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

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

1. Introduction

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

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

The POEMS project is building an initial library of models, at multiple levels of granularity, for modeling scalable architectures like those envisaged under the DOE ASCI program, and for modeling complex adaptive systems like those envisaged under the DARPA GloMo and Quorum programs. The project supports evaluation of component behaviors through analytic models and through application of discrete event simulation at multiple levels of detail. The analytic models include deterministic task graph analysis [Adve '93], and the LogP [Culler et. al. '93] and LoPC models [Frank et. al. '97]. POEMS incorporates parallel evaluation of discrete event simulation models through application of the Maisie simulator [Bagrodia et al '94]. The project is also building a knowledge base of performance data that can be used to estimate the performance properties of widely-used algorithms as a function of architectural characteristics.

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

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

2. POEMS Methodology

The major elements of the conceptual framework of POEMS are a general model of parallel computation and the use of associative objects as the representation basis for components. The models developed in POEMS span three domains: *application, operating system*, and *hardware*. The application domain represents parallel computation as a dynamic task graph where nodes represent sequential computation units and edges represent dependencies. The operating system domain provides the models for process management, inter-process communication, and parallel file system. For systems that do not support dynamic process creation or multi-threaded execution (e.g., MPI), the graph model for the application domain can be collapsed into a static number of process models. The hardware domain provides models for the processor and memory components, where the latter includes models for cache memory as well as shared memory hierarchies.

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

2.1 General Model of Parallel Computation

The POEMS representation is based on a parallel model of computation where all units of computation are assumed to execute in parallel unless constrained to execute in some partial (or total) order by specification of relationships among them or constraints upon them. This model of computation [Pandey & Browne '94] is an extension and adaptation of Pratt's [Pratt 86] POMSET model of parallel computation. A task level data flow representation of this model of computation is obtained by defining the nodes in the data flow system to have extended dependence relationships. Synchronous parallel execution is obtained by extending the data dependence relationships (and specifications of the firing rules for the nodes) to regard a clock as a unit of computation.

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

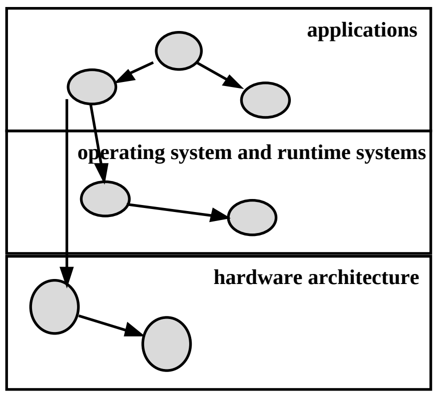

Figure 1: Multi-Domain Dependence Graph

A node executes when the computation reaches a state where the "firing rule" evaluates to true. Each arc has associated with it a data type specification, which we call a transaction. The model has an initialization that creates instances of the data types and places them on its outgoing arcs. The execution of this model of computation is the traversal of this graph structure by the transactions.

2.2 Automatic Task Graph Construction

The POEMS generalized task graph representation provides workload information at various levels of abstraction for both analytic and simulation models. Furthermore, the representation is designed to be computable *automatically* by a parallelizing compiler, but may be computed manually for explicitly parallel programs. The representation must also be able to capture adaptive codes where the parallel structure depends not only on the program input but also on intermediate computational results. Finally, the representation must scale to support realistic problem sizes for large systems.

The application representation in POEMS is based on a combination of *static* and *dynamic* task graphs. The static task graph is a compact, symbolic graph representation of the parallel structure of the program, including the parallel tasks, precedence edges between tasks, and a subset of the control flow (including loops). It is independent of program input values. The dynamic task graph is a directed *acyclic* graph that captures the precise parallel structure of an execution of the application for a given input [Adve 93]. The dynamic task graph is important for supporting detailed and precise modeling of parallel program behavior. For programs where the tasks are statically scheduled on to the processes (as in most message-passing programs), the dynamic task graph can be represented very concisely by combining all the tasks allocated to a particular process between synchronization points into a single “task.” For other cases, the dynamic graph is computed on demand (from the static task graph) during model evaluation.

Data-parallel compiler technology from the Rice University dHPF compiler project [Adve and Mellor-Crummey 1998] will be used to compute the static task graph representation for key example programs, including High Performance Fortran (HPF) as well as manually parallelized programs. The key steps required to construct the static graph are selecting the units that will be represented as tasks, and identifying the precedences, communication and synchronization patterns between the tasks. There are some significant challenges to be faced in constructing the graph. First, computing a symbolic representation of the synchronization and communication patterns requires a powerful symbolic analysis capability, potentially greater than that required for parallelization alone. As one example, in the Sweep3D application described in section 3, eight different communication patterns (for the eight octants) are encoded in a single piece of code using three symbolic “direction” parameters, and these direction values are computed outside the sweep function. Second, the compiler must recognize and capture non-static behavior where the task graph may depend on intermediate computational results. For example, in Sweep3D, a different flux computation is used after the simulation has proceeded for a certain number of time steps. This requires alternative sets of tasks and precedences to be captured in the graph, along with the governing condition. Finally, in manually parallelized codes, the key additional challenge is to extract as much static information as possible about the communication and synchronization patterns in the program. In general, these might be obscured by SPMD control flow, procedure boundaries, and non-standard communication libraries. The dHPF compiler infrastructure provides a combination of symbolic analysis, interprocedural analysis, and communication analysis capabilities that will enable us to address these challenges.

Static and dynamic task graphs for the Sweep3D program are given in section 3.2. These graphs, constructed by hand, have aided in identifying the issues that need to be addressed in automating the task graph generation, as described above. The next step is to address these issues for generating the task graph in the dHPF compiler.

2.3 Task Execution Descriptions (TEDs)

The size of the multiprocessor systems and applications that will be modeled by POEMS requires that analytic modeling be used as much as possible and that detailed simulation be used to support and refine the analytic results, focusing on the most important regions of the design space. With this in mind, a Task Execution Description, TED, represents the modeled execution of a task, and is associated with each node of an application's task dependency graph. A TED relates the task to the functional definition of a hardware domain subsystem comprised of processor, memory, and transport components, which currently supports the quantification of the execution time of a task via detailed instruction-level simulation. Obtaining TEDs from instruction-level simulation is further described in section 4.4.

2.4 Compositional Modeling Using Associative Objects

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

An associative object is a "standard object" encapsulated with an associative interface [Shlaer92, Rumbaugh91, Booch et. al. 97]. The key concepts are that objects are defined in the context of a semantic domain and the properties of objects are defined in terms of the attribute set of a semantic domain.

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

An associative interface specifies all of the interactions in which a component can participate via two elements: an "accepts" interface for the services that it implements and a "requests" interface which specifies the services it requires.. Interfaces are specified in terms of the attributes that define the behavior and the states of the standard objects. An object that has associative interfaces is said to be an associative object and an object that interacts through associative interfaces is said to have associative interactions. An associative interface is an extension of the associative model of communication [Bayerdorffer 93, Bayerdorffer 95] to define complex dynamic interactions among object instances.

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

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

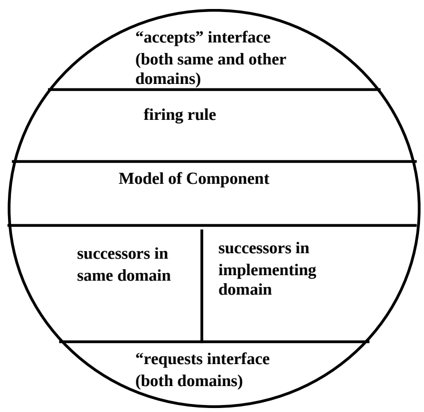

Figure 2: Schematic of a Node of a Dependence Graph as an Associative Object

2.5 Performance Recommender

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

We are using a kernel (IFESTOS), developed at Purdue [Ramakrishnan 97], that supports the rapid prototyping of recommender systems. IFESTOS abstracts the architecture of a recommender system as a layered system with clearly defined subsystems for problem formulation, knowledge acquisition, performance modeling and knowledge discovery. The designer of the recommender system first defines a database of application classes (problems) and computation class instances (methods). The data acquisition subsystem generates performance data by invoking the appropriate applications (*e.g.*, Sweep3D). The performance data management subsystem provides facilities for the selective editing, viewing and manipulation of the information so generated. Performance analysis is performed by traditional attribute—value statistical techniques and `mining' this information produces useful rules that can be used to drive the actual recommender system. This approach has been successfully demonstrated for problem domains in numerical quadrature and elliptic partial differential equations [Ramakrishnan 97]. We are currently applying it to the Sweep3D application, described next.

3. POEMS Driver Application: Sweep3D

We consider the problem of predicting the performance of an ASCI kernel application called Sweep3D on large scale parallel architectures such as the IBM SP2 or the SGI Origin 2000. This section contains a brief description of this application; Section 4 discusses how the performance analysis tools in POEMS are being applied to Sweep3D.

3.1 Overview of Sweep3D

The Sweep3D kernel is a solver for the three dimensional time independent, neutral particle transport equation on an orthogonal mesh [Koch et al 1992]. 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 on other quantities such as local sources, cross section data, and geometric factors. The flux of a given cell cannot be computed until all of its upstream neighbors have been computed, implying a recursive or wavefront structure. This structure is repeated along the eight diagonal directions through the cube (referred to as octants). Figure 4 shows how this 3D algorithm is partitioned on a 2D processor grid (i and j dimensions). A wavefront consists of a block of 'k' cells and a block of angle values for each octant. The wavefront in the figure is originating from the lower left corner of the processor grid (octant 7) and has three cells per k-block. The sweep in octant 7 consists of additional wavefronts for each of the angle blocks until the last k-block has been computed on the processor in the upper right.

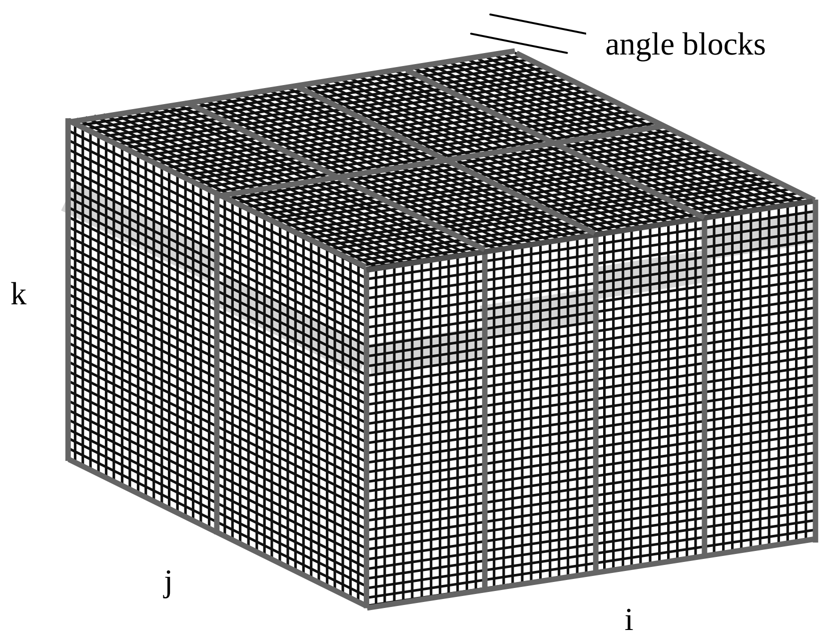

Figure 4: Mapping the Wavefronts of Sweep3D on a 2D Processor Grid

3.2 Sweep3D Task Graph

The static and dynamic task graphs for the sweep phase of Sweep3D on a 3x3 processor grid are illustrated in Figures 5 and 6. Figure 5 shows a hierarchical view of the static task graph. On the right, two of the eight octants in each time step of Sweep3D are displayed, and the computation and communication tasks performed by each processor for each octant are shown on the left. Figure 6 shows the dynamic task graph for two wavefronts *within* one octant. Communication tasks are circles numbered with a send/receive processor pair, and computation tasks are squares numbered by a computing processor. The critical path for the two wavefronts are the two long vertical sequences of tasks.

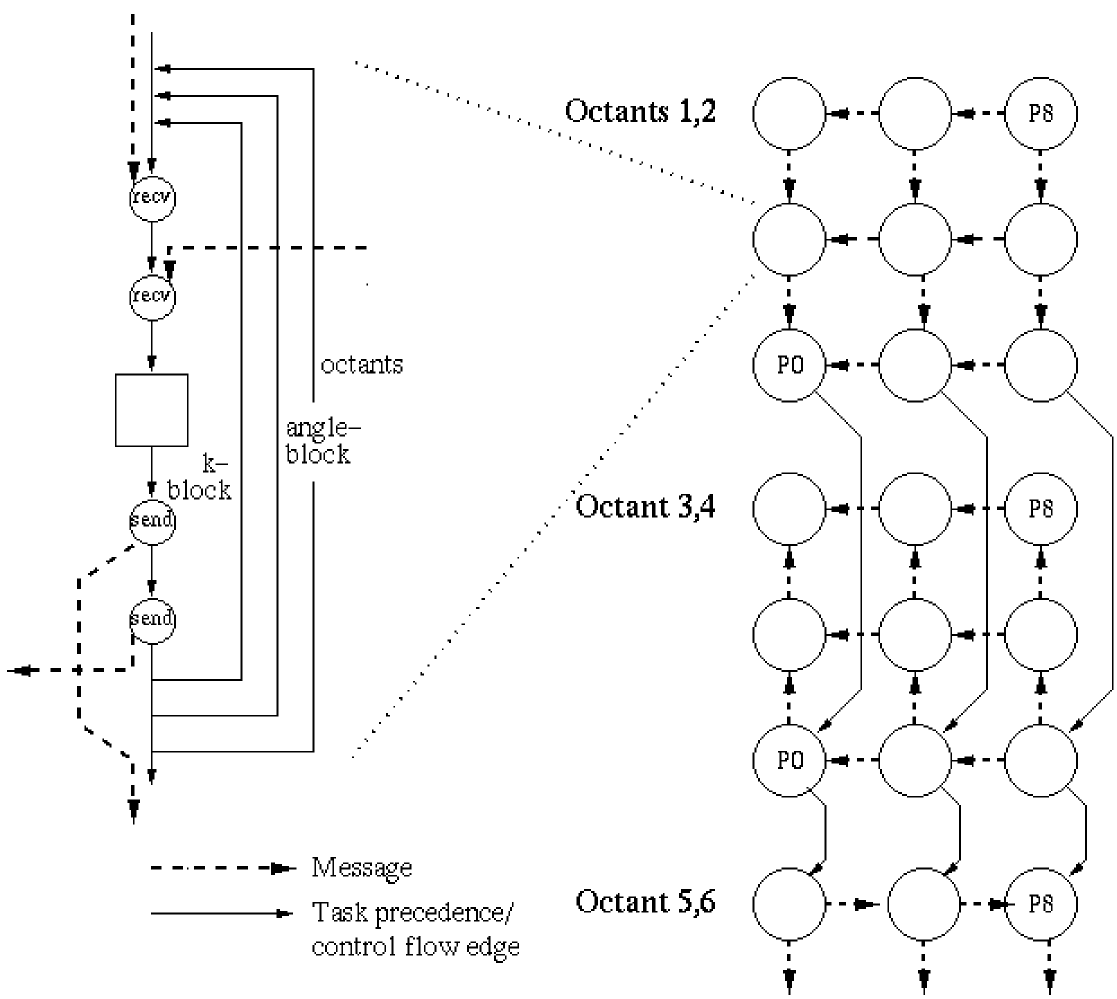

Figure 5: Static Task Graph for Sweep3D on a 3x3 processor grid

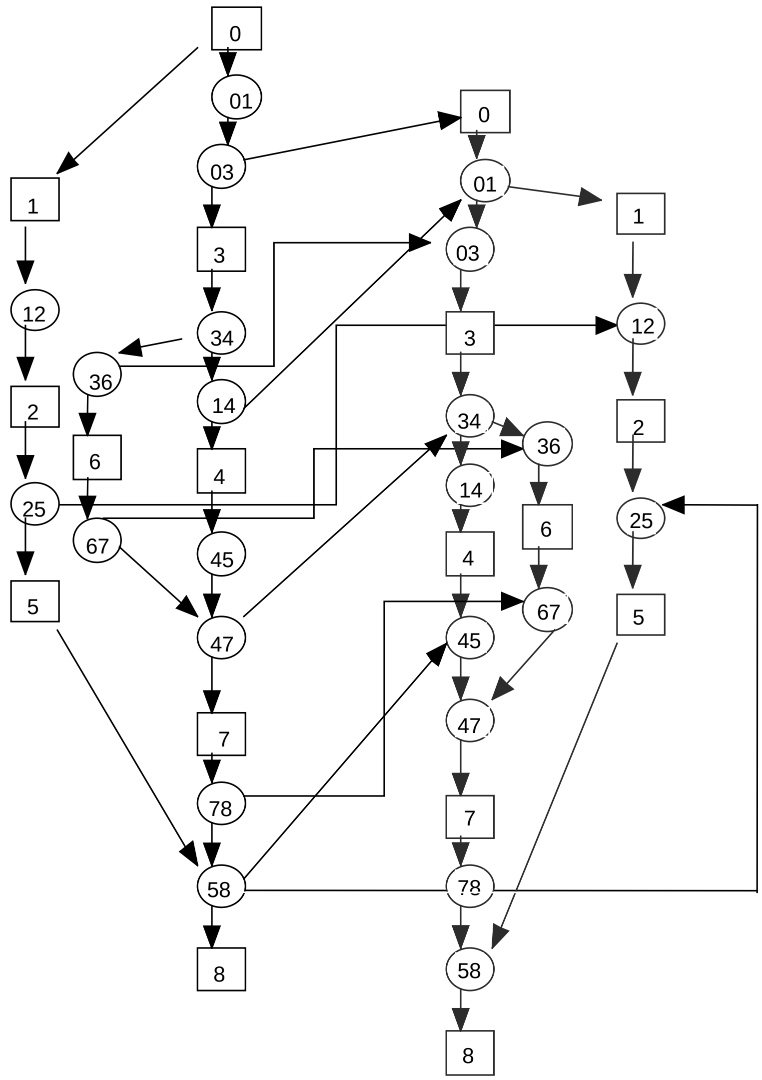

Figure 6: Dynamic Task Graph for Sweep3D on a 3x3 Processor Grid

4. POEMS Performance Analysis Tools

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

4.1 LogP and LoPC

The analytical modeling capabilities in POEMS include the LogP model [Culler *et. al.* 93] and an extension of LogP known as LoPC [Frank *et. al.* 97]. The task graph for a given application elucidates the principal structure of the code, including the communication events, from which it is relatively easy to generate the LogP or LoPC equations. We illustrate this approach using the Sweep3D application.

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

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

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

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

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

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

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

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

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

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

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

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

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

4.2 Deterministic Task Graph Analysis

Another analytical model in POEMS is parallel program performance prediction based on deterministic task graph analysis [Adve 1993]. The inputs to the model are a task-graph that describes the tasks and synchronization behavior of an application; a description of the task scheduling method used to allocate tasks to processors; and parameters describing the computational cost and average communication rates of each task.

The model is a two-level hierarchical model in which the application is modeled as a task graph with deterministic total task times, and contention in the underlying parallel system is modeled using a stochastic queueing network. The queueing network model is solved using AMVA to compute the mean values of the task residence times, and these are used as deterministic task execution times in analyzing the task graph model [Adve &Vernon 1993]. We use a modified critical path analysis to analyze the task graph, modified to account for task scheduling when the number of ready tasks exceeds the number of available processors. The assumption of deterministic task times is essential to enable a very efficient solution of the task graph, and to make it practical to solve the queueing network for every combination of tasks in execution (viz., O(N) times for a task graph with N tasks).

The deterministic task graph analysis provides detailed information about the impact of task synchronization, task scheduling, and average communication costs (including contention), on program performance. It has been shown to be efficient and consistently accurate for several shared-memory programs (including three from the Splash benchmarks [Singh *et. al*. 92]). It has also been shown to be useful for *predicting* the impact of program design changes, including the benefits of sophisticated dynamic and semi-static scheduling algorithms that improve communication locality as well as load-balancing [Adve 93]. Task graph analysis of the Sweep3D application is in progress. [Note to the referees: it is likely that some preliminary results of the task graph analysis will be available in time for the final version of the paper.]

4.3 Execution-driven System Simulation

POEMS includes a modular, execution-driven, parallel program simulator called MPI-SIM that has been developed at UCLA [Prakash 96]. MPI-SIM can evaluate the performance of existing MPI programs as a function of various hardware and system software characteristics that include the number of processors, interconnection network characteristics, or message-passing library implementations. The simulator can also be used to evaluate the performance of parallel file systems and I/O systems [Bagrodia et. al. 97]. Supported capabilities include a number of different caching algorithms, collective I/O techniques, cache replacement algorithms, and I/O device models.

MPI-SIM simulates the application program by using direct execution to simulate local code fragments and using parallel discrete-event simulation to simulate communication (point-point and collective operations) and I/O operations. Detailed simulations of parallel programs, particularly when simulated for large hardware configurations, are notoriously slow. MPI-SIM uses a set of parallel conservative synchronization protocols together with a number of optimizations to execute the parallel program models on parallel architectures.

MPI programs execute as a collection of single threaded processes, and, in general, the host machine will have fewer processors than the target machine (for sequential simulation, the host machine has only one processor). This requires that the simulator support multithreaded execution of MPI programs. We have developed MPI-LITE, a portable library for multithreaded execution of MPI programs, for this purpose. A preprocessor replaces all MPI calls by equivalent calls to corresponding routines in the simulator. The simulator does not directly simulate every MPI call. Rather, all collective communication functions 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 are identical to how they are implemented on the target architecture. MPI-SIM has been validated against the NAS MPI benchmarks and has demonstrated excellent performance improvement with parallel execution against these benchmarks [Prakash '96].

Section 5 presents results on the scalability and performance of sweep3D as predicted by MPI-SIM. A facility to simulate the task graph notation expressed as an MPI program is currently being developed. Although the use of direct execution speeds up the simulator, it also restricts its ability to evaluate the performance of the application as a function of alternative processor and memory subsystem designs. As described in the next section, POEMS is also developing a simulator for the memory and processor subsystems as part of its hardware domain models. These models will be integrated with MPI-SIM to predict the performance of actual programs or program designs as a function of proposed next-generation memory or processor designs.

4.4 Execution-driven Analysis of the Memory Hierarchy

The POEMS hardware modeling environment is being designed to allow a user to define the computer architecture to be modeled and the modeling methodology to be used. By specifying the design-specific attribute values of processor, memory, and transport components, which are described below, the user defines the architecture to be evaluated. By specifying the model-specific attribute values of these hardware domain components, the user defines the evaluation method to be used.

Given the design attributes of a realized component (such as an existing processor), TEDs that provide parameter values for less detailed models of the processor can be obtained through system measurement, as for Sweep3D on existing SP/2 and Origin 2000 architectures. Given the design attributes of a yet-to-be-realized component, e.g., a new cache design, an execution-driven simulation is required to quantify the execution time and interprocessor communication events of a task on the new component. The POEMS project is currently investigating various existing execution-driven simulators that can provide this functionality.

5. Preliminary Results for Sweep3d

This section summarizes results from the application of a subset of the tools described in the previous section to study the performance of the Sweep3D application. The MPI code previously developed for this application was the point of departure for the study. The original application was written using Fortran. A version was generated in MPI with C to allow the application to be evaluated with the simulator. The goal of the initial study was to investigate how well Sweep3D performance scales with the number of processors on the IBM SP2 and the SGI Origin 2000. Availability of both Fortran and C versions of the program allow us to investigate this question for both implementations.

Clearly, measurement is restricted by both the size and the IPC characteristics of the available hardware. By using the LogP/LoPC model and the MPI-SIM simulator we can evaluate performance scalability for larger numbers of processors and improved communication primitives. The Sweep3D task graph aided in developing the LogP/LoPC model as well as in measuring the parameter values (e.g. computation granularity of the code) for all of the models. Preliminary results of the analytical and simulation studies are given below.

Figure 7 shows the validation of the LogP/LoPC model against SGI Origin2000 measurements. The problem size per processor is fixed in this experiment (i.e., the total problem size increases with the number of processors). Since the length of the critical path is equal to the sum of the dimensions of the grid, the runtime should increase linearly in $P_i + P_j$, as is observed. Furthermore, the breakdown of the total computation time and communication time predicted by the model shows that Sweep3D is heavily cpu-bound.

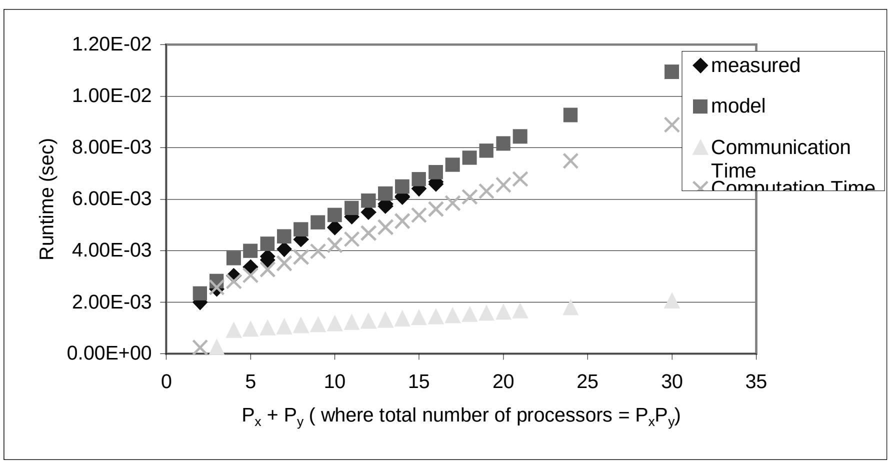

Figure 7. LogP/LoPC model validatation on SGI Origin2000

(subgrid size 5x5x10 per processor, k-blocking factor = 1; single octant execution time)

Validations of the LogP/LoPC model and the MPI-SIM model against the SP/2 measures, for fixed total problem size, are shown in Figure 8 and 9, respectively. We are currently investigating why the model estimates for the SP/2 architecture appear to be low by a constant equal to 1.8 seconds. In any case, the simple analytic model appears to predict the performance behavior to within this constant factor extremely well, and the MPI-SIM model produces extremely accurate results. In practice one would want to scale the

problem size more aggressively, but this more conservative scaling experiment "stress-tests" the accuracy of the models with respect to interprocessor communication.
[Note to reviewers: we expect to have LogP/LoPC validations on the SP/2 for up to 128 processors in the final version of the paper; the SP/2 at LLNL was unavailable during 3/8-3/15 which prevented us from completing those runs prior to submitting the paper.]

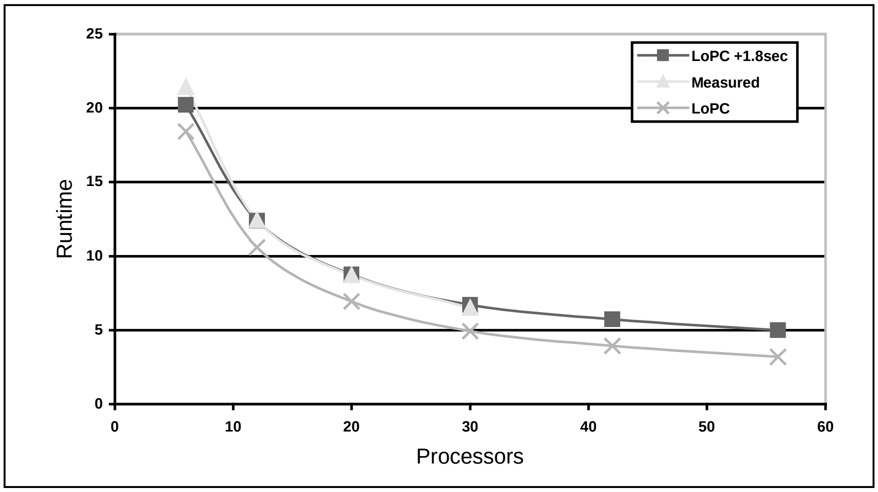

Figure 8. LogP/LoPC model validation on an IBM SP/2.
(grid size: 50^3, k-blocking factor =10, angle-blocking factor =3)

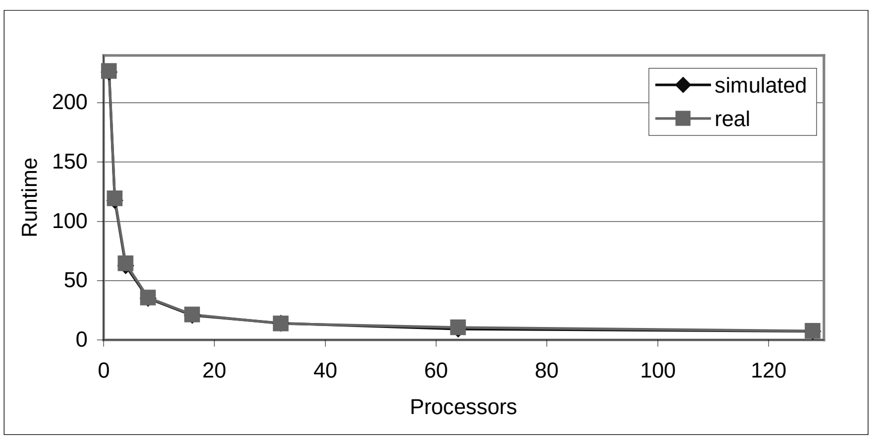

Figure 9. MPI-SIM validation and scalability results.
(50^3, k-blocking factor =10, angle-blocking factor =3)

We have also investigated the performance of Sweep3D as a function of message latencies as shown in figures 10 and 11. Five different latencies were used for these studies: 0, *sp*, 10*sp*, *50sp* and 100*sp*, where *sp* is the current latency on the SP/2. Surprisingly, the actual SP/2 latency appears to have very small impact on the execution time of the Sweep3D kernel for configurations with up to 64 processors (as shown by both the MPI-SIM and LoPC predictions). For the MPI-SIM experiment based on the C version of Sweep3D, the differences in performance only became apparent when the latency was increased to 50 times that of an SP/2. The LoPC results based on the Fortran version of Sweep3D show that the execution time is a bit more sensitive to message latency, which makes sense given the smaller overall execution time of the Fortran code.

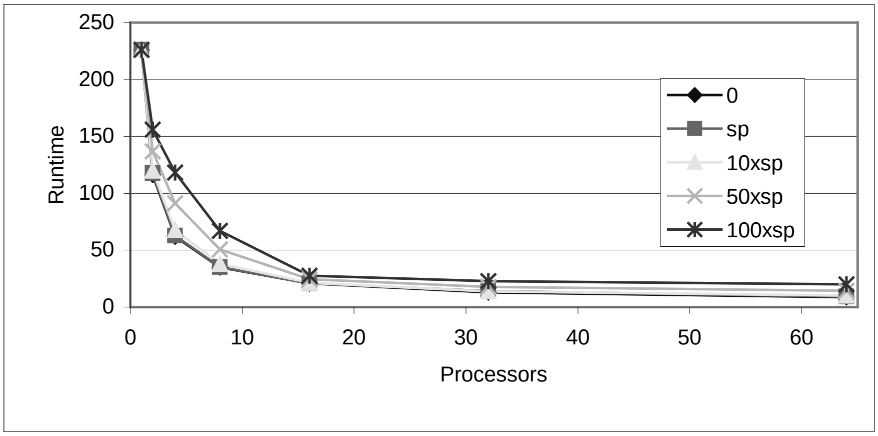

Figure 10. Predicted Impact of Communication Latency on Sweep3D (C code)

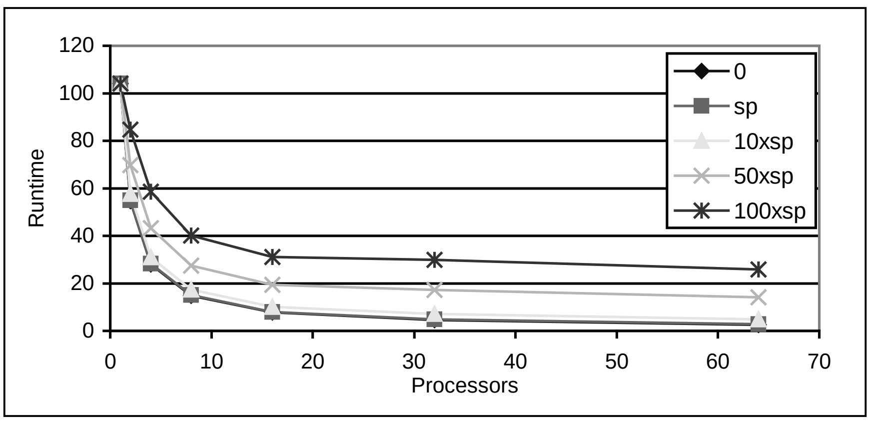

Figure 11. Predicted Impact of Communication Latency on Fortran Sweep3D

6. Related Work

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

The conceptual framework for POEMS is a synthesis from models of naming and communication [Bayerdorffer93,95], CAD [Cadence, Mentor Graphics], software frameworks [Myers97], parallel computation [Newton92], object-oriented analysis [Shlaer92], data mediation [Weiderhold95] and intelligent agents [Woodbridge95] There are many research projects focusing on performance modeling. The most closely related projects to POEMS are probably the Maisie parallel discrete-event simulation framework and its use in parallel program simulation [Bagrodia94, Prakash 96], SimOS [Rosenblum95], and RSIM [Pai 97b] and the earlier work in program simulators, direct execution simulators, and parallel discrete-event simulation.

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

There are many effective commercial products for simulation modeling of computer and communication systems. We refer to the *March 1994 IEEE Communications Magazine* for a survey of commercial products.

7. Conclusions

The POEMS project is creating a problem solving environment for end-to-end performance models of complex parallel and distributed systems and to apply this environment for performance prediction of application software 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 illustrated the POEMS modeling methodology and approach, by using a number of the POEMS tools for performance prediction of the Sweep3D application kernel selected by Los Alamos National Laboratory for evaluation of ASCI architectures. The paper validated the performance predicted by the analytical and simulation models against the measured appication performance. The Sweep3D kernel used for this study is an example of a *regular* cpu-bound application. Current research involves developing out-of-core and adaptive versions of the Sweep3D as well as further results for scaling the regular application and for using improved communication primitives on the SP/2 and Origin 2000. Reusable versions of the analytic and simulation models, parameterized for three-dimensional wavefront applications, will form the initial component model library for POEMS. [Note to the reviewers: this library should be available in September.]

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

Acknowledgements:

A number of people from the member institutions represented by the POEMS team contributed to the work: Thomas Phan and Steven Docy from UCLA….

This work was supported by DARPA/ITO under Contract N66001-97-C-8533, "End-to-End Performance Modeling of Large Heterogenous Adaptive Parallel/Distributed Computer/Communication Systems," 10/01/97 - 09/30/00 and by a NSF grant titled Design of Parallel Algorithms, Language, and Simulation Tools Award ASC9157610, 08/15/91 - 7/31/98.

References

[Adve 93] Adve, V. S., "Analyzing the Behavior and Performance of Parallel Programs", University of Wisconsin-Madison, UW CS Tech. Rep. #1201", October 1993.

[Adve & Vernon 93] Adve, V. S., and M. K. Vernon, "Influence of Random Delays on Parallel Execution Times", *Proc. 1993 ACM SIGMETRICS Conference on Measurement and Modeling of Computer Systems*, May 1993, pp. 61-73.

[Adve & Mellor-Crummey 98] Adve. V. S., and J. Mellor-Crummey, "Using Integer Sets for Data-Parallel Program Analysis and Optimization," SIGPLAN98, Montreal, June 1998.

[Bagrodia & Liao 94] Bagrodia, R., and W. Liao, "Maisie: A Language for design of efficient discrete-event simulations", *IEEE Transactions on Software Engineering*, April 1994.

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

[Bayerdorffer 93] B. Bayerdorffer, "Associative Broadcast and the Communication Semantics of Naming in Concurrent Systems", PhD. dissertation, Dept.of Computer Sciences, Univ. of Texas at Austin. Dec. 1993.

[Bayerdorffer 95] B. Bayerdorffer, "Distributed Programming with Associative Broadcast", *Proc. 28th HICSS '95*, Jan. 1995.

[Booch *et. al.* 97] G. Booch, J. Rumbaugh and I. Jacobson "Unified Modeling Language User Guide" (Addison-Wesley, Englewood Cliffs, 1997)

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

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

[Koch *et. al*. 92] 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.

[Newton92] P. Newton and J.C. Browne, ÒThe CODE 2.0 Graphical Parallel Programming Language,Ó Proc. ACM Int. Conf. on Supercomp, July, 1992

[Pai 97] RSIM: An Execution-Driven Simulator for ILP-Based Shared-Memory Multiprocessors and Uniprocessors, Vijay S. Pai, Parthasarathy Ranganathan, and Sarita V. Adve, IEEE Technical Committee on Computer Architecture newsletter, Fall 1997

[Pandey & Browne 93] Raju Pandey and James C. Browne. "Event-based Composition of Concurrent Programs." *Workshop on Languages and Compilers for Parallel Computation, Lecture Notes in Computer Science 768*, Springer Verlag, 1993.

[Pandey & Browne 94] Raju Pandey and James C. Browne. "A Compositional Approach to Concurrent Object-Oriented Programming," *IEEE Int'l. Conference on Computer Languages*, IEEE Press, May 1994.

[Prakash 96] Prakash, S., *Performance Prediction of Parallel Programs*, Ph.D. thesis, UCLA Computer Science Dept. December 1996.

[Pratt 86] V. Pratt. "Modeling concurrency with partial orders", *Int'l. J. on Parallel Programming*, 15, pp 213-223.

[Ramakrishnan 97], Ramakrishnan, N., "Recommender Systems for Problem Solving Environments}", Ph.D. thesis, Dept. of Computer Sciences, Purdue University, 1997.

[Rumbaugh et. al. 91] J. Rumbaugh, et.al. *Object-Oriented Modeling and Design*, Prentice-Hall, Englewood Cliffs, NJ, 1991.

[Ros95] 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.

[Shlaer & Mellor 92] S. Shlaer and S. Mellor “Object Lifecycles: Modeling the World in States” (Yourdon Press, New York, 1992)

[Singh et. al. 92] Singh, J. P., W.-D. Weber and A. Gupta, "SPLASH: Stanford Parallel Applications for Shared-Memory", Computer Architecture News, Vol. 20 (1), March 1992, pp. 5-44.

[Wiegerhold 95] G. Wiederhold "Mediation in Information Systems: Research Directions in Software Engineering" ACM Computing Surveys, June, 1995.

[Woodbridge 95] M. Woodbridge and N.R Jennings "Intelligent Agents: Theories, Architectures and Languages" LNAI 890 (Springer-Verlag, Berlin, 1995)