interfacing and integration of tools requires that the output of one tool be the input of another tool. Successful interfacing and integration of tools, each of which "understands" different type sets, requires that the output of a source tool (which will be in the type set it “understands”) must be mapped to the type set "understood" by its target tool. The information necessary to support these mappings must be defined in the external interfaces from which the tools are invoked.

Manual composition of component models to create system models is now accomplished on an ad hoc basis by humans reading documentation and applying their experience and knowledge to select component models with properties and behaviors such that the components can be integrated. Manual interfacing of evaluation tools that utilize different representations of components is now accomplished by manual specification of mappings from outputs to inputs and hand coding of these mappings. Design and implementation of these mappings is often a very laborious and error-prone process, sufficiently so that these tasks are seldom attempted. A major reason for the difficulty of these tasks is that there is no consistent specification language in which to describe the properties and behaviors of either components or tools. The human system model composer and evaluation tool integrator must resolve the ambiguities and inconsistencies of mappings among languages with different semantics on an ad hoc, case-by-case basis. Automation of component model composition and tool interfacing and integration will require that the interfaces of component models and evaluation tools be specified in a common representation with semantics sufficiently precise to support automated translations.

The POEMS Specification Language (PSL) [16] is the representation used in POEMS to specify the information necessary to support composition of component models and interfacing and integration of evaluation tools. PSL is an interface specification language. Components and tools are treated as objects and each is encapsulated with a PSL-specified interface. PSL-specified interfaces, called associative interfaces, are a common language for expressing both component model composition and mappings across evaluation tool interfaces. An object encapsulated with an associative interface is called a compositional object. Associative interfaces and compositional objects are defined and described in Section 3.3.2. An example of PSL specifications is given in Section 3.3.4.

A compiler for PSL-specified interfaces that has access to a library of component models and evaluation tools can automatically compose a set of component models selected by a performance engineer into a system model and generate the mappings among the outputs and inputs of the evaluation tools. The compiler for PSL programs will generate an instance of the generalized hierarchical dependence graph defined in Section 3.2. A feasibility demonstration prototype of a PSL compiler has been developed [16] and a more capable version of the PSL compiler is under development.

PSL-specified associative interfaces also facilitate manual composition and integration processes since they provide a systematic framework for formulation of component model and evaluation tool interfaces. The applications of the POEMS methodology to performance studies of Sweep3D given in Sections 5 and 6 are manually executed examples of the model composition and evaluation tool interfacing methods and concepts defined and described in this section and in Sections 4 and 5.

3.2 General Model of Parallel Computation

Given a PSL program, the PSL compiler generates an instance of a generalized hierarchical dependence graph. This model of parallel computation and its use in POEMS system modeling are described in this section after introducing some basic definitions and terminology.

3.2.1 Definitions and Terminology

Generalized Hierarchical Dependence Graph: A general graphical model of computation in which each node represents a component of the computation and each edge represents a flow of information from node to node. The graph is hierarchical in that each node can itself be an interface to another generalized dependence graph. Each edge in such a graph may represent either a dataflow relationship or a precedence relationship. For each node, an extended firing rule (that may include local and global state variables in addition to input data values) specifies when each node may begin execution. For a precedence edge, the node at the head of the edge may not begin execution until the node at the tail has completed.

Dynamic Task Graph: An acyclic, hierarchical dependence graph in which each node represents a sequential task, precedence edges represent control flow or synchronization, and dataflow edges (called communication edges) represent explicit data transfers between processes.

Static Task Graph: A static (symbolic) representation of the possible dynamic task graphs of a program, in which each node represents a set of parallel tasks and each edge represents a set of edge instances. Unlike the dynamic task graph, this graph includes loop and branch nodes to capture logical control flow (and hence the graph may contain cycles).

3.2.2 Dependence Graph Model of Parallel

Systems

The POEMS representation of a parallel system is a hierarchical dependence graph in which the nodes are instances of compositional objects and the edges represent flows of information from node to node. As shown in Figure 3.2, the nodes may be defined in different domains and a node in one domain may invoke nodes in both its own and implementing domains. For example, the implementing domains for an application are the OS/runtime system and hardware domains.

The graph model is executable, where an execution represents a model solution for the specified application software and system configuration. The nodes and edges may be instantiated during execution, thus defining a dynamic instance of the graph that captures the behavior of the entire system during the execution being modeled. This dynamic graph may or may not be constructed explicitly at runtime, depending on the specified solution methods.

A node executes when the computation reaches a state in which its associated "firing rule" evaluates to true. A firing rule is a conditional expression over the state of the input edges and the local state of a node. Each edge has associated with it a data type specification, which is called a transaction. Clock-driven execution is obtained by adding to the data dependence relationships (and specifications of the firing rules for the nodes) an edge between each source sink node pair carrying the current time of each source node. Each node executes the Lamport [23] distributed clock algorithm to determine the current time.

3.2.3 Common Application Representation

The POEMS application representation is designed to provide workload information at various levels of abstraction for both analytical and simulation models. More specifically, the representation is designed to meet four key goals [3]. First, it should provide a common source of workload information for the various modeling techniques envisaged in POEMS. Second, the representation should be computable *automatically* using parallelizing compiler technology. Third, the representation should be concise and efficient enough to support modeling terascale applications on very large parallel systems. Finally, the representation should be flexible enough to support performance prediction studies that can predict the impact of changes to the application, such as changes to the parallelization strategy, communication, and scheduling. The design of the representation is described in more detail in [3], and is summarized briefly here.

The application representation in POEMS is based on a combination of *static* and *dynamic* task graphs. As defined earlier, the static task graph (STG) is a compact, symbolic graph representation of the parallel structure of the program. The nodes represent computational tasks, CPU components of communication, or control flow. The graph also includes extensive symbolic information in order to capture symbolically the structure of the parallel computation as a function of parameters such as the number of processors and relevant program input variables. For example, each static task node contains a symbolic integer set that describes the set of task instances that are executed at runtime (e.g., the following set can be used to denote a task node with P instances: $\{[i] : 0 \le i \le P-1\}$). Each edge between parallel task nodes contains a symbolic integer mapping that describes the edge instances connecting task instances (e.g., the mapping $\{[i \rightarrow j] : 1 \le j \le P-1 \wedge j = i+1\}$ denotes P-1 edge instances connecting instance i of the node at the tail to instance i+1 of the node at the head). Finally, control-flow nodes contain symbolic information about loop bounds or branching conditions. Together, this information enables the STG to capture the parallel structure of the program while remaining independent of program input values. In addition, each computational task also has an associated scaling function describing how the computational work for the task scales as a function of relevant program variables. For example, this information can be used to support simple extrapolation of performance metrics as a function of input size.

The dynamic task graph, which is instantiated from the static 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 many modeling techniques, however, the dynamic task graph need not be instantiated explicitly but the same information can be obtained by traversing the static task graph at model runtime. This is a key capability already available in the graph-based runtime system on which the POEMS implementation will be based.

3.2.4 Sweep3D Task Graph

The task graph concepts are illustrated by the static task graph for the sweep phase of Sweep3D, shown on the left-hand side of Figure 3.3. This task graph was generated manually and is a simplified version of the graph that would be generated using the task graph synthesis techniques in the dHPF compiler, described in Section 4.1. (The actual static task graph differs mainly in that there are additional small computational tasks between the communication tasks, and the k-block actually consists of several computation and a few control-flow nodes.)

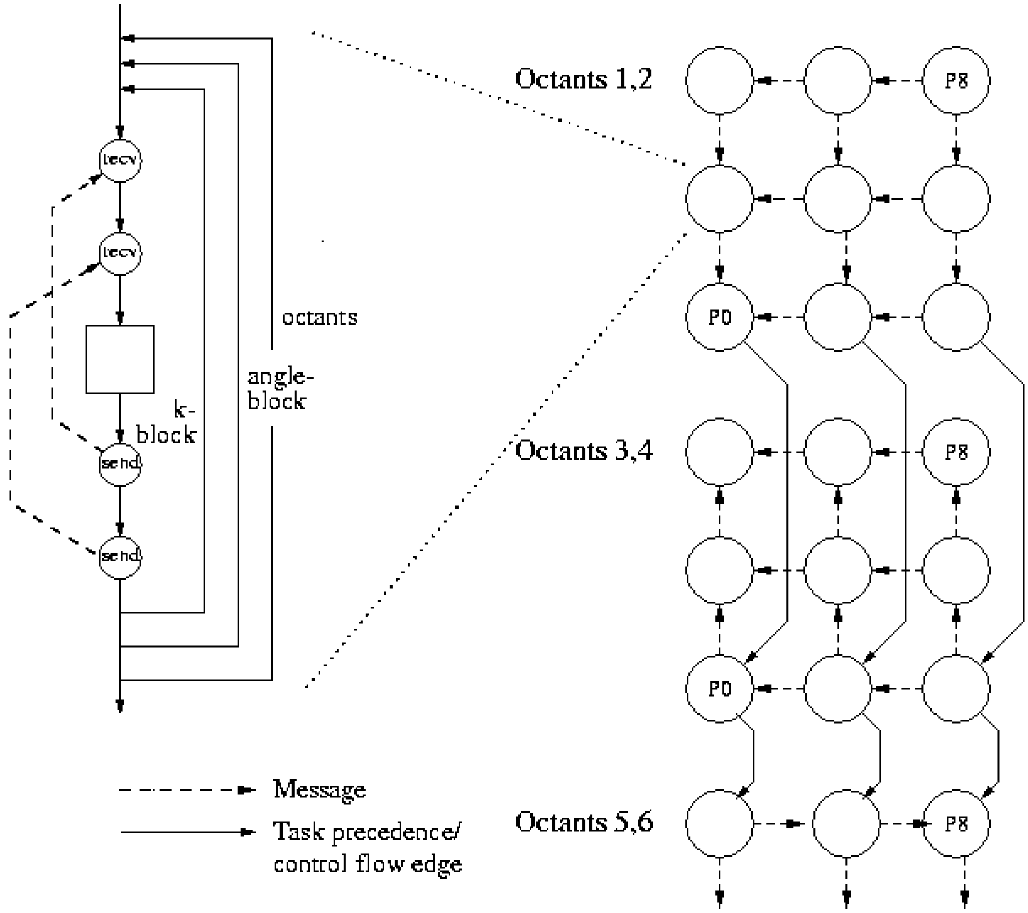


Figure 3.3. Static Task Graph for Sweep3D and the Dynamic Communication Structure on a 3x3 Processor Grid.

The right-hand side of Figure 3.3 shows the dynamic communication pattern of the sweep phase that would be realized assuming a 3×3 processor grid. This shows four of the eight octants in each time step of Sweep3D, and the computation and communication tasks performed by each processor for each octant can be seen on the left. An illustration of the dynamic task graph and a more detailed description can be found in [3].

3.3 Modeling Using Compositional Objects

The interfaces of compositional objects carry sufficient information to enable compiler-based integration of multi-domain, multi-paradigm, multi-resolution component models into a system model. Section 3.3.1 defines compositional objects. Section 3.3.2 defines the interfaces of compositional objects. Section 3.3.3 describes how compositional objects are incorporated into the dependence graph model of computation and Section 3.3.4 illustrates compositional modeling with an example from Sweep3D.

3.3.1 Compositional Objects

The POEMS Specification Language and programming environment enable creation of performance models as instances of the general dependence graph model of parallel computation. The nodes of the graph are instances of compositional objects that represent components. Compositional objects are defined in the context of an application semantic domain, an OS/runtime semantic domain, and/or a hardware semantic domain. The properties of objects are defined in terms of the attribute set of the appropriate semantic domain.

Each component is specified by a set of attributes. A component model is an instantiation of a component. There may be many instances of a given component model each with the same set of attributes, but with different values bound to these attributes.

3.3.2 Associative Interfaces

An associative interface is an extension of the associative model of communication [11,12] used to specify complex, dynamic interactions among object instances. An associative interface specifies the functions it implements, domain-specific properties of the functions it implements, the functions it requires, and the domain-specific properties of the functions it requires.

An associative interface has two elements: an "*accepts*" interface for the services that the component model implements and a "*requests*" interface that specifies the services the component model requires. Interfaces are specified in terms of the attributes that define the behavior and the states of standard objects [13,31,33]. An object that has associative interfaces is said to be a compositional object and an object that interacts through associative interfaces is said to have associative interactions.

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 a domain. 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 mode of interaction such as call-return or data flow (transfer of control) and/or 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 a domain. The selector references attributes from the profiles of target objects. When evaluated using the attributes of a profile, a selector is said to match the profile whenever it evaluates to true. A match that causes the selector to evaluate to true selects an object as the target of an interaction. The parameters of the transaction in the match should either conform or the POEMS component model library must include a routine to translate the parameters of the transaction in the *requests* interface instance to the parameters of the transaction in the matched *accepts* interface and vice versa. A compositional object may have multiple *accepts* and *requests* in its associative interface. Multiple accepts arise when a component model implements more than one behavior. A component model may have a request instance for a service from an implementing domain and a request instance for continuation of execution in its own domain.

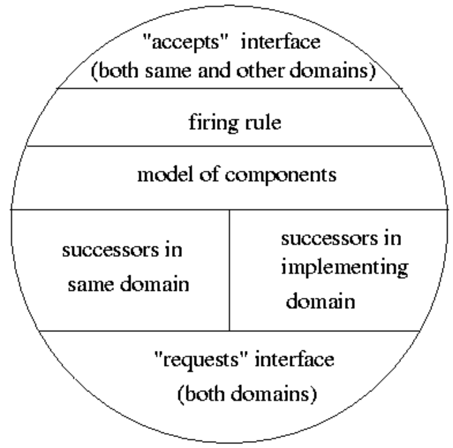


Figure 3.4. A Node of a Dependence Graph as a Compositional Object.

3.3.3 Mapping of Compositional Objects to Dependence Graphs

POEMS compositional objects are defined by encapsulating "standard objects" [13,31,33] with associative interfaces and data-flow execution semantics. Figure 3.4 shows the structure of a POEMS compositional object. The edges of the generalized dependence graph defined in Section 3.2 are derived by matching request interfaces with accepts interfaces. The selector must evaluate to true and the transaction must be conformable for an edge to be created. Transactions are conformable if the parameters of the transaction can be mapped to one another. The requirement for the mapping of parameters arises when component models in a system model either are defined at different levels of resolution or use different paradigms for evaluation. The POEMS compiler uses a library of domain-aware type coercions to implement mappings when required. The firing rule for a component model is derived by requiring that all of the parameters in the transaction in the interface be present on its input edges. Requests and accepts interfaces can be matched when a system model is compiled or at runtime.

The matching of *selectors* in *requests* to *profiles* in *accepts* thus composes a dynamic data flow graph and controls the traversal of the graph that models the execution behavior of the system.

Note that a component model at the application or OS/runtime level may have dependence arcs to and from its own and implementing levels.

3.3.4 Illustration of Compositional Modeling for Sweep3D

As an example of compositional modeling, the specification of two of the component models that are composed to create an end-to-end model of Sweep3D are illustrated in Figures 3.5 and 3.6. The target hardware system is an IBM SP/2, except that the memory hierarchy of the 604e processors is modeled as being different from that of the SP/2. (This modification is particularly relevant because measurements have shown that Sweep3D utilizes

Accepts:

Profile = {*domain* = application[Sweep3D], *node type* = k-block, *evaluation mode* = simulation[SimpleScalar] };

Transaction = { k-block(k-block-code) };

Protocol = *dataflow*;

Body:

/* Invocation of the SimpleScalar simulator to execute the instructions that comprise the computation of this k -block. The instructions for the k-block (and the other nodes) have been previously loaded into SimpleScalar's data space as the memory contents of the processor/memory pair being simulated. Initialization of SimpleScalar is done using an initialization component in the POEMS database. The parameter of the transaction is a pointer to the base address in memory for the instructions. */

Requests:

Selector = {*domain* = hardware [processor/memory], *node type* = N/A, *evaluation mode* = simulation [SimpleScalar]};

Transaction = {execute_code_block(k-block-code};

Protocol = *call-return*;}

Selector = {*domain* = Application[Sweep3D], *node type* = send, *evaluation mode* = simulation[MPI-Sim]};

Transaction = {MPI-send(destination, message, mode)};

Protocol = *dataflow* ;

Figure 3.5. Specification of Compositional Objects: an Example Using the Representation of a "k-block" Node of the Sweep3D Task Graph.

only about 20%-25% of the available cycles on a single processor in high-performance systems [35].)

This example assumes that the Sweep3D application being modeled is represented by the task graph of Figure 3.3 with "recv" nodes, "k-block" nodes, and "send" nodes. It also is assumed that the analyst has specified that execution of the compute operations be modeled using detailed instruction-level simulation, including a detailed simulation of memory access, and that the execution of communication and synchronization operations be modeled using MPI simulation.

This example involves the application domain, the OS/runtime domain, and the hardware domain. (In the interest of brevity and clarity, the syntax is a simplified form of the actual syntax of PSL.) In the example that follows, the PSL interfaces for a "k-block" node of the dependence graph and the first "send" node after a "k-block" node are specified. In the specification for the "k-block" component model the profile element "domain = application[Sweep3D]" specifies that this component is from the Sweep3D application. "Evaluation mode = simulation[SimpleScalar]" specifies that the evaluation of this component will be done by the SimpleScalar simulator. In this case the "k-block" is a sequence of instructions to be executed by the SimpleScalar simulator. The PSL compiler composes these component model specifications into an instance of the generalized hierarchical dependence graph defined in Section 3.2.1.

Accepts:

Profile = { *domain* = application[Sweep3D], *node type* = send, *evaluation mode* = simulation[MPI-Sim] };

Transaction = {MPI-send(destination, message, mode)};

Protocol = *dataflow*;

Body:

/* Invocation of the MPI-Sim simulator to simulate the modified MPI send operation specified in the transaction. */

Requests:

Selector={ *domain* = runtime, *node type* = N/A, *evaluation mode* = simulation[MPI-Sim] };

Transaction ={MPI-send(destination, message, mode) };

Protocol = *call-return*;

Selector = { *domain* = Application[Sweep3D], *node type* = send, *evaluation mode* = simulation[MPI-Sim] };

Transaction = {MPI-send(destination, message, mode)};

Protocol = *dataflow*;

Figure 3.6. Specification of Compositional Objects: an Example Using the Representation of a "send" node of the Sweep3D Task Graph.

The *accepts* interface for the compositional object representing the k-block node is straightforward. The k-block node has been generated by the task graph compiler and belongs to the application domain. The identifier "k-block" specifies a set of instructions that are to be executed. The *requests* interface for the k-block node has two request instances, one for the implementing (hardware) domain for the k-block and a second request instance continuing the flow of control to the next send node in the application domain.

The first request in the *requests* interface selects the SimpleScalar simulator to execute the instructions of the k-block node and evaluate the execution time of the code for the k-block node. The transaction associated with the first selector invokes an execute code block entry point defined for SimpleScalar. The protocol for this selector is call-return since SimpleScalar must complete its execution and return control to the k-block node before the k-block node can transfer the computed data to the send node and send

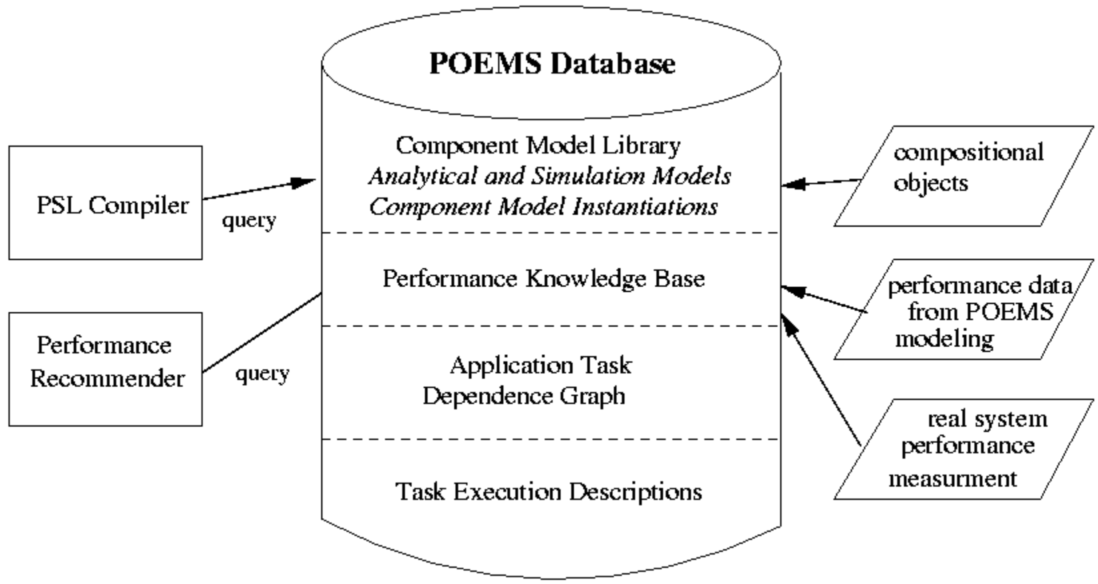


Figure 3.7. Overview of the POEMS Database.

node execution can be initiated. SimpleScalar has been encapsulated in the POEMS library as a single node dependence graph in the hardware domain.

The second *selector* in the requests interface selects the send node immediately following the k-block instance in the task graph to continue execution of the system model. The transaction in this request invokes the MPI-Sim simulator to evaluate the execution time of the MPI-send function defined in MPI-Sim. The protocol is the "data-flow" protocol for data-flow graphs; in this case, the control follows the data along the arc that is defined by the matching of the request and the accept interfaces. Note that even though MPI-Sim is in the runtime domain we have assigned the send node to the application domain. This choice was made because the send node is constructed as a part of the application by the task graph compiler (See Section 4.1 for a description of the task graph compiler.) This is the most direct translation of the task graph into a data-flow graph. MPI-Sim is encapsulated in the POEMS component model library as a single-node graph. MPI-Sim is invoked from the send node of the application send node with call-return semantics. This encapsulation of the runtime domain MPI-Sim node in the application node allows the send node to pass control to its successor send node as soon as the MPI-Sim simulator has completed execution.

For this example, the representation of the first "send" node following the k-block node in the task graph for Sweep3D is given in Figure 3.6. As shown, the profile in the *accepts* interface of the send component model specifies that it derives from the Sweep3D application, is a "send" node, and is to be evaluated using MPI-Sim. The transaction specifies an entry point of MPI-Sim.

The selector in the first request instance matches the MPI-Sim simulator and the transaction specifies the entry point for the MPI-send operation. The protocol is call-return because MPI-Sim must complete execution and return control to the application before this send node can transfer control to the next send node. The selector of the second request instance identifies the next node to which control should be transferred.

The accepts and requests interfaces of each compositional object or task graph node direct the search of the POEMS database by the POEMS compiler to identify appropriate component models to be used to instantiate components and link them to generate this segment of the system model. (The current version of the POEMS Specification Language compiler [16] does not yet interface to the POEMS database. It accesses component models from a Unix file system.) This database search provides a link to the code that implements an instance of the specified component model. In this case, the code that implements the k-node component model is the SimpleScalar simulator and the code that implements the send node is MPI-Sim. (See Section 4 for a description of SimpleScalar and MPI-Sim.) The accepts and requests interfaces, the link to the code implementing the instance of a component model, and other parameters associated with the component model will be stored in a Task Execution Description (TED) (see Section 3.4) in the POEMS database.

To execute the specified system model, SimpleScalar and MPI-Sim execute as processes on a host processor. SimpleScalar runs as the executing program for which MPI-Sim is modeling communication. SimpleScalar takes as input the executable file of the k-block node, which is stored in simulated memory. At the conclusion of executing the “k-block” node, the POEMS environment invokes the MPI-send module of MPI-Sim. A special built-in interface procedure that links SimpleScalar and MPI-Sim copies the data to be transmitted from the simulated memory of SimpleScalar into the real memory of the host, which allows MPI-Sim to model the communication operation.

3.4 Task Execution Descriptions (TEDs)

Considerable information is needed to characterize the behavior and properties of each component and different instances of each component model will have different attribute values. For example, there may be two instances of a component, one that is analytically evaluated and one that is evaluated by simulation. Analytical modeling may require parameter values that specify task execution time, while simulation of a single task requires either an executable representation of the task or its memory address trace. As a result, these two instances of the same component require different modes of integration into a system model. The information necessary to accomplish these integration functions must be associated with each component instance. POEMS will use a Task Execution Description (TED) to describe the modeled execution of a task; a TED is associated with each node of a task graph.

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 example, a TED would define the input parameters for SimpleScalar that would enable the creation a particular instantiation of the SimpleScalar component model.

3.5 Performance Recommender

The POEMS Performance Recommender system facilitates the selection of computational parameters for widely used algorithms to achieve specified performance goals. For example, in the Sweep3D context, the system 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 processors). Capturing the results of system measurement as well as modeling studies (discussed in Section 5), this facility can provide insight into how inter-relationships among variables and problem features affect application performance. It functions at several levels ranging from the capture of analytical and simulation model results to those of the measured application.

POEMS is using a kernel (IFESTOS), developed at Purdue [29], 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 application (e.g., Sweep3D). The performance data management subsystem provides facilities for the selective editing, viewing, and manipulation of the generated information. 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 [29]. Currently it is being applied to the Sweep3D application.

3.6 POEMS Database

The volume and complexity of the data environment for POEMS make the POEMS database a critical component of the project. In fact, POEMS as a tool could not be built without a database as a searchable repository for a wide spectrum of model and performance data. The POEMS database will be the repository for:

a) The component model definitions as compositional objects and component model definitions as instances of the component models.

b) Static task graphs for the applications, generated by the extended dHPF compiler.

c) The Task Execution Descriptions, which characterize each component model, as discussed in Section 3.3.4.

d) The knowledge base, which will guide analysts and designers in the development of total systems.

e) Measurements of performance in several formats included measurements and predictions of execution time.

The POEMS Specification Language compiler will be interfaced to the database as will the Performance Recommender. Figure 3.7 is a schematic of the role of the database in POEMS.

4 THE INITIAL POEMS PERFORMANCE ANALYSIS TOOLS

Currently, several performance analysis tools are being integrated in POEMS. These include the following tools, which are described briefly in this section: an automatic task graph generator [3], the LogGP [5] and LoPC [18] analytic models, the MPI-Sim simulator [28], and the SimpleScalar instruction-level, processor/memory architecture simulator [14]. Each tool contains capabilities for modeling the application, OS/runtime, and hardware domains. Together, these tools provide the capability to analyze, with a fairly high degree of confidence, the performance of current and future applications and architectures, for very large and complex system configurations. As a pre-requisite to developing multi-paradigm models, each of these tools has been used to develop a single-paradigm, multi-domain model of Sweep3D. This has allowed us to understand the unique role that each paradigm can play in total system performance analysis.

4.1 Automatic Task Graph Generator

A comprehensive performance modeling environment like POEMS will be used by system designers in practice only if model construction and solution can be largely automated. For an existing application, one critical step towards this goal is to automatically construct the application representation described in Section 3.2. Data-parallel compiler technology from the dHPF compiler project [2] has been extended to compute the task-graph-based application representation automatically for High Performance Fortran (HPF) programs. In normal use, the dHPF compiler compiles a given HPF program into an explicitly parallel program in SPMD (Single Program Multiple Data) form, for message-passing (MPI), shared-memory (e.g., pthreads), or software distributed shared-memory (TreadMarks [6]) systems. The synthesized task graphs represent this explicitly parallel program. In the future, the task graph synthesis will be extended to handle existing message-passing programs as well.

There are three key innovations in the compiler support for task graph construction:

- *The use of symbolic integer sets and mappings*: These are critical for capturing the set of possible dynamic task graphs as a concise, static task graph. Although this is a design feature of the representation itself, the feature depends directly on the integer set framework technology that is a foundation of the dHPF compiler [2]. The construction of these sets and mappings for an STG is described in more detail in [3].
- *Techniques for condensing the static task graph*: The construction of static task graphs in dHPF is a two-phase process [3]. First, an initial static task graph with fine-grain tasks is constructed directly from the dHPF compiler's internal program representation and analysis information. Second, a pass over the STG condenses fine-grain computational tasks (e.g., single statements) into coarse-grain tasks (e.g., entire loop iterations or even entire loop nests). The degree of granularity clearly depends on the goals of the modeling study, and can be controlled as discussed below.
- *Integer set code generation techniques for instantiating a dynamic task graph for a given input:* Although this step is conceptually performed outside the compiler, it can be done in a novel, highly-efficient, manner for many programs using dHPF's capability of generating code to enumerate the elements of an integer set or mapping. This also is explained further below.

The goal of condensing the task graph is to obtain a representation that is accurate, yet of a manageable size. For example, it may be appropriate to assume that all operations of a process between two communication points constitute a single task, permitting a coarse-grain modeling approach. In this case, in order to preserve precision, the scaling function of the condensed task must be computed as the symbolic sum of the scaling functions of the component tasks, each multiplied by the symbolic number of iterations of the surrounding loops or by the branching probabilities for surrounding control-flow, as appropriate. Note, however, that condensing conditional branches can introduce fairly coarse approximations in the modeling of task execution times. The compiler, therefore, takes a conservative approach and does not collapse branches by default. Instead, it is desirable to allow the modeler to intervene and specify that portions of the task graph can be collapsed even further (e.g., by inserting a special directive before control flow that can be collapsed). Even in the default case, however, the compiler can increase the granularity of tasks further in some cases by moving loop-invariant branches out of enclosing loops, a standard compiler transformation. For example, this would be possible for a key branch within the *k*-block of Sweep3D.

Constructing the DTG for a given program input requires symbolic interpretation of the parallel structure and part of the control flow of the program. This interpretation must enumerate the loop iterations, resolve all dynamic instances of each branch, and instantiate the actual tasks, edges, and communication events. For many regular, non-adaptive codes, the control flow (loop iterations and branches) can be determined uniquely by the program input, so that the DTG can be instantiated *statically*.

$$\textbf{Send (message length} < \textbf{4KB)} = \mathbf{o} \tag{1a}$$

$$\textbf{Send (message length} \geq \textbf{4KB)} = \mathbf{o_s + L + o_s + o_s + L + o_{\mathit{l}}} \tag{1b}$$

$$\textbf{Receive (message length} < \textbf{4KB)} = \mathbf{o} \tag{2a}$$

$$\textbf{Receive (message length} \geq \textbf{4KB)} = \mathbf{o_s + L + o_{\mathit{l}} + (message\_size \times G_{\mathit{l}}) + L + o_{\mathit{l}}} \tag{2b}$$

$$\textbf{Total Comm (message length} < \textbf{4KB)} = \mathbf{o + (message\_size \times G) + L + o} \tag{3a}$$

$$\textbf{Total Comm (message length} \geq \textbf{4KB)} = \mathbf{o_s + L + o_s + o_s + L + o_{\mathit{l}} + (message\_size \times G_{\mathit{l}}) + L + o_{\mathit{l}}} \tag{3b}$$

$$\mathbf{W_{i,j} = W_g \times mmi \times mk \times it \times jt} \tag{4}$$

$$\mathbf{StartP_{i,j} = max\,(StartP_{i-1,j} + W_{i-1,j} + Total\_Comm + Receive,\ StartP_{i,j-1} + W_{i,j-1} + Send + Total\_Comm)} \tag{5}$$

$$\mathbf{T_{5,6} = startP_{1,m} + 2[(W_{1,m} + Send_E + Receive_N + (m-1)L) \times \#k\text{-}blocks \times \#angle\text{-}groups]} \tag{6}$$

$$\mathbf{T_{7,8} = startP_{n-1,m} + 2[(W_{n-1,m} + Send_E + Receive_W + Receive_N + (m-1)L + (n-2)L) \times \#k\text{-}blocks \times \#angle\text{-}groups] + Receive_W + W_{n,m}} \tag{7}$$

$$\mathbf{T = 2\,(T_{5,6} + T_{7,8})} \tag{8}$$

Figure 4.1. LogGP Model of MPI Communication and the Sweep3D Application.

(Again, in some cases, a few data-dependent branches may have to be ignored for approximate modeling, under the control of the modeler as proposed above.) To instantiate parallel tasks or edges, the elements of the corresponding integer set or mapping must be enumerated. The key to doing this is that, for a given symbolic integer set, the dHPF compiler can synthesize code to enumerate the elements of that set [2]. (Any mapping can be converted into an equivalent set for code generation.) We exploit this capability and generate a separate procedure for each set, parameterized by the symbolic program variables that appear in the set; typically these are process identifiers, program input variables, and loop index variables. Then this generated code is compiled separately and linked with the program that performs the instantiation of the DTG. When instantiating dynamic task instances for a given static task, the code for that task's symbolic set is invoked and is provided with the *current values of the symbolic parameters* at this point in the interpretation process. This directly returns the required index values for the task instances. Edge instances of a static edge are instantiated in exactly the same manner, from the code for the integer set mapping of that static edge.

In some irregular and adaptive programs, the control flow may depend on intermediate computational results of the program, and the DTG would have to be instantiated dynamically using an actual or simulated program execution. The DTG for a given input can be either synthesized and stored offline for further modeling with any model, or instantiated on the fly during model execution for modeling techniques such as execution-driven or instruction-driven simulation. Both these approaches will be key for multi-paradigm modeling of advanced adaptive codes.

The automatic construction of the static task graph has been exploited directly in integrating a task-graph-based model with MPI-Sim for improving the scalability of simulation, as described in Section 6.1. The automatic construction of the dynamic task graph makes it possible to do automatic analytical modeling of program performance, using deterministic task graph analysis [1].

4.2 LogGP/LoPC

The task graph for a given application elucidates the principal structure of the code, including the inter-processor communication events, from which it is relatively easy to derive the LogGP or LoPC model equations. The approach is illustrated by deriving the LogGP model of the Sweep3D application that uses blocking MPI send and receive primitives for communication. The task graph for the sweep (or main) phase of this application is given in Figure 3.3.

The LogGP model of the Sweep3D application is given in Figure 4.1. The hardware domain is modeled by three simple parameters: L, G, and W_g, which are defined below.

The first six equations, (1a) through (3b), model the runtime system components used by the Sweep3D application. That is, these equations model the MPI communication primitives as they are implemented on the SP/2 that was used to validate the model. Equations (4) through (8) model the execution time of the application as derived from the dynamic task graph.

Equations (1a) through (3b) reflect a fairly precise, yet simple, model of how the MPI-send and MPI-receive primitives are implemented on the SP/2. Information about how the primitives are implemented was obtained from the author of the MPI software. For messages smaller than four kilobytes, the cost of a send or receive operation is simply the LogGP processing *overhead* (o) parameter. The total communication cost for these messages (equation 3a) is the sum of the send processing overhead, the message transmission time (modeled as the network *latency* (L), plus the message size times the *gap per byte* (G) parameter), and the receive processing overhead.[1] A subscript on the processing overhead parameter denotes the value of this parameter for messages smaller (o_s) or larger (o_l) than one kilobyte. When the subscript is omitted, the appropriate value is assumed. For messages larger than four kilobytes (equation 3b), an additional handshake is required. The sending processor sends a short message to the receiving processor, which is acknowledged by a short message from the receiving processor when the receive system call has been executed and the buffer space is available to hold the message. After that, the message transfer takes place. In this case, the Send cost (1b) or Receive cost (2b) is the duration of the communication event on the processor where the corresponding MPI runtime system call occurs. Further details about the accuracy of these communication models and how the parameter values were measured are given in [36].

The equations that model the MPI communication primitives might need to be modified for future versions of the MPI library, or if Sweep3D is run on a different message-passing system or is modified to use non-blocking MPI primitives. The equations illustrate a general approach for capturing the impact of such system modifications.

Equations (4) through (8) model the application execution time, taking advantage of the symmetry in the Sweep3D task graph (see Figure 3.3). For simplicity, the Sweep3D application model presented here assumes each processor in the $m \times n$ Sweep3D processor grid is mapped to a different SMP node in the SP/2. In this case, network latency, L, is the same for all (nearest-neighbor) communication in Sweep3D. As explained in [36], the equations that model communication can be modified easily for the case when 2x2 regions of the processor grid are mapped to the same SMP node.

Equation (4) models the time required for a single task to compute the values for a portion of the grid of size mmi $\times$ mk $\times$ it $\times$ jt. In this equation, W_g is the measured time to compute one grid point, and *mmi*, *mk*, *it*, and *jt* are the Sweep3D input parameters that specify the number of angles and grid points per block per processor.

Equation (5) models the precedence constraints in the task graph for the sweeps for octants 5 and 6, assuming the processors are numbered according to their placement in the two-dimensional grid, with the processor in the upper left being numbered (1,1). Specifically, the recursive equation computes the time that processor $p_{i,j}$ begins its calculations for these sweeps, where i denotes the horizontal position of the processor in the grid. The first term in equation (5) corresponds to the case where the message from the West is the last to arrive at processor $p_{i,j}$. In this case, due to the blocking nature of the MPI primitives, the message from the North has already been sent but cannot be received until the message from the West is processed. The second term in equation (5) models the case where the message from the North is the last to arrive. Note that $StartP_{1,1} = 0$, and that the appropriate one of the two terms in equation (5) is deleted for each of the other processors at the east or north edges of the processor grid.

The Sweep3D application makes sweeps across the processors in the same direction for each octant pair. The critical path time for the two right-downward sweeps is computed in equation (6) of Figure 4.1. This is the time until the lower-left corner processor $p_{1,m}$ has finished communicating the results from its last block of the sweep for octant 6. At this point, the sweeps for octants 7 and 8 (to the upper right) can start at processor $p_{1,m}$ and proceed toward $p_{n,1}$. The subscripts on the Send and Receive terms in equation (6) are included only to indicate the direction of the communication event, to make it easier to understand why the term is included in the equation.

Since the sweeps from octants 1 and 2 (in the next iteration) will not begin until processor $p_{n,1}$ is finished, the critical path for the sweeps for octants 7 and 8 is the time until all processors in the grid complete their calculations for the sweeps. Due to the symmetry in the Sweep3D algorithm, captured in the task graph, the time for the sweeps to the Northeast is the same as the total time for the sweeps for octants 5 and 6, which is computed in equation (7) of the figure. Due to the symmetry between the sweeps for octants 1 through 4 and the sweeps for octants 5 through 8, the total execution time of one iteration is computed as in equation (8) of Figure 4.1. Equation (6) contains one term [$(m-1)L$], and the equation (7) contains

[1] The communication structure of Sweep3D is such that we can ignore the LogGP *gap* (g) parameter, since the time between consecutive message transmissions is greater than the minimum allowed value of inter-message transmission time.

two terms [$(m–1)L$ and $(n-2)L$], that account for synchronization costs, as explained in [36].

The input parameters to the LogGP model derived above are the L, o, G, P, and W_g parameters. The first three parameters were derived by measuring the round-trip communication times for three different message sizes on the IBM SP system, and solving equations (3a) and (3b) with the derived measures (see [36] for details). The $W_{i,j}$ parameter value was measured on a 2x2 grid of processors so that the work done by corner, edge, and interior processors could be measured. In fact, to obtain the accuracy of the results in this paper, $W_{i,j}$ for each per-processor grid size was measured to account for differences (up to 20%) that arise from cache miss and other effects. Since the Sweep3D program contains extra calculations ("fixups") for five of the twelve iterations, $W_{i,j}$ values for both of these iteration types were measured. Although this is more detailed than the creators of LogP/LogGP may have intended, the increased accuracy is substantial and needed for the large-scale performance projections in Section 5. In the future, other POEMS tools will be used to obtain these input parameters, as explained in Section 6.3.

The LogGP model of the SP/2 MPI communication primitives is shown to be highly accurate in [36]. Selected validations and performance projections of the LogGP model of Sweep3D are presented in Section 5.

4.3 MPI-Sim: Direct Execution-Driven System Simulation

POEMS includes a modular, direct execution-driven, parallel program simulator called MPI-Sim that has been developed at UCLA [10, 28]. 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, and message-passing library implementations. The simulator also can be used to evaluate the performance of parallel file systems and I/O systems [8]. Supported capabilities include a number of different disk caching algorithms, collective I/O techniques, disk cache replacement algorithms, and I/O device models. The parallel discrete-event simulator uses a set of conservative synchronization protocols together with a number of optimizations to reduce the time to execute simulation models.

MPI-Sim models the application and the underlying system. An application is represented by its local code blocks and their communication requirements. The local code block model is evaluated by direct execution. MPI programs execute as a collection of single threaded processes, and, in general, the host machine has fewer processors than the target machine. (For sequential simulation, the host machine has only one processor). This requires that the simulator supports multithreaded execution of MPI programs. MPI-LITE, a portable library for multithreaded execution of MPI programs, has been developed for this purpose.

The MPI communication layer, which is part of the runtime domain, is simulated by MPI-Sim in detail; buffer allocation and internal MPI synchronization messages are taken into account. The simulator does not simulate every MPI call, rather all collective communication functions are first 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. A preprocessor replaces all MPI calls by equivalent calls to corresponding routines in the simulator. The physical communications between processors, which are part of the hardware domain are modeled by simple end-to-end latencies, similar to the communication latency parameter in the LogP model.

For many applications, these runtime domain and hardware domain communication models are highly accurate [10,28]. MPI-Sim has been validated against the NAS MPI benchmarks and has demonstrated excellent performance improvement with parallel execution against these benchmarks [28]. As shown in Section 5, it also is highly accurate for the Sweep3D application, and the simulation results (which reflect fairly detailed modeling of the runtime domain) greatly increase confidence in the scalability projections of the more abstract LogGP model.

4.4 Instruction-Level Simulation

As stated above, the system model provided by MPI-Sim is based on direct execution of computational code and simulation of MPI communication. As a result, the processor/memory architecture of the target system to be evaluated must be identical to that of the host system.

To provide performance evaluation of applications on alternative (future) processor and memory subsystem designs, the POEMS component model library includes processor, memory, and transport component models that are evaluated using instruction-level, discrete-event simulation. These models are in the process of being composed with MPI-Sim to predict the performance of programs for proposed next-generation multiprocessor systems. The results of the more abstract task graph, LogGP/LoPC, and MPI-Sim analyses can be used to identify the most important regions of the design space to be evaluated with these, more detailed, hardware component models. The detailed hardware models can be used to validate the more abstract models, and can provide parameter values for future processor/memory architectures needed in the more abstract models.

In order to provide this kind of modeling capability, a simulator that models instruction-level parallelism is essential. The "sim-outorder" component of the SimpleScalar tool set [14] meets this requirement for simulating complex modern processor/memory architectures. As a result, it is being integrated in the POEMS environment. sim-outorder can be used to model state-of-the-art superscalar processors, which support out-of-order instruction issue and execution. The processor attributes of these hardware processor/memory subsystem component models include the processor fetch, issue, and decode rates, number and types of functional units, and defining characteristics of the branch predictor. Currently, its integrated memory components include level-one and level-two instruction and data caches, a translation-lookaside buffer, and main memory. The simulator is fairly portable and customizable with reasonable effort.

Currently, the MPI-version of Sweep3D successfully executes on multiple instantiations of sim-outorder, each executed by a separate MPI process. This simulation produces estimates of the parameter value called $W_{i,j}$ in the LogGP model, thus enabling the LogGP model to predict the performance of Sweep3D on alternative processor-cache architectures. This simulation capability also demonstrates a proof-of-concept for integrating the sim-outorder and MPI-Sim modeling tools. The future integration of MPI-Sim and sim-outorder will support detailed simulation of both the OS/runtime domain and the hardware domain.

5 APPLICATION OF POEMS TO PERFORMANCE ANALYSIS OF SWEEP3D

The goal of the POEMS performance modeling system is to enable complete end-to-end performance studies of applications executed on specific system architectures. To accomplish this, the system must be able to generate information that can be used to identify and characterize performance bottlenecks, analyze scalability, determine the optimal mapping of the application on a given architecture, and analyze the sensitivity of the application to architectural changes. This section presents a representative sample of the results obtained in modeling Sweep3D. These results are for the IBM SP system.

5.1 Model Validations

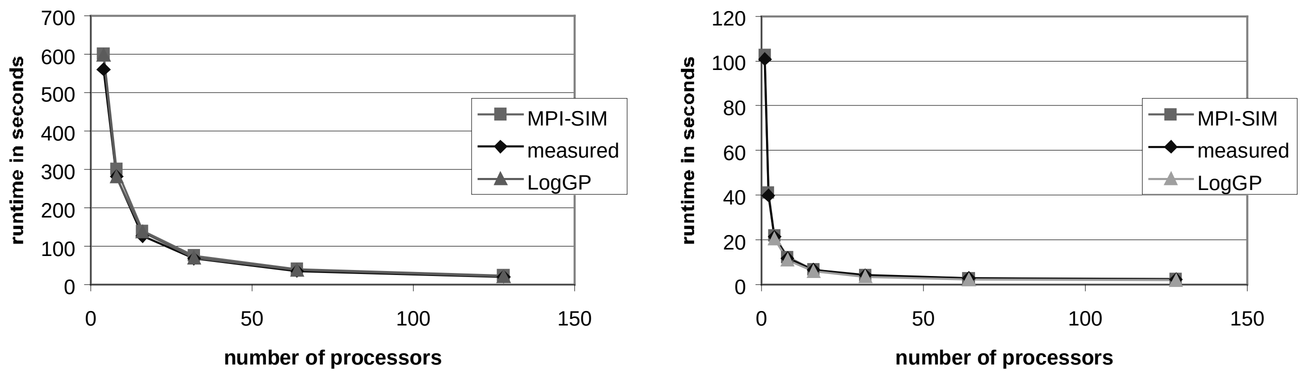
We first demonstrate the accuracy of the performance models described above. Both MPI-Sim and LogGP model Sweep3D accurately for a variety of application parameters such as the mk and mmi parameters that define the size of a pipelined block, different total problem sizes, and number of processors. Figures 5.1(a) and (b) present the measured and predicted execution time of the program for the 150^3 and 50^3 total problem sizes respectively, as a function of the number of processors. For these results, the k-blocking factor (mk) is 10 and the angle-blocking factor (mmi) is 3. Both the MPI-Sim and LogGP models show excellent agreement with the measured values, with discrepancies of at most 7%. The next section shows that these two problem sizes have very different scalability, yet Figure 5.1 shows that the LogGP and MPI-Sim estimates are highly accurate for both problem sizes.

5.2 Scalability Analysis

It is important to application developers to determine how well an application scales as the number of processors in the system is increased. On today's systems, the user could conduct such studies by measuring the runtime of the application as the number of processors is increased to the maximum number of processors in the system, e.g., 128 (see Figure 5.2(a)). The figure clearly shows that the small problem size (50^3) cannot efficiently exploit more than 64 processors. On the other hand, the larger problem size shows excellent speedup up to 128 processors. However, due to the system size limitation, no information about the behavior of larger problem sizes on very large numbers of processors is available to the user. Modeling tools enable users to look beyond the available multiprocessor systems. Figure 5.2(b) shows the projections of LogGP and MPI-Sim to hardware configurations with as many as 2500 processors. Although the application for the 150^3 problem size shows good performance for up to 900 processors, it would not perform well on machines with a greater number of processors.

Figure 5.2 also demonstrates the excellent agreement between the analytical LogGP model and the MPI-Sim simulation model. Each model was independently validated for a variety of application configurations on as many as 128 processors, and the good cross-validation between the models for up to 2500 processors increases our confidence in both models.

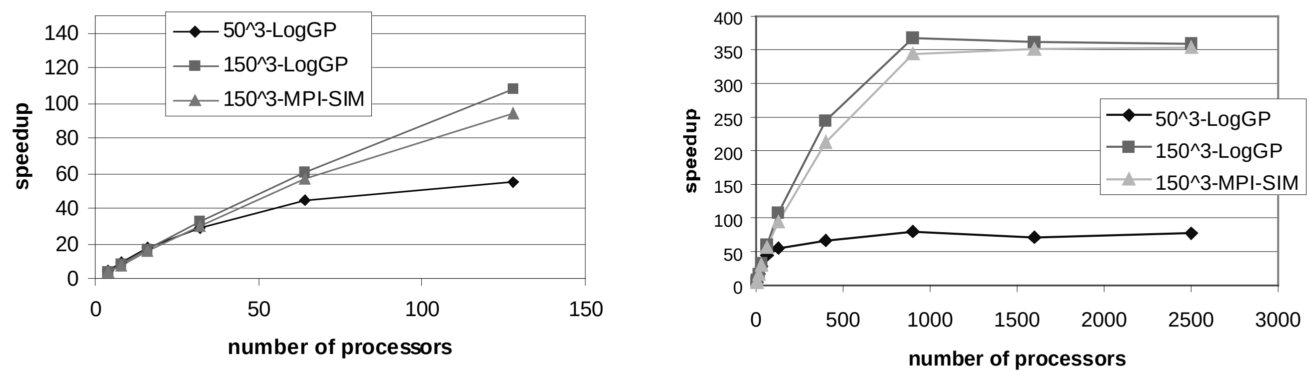
Figure 5.3(a) demonstrates the projective capability of the simulation and analytical models. This figure shows the measured and estimated performance as a function of the number of processors, for a fixed per-processor grid size of 14×14×255. For this per-processor grid size the total problem size on 400 processors is approximately 20 million grid points, which is one problem size of interest to the application developers. Obviously, measurement is limited to the size of the physical system (128 processors). The maximum problem size that can be measured, with the per-processor grid size of 14×14×255 is 6.4 million cells. MPI-Sim can extend the projections to 1,600 processors (80 million cells) before running out of memory because



(a) Validation for Problem Size 150x150x150.

(b) Validation for Problem Size 50x50x50.

Figure 5.1. Validation of LogGP and MPI-Sim.



(a) Speedup for up to 128 Processors.

(b) Speedup for up to 2500 Processors.

Figure 5.2 Projected Sweep3D Speedup.

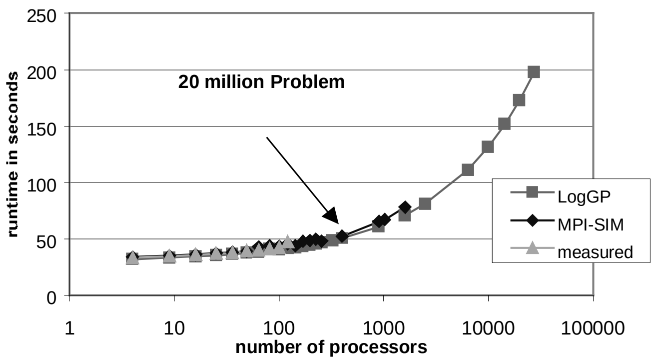
the simulator needs at least as much aggregate memory as the application it models. (In Section 6.1, we describe compiler-based techniques that eliminate the memory bottleneck for simulation of regular programs such as Sweep3D and thus greatly increase the system and problem sizes that can be simulated.) Finally, LogGP can take the projections to as much as 28,000 processors (i.e., for the given application configuration parameters, a 1.4 billion problem size).

One important issue addressed in the figure is the validity of the model projections for very large systems. In particular, the close agreement between measurement and the MPI-Sim and LogGP models for up to 128 processors and the close agreement between MPI-Sim and LogGP for up to 1,600 processors greatly increases our confidence in the projections of MPI-Sim and LogGP for large problem sizes of interest to the application developers.

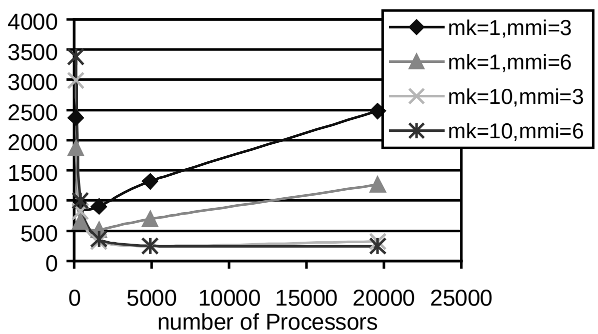
The mutual validation between the two different models is one key way in which simulation and analytical modeling complement each other. Furthermore, the two models have complementary strengths. The key strength of MPI-Sim is that it can be used to study program performance by users with little or no modeling expertise, and can be used to study the performance impact of detailed changes in the design of Sweep3D or in the implementation of the MPI communication primitives. The key strength of the LogGP model is that it can project the performance of design changes *before* the changes are implemented, and for the largest problem sizes of interest.

5.3 Application Mapping

For applications such as Sweep3D, which allow varying the degree of pipelining, it is often important to explore not only how many resources are needed to achieve good results, but also how best to map the application onto the machine. Figure 5.3(b) shows how LogGP explores the parameter space for the 20 million-cell Sweep3D problem



(a) Projective Capability of Analysis and Simulation (mk=10,mmi=3).



(b) Effect of Pipeline Parameters.

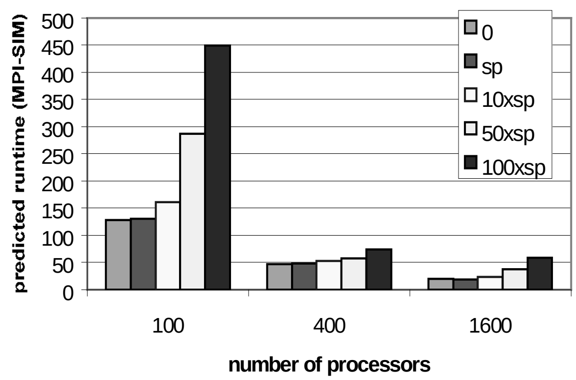
Figure 5.3. Projective Capabilities and Model Projections for Sweep3D, 20 Million Cell Problem.

on up to 20,000 processors. Here the k- and angle-blocking factors are varied, which results in various degrees of pipelining in the algorithm. The graph shows that for a small number of processors (less than 400) the choice of blocking factors is not very important. However, as more processors are used and the grid size per processor decreases, the degree of pipelining in the k-dimension has a significant impact, resulting in poor performance for the smallest blocking factor (mk=1). For this blocking factor, the performance of the application also is sensitive to the angle-blocking factor. However, when the k-blocking factor is properly chosen (mk=10), the impact of the mmi parameter value is negligible. The results also indicate that the optimal operating point for this total problem size is perhaps one or two thousand processors; increasing the number of processors beyond this number leads to greatly diminished returns in terms of reducing application execution time. This projected optimal operating point is a key design question that was unanswered prior to the POEMS analysis.

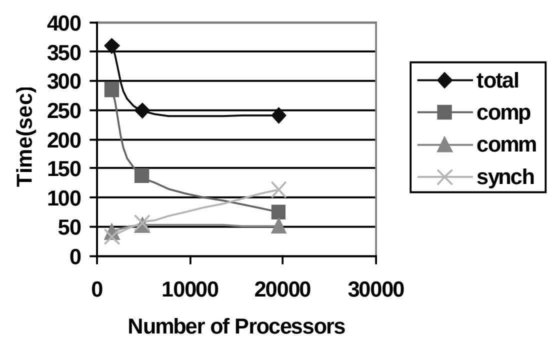
5.4 Communication Requirements Analysis

Another issue of interest is the application's performance on a system with upgraded communication capability. A related issue is the application's performance on high-performance tightly-coupled multiprocessor systems versus its performance on a network of loosely-coupled workstations. Both simulation and analytical models can be used to study these issues by modifying their communication components to reflect changes in the communication latency of the system.

As depicted in Figure 5.4, MPI-Sim was used in this way to show the impact of latency on the performance of the 20 million-cell Sweep3D. In this experiment, the latency was set to *n* times the latency of the IBM SP (denoted *n*×SP in the figure), for a range of values of *n*. The MPI-Sim projections shown in Figure 5.4(a) indicate that if the communication latency is 0, not much improvement is gained. In addition, these results show that for large numbers of processors, 400 and over, the application may



(a) Latency Variation (MPI-Sim).



(b) Communication Costs (LogGP).

Figure 5.4 Effect of Communications on Sweep3D, 20 Million Cell Problem.

perform well on a network of workstations, because even if the latency is increased to 50 times that of the SP, performance does not suffer significantly. However, as Figure 5.3(b) demonstrates, the performance of the 20 million-cell Sweep3D does not improve when more than 5000 processors are used. Since the grid-size per processor diminishes, reducing the computation time, and latency does not seem to be a factor, one would expect performance gains to persist. The detailed communication component in the LogGP model provides an explanation. As can be seen in Figure 5.4(b), the computation time decreases, but the communication time remains flat and, more importantly, the synchronization costs resulting from blocking sends and receives dominate the total execution time as the number of processors grows to 20,000. This implies that modifications to the algorithm are necessary to effectively use large numbers of processors. In addition, the use of non-blocking communication primitives might be worth investigating. Again these are key design results produced by the POEMS modeling and analysis effort.

6 RESEARCH IN PROGRESS - INTEGRATION OF MODELING PARADIGMS

One goal of the POEMS methodology is to facilitate the integration of different modeling paradigms. The use of compositional objects and the task graph abstraction as a workload representation together provide the framework needed for this integration. Compositional objects facilitate the use of different models for a particular system or application component. The task graph explicitly separates the representation of sequential computations (tasks) from inter-process communication or synchronization. This separation directly enables combinations of modeling paradigms where different paradigms are used to model various tasks as well as the parallel communication behavior. Other combinations (e.g., combining analysis and simulation to model the execution of a particular sequential task) can be performed with some additional effort, using task component models that are themselves composed of submodels. This section presents an overview of some of the multi-paradigm modeling approaches that are being designed and evaluated for inclusion in the POEMS environment.

6.1 Integrating Task Graph and Simulation Models

When simulating communication performance with a simulator such as MPI-Sim, the computational code of the application is executed or simulated in detail to determine its impact on performance. State-of-the-art simulators such as MPI-Sim use both parallel simulation and direct-execution simulation to reduce overall simulation time greatly, but these simulators still consume at least as much aggregate memory as the original application. This high memory usage is a major bottleneck limiting the size of problems that can be simulated today, especially if the target system has many more processors than the host system used to run the simulations.

In principle, only two aspects of the computations are needed to predict communication behavior and overall performance: (a) the *elapsed time* for each computational interval and (b) those *intermediate computational results* that affect the computation times and communication behavior. We refer to the computations required for part (b) as "essential" computations (signifying that their *results* affect program performance), and the rest of the computations as "non-essential." Essential computations are exactly those that affect the control-flow (and therefore computation times) or the communication behavior of the program. Assume for now that computation times for the non-essential computations can be estimated analytically using compiler-driven performance prediction. Then, if the essential computations can be isolated, the non-essential computations can be ignored during the detailed simulation, and the data structures used exclusively by the non-essential computations can be eliminated. If the memory savings are substantial *and* the simplified simulation is accurate, this technique can make it practical to simulate much larger systems and data sets than is currently possible even with parallel direct-execution simulation.

We have integrated the static task graph model with the MPI-Sim simulator (plus additional compiler analysis) in order to implement and evaluate the technique described above [4]. Compiler analysis is essential because identifying and eliminating the "non-essential" computations requires information about the communication and control-flow in the application. The static task graph model serves two purposes. First, it provides an abstract representation for the compiler analysis, in which the computation intervals (tasks) are clearly separated from the communication structure. Second it serves as the interface to MPI-Sim (in the form of a simplified MPI program that captures the task graph structure plus the "essential" computations).

Briefly, the integration works as follows. The static task graph directly identifies the computational intervals: these simply correspond to subgraphs containing no communication. First, the compiler identifies the *values* (i.e, the uses of variables) that affect the control-flow within the computation intervals, and the values that affect the communication behavior (note that this does not yet include the values being communicated). The compiler then uses a standard technique known as program slicing [20] to identify the subset of the computations that affect these variable values; these are exactly the essential computations.

Second, the compiler computes a symbolic expression representing the elapsed time for each non-essential computation interval. These task time estimates are similar to equation (4) of the LogGP model, but derived automatically from program analysis. The W_{ij} parameters can be measured directly, or estimated by more detailed analysis, or simulated using SimpleScalar as described in Sections 5.2 or 6.2. The current implementation simply measures these values by generating an instrumented version of the original source code for one or more relatively small problem sizes.

Finally, the compiler transforms the original parallel code into a simplified MPI program that has exactly the same parallel structure as the original task graph, but where the non-essential computations are replaced by function calls to a special simulator function. MPI-Sim has been augmented to provide this special simulator function, which takes an argument specifying a time value and simply advances the simulation clock for the current process by that value. The symbolic estimate derived by the compiler is passed as an argument to the function.

Preliminary results demonstrating the scalability of the integrated simulator can be seen in Figure 6.1. The per-processor problem size is fixed (6×6×1000) in the figure so that the total problem size scales linearly with the number of processors. The original MPI-Sim could not be used to simulate applications running on more than 400 processors in this case (i.e., an aggregate 14.4 million problem size), whereas the integrated task graph simulator model scaled up to 6400 processors (i.e., a 230 million problem size). The primary reason for the improved scalability of the simulation is that the integrated model requires a factor of 1760× less memory than the original simulator! The total simulation time is also improved by about a factor of 2. Finally, the integrated model has an average error under 10% for this problem, compared with an average error of 3.6% for the original simulator. In fact, for other cases we have looked at, the two approaches are comparable in their accuracy [4].

The composition of models described above was developed manually because substantial research issues were involved. In practice, the POEMS methodology for component model composition and evaluation tool integration described in Section 3 can be applied to perform this composition. The dHPF compiler would first generate a modified static task graph that creates separate tasks for intervals of essential and non-essential computations. (The code for each graph node in PSL is encapsulated in a separate function, mainly for implementation convenience.) The evaluation tool for the communication operations is of course the MPI-Sim simulator. The evaluation tool for the essential task nodes,

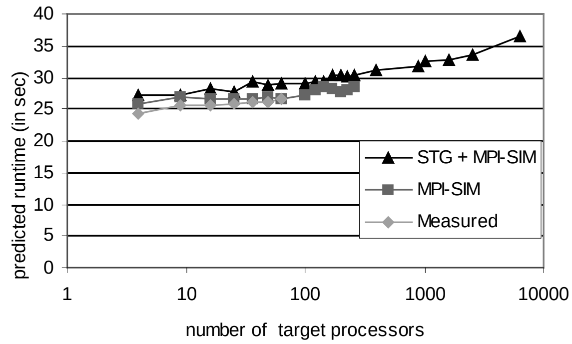


Figure 6.1. Aggregate Problem and System Sizes that can be Simulated with the Integrated Static Task Graph + MPI-Sim Model; Per-processor Problem Size is Fixed at 6×6×1000

including essential control-flow nodes, would be direct execution within the POEMS runtime system. Last and most important, several different evaluation tools can be used for the non-essential task nodes: compiler-synthesized symbolic expressions parameterized with measurements as in our experiments above (in fact, the POEMS environment would further simplify the measurement process), purely analytical estimates derived by the compiler, or the SimpleScalar processor simulator. Furthermore, different evaluation tools could be used for different nodes. For example, the SimpleScalar simulations could be used for a few instances of each node in order to model cache behavior on a hypothetical system, and these results used as parameter values in the symbolic expressions for modeling other instances of the node.

The specification of this integrated model in PSL is closely analogous to the example given in Section 3.3. The component models for the task graph nodes are MPI-Sim for communication nodes, direct execution for the essential computation nodes, and one of the evaluation methods described above for each of the non-essential computation nodes. The *accepts* interfaces for the component models implemented through MPI-Sim identify the position of the given computation or communication node in the static task graph, and the function to be modeled or the signature of the MPI component to be simulated. The *requests* interfaces specify the evaluation tools and the successor nodes in the task graph. The *accepts* interfaces of the non-essential computation nodes specify the function being encapsulated, the position in the static task graph of the given computation node, and an evaluation mode. The *requests* interfaces of the computation nodes specify the evaluation tools, the next communication node in the task graph, and the signature of the desired successor communication function. The PSL compiler would automatically generate a single executable that invokes the appropriate model components at runtime. An execution of this program essentially can be envisioned as synthesizing the dynamic task graph from the static task graph on the fly and traversing it, alternating between execution of computation nodes and communication nodes, where each node is evaluated or executed by its component model.

6.2 Integration of MPI-Sim and the LogGP Models

As discussed in Section 5, both analytical and simulation techniques can predict the performance of large-scale parallel applications as a function of various architectural characteristics. Simulation can model system performance at much greater levels of detail than analytical models, and can evaluate application performance for architectural modifications that would change the analytical model input parameters in unknown ways. As a simple example, MPI-Sim results were used to gain confidence in the very large-scale projections of Sweep3D performance from the LogGP model. On the other hand, because of the resource and time constraints of simulation, analytical models can elucidate the principal system performance parameters and can provide performance projections for much larger configurations than is possible with simulation models. Because of these complementary strengths, significant benefits can be derived from combining the two approaches.

One key advantage to further integrating MPI-Sim and the LogGP models is that performance of large-scale applications with modified implementations of the MPI communication primitives can be evaluated. For example, the MPI runtime software implementation on the SP/2 has not yet been optimized for communication among the processors in the same SMP node. For the experiments reported in Section 5, the Sweep3D processes were mapped to different nodes in the SP/2, in order to utilize the efficient MPI communication between nodes. To obtain performance estimates of Sweep3D for next generation systems, MPI-Sim component models can be used to simulate the efficient communication that is expected to be available in future versions of the MPI runtime library. The measured communication costs from these simulations can then be used in the LogGP model, appropriately updated to reflect non-uniform intra-node/inter-node communication costs [36], to predict application scalability when Sweep3D uses intra-node as well as inter-node communication in the SP/2.

The Sweep3D application that uses MPI communication primitives is accurately modeled by the LogGP and MPI-Sim models, which assume no significant queuing delays at the network interfaces or in the interconnection network switches. Other applications, including the shared-memory version of Sweep3D, may require estimates of queuing delays in the network, or the network interfaces, in order to achieve accurate performance models. In many previous studies, analytical models of contention in interconnection networks have proven to be both very efficient to evaluate and highly accurate. Thus, a detailed analytical component model for the interconnection network might be composed with (1) a more abstract model of the application running time (as in the LoPC model [18]), and/or (2) an MPI-Sim component model of the MPI runtime system communication primitives.

6.3 Integration of SimpleScalar with MPI-Sim and LogGP

Although both MPI-Sim and LogGP have flexible communication components, the processor model in both systems is simple and relies on measurement of the local code blocks on an existing processor and memory system. To allow the POEMS modeling capabilities to be expanded to include assessment of the impact of next generation processors and memory systems, detailed component models based on SimpleScalar are under development. These models interact with the MPI-Sim and LogGP component models to project the impact of processor and memory system architecture changes on the execution time of large-scale Sweep3D simulations (on thousands of processor nodes).

The modeling of specific processors by SimpleScalar still needs to be validated. For example, the POEMS SimpleScalar model of the Power604e is not exact because of certain aspects of the 604e; e.g., the modeled ISA is not identical to that of the 604e. A key question is whether the approximate model of the ISA is sufficiently accurate for applications such as Sweep3D. Several validation methods are under consideration:

1) execute narrow-spectrum benchmarks on a 604e and on SimpleScalar configured as a 604e to validate the memory hierarchy design parameters;
2) use on-chip performance counters to validate the microarchitecture modeling; and
3) compare measured Sweep3D task times on the SP/2 with task execution times attained by running multiple SimpleScalar instances configured as 604es under MPI.

One example of combined simulation and analysis, is to use the LogGP model to obtain application scalability projections for next-generation processor and cache architectures, by using estimates of task execution times for those architectures that are derived from SimpleScalar simulations. For example, in the Sweep3D application, a SimpleScalar component model can produce the $W_{i,j}$ values for a fixed per-processor grid size on a 2×2

processor grid, and then the LogGP component model can be run using the estimates for various per-processor grid sizes to project the performance scalability of the application. This is one of the near-future goals of the POEMS multi-paradigm modeling efforts. As another example, instead of simulating or directly executing the MPI communication primitives, a LogGP component model of the MPI communication costs might be composed with a SimpleScalar component model of the node architecture.

7 RELATED WORK

The conceptual framework for POEMS is a synthesis from models of naming and communication [11, 12], Computer-Aided Design (CAD), software frameworks, parallel computation [24], object-oriented analysis [33], data mediation [38] and intelligent agents. In this section, however, we focus on research projects that share our primary goal of end-to-end performance evaluation for parallel systems. 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 at http://www.cs.utexas.edu/users/poems.

The most closely related projects to POEMS are probably the Maisie/Parsec parallel discrete-event simulation framework and its use in parallel program simulation [7, 9, 27, 28], SimOS [32], RSIM [25,26], PACE [21], and the earlier work in program simulators, direct-execution simulators, and parallel discrete-event simulation. In addition, an early system that shared many of the goals of the POEMS modeling environment, but did not incorporate recent results from object-oriented analysis, data mediation and intelligent agents, nor the range of modern analytic and simulation modeling tools being incorporated in POEMS, was the SARA system [17]. SimOS simulates the computer hardware of both uniprocessor and multiprocessor computer systems in enough detail to run an entire operating system, thus, providing a simulation environment that can investigate realistic workloads. Different modes of operation provide a trade-off between the speed and detail of a simulation. Thus, SimOS supports multi-domain and multi-resolution modeling, but unlike POEMS, it primarily uses a single evaluation paradigm, namely, simulation. RSIM supports detailed instruction-level and direct-execution simulation of parallel program performance for shared memory multiprocessors with ILP processors. PACE (Performance Analysis and Characterisation Environment) is designed to predict and analyze the performance of parallel systems defined by a user, while hiding the underlying performance characterization models and their evaluation processes from the user.

None of the above projects supports the general integration of multiple paradigms for model evaluation, a key goal of POEMS. The conceptual extensions used to achieve this in POEMS are a formal specification language for composition of component models into a full system model, a unified application representation that supports multiple modeling paradigms, and automatic synthesis of this workload representation using a parallelizing compiler. The alternative modeling paradigms support validation and allow different levels of analyses of existing and future application programs within a common framework.

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

8 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 executed on 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 illustrates 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 validates 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. Future development of POEMS methods and tools will be largely driven by MOL [30], which is a modular program that implements the Method of Lines for solving partial differential equations. It is designed to be a "simple" program (less than 1000 lines of code) which has all the features of a "sophisticated" dynamic code. Features that can be varied easily include (1) work load needed to maintain quality of service, (2) number of processors needed, (3) communication patterns, (4) communication bandwidth needed, (5) internal data structures, etc…

In addition to continuing efforts on the preceding topics, several interesting research directions are being pursued in the project. First, the POEMS modeling framework and tools 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. As the next step, the project is enhancing the multi-paradigm modeling capability in POEMS, in which 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 invoked automatically to derive analytical model input parameters. Several initial examples of such integrated modeling approaches were described in Section 7. Innovations in parallel discrete-event simulation technology to reduce the execution time for the integrated models will continue to be investigated, with and without compiler support. The integration of compiler support with analytical and parallel simulation models will enable (to our knowledge) the first fully-automatic, end-to-end performance prediction capability for large-scale parallel applications and systems.

ACKNOWLEDGMENT

A number of people from the member institutions represented by the POEMS team contributed to the work. In particular, the authors acknowledge Adolfy Hoisie, Olaf Lubeck, Yong Luo, and Harvey Wasserman of Los Alamos National Laboratory for suggesting the Sweep3D application, providing significant assistance in understanding the application and the performance issues that are of importance to the application developers, and providing key feedback on our research results. We also wish to thank 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. Thanks also to Lawrence Livermore Laboratory for providing extensive computer time on the IBM SP/2.

This work was supported by DARPA/ITO under Contract N66001-97-C-8533, "End-to-End Performance Modeling of Large Heterogeneous Adaptive Parallel/Distributed Computer/Communication Systems," 10/01/97 - 09/30/00, and by an 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 J. Mellor-Crummey, "Using Integer Sets for Data-Parallel Program Analysis and Optimization", *Proc. SIGPLAN98 Conf. on Prog. Lang. Design and Implementation*, Montreal, June 1998.

[3] Adve, V. S., and R. Sakellariou, "Application Representations for Multi-Paradigm Performance Modeling", *International Journal of High Performance Computing Applications* 14(4), 2000.

[4] Adve, V. S., R. Bagrodia, E. Deelman, T. Phan and R. Sakellariou, "Compiler-Supported Simulation of Highly Scalable Parallel Applications", *SC99: High Performance Computing and Networking, Portland, OR, Nov. 1999.*

[5] Alexandrov, A., M. Ionescu, K. E. Schauser, and C. Scheiman, "LogGP: Incorporating Long Messages into the LogP Model", *Proc. 7th Ann. ACM Symp. on Parallel Algorithms and Architectures (SPAA '95),* Santa Barbara, CA, July 1995.

[6] Amza, C., A. Cox, S. Dwarkadas, P. Keleher, H. Lu, R. Rajamony, W. Lu, and W. Zwaenepoel, "TreadMarks: Shared Memory Computing on Networks of Workstations", *Computer,* 29(2), Feb. 1996, pp. 18-28.

[7] Bagrodia, R., and W. Liao, "Maisie: A Language for Design of Efficient Discrete-event Simulations", *IEEE Tran. on Software Engineering*, 20(4), Apr. 1994.

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

[9] Bagrodia, R., R. Meyer, M. Takai, Y. Chen, X. Zeng, J. Martin, B. Park, and H. Song, "Parsec: A Parallel Simulation Environment for Complex Systems", *Computer*, 31(10), Oct. 1998, pp. 77-85.

[10] Bagrodia, R., E. Deelman, S. Docy, and T. Phan, "Performance Prediction of Large Parallel Applications Using Parallel Simulations", *7th ACM SIGPLAN Symp. on the Principles and Practices of Parallel Programming (PPoPP '99)*, Atlanta, May 1999.

[11] 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.

[12] Bayerdorffer, B., "Distributed Programming with Associative Broadcast", *Proc. of the Twenty-eighth Hawaii International Conf. on System Sciences*, Jan.

1995, pp. 525-534.

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

[14] Burger, D., and T. M. Austin, “The SimpleScalar Tool Set, Version 2.0”, Univ. of Wisconsin-Madison, UW CS Tech Rpt. #1342, June 1997.

[15] 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, CA, May 1993, pp. 1-12.

[16] Dube, A., “A Language for Compositional Development of Performance Models and its Translation”, Masters Thesis, Dept. of Computer Science, Univ. of Texas at Austin, Aug., 1998.

[17] Estrin, G., R. Fenchel, R. Razouk, and M. K. Vernon, “SARA: Modeling, Analysis, and Simulation Support for Design of Concurrent Systems”, *IEEE Trans. on Software Engineering, Special Issue on Software Design Methods*, **SE-12**(2), Feb. 1986, pp. 293-311.

[18] 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, NV, June 1997, pp. 62-73.

[19] Hoisie, A., O. M. Lubeck, and H. J. Wasserman, “Performance and Scalability Analysis of Teraflop-Scale Parallel Architectures Using Multidimensional Wavefront Applications”, *Proc. Frontiers ‘ 99*.

[20] Horwitz, S., T. Reps, and D. Binkley, “Interprocedural Slicing Using Dependence Graphs,” *ACM Trans. on Programming Languages and Systems 12(1)*, Jan. 1990, pp. 26-60.

[21] Kerbyson, D. J., J. S. Harper, A. Craig, and G. R. Nudd, “PACE: A Toolset to Investigate and Predict Performance in Parallel Systems”, European Parallel Tools Meeting, ONERA, Paris, Oct. 1996.

[22] 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.

[23] Lamport, L., "Time, Clocks and the Ordering of Events in a Distributed System" Communications of the ACM, **21**(7), (July 1978) pp. 558-565.

[24] Newton, P., and J. C. Browne, “The CODE 2.0 Graphical Parallel Programming Language”, *Proc. ACM Int. Conf. on Supercomputing*, July 1992, pp. 167-177.

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

[26] 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, TX, Mar. 1997, pp. 72-83.

[27] Prakash, S. and R. Bagrodia, “Parallel Simulation of Data Parallel Programs”, *Proc. 8th Workshop on Languages and Compilers for Parallel Computing*, Columbus, OH, August 1995.

[28] Prakash, S. and R. Bagrodia, “Using Parallel Simulation to Evaluate MPI Programs”, *Proc. Winter Simulation Conf.*, Washington D.C., Dec. 1998.

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

[30] Rice, John., *Numerical Methods, Software and Analysis*, Academic Press, 2nd Edition, New York, 1993, Chapter 10.2.E, pp.524-527

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

[32] 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.

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

[35] Sun, X.H., D. He, K. W. Cameron and Y. Luo, “A Factorial Performance Evaluation for Hierarchical Memory Systems”, *Proc. Int’l Parallel Processing Symposium* (*IPPS'99*), San Juan, PR, Apr. 1999.

[36] Sundaram-Stukel, D., and M. K. Vernon, “Predictive Analysis of a Wavefront Application Using LogGP”, *Proc. 7th ACM SIGPLAN Symp. on the Principles and Practices of Parallel Programming (PPoPP ’99)*, Atlanta, May 1999.

[37] Vernon, M. K., E. D. Lazowska, and J. Zahorjan, “An Accurate and Efficient Performance Analysis Technique for Multiprocessor Snooping Cache-Consistency Protocols,” *Proc. 15th Annual Int’l. Symp. on Computer Architecture*, Honolulu, May 1988, pp. 308-315,.

[38] Wiederhold, G., “Mediation in Information Systems; in Research Directions in Software Engineering”, *ACM Computing Surveys*, **27**(2), June 1995, pp. 265-267.