International Journal of Computational Intelligence Research.
ISSN 0973-1873 Vol.4, No.1 (2008), pp. 17–26

A Survey of the State of the Art in Performance Modeling and Prediction of Parallel and Distributed Computing Systems

Sabri Pllana, Ivona Brandic and Siegfried Benkner

University of Vienna, Institute of Scientific Computing
Nordbergstrasse 15, 1090 Vienna, Austria
{pllana,brandic,sigi}@par.univie.ac.at

Abstract: **Performance is one of the key features of parallel and distributed computing systems. Therefore, in the past a significant research effort was invested in the development of approaches for performance modeling and prediction of parallel and distributed computing systems. In this paper we identify the trends, contributions, and drawbacks of the state of the art approaches. We describe a wide range of the performance modeling approaches that spans from the high-level mathematical modeling to the detailed instruction-level simulation. For each approach we describe how the program and machine are modeled and estimate the model development and evaluation effort, the efficiency, and the accuracy. Furthermore, we present an overall evaluation of the described approaches.**

1. Introduction

The solution of resource-demanding scientific and engineering computational problems involves the execution of programs on parallel and distributed computing machines, in order to solve large problems or to reduce the time to solution for a single problem [13]. However, the development of this kind of computing systems is an expensive and time-consuming endeavor. For instance, the development cost of the Earth Simulator Center (ESC) [10] was about US$350 million [43], and its development took about five years. While the life of parallel and distributed computing machines is commonly up to five years long, the life of parallel and distributed programs is up to 30 years [9]. Therefore, it is important to have means for the performance evaluation of programs not only on existing machines, but also on machines that are under development or being planned.

Because the performance is a key indicator of computing systems, the performance evaluation was a preoccupation of many computer scientists in the past [1, 35, 18, 23, 25]. The commonly used techniques for the performance evaluation of computing systems include: measurement, mathematical modeling, and simulation. Each of these techniques has its limitations. Measurement techniques require that the system under study is available for experimentation, and their applicability is limited to only existing systems. Mathematical performance models usually lack the system's structural information, and therefore, are not suitable for the model based performance analysis. The model-based performance analysis involves the modification of structure of the model to reflect system structural changes, in order to predict what would be the performance of the system under study if its structure is changed. Detailed simulation models demand large computational and storage resources, and their evaluation may be so slow that the performance assessment of real-world programs is impractical.

In this paper we systematically present a collection of the state of the art approaches for performance modeling and prediction of parallel and distributed computing systems. The range of the approaches for performance modeling and prediction that we cover spans from the high-level mathematical modeling to the detailed instruction-level simulation. For each approach we describe how the program and machine are modeled and estimate the model development and evaluation effort, the efficiency and the accuracy. In addition, we evaluate the suitability of the presented approaches for the model-based performance analysis of parallel and distributed computing systems.

The rest of this paper is organized as follows. Preliminaries are described in Section 2. Section 3 describes a collection of approaches for performance modeling and prediction of computing systems. An evaluation of the described approaches is presented in Section 4. Finally, Section 5 presents some concluding remarks.

2. Preliminaries

The terms *workload*, *machine*, and *system* are used frequently in the literature, but not always with the same meaning. In the context of this paper the term *workload* indicates a distributed and parallel program, whereas the term *machine* indicates a distributed and parallel computing

architecture. The term *system* indicates a computing system. In our approach, the workload model and the machine model are considered as integral parts of the computing system model (see Figure 1).

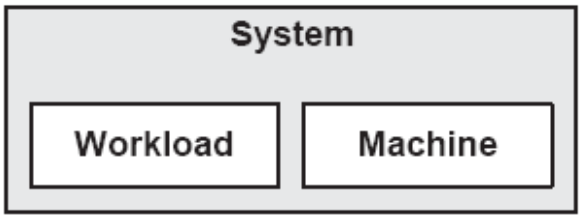


Figure 1 : Computing system model.

In this paper, for each performance modeling and prediction approach, we describe how the workload and machine are modeled. In addition, the following properties of approaches are discussed:

Model development effort indicates the endeavor of the user of a specific approach for building the model.

Model evaluation effort indicates the time and the computing resources that are needed to evaluate the model.

Efficiency of the approach considers the effort needed to develop and evaluate the model. For instance, the efficiency of an approach is high if the model development and evaluation effort is low.

Accuracy indicates the exactness of an approach. For estimation of the accuracy is commonly performed a comparison of prediction results with measurement results.

2.1 Performance-oriented program development

There is a widening gap between the maximal theoretical performance of a computing machine and the achieved performance when a certain program is executed [13]. This gap may be reduced by tuning the performance of a program for a specific computing machine. The procedure for improvement of the program performance involves multiple cycles of code editing, compilation, and execution (see Figure 2(a)). For real-world programs this procedure of code-based performance tuning is time-consuming, expensive and error-prone. Furthermore, its applicability is limited to the available computing machines.

Model based performance tuning involves the model editing, model transformation to a form that is suitable for evaluation, and model evaluation (see Figure 2(b)). In this manner, instead of making experiments with the real program code on the real computing machine, we are able to experiment with the model of the program and the model of the computing machine. Commonly, the model-based performance tuning is: efficient, inexpensive, and non intrusive. Moreover, it is applicable not only to the available computing machines but also to those that are under development or being planned.

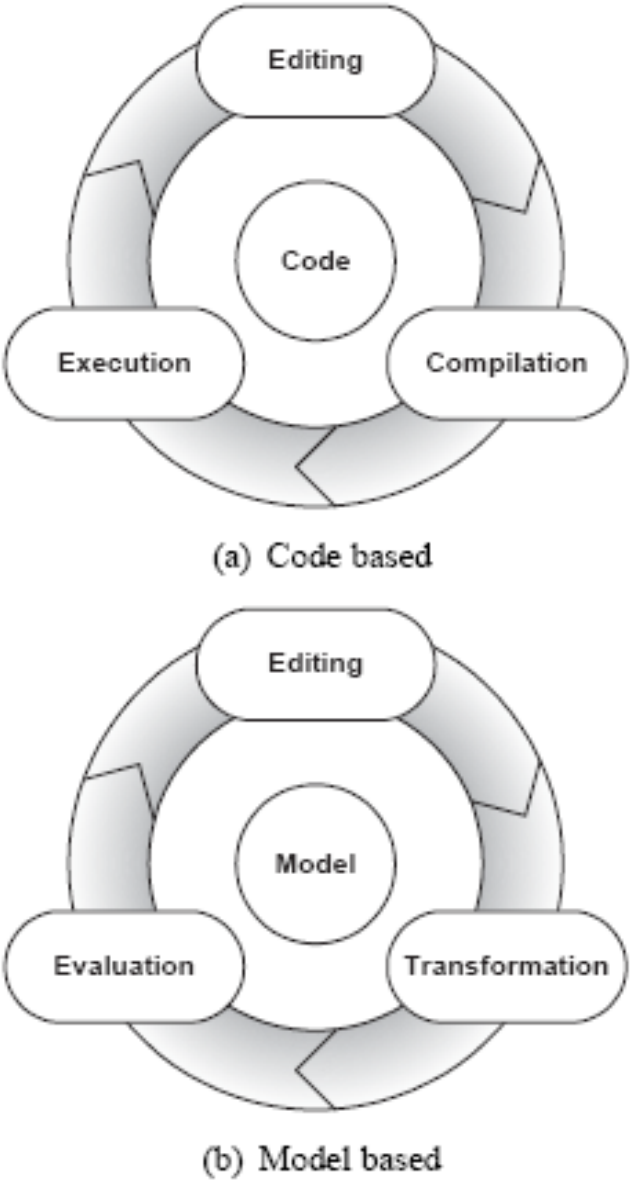


Figure 2 : Performance tuning.

3. Performance Modeling and Prediction Approaches

In this section we present some of the most relevant approaches for performance modeling and prediction of parallel and distributed computing systems.

3.1 A1: PAL

This approach for the performance modeling and prediction of distributed computing systems is commonly used by the Performance and Architecture Laboratory (PAL) at Los Alamos National Laboratory [24, 25].

The PAL approach expresses the execution time of a program on a machine as a parameterized mathematical model. The model parameters characterize the problem size S, and computational and communicational capabilities of the machine M. The program execution time $T_{ProgExec}$ is estimated as follows,

$$T_{ProgExec}(S,M) = T_{Comp}(S,M) + T_{Comm}(S,M) + T_{MemCont}(S,M) \quad (1)$$

where T_{Comp} is the computation time, T_{Comm} is the communication time, and $T_{MemCont}$ is the time spent for memory contention within a multiprocessor node. By expressing the performance of the whole program with a mathematical expression the structural information (for instance the control flow) of the program is not preserved. For this reason, this approach may not be suitable for the model-based performance evaluation of various program designs.

Workload model. The development of the workload model is based on a detailed analysis of the source code of the program (such as counting computation and communication operations). The modeling procedure results with a parameterized mathematical model (see Equation 1).

Machine model. Machine is not modeled separately. Machine is characterized by a set of parameters such as number of processors per node, number of nodes, number of communication links per node, communication latency and bandwidth, and sequential processing capacity. These parameters serve as input for themodel that predicts the program execution time.

Model development effort. PAL modeling approach requires a careful analysis of the program control flow and data structures. Furthermore, computer measurement techniques are used for determining the values of various machine parameters such as communication latencies and bandwidths, sequential processing capacity, and memory contention. Therefore, the modeling effort for this approach is considered to be high.

Model evaluation effort. The modeling procedure results with a mathematical expression, which is suitable for fast evaluation. Therefore, the model evaluation effort for this approach is low.

Efficiency. We consider that the efficiency of this approach is medium, since the model development effort is high but the model evaluation effort is low.

Accuracy. In [25] authors of PAL approach report performance prediction results for various systems with average accuracy between 5% and 11%. We should emphasize that the performance effects of the overlapping of computation and communication phases are not considered. Therefore, models that are built based on PAL approach may provide accurate results only for that class of programs for which the overlapping of computation and communication does not affects significantly the overall program performance.

3.2 A2: PEVPM

Performance Evaluating Virtual Parallel Machine (PEVPM) [14, 16] is a performance modeling system for message-passing programs. The basic idea behind PEVPM is to use statistical distributions to model the performance of the message passing operations, such as *send* and *receive*. The use of PEVPM is limited to message passing programs. Therefore, it is not possible to use it for shared memory programs, or mixed message passing and shared memory programs.

Authors of PEVPM state that, because of the network contention, the completion of message-passing operations varies over time. Therefore, by using statistical distributions instead of the average completion time the effect of network contention is captured. However, authors of PEVPM do not provide an empirical or theoretical proof that the use of statistical distributions instead of mean values to model the execution time of message passing operations results with a better estimation of the programs execution time. On the other hand, their basic claim of prediction accuracy improvement is not supported by fundamental probability theorems [22].

In order to generate statistical distributions for message passing operations, authors of PEVPM have developed a special-purpose benchmarking tool MPI Bench [15]. This tool measures the performance of operations of an MPI implementation on a particular machine, and automatically fits the measurement data with statistical distributions. For each MPI operation is generated one statistical distribution.

Workload model. During the workload modeling procedure the user manually inserts in the source code *performance directives* as comments. Basically, these directives describe all computation and communication operations of the program. In addition, these directives partially describe the program control flow. Since PEVPM provides no compiler support, the user manually generates the *driving program* for the simulation, by extending the program code with *directives*.

Machine model. PEVPM comprises a simple machine model. The machine is modeled as a set of processors each having one send queue and one receiving queue. The memory hierarchy is not modeled. The network topology is not modeled as well.

Model development effort. An obstacle for the usage of PEVPM is the difficulty of performance model description; the user describes the model textually by manually modifying the source code of the program that is modeled. For real world codes that may comprise thousands of lines of codes this approach is impractical. Furthermore, since PEVPM workload models correspond closely (virtually to each program statement correspond one or more directives) to the modeled program, it is difficult to build performance models at a higher level of abstraction.

Model evaluation effort. Commonly the time needed to evaluate a PEVPM model is shorter than the time needed to execute the real program on a real machine.

Efficiency. We consider that the efficiency of this approach is medium, since the model development effort is high but the model evaluation effort is low.

Accuracy. The prediction accuracy of PEVPM is illustrated for small programs such as Jacobi iteration. Since PEVPM workload model is based on platform-dependent performance models of communication/computation operations and because of the workload model matches closely the program code the expected prediction accuracy is relatively high. The authors of PEVPM report that commonly the prediction error is within 5%. The machine model of PEVPM does not consider the memory hierarchy and the network topology. Therefore, in the case that the program performance is influenced by these subsystems the prediction accuracy may be substantially lower.

3.3 A3: PP

Performance Prophet (PP) [36] is a tool for performance modeling and prediction of parallel and distributed computing systems. This tool provides a graphical user interface, which alleviates the problem of specification and modification of the performance model. The user specifies graphically the performance model using the Unified Modeling Language (UML) [35, 33]. Afterwards, PP automatically transforms the performance model from UML to C++ and evaluates it by simulation.

PP is suitable for exploring a large set of possible program's versions, because of its efficiency of model building and evaluation. The rapid performance model evaluation capability of PP is due to a methodology that involves model simplification, and the combination of mathematical modeling with discrete event simulation. The aim is to combine the model evaluation efficiency of mathematical performance models with the structure awareness of simulation models. The behavior of the whole computing system is split-up into *action states* and *waiting states*. Mathematical modeling is used for modeling the performance behavior of action states, whereas the performance behavior of waiting states is simulated.

Workload model. The workload model is specified graphically based on UML.

Machine model. The machine model for clusters of SMP's is composed automatically based on the user specified parameters such as the number of nodes and the number of processors per node.

Model development effort. Commonly, the model is composed rapidly from the existing building blocks. In this case the model development effort is low. However, if the user needs to develop new building blocks, then the model development effort is increased significantly. Therefore, in the general case the model development effort is medium.

Model evaluation effort. The time needed for model evaluation is reduced, by (1) simplifying the model, and by (2) combining mathematical modeling and discrete event simulation. Therefore, the model evaluation effort is low. In a case study presented in [36], the model evaluation with PP on a single processor workstation was several thousand times faster than the execution time of the real program on a real cluster.

Efficiency. We consider that the efficiency of this approach is high, since the model development effort is medium but the model evaluation effort is low.

Accuracy. Authors of PP have assessed the accuracy of PP by modeling and simulating a real-world material science program that comprises about 15,000 lines of code [36]. The average prediction accuracy was about 7%.

3.4 A4: PACE

The Performance Analysis and Characterization Environment (PACE) [32, 34] is a tool-set for the performance prediction of distributed systems. Characterization Instrumentation for Performance Prediction of Parallel Systems (CHIP3S) is a language that PACE uses for description of the performance information of software and hardware components.

Commonly used performance evaluation techniques for computing systems include: measurement, analytical modeling (that is mathematical modeling), and simulation. PACE uses measurement and analytical modeling techniques. Analytical models usually have the form of a simple expression. Their advantage is the low evaluation cost, but they lack the structural information of the system.

Workload model. The source code is analyzed and translated into CHIP3S procedures. A compiler is used to translate CHIP3S into C language code.

Machine model. The machine configuration is specified in Hardware Model Configuration Language (HMCL). For instance, a cluster of 32 SunUltra10 nodes is specified as follows,

SunUltra10 cluster_a[32];

where SunUltra10 specifies the type of the node.

Model development effort. The model development with PACE involves an extensive analysis of the program code and machine. Therefore, the model development effort for this approach is high.

Model evaluation effort. The final result of the PACE modeling approach is a single executable file that represents the whole computing system. Some model parameters can be set as command line options at the model execution time. During the model execution, a set of symbolic expressions that represent various components of the system is evaluated. Therefore, the model evaluation effort is low.

Efficiency. We consider that the efficiency of this approach is medium, since the model development effort is high but the model evaluation effort is low.

Accuracy. In [20] authors of PACE report that the average prediction accuracy of PACE was 5%.

3.5 A5: POEMS

The aim of Performance Oriented End-to-end Modeling System (POEMS) [1] project (period of performance 1997-2000) was to develop an environment for performance modeling of parallel computing systems.

POEMS proposed a methodology for the evaluation of system model using multiple evaluation tools. The model of system is composed of *component models*. POEMS authors state that each component of the system model may be evaluated by the corresponding evaluation tool; the output of a tool serves as input for the subsequent tool [1]. We consider that in the general case the component models may be of different kinds and at different levels of abstraction, and therefore the output of an evaluation tool may not be *interpretable* for the subsequent evaluation tool.

Workload model. POEMS devised a graphical representation for parallel programs, which is based on task graphs. Each node of the task graph may represent a set of parallel tasks. Edges of the task graph may represent data flow or task precedence. However, POEMS does not provide the corresponding tool-support for graphical model composition.

Machine model. The processor and the memory subsystem are simulated with Simple Scalar [5], the network is simulated based on Parsec simulation language, and I/O subsystem is simulated with PIOSIM [3].

Model development effort. A relevant outcome of the POEMS project is an automatic task graph generator for High Performance Fortran (HPF) programs [2]. The tool support is provided by an extended version of *dHPF* [7] compiler. This approach supports the automatic model development for HPF programs. However, the automatic development of the machine model is not adequately addressed. The claim that the machine model may be automatically composed from existing components is not of particular practical relevance, if these components are not already developed.

Model evaluation effort. It is difficult to estimate, because of the large spectrum of addressed abstraction levels and proposed model evaluation tools. For equation solvers the model evaluation effort is low. But, for detailed simulators the model evaluation effort may be very high both in terms of the time and computing resources needed to evaluate the model.

Efficiency. This approach aims to incorporate a large spectrum of the performance models from the high-level mathematical models to the instruction-level simulators. Therefore, it is hard to estimate the efficiency of this approach.

Accuracy. It may be anywhere from low to high. It depends on the model abstraction level, system modeling process, and the used model evaluation tools.

3.6 A6: PMaC

In this section we describe the Performance Modeling and Characterization (PMaC) [37, 41] framework for performance prediction of message passing programs. PMaC approach involves the determination of the *machine profile* and the *program signature*. A machine profile comprises the information on how fast the machine can perform basic operations (for instance memory *store* and *load*). A program signature comprises the information on the quantity and the type of basic program operations. Basically, the machine profile determines the machine computational capacity, whereas the program signature determines the program computational requirements. The execution time of a program is estimated as follows,

$$T_{ProgExec} = \sum_{i=1}^{n} O_i \cdot T_i$$

where T_i is the execution time of operation O_i.

Tools that are used within PMaC framework include Machine Access Pattern Signature (MAPS) [29], MetaSim [30], Pallas MPI Benchmark (PMB) [38], MPIDtrace [31], and Dimemas [8, 12]. MAPS is used for evaluation of the performance of memory hierarchy. MetaSim tracer is used for collection of the information on store/load and floating point operations of a program. PMB is used for performance evaluation of MPI communication operations. MPIDtrace is used for collection of the information on communication operations of a program. Dimemas is a simulator for message passing programs.

Authors of PMaC framework propose a technique for model simplification that is based on program trace sampling [6]. This means that for each interval i of the program trace only a sample of the first s elements is considered. For instance, for a trace that comprises 1000000 elements, with i = 100000 and s = 1000, only these elements are considered {[1, 1000], [100001, 101000], [200001, 201000] ,..., [900001 : 901000]}. This technique may have been inspired from signal sampling in electrical engineering. But, we consider that this technique may not be suitable for modeling computer programs, because a critical performance information that may strongly influence the performance of the whole program may be left out by just sampling periodically the program trace.

Workload model. The program is represented as a trace of computation and communication operations.

Machine model. The machine is characterized by measuring the capacity of performing the basic computation and communication operations.

Model development effort. It involves the program instrumentation, program execution, and execution of benchmarks for the performance evaluation of memory hierarchy and the interconnection network of machine. Therefore, the model development effort for this approach is medium.

Model evaluation effort. The model evaluation involves the simulation of the interconnection network with Dimemas. Therefore, the model evaluation effort for this approach is medium.

Efficiency. We consider that the efficiency of this approach is medium, since the effort for the model development and evaluation is medium.

Accuracy. Authors of PMaC approach report that even for small programs, such as matrix-vector multiply, the

prediction error is up to 25%. Relatively high prediction errors may be due to the use of platform-independent models.

3.7 A7: CLUE

Cluster Evaluator (CLUE) [26, 17] is an execution-driven simulator that is used for performance evaluation of message passing programs on cluster computer architectures.

CLUE is based on Machine Independent Simulation System for PVM (MISS-PVM) [27]. Parallel Virtual Machine (PVM) is a software system that supports the message passing programming paradigm [44].

CLUE is used for performance evaluation of message passing programs that are developed based on PVM. The simulation is driven by the executing the slightly modified program, whose performance is analyzed. In order to simulate the performance of a program with CLUE, the kind of the header files is changed in the program source code and the program is linked with CLUE libraries. During the program execution PVM calls are redirected to CLUE. CLUE simulates the performance behavior of the call, and then passes the call to PVM. During the simulation CLUE generates a trace of communication events, which may be used for postmortem performance analysis.

Workload model. The program whose performance is analyzed acts as a workload for CLUE.

Machine model. Machine is characterized by a set of parameters that are stored in a configuration file. These parameters determine the communication and computation properties of the machine. The effects of the network contention are not considered.

Model development effort. The development of the workload model is strait forward. It involves only the modification of the program header. The development of the machine model involves the determination of the parameters that affect the performance of the machine.

Model evaluation effort. One of the drawbacks of execution driven-simulation is that the time needed for model evaluation strongly depends on the execution time of the real program. For instance, if the execution time of a real world program is several weeks, then the time needed for model evaluation is several weeks as well. Furthermore, real-world programs may have high memory requirements. This means that a large amount of memory may be required for the model evaluation.

Efficiency. We consider that the efficiency of this approach is medium, since the model development effort is low but the model evaluation effort is high.

Accuracy. CLUE is validated for ScaLAPACK [4, 40] routines such as *matrix multiplication*. Authors of CLUE state that the average prediction accuracy of CLUE was 10% (see [28]).

3.8 A8: POSE

Parallel Object-oriented Simulation Environment (POSE) [46] is a parallel discrete event simulator [11]. POSE is used for simulation of the performance behavior of programs that are executed on large-scale machines such as IBM Blue Gene [47, 19]. The Blue Gene/L [19], which is listed as number one in the list of 500 hundred most powerful computers in the world (TOP500 [45], November 2006), contains 131,072 processors. A detailed simulation of such large machines may require processing and memory resources that are not available on a singe processor machine. Therefore, such simulations are executed on multiprocessor machines.

POSE is implemented based on Charm++ [21], which is a parallel C++ library. The simulation entities of POSE are represented as Charm++ objects. Each object has a data member for tracking the simulation time and a set of methods for event handling. Based on POSE and the Charm++ runtime environment a specific simulator that simulates the Blue Gene/L machine was developed [47]. The rest of this section provides a discussion of properties of the Blue-Gene/L simulator.

Workload model. The Blue Gene/L simulator is an execution-driven simulator. It supports the simulation of programs that are written based on Charm++ or MPI.

Machine model. Machine is modeled as a set of interconnected nodes. Each node may have a set of processors.

Model development effort. Charm++ or MPI programs may be used as input for the simulator. In this case the workload modeling effort is low. However, in order to evaluate different program versions the user has to restructure the source of the program. This process may be time consuming for the real-world programs. The effort for the development of a detailed machine model that supports execution-driven simulation is high.

Model evaluation effort. Commonly sequential discrete event simulators are about 10 times faster than parallel discrete event simulators if a single-processor machine is used for model evaluation. This is due to the synchronization and communication overhead of parallel discrete event simulators. If a machine with more than 10 processors is available for model evaluation, then parallel simulators outperform sequential simulators (see [46]).

Efficiency. We consider that the efficiency of this approach is low, since the effort for the model development and evaluation is high.

Accuracy. We were unable to find a comparison of prediction results of Blue Gene/L simulator with measurement results on the real Blue Gene/L machine. The prediction accuracy of this approach may be anywhere from low to high, depending on the fidelity of the machine model.

3.9 A9: RSIM

Rice Simulator for ILP Multiprocessors (RSIM) is a simulator of cache-coherent non-uniform memory access (CCNUMA) shared-memory machines [39, 18]. A distinguishing feature of RSIM is the ability to simulate processors that use instruction-level parallelism (ILP). ILP

processors are capable of executing multiple instructions in parallel.

RSIM supports SPARC processors [42]. As input for RSIM simulator may serve programs that are compiled and linked (that is *executables*) on SPARC/Solaris systems. During the program simulation, RSIM interprets the executable of the program. The output of RSIM includes the number of executed cycles, and statistics on the utilization of components of the machine.

RSIM comprises a detailed (that is a cycle-level) machine model that allows the analysis of the performance effects of architectural parameters. Therefore, it is suitable to evaluate various designs of CC-NUMA shared-memory machines. However, because the simulation of the program execution with RSIM is very slow (several thousands times slower than the program execution on the real machine), it is not suitable for evaluation of various designs of real-world programs.

Work load model. The executable of the program serves as a workload. The RSIM simulator takes as input the programs that are compiled and linked on SPARC/Solaris systems.

Machine model. The main components of the machine model include processors, the memory hierarchy (that is L1 cache, L2 cache, local and remote memory), and the interconnection network.

Model development effort. The development of the workload model is strait forward. Basically, it involves program compiling and linking. But, the effort for development of the cycle-level machine model is high.

Model evaluation effort. Authors of RSIM report that several thousands times more time was needed to simulate a program with RSIM, than to execute the program on a real machine. For instance, for a program that performs *LU matrix decomposition* the RSIM simulation was 7100 times slower than the program execution on the real machine. Therefore, the model evaluation effort for this approach is high.

Efficiency. We consider that the efficiency of this approach is low, since the effort for the model development and evaluation is high.

Accuracy. The model of the computer architecture that is used by RSIM does not match exactly the architecture of any existing machine. This makes difficult the task of estimation of the prediction accuracy of RSIM, since it is not possible to compare simulation results with results that are obtained from the real computing system.

4. Evaluation

In this section we evaluate the suitability of the approaches, which we described in Section 3, for the model-based performance analysis of parallel and distributed computing systems. The time spent for the model development and evaluation should be short, in order to explore a large set of design alternatives within a reasonable time. The approaches should provide performance prediction results with an *accuracy* that makes possible the *comparison* of various design alternatives. We evaluate the approaches based on the following properties: (1) model development effort, (2) model evaluation effort, (3) efficiency, and (4) accuracy. The efficiency of an approach is deduced based on the model development and evaluation effort.

Table 1 shows an estimation of values of the properties of approaches. The values of properties are obtained from our reasoning about each approach in Section 3. Approaches are arranged in the table in the order of their appearance in Section 3.

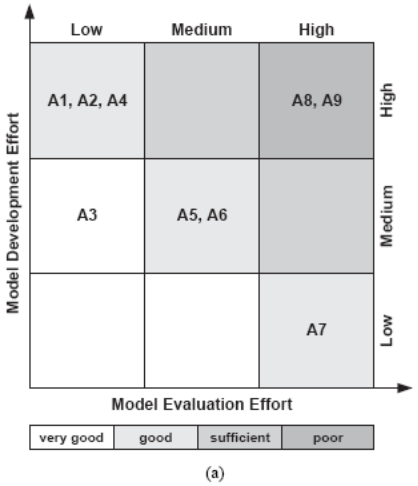
Table 1: A summary of the properties of approaches for the performance modeling and prediction of parallel and distributed computing systems.

ID	Mod. Eval.	Mod.	Dev. Efficiency	Accuracy
A1	Low	High	Medium	Medium
A2	Low	High	Medium	Medium
A3	Low	Medium	High	Medium
A4	Low	High	Medium	Medium
A5	Medium	Medium	Medium	Medium
A6	Medium	Medium	Medium	Low
A7	High	Low	Medium	Medium
A8	High	High	Low	Medium
A9	High	High	Low	High

Figure 3 depicts the assessment results of approaches based on the values of their properties (see Table 1).

Figure 3(a) provides an assessment of approaches by considering the model development and evaluation effort. Approaches A8 and A9 received the grade *poor*; A1, A2, A4, A5, A6 and A7 received the grade *good*; and A3 received the grade *very good*.

Figure 3(b) provides an assessment of approaches by considering the accuracy and efficiency. A6 and A8 received the grade *sufficient*; A1, A2, A4, A5, A7 and A9 received the grade *good*; and A3 received the grade *very good*.



(a)

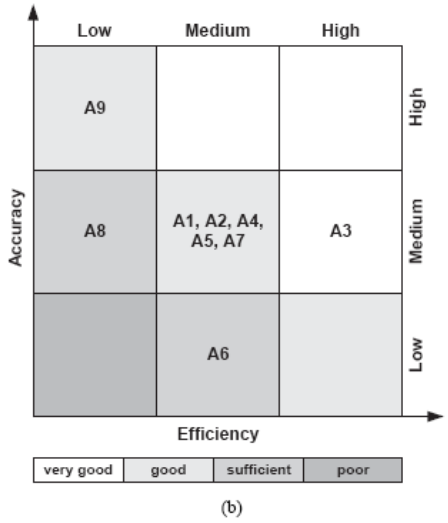


Figure 3 : The assessment of approaches based on (a) model development effort and model evaluation effort, (b) accuracy and efficiency.

5. Conclusions

Performance is a key indicator of parallel and distributed computing systems. Therefore, the performance evaluation of computing systems was a preoccupation of many computer scientists in the past. In this paper we have systematically presented the state of the art approaches for performance modeling and prediction of parallel and distributed computing systems. We have discussed the contributions and drawbacks of each approach. In what follows we summarize some of the issues that we have identified.

Most of approaches for the performance modeling of parallel and distributed programs are of limited use to support performance-oriented software engineering because of the following reasons: (1) the use of a notation that is not based on widely accepted standards, and (2) the requirement that the software engineer has a thorough understanding of the underlying performance modeling technique. Some approaches aim to bridge this gap between the performance modeling and the software engineering by incorporating UML.

Most of approaches are able to cope only with small programs such as matrix-vector multiplication. There are several reasons for the lack of scalability: (1) a very complex code analysis is used during the workload modeling that does not scale up to the size and complexity of the real world programs, (2) a detailed machine model is used that is so slow that makes impractical the simulation of real-world programs, or (3) for the model evaluation are required very large resources (processors and memory) that may not be available. Some approaches have addressed this issue by using model simplification techniques, combination of mathematical modeling with discrete event simulation, and by using a simple machine simulation model.

Performance models that represent the whole program and machine as a symbolic expression lack the structural information. Consequently, it is difficult to identify the part of system that is responsible for the sub optimal performance. Some approaches support the development of performance models at various levels of abstraction. For instance, for workload modeling are used UML activity diagrams. An activity may represent a single instruction, or larger blocks of the program (for instance a *loop*), or the whole program. Furthermore, some approaches use discrete-event simulation to describe the structure of system and the interaction among its components.

In future we plan to supplement our survey by including other approaches for performance modeling and prediction of parallel and distributed computing systems.

Acknowledgments

The work described in this paper was partially supported by the Austrian Science Fund (FWF) under the project AURORA.

Bibliography

[1] V. Adve, R. Bagrodia, J. Browne, E. Deelman, A. Dube, E. Houstis, J. Rice, R. Sakellariou, D. Sundaram-Stukel, P. Teller, and M. Vernon. POEMS: End-to-End Performance Design of Large Parallel Adaptive Computational Systems. *IEEE Transactions on Software Engineering*, 26:1027–1048, November 2000.

[2] V. Adve, R. Bagrodia, E. Deelman, T. Phan, and R. Sakellariou. Compiler-Supported Simulation of Highly Scalable Parallel Applications. In *Proceedings of SC99*, Portland, Oregon, USA, 1999. ACM Press.

[3] R. Bagrodia, S. Docy, and A. Kahn. Parallel Simulation of Parallel File Systems and I/O Programs. In *SC97 (SuperComputing 97)*, San Jose, CA, 1997. ACM.

[4] L. Blackford, J. Choi, A. Cleary, E. Azevedo, J. Demmel, I. Dhillon, J. Dongarra amd S. Hammarling, G. Henry, A. Petitet, K. Stanley, D. Walker, and R. Whaley. ScaLAPACK: A Linear Algebra Library for Message-Passing Computers. In *The Eighth SIAM Conference on Parallel Processing for Scientific Computing (PPSC 1997)*, Minneapolis, Minnesota, USA, 1997. SIAM.

[5] D. Burger and T. Austin. The SimpleScalar Tool Set, Version 2.0. Technical Report CS-TR-1997-1342, Computer Sciences Department, University of Wisconsin-Madison, 1997.

[6] L. Carrington, A. Snavely, X. Gao, and N. Wolter. A Performance Prediction Framework for

Scientific Applications. In *International Conference on Computational Science (ICCS 2003)*, Melbourne, Australia, June 2003. Springer.

[7] dHPF Compiler. Department of Computer Science, Rice University, http://www.cs.rice.edu/dsystem/dhpf/.

[8] Dimemas. http://www.cepba.upc.edu/dimemas/.

[9] J. Dongarra. An Overview of High-Performance Computing and Challenges for the Future. http://www.netlib.org/utk/people/JackDongarra/talks.htm, May 9 2006.

[10] The Earth Simulator Center. http://www.es.jamstec.go.jp/esc/eng/.

[11] R. Fujimoto. *Parallel and Distributed Simulation Systems*. John Willey, 2000.

[12] S. Girona, J. Labarta, and R. Badia. Validation of Dimemas Communication Model for MPI Collective Operations. In *7th European PVM/MPI*, volume 1908 of *Lecture Notes in Computer Science*, Balatonf¨ured, Hungary, September 2000. Springer-Verlag.

[13] S. Graham, M. Snir, and C. Patterson. *Getting Up to Speed: The Future of Supercomputing*. The National Academies Press, 2004.

[14] D. Grove. *Performance Modelling of Message-Passing Parallel Programs*. PhD thesis, Department of Computer Science, University of Adelaide, Australia, 2003. PhD Thesis.

[15] D. Grove and P. Coddington. Precise MPI Performance Measurement Using MPIBench. In *HPC Asia*, pages 24–28, Gold Coast, Australia, September 2001.

[16] D. Grove and P. Coddington. Performance Modeling and Evaluation of High-Performance Parallel and Distributed Systems. *Performance Evaluation*, 60(1–4):165–187,May 2005.

[17] H. Hlavacs, D. Kvasnicka, and C. Ueberhuber. CLUE - A tool for Cluster Evaluation, Distributed and Parallel Systems. In *DAPSYS 2000*, pages 61–64, Balatonfuered, Lake Balaton, Hungary, September 2000. Kluwer Academic Publishers.

[18] C. Hughes, V. Pai, P. Ranganathan, and S. Adve. RSIM: Simulating Shared-Memory Multiprocessors with ILP Processors. *IEEE Computer*, 35(2):40–49, February 2002.

[19] IBM Research: Blue Gene Project. http://www.research.ibm.com/bluegene/.

[20] S. Jarvis, D. Spooner, H. Keung, J. Cao, S. Saini, and G. Nudd. Performance Prediction and its use in Parallel and Distributed Computing Systems. In *IEEE/ACM International Workshop on Performance Modelling, Evaluation and Optimization of Parallel and Distributed Systems held as part of the 17th IEEE International Parallel and Distributed Processing Symposium 2003 (IPDPS03)*, Nice, France, April 2003. IEEE Computer Society.

[21] L. Kale and S. Krishnan. *Parallel Programming using C++*, chapter Charm++: Parallel Programming with Message-Driven Objects. MIT Press, 1996.

[22] O. Kallenberg. *Foundations of Modern Probability*. Springer-Verlag, 1997.

[23] H. Karatza. *Applied System Simulation: Methodologies and Applications*, chapter Simulation of Parallel and Distributed Systems Scheduling, Concepts, Issues and Approaches. Springer, 2003.

[24] D. Kerbyson, H. Alme, A. Hoisie, F. Petrini, H. Wasserman, and M. Gittings. Predictive Performance and Scalability Modeling of a Large-scale Application. In *ACM/IEEE conference on Supercomputing (SC 2001)*, Denver, CO, USA, November 2001. ACM.

[25] D. Kerbyson, A. Hoisie, and H. Wasserman. Use of Predictive Performance Modeling During Large-Scale System Installation. *Parallel Processing Letters*, 15(4), December 2005.

[26] D. Kvasnicka, H. Hlavacs, and C. Ueberhuber. Simulating Parallel Program Performance with CLUE. In *International Symposium on Performance Evaluation of Computer and Telecommunication Systems (SPECTS)*, pages 140–149, Orlando, Florida, USA, July 2001. The Society for Modeling and Simulation International.

[27] D. Kvasnicka and C. Ueberhuber. Developing Architecture Adaptive Algorithms Using Simulation with MISS-PVM for Performance Prediction. In *International Conference on Supercomputing (ICS'97)*, Vienna, Austria, 1997. ACM.

[28] D. Kvasnicka and C. Ueberhuber. Simulating Architecture Adaptive Algorithms with MISS-PVM. AURORA TR-1997-06, University of Vienna, VCPC, June 1997.

[29] Machine Access Pattern Signature (MAPS). http://www.sdsc.edu/PMaC/MAPs/maps.html.

[30] MetaSim. http://www.sdsc.edu/PMaC/MetaSim/metasim.html

[31] MPIDtrace. http://www.cepba.upc.edu/.

[32] G. Nudd, D. Kerbyson, E. Papaefstathiou, S. Perry, J. Harper, and D. Wilcox. PACE - A Toolset for the Performance Prediction of Parallel and Distributed Systems. *International Journal of High Performance Computing Applications*, 14(3):228–251, 2000.

[33] Object Management Group (OMG). UML 2.0 Superstructure Specification. http://www.omg.org, August 2005.

[34] Performance Analysis and Characterisation Environment (PACE). http://www.dcs.warwick.ac.uk/research/hpsg/pace/paceintroduction.html.

[35] S. Pllana and T. Fahringer. UML Based Modeling of Performance Oriented Parallel and Distributed Applications. In *Proceedings of the 2002 Winter Simulation Conference*, San Diego, California, USA, December 2002. IEEE.

[36] S. Pllana and T. Fahringer. PerformanceProphet: A Performance Modeling and Prediction Tool for Parallel and Distributed Programs. In *The 2005 International Conference on Parallel Processing (ICPP- 05). Performance Evaluation of Networks for Parallel, Cluster and Grid Computing Systems.*, Oslo, Norway, June 2005. IEEE Computer Society.

[37] Performance Modeling and Characterization (PMAC). http://www.sdsc.edu/PMaC/.

[38] Pallas MPI Benchmarks (PMB). http://www.pallas.com/e/products/pmb/.

[39] Rice Simulator for ILP Multiprocessors (RSIM). http://rsim.cs.uiuc.edu/rsim/.

[40] The ScaLAPACK Project. http://www.netlib.org/scalapack/.

[41] A. Snavely, L. Carrington, N. Wolter, J. Labarta, R. Badia, and A. Purkayastha. A Framework for Performance Modeling and Prediction. In *The 2002 ACM/IEEE conference on Supercomputing*, Baltimore, Maryland, USA, November 2002. ACM.\

[42] SPARC International. http://www.sparc.org/.

[43] Simulating the Planet Earth. http://www.nec.com/global/features/index9/index.html.

[44] V. Sunderam, A. Geist, J. Dongarra, and R. Manchek. The PVM Concurrent Computing System: Evolution, Experiences, and Trends. *Parallel Computing*, 20(4):531–545, 1994.

[45] TOP500 Supercomputer Sites. http://www.top500.org/.

[46] T.Wilmarth, G. Zheng, E. Bohm, Y.Mehta, N. Choudhury, P. Jagadishprasad, and L. Kale. Performance Prediction using Simulation of Large-scale Interconnection Networks in POSE. In *2005 Workshop on Principles of Advanced and Disctributed Simulation (PADS)*,Monterey, California, June 2005. IEEE Computer Society.

[47] G. Zheng, G. Kakulapati, and L. Kale. BigSim: A Parallel Simulator for Performance Prediction of Extremely Large Parallel Machines. In *18th International Parallel and Distributed Processing Symposium (IPDPS 2004)*, Santa Fe, New Mexico, USA, April 2004. IEEE Computer Society.