Developing Architecture Adaptive Algorithms using Simulation with MISS-PVM for Performance Prediction

Dieter F. Kvasnicka, Christoph W. Ueberhuber
Institute for Applied and Numerical Mathematics
University of Technology Vienna
dieter@titania.tuwien.ac.at

Abstract

The idea of ARCHITECTURE ADAPTIVE ALGORITHMS was introduced in order to reduce porting overhead in parallel algorithms. The AAA presents a way for algorithms to adapt themselves to the computer system on which they are running without the assistance of the application programmer. For the development of these algorithms it is, nevertheless, necessary to obtain reliable and reproducible execution time information. A good method for achieving these results is to simulate the hardware. In this paper a new simulator (the *Machine Independent Simulation System for PVM 3*) is presented and its usefulness for this type of application is investigated.

1 Introduction

Since parallel programming is more complicated than sequential programming it is only advisable to use it when the performance of sequential programming comes to its limits. When developing parallel programs one has to cope with problems that do not occur when developing sequential programs. One of these problems is that parallel programs need not only to be portable in the sense that they run on a variety of parallel platforms, but they also have to run efficiently on these platforms.

This paper starts by describing the reasons why ARCHITECTURE ADAPTIVE ALGORITHMS (AAAs) were developed and by explaining why architecture adaptivity is important to parallel programming. The special needs of AAA developers are then discussed. These needs may be satisfied by use of simulation tools. Chapter 3 gives an overview of existing simulators, points out different attempts to overcome simulation obstacles, and evaluates these attempts in terms of AAA development.

The next chapter introduces a new simulation engine, the *Machine Independent Simulation System for PVM 3* (MISS-PVM), which has been especially designed for the purposes of the development of AAAs. This engine is able to simulate a wide variety of parallel computers which follow the message passing paradigm. It is based on a combination of program execution and simulation. The execution part profits from exact time measurements taken from actual parallel computers. The simulation part is implemented in such a way that different parallel target computers can be parametrized and evaluated without significant overhead.

In chapter 5 a model for an IBM SP2 and simulation runs on an SGI Power Challenge XL are described. In chapter 6 the model parameters are determined. The validation of the model by use of a matrix triangularization algorithm is the topic of chapter 7.

To appear in the 11th ACM Conference on Supercomputing July 1997, Vienna, Austria.

2 AAA

The fundamental idea behind the ARCHITECTURE ADAPTIVE ALGORITHMS methodology was started in 1967 by Peter Naur [16]:

> We have to admit a new kind of elements, *Environment Enquiries*, in our common programming languages. These should be designed to place a carefully chosen set of information about the available equipment at the disposal of the programmer, for use in directing the control of the process.

When this quotation was written (thirty years ago), environment enquiries were supposed to yield information about machine numbers (radix, minimum and maximum exponent, etc.) and computer arithmetic. By taking this information into account, an algorithm can adapt itself to machine characteristics and thus significantly increase its portability. After long discussions arithmetic enquiry functions were included in the Fortran language definition.

Multiprocessor algorithms need environment enquiries to provide information about the processors, communication channels, and memory hierarchies available within the computing environment. By taking this information into account, a portable parallel algorithm can adapt its behavior to environment characteristics and thus significantly increase its efficiency (Krommer, Ueberhuber [13]).

This adaptation can be either done (i) during compile time or (ii) during execution time. The first approach should be used with sequential algorithms because it improves overall performance. With parallel algorithms, however, it is more likely that the hardware characteristics may change after compile time (for instance, the number of available processors may change). Therefore adaptation to hardware parameters should be made at execution time.

In a workstation cluster, for example, some machines can temporarily be exclusively assigned to certain users or applications and, thereby, become unavailable to others. As such changes may occur not only in between, but also during program runs, program adaptation has to take place at execution time.

Performance Prediction for AAA Development The development of parallel AAAs is faced with one major obstacle: parallel hardware exists in discrete form only. It cannot be parametrized to meet the requirements of the AAA designer.

This difficulty may be overcome by the simulation of parallel hardware. Variation of the *number of available processors,* of the *processor's performance,* of *network topologies* and of the *speed of interconnection hardware* must be possible for the simulation environment to meet the specific needs of the algorithm designers.

3 Performance Prediction using Simulation

This chapter describes several attempts to cope with the difficult problem of creating a fast and accurate simulation of parallel computers. It is not meant to give a full overview of existing simulation systems, but to point out directions in which the simulation of parallel computers is developing.

Hardware Simulation Simulation can be done at different levels of abstraction, resulting in different levels of achievable accuracy. One of the low-level hardware simulators is TALISMAN (Bedichek [3]). It reaches quite high accuracy at the cost of 100 host instructions per simulated instruction.

Hardware Emulation An alternative way to speed up model execution is to use hardware emulation instead of simulation. Barroso et al. [2] present their emulator RPM, which speeds up model execution by several orders of magnitude. Another advantage of emulation is that the code does not have to be instrumented to be processed by the simulator. Drawbacks are, however, time consuming development of the hardware, and the limited adaptability to changing hardware parameters.

Analytic Models Analytic methods present another way to achieve estimates on parallel program execution time. General analytic models are very hard to obtain, but it can be shown in special cases that they achieve good accuracy (Peterson, Chamberlain [17]). They also provide better insight into the parametrization of the application.

Trace-Driven versus Execution-Driven Simulation Trace-driven simulation is based on a trace file obtained by actually running the program on a parallel computer. Time information is obtained by simulation runs using the trace file. The accuracy of trace-driven simulation is evaluated in Goldschmidt, Hennessy [12]. They state that this method of simulation is inaccurate unless time dependencies are eliminated from the traces. This implies that dynamic load balancing algorithms cannot be accurately modeled in trace-driven simulators. In this case direct simulation has to be used.

Execution-driven simulation is a simulation that is performed during the execution of the parallel program. In Davis et al. [8] trace-driven and execution driven simulation are compared and evaluated with the simulation system TANGO. Execution-driven simulation turns out to be the most accurate by far. The authors also discuss causes of simulation overhead.

The simulation system TANGO can be used for algorithm studies as well as for hardware evaluation. It runs on a single processor and emulates target machines by multiplexing.

Another simulator is PROTEUS (Brewer, Weihl [4], Brewer et al. [5]). The strong points of PROTEUS are *versatility, trivial repeatability, detailed nonintrusive data collection* and *debugging.* PROTEUS uses multiple implementations of modules to provide different combinations of accuracy and performance. It is one of the first simulators which provides an environment fast enough to develop real parallel applications including debugging, testing and some tuning. On many current parallel machines the repeatability of benchmarks is rather poor, especially when compared to simulation. The effects of different parameters can be determined even faster by simulation than by executing the program on a parallel computer. Of course, the simulation system has to be fast enough.

Benchmarks The simulation of existing parallel machines can only be done accurately when the performance data is of sufficient accuracy. So accurate benchmarks have to be developed. The GENESIS benchmark suite (Addison et al. [1]) was constructed to provide portable benchmarks for different distributed memory systems using the PARMACS macros. In this paper the difficulties in gaining accurate numbers for the evaluation of parallel machines are shown. It is important to remember that it is useless to strive for a simulation which is more accurate than the data accuracy. There is no sense in creating a model which is accurate up to the tenth of a percent when the parameters are reliable only up to ten percent.

Invasiveness When measuring the performance data of parallel machines, every act of measuring has some influence on the system. An overview of the invasiveness of measurements at different hardware platforms and using different applications can be found in Naim, Hey [15], with special emphasis on the invasiveness of PARMACS.

Another parallel benchmark specification can be found in Figueira, Hernandez [11]. Their main goal was to estimate the performance of a given application before fully developing the application for the target system. This has to be done for every application. Surprisingly, the overhead incurred is rather low compared to full simulation.

Wabnig, Haring [20] also concentrate on the crucial point of determining performance estimates in early states of the parallel program development. Their tool PAPS is used before any actual coding on real hardware.

Simulation Environments Some simulators are embedded in whole parallel programming environments like DIMEMAS, which is described in Labarta et al. [14].

Another simulation project is the WISCONSIN WIND TUNNEL (Burger, Wood [6], Reinhardt et al. [18]). This simulator predicts the performance of shared-memory machines using several different network simulation models. A two-fold technique is used: the program is executed (cache hits remain unchanged), and cache misses are simulated by trapping to the WWT. The authors show that more expensive models (in terms of execution time) have to be used in the presence of high network loads or non-uniform traffic patterns to achieve satisfactory accuracy.

Candlin, Phillips [7] use statistical techniques to identify the way in which program characteristics influence performance and also include the effects of the runtime environment.

Scalability *Scalability* is a point of major interest in parallel application development. The use of simulators can provide insight into this important matter (Sivasubramaniam et al. [19]). The tool Spasm copes with the problem of scalability. Especially those effects which can be neglected in simulation systems for small scale machines (*contention, synchronization,* or *load balancing*) have to be taken into account here. The authors established several performance metrics like speedup, scaled speedup, sizeup, or an isoefficiency function and an application for scalability problems.

Lapse is an example for a simulator which uses parallel simulation on a small number of processors to predict the performance on large numbers of processors (Dickens et al. [9] and Dickens et al. [10]). Lapse distinguishes between *execution threads* and *communication.* Only the communication has to be simulated. Therefor there are two extremes in the simulation: if communication is rare, simulation is (at first sight) slowed down only by the number of processors; if communication is frequent, the simulation is also slowed down by communication and simulation overhead. A slowdown of 2-15 is also achieved by other simulators.

The Evaluation of Existing Systems Simulation systems which really execute code on a parallel machine (because of performance reasons and the fact that often data dependent programs have to be simulated), which is easy to use, which provides its full functionality without instrumenting the source code of a program (which would be much overhead when simulating whole existing parallel program libraries), which may adapt to a varying environment during program execution, and which provides *environment enquiries* (chap. 2) are necessary for the development of AAAs. Some of the above mentioned systems are fast enough and sufficiently developed to be easy to use. However, for the sort of flexibility needed for AAA development, no available simulation system is able to fully meet our requirements.

4 MISS-PVM

The execution driven, discrete-event *Machine Independent Simulation System for PVM 3* (*MISS-PVM*) was developed to provide an easy-to-use and fast simulation environment which is able to obtain performance estimations from parallel computers by executing parallel programs on host machines which do not necessarily have to have the same architectures as the target computer.

The advantages of MISS-PVM in comparison to other approaches of simulating parallel architectures are: MISS-PVM has high performance; existing applications (using PVM 3) can be directly simulated with hardly any changes in the code; and the flexibility of the PVM 3 execution model is fully exploited. MISS-PVM is very suitable for problems which can be solved on networks of workstations (even heterogeneous ones). It is also useful for simulating real parallel computers as long as the programs use the message passing paradigm. This does not mean that the machine has to be of the distributed memory type. Efficient implementations of PVM on shared memory machines are available. The main hosts of MISS-PVM are the SGI and HP multiprocessors.

The excellent performance of MISS-PVM is due to the fact that sequential threads are directly executed and therefore do not create communication overhead. Also the simulation of message-passing is quickly done. The model is implemented as C functions and is linked to the executable program as a library.

The message passing tool PVM was chosen because virtual machines, their process numbers and their structures may change dynamically in the PVM model. The new message passing standard MPI (Walker [21]) does not provide this flexibility.

MISS-PVM uses the PVM communication calls with a special feature: the code which contains calls to PVM-routines can be used for simulation without (or just very few) adaptations to the source code. The output of the simulation system may be visualized after the program run. Simulation models are implemented as library calls, which return time estimates for communication and computation times when called with appropriate parameters. Different target computers are compared by using different simulation libraries when linking the executable program. Several parameters of the target computer are read from a file each time a new process is started.

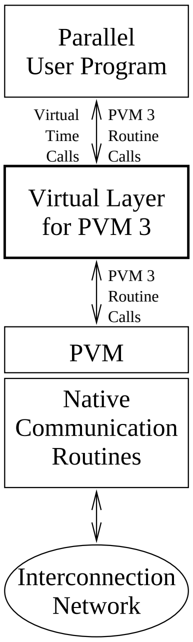

Figure 1: The Virtual Layer of MISS-PVM is situated between the Parallel User Program and PVM 3 without affecting the behavior of either one.

Figure 1 depicts the new level between the user program and PVM 3: the MISS-PVM library is called *Virtual Layer for PVM 3.* It uses PVM 3 calls and does not contain machine dependent routines. The *Virtual Layer for PVM 3* has the obvious advantage of being able to run on a variety of machines and makes virtual time, virtual machines, and output generation for graphical post mortem visualization available. The user program as well as the PVM 3 level remain unchanged. The only difference is that an include file redirects PVM 3 routine calls. In C the redirection works with compiler directives of the form `#define pvm_recv pvmV_recv`.

For Fortran programs a compatibility library was created. These do not even have to make changes in the source. Only in the Makefile the PVM library has to be converted into the MISS-PVM library.

The virtual layer is able to create an output file which is used for post mortem visualization (after postprocessing). This output file can also be written in a format which is readable by the program developer. For switching between formats global variables have to be changed.

The Concepts of Virtual Time and Virtual Hosts All the processes running in parallel run independent of each other and each process has its own virtual (simulated) time. This time is exchanged whenever it is necessary to synchronize their virtual time with other processes. This is a one-way communication: in case timing conflicts might occur, one process sends its virtual time to another process, which advances its own virtual time to avoid problems. The main difficulty is that tachyons, messages which are received before (according to the virtual time of the receiving process) they have been sent (according to the virtual time of the sending process), may occur.

Every process may run on virtual hosts with different characteristics. When starting a new process, MISS-PVM reads several host parameters from a file and, depending on the data found in this file, passes the name of the host and other parameters used in the `pvm_spawn()` function call to PVM 3 (maybe in a modified form). This is flexible enough to simulate heterogeneous workstation environments on shared-memory multiprocessors or shared-memory computers (using message-passing communication calls) on clusters of workstations.

MISS-PVM Library There are four kinds of MISS-PVM library routines:

1. *delay routines*: The simulation model is created in the form of several routines which estimate time. These may, if necessary, also be accessed by the user program. This is one of the main entry points for environment enquiries when developing AAAs.
2. *timing routines*: A machine timing routine is redirected to MISS-PVM in order to return simulation time instead of real time (`gettimeofday`). This timer is the *fastest way to obtain simulation results.*
3. *pvmV-routines (Virtual PVM routines):* Normal PVM-calls are redirected to these routines to call the simulation system when necessary. These are the only entrance points for simulation.
4. *pvmS-routines (MISS-PVM service routines):* These routines work on a lower level than the Virtual PVM routines. Some may be accessed by the user program. There are also global variables which may be read (and even set) directly by the user program.

Performance One of the advantages of MISS-PVM is its high performance, especially when run on a shared memory parallel computer.

Programs with *few communication* requirements run without any decrease in performance compared to running without MISS-PVM, because only communication is simulated.

Programs with *higher communication* requirements on the other hand improve when running on a shared memory architecture. If a process has to wait because of a blocking receive statement it will be delayed by the simulation layer of MISS-PVM until the message has arrived. On parallel processors with distributed memory this would mean that the processor has to idle. (Using the multi-tasking kernels of parallel computers of the last generation has the obvious advantage that the processors can at the same time do work related to other users. Because of performance reasons this option should normally only be chosen in the development phase. Previously parallel computers only had dedicated processors which had to idle when the single user did not use them 100 percent.) On a shared memory computer, other threads of the same task are simulated to be on another node. They can run on the processor while it is waiting to perform the first task.

Taking this behavior into account it is possible to simulate several processors of a bigger machine on only a few processors of the host machine which can be very advantageous (Dickens et al. [9] and Dickens et al. [10]).

Intrusiveness To improve timing estimations, the time used for *timing routines* and for *simulation functions* has to be taken into account. On computers with accurate and reliable timers this can be easily achieved as straightforward measurements (i. e. on the SGI Power Challenge XL which measures user time with an accuracy of up to microseconds).

5 The Model

The parameters of a simulation model are on one hand *computation time* (a performance factor) and on the other hand *communication time*.

Computation Two methods for timing the computational parts of the simulated code are

1. *Measuring the execution time on the simulation host (Default)*: MISS-PVM automatically takes the time between two consecutive calls of any PVM-routine (or at least of most). This time is modified by a constant factor to estimate the actual execution time on the target computer. It is also decremented by the time spent in the timing routine itself. Therefore the simulation should run on a machine which is able to measure program timing information in microseconds (like the SGI Power Challenge XL, but unlike most workstations).

   Due to the form of the computation measurement, special care has to be taken about *optimization levels,* which can change the prediction of execution time by high amounts. The computation factor only represents the relation between one simulation and one execution time at once. (For instance the fraction between fully optimized programs on the target and the simulation host.) The optimization level also has a strong effect on the ratio of computation to communication. The program developer has to be aware of this effect when choosing the optimization level especially in the development phase when it might be necessary to compile with debug information.
2. *Time estimations included in the source code*: To increase the accuracy and the performance of the simulation automatic computation time estimation may be turned off for some parts of the code. Time estimated by use of the measurements of the target processors is inserted in form of MISS-PVM service routine calls. This is

a dangerous approach, however, as it is with all low-level patches.

It is used to improve the performance behavior of a simulation run by replacing computing loops of an algorithm by time estimations. This obviously changes the results of the program, so it must not be done for programs which terminate in a data dependent way.

One of the few disadvantages of MISS-PVM is that cache effects are not taken into account. This may lead to seriously inaccurate predictions when choosing the first timing measurements method. Cache effects can, however, often be easily recognized in figures (with varying problem size), so that at least a judgement of the error should be possible by examination of the simulation results. The so achieved accuracy is sufficient for the development of AAAs.

Communication The communication model is more complex. Communication time dependencies on the message length are nonlinear, it also depends on the distance between the sending and the receiving processors (number of hops), on the contention of the network, the length of the physical connection (which turns out to become less important with the newest parallel computer generation), on the type of the processors (in a heterogeneous network), on the packing model (PVM supports several packing models), and on the type of interconnection network. Parallel computers with heterogeneous networks are available, like some IBM's RS/6000 Scalable Powerparallel Systems (IBM SP2). Normally PVM communicates by means of sockets; on some shared-memory machines, however, it makes use of the shared memory as well.

Model Parameters The timing estimation $t = \alpha + \beta * L_{\text{MessageLength}}$ may be very successful when used for a simple model. The two parameters α and β are easily found and in a first approximation the communication time will linear with the message length.

The *parameters* of the simulation are read from a file during the startup phase of the simulation (in the current implementation of MISS-PVM).

6 The Determination of Simulation Parameters

To estimate communication time the program `timing` which is already in the PVM package has been used in a modified form. Additionally further measurements were made to obtain values for execution time of PVM functions.

The program `timing` consists of a master (`timing`) and a slave (`timing_slave`). The master sends packets of different sizes to the slave which immediately who echos back. So the measured time contains sending a message of the reported size plus a return message of minimal size. The time needed for the packing and sending of each message is measured and reported. Averages are computed but not representative since they often over-estimate the communication time caused by delays at single intervals. These delays are due to interrupts caused by the operating system.

The figures 2 to 6 show the performance of PVM on different hardware: the SGI Power Challenge XL with sockets (fig. 2) and shared memory (fig. 3) and the communication time on the high performance switch of the IBM SP2 in its current version TB3 at three zoom levels: up to 400 000 bytes (fig. 4), up to 32 768 bytes (fig. 5), and up to 5000 bytes (fig. 6). Each of the zoom levels was chosen in order to show different effects.

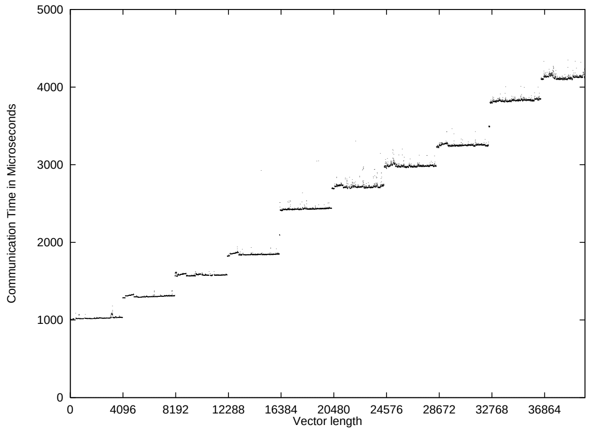

Figure 2: Communication Time on an SGI Power Challenge XL with Socket Communication (in a typical case)

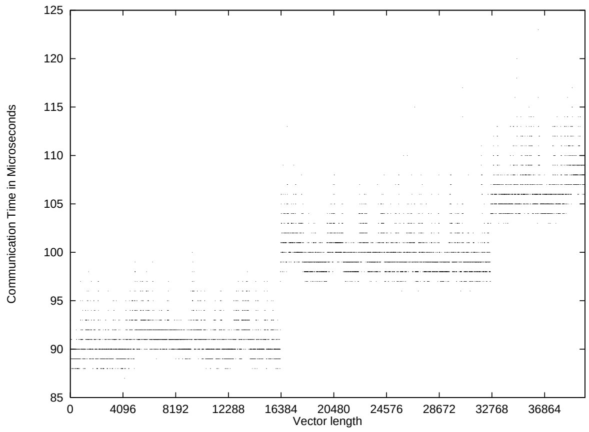

Figure 3: Communication Time on an SGI Power Challenge XL with Shared memory (all measurements are done several times)

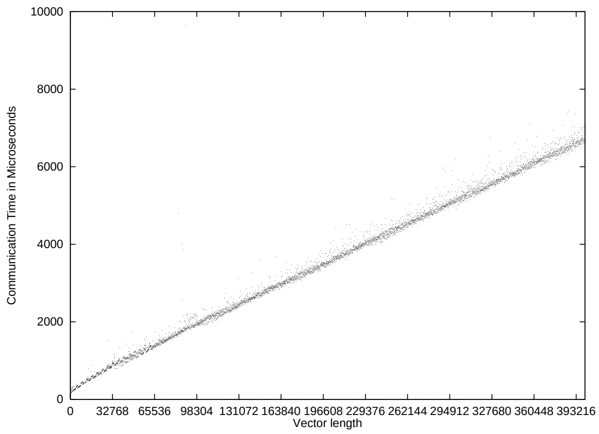

Figure 4: Communication Time on an IBM SP2 using the High Performance Switch TB3 (linear up to a message length of 400 000)

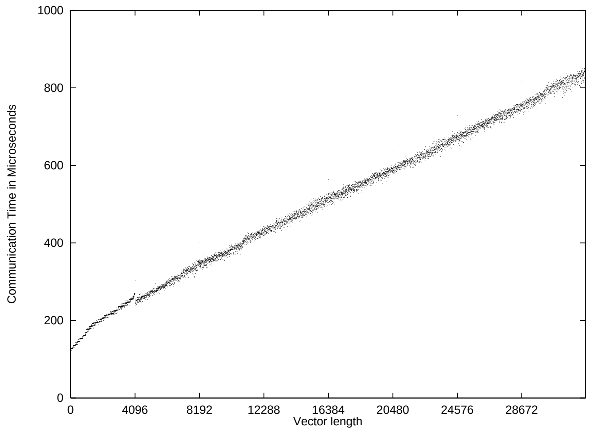

Figure 5: Communication Time on an IBM SP2 using the high performance switch TB3 (two effects are visible: (i) if the message length is 4096x+4 bytes then the communication time is 60 microseconds longer, and (ii) if the message length is under 4096 totally different characteristics apply)

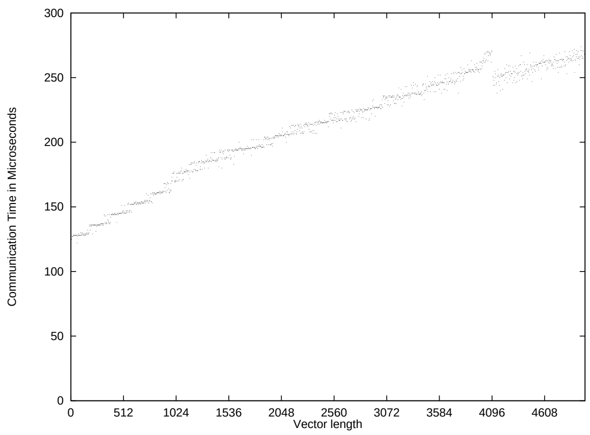

Figure 6: Communication Time on an IBM SP2 using the high performance switch TB3 (which is nonlinear in two ways: (i) the message time is piecewise linear in three parts: 0..1400, 1400..4096, 4096..; and (ii) there are smaller "pieces" with "steps")

The given time values include packing, sending and waiting for a reply. The models work with piecewise linear interpolation (table 1). The values in this table do not reflect the fastest communication possible on this computer, but the same type of communication which is actually used in available programs.

Message Length (Byte)	Time (Microseconds)
$x < 1400$	$62 + 3x/70$
$1400 < x < 4096$	$100 + x/40$
$4096 < x < 11000$	$180 + 21x/1100$
$11000 < x$	$210 + 9x/5500$

Table 1: Transmission Model for the SP2 High Performance Switch

7 Validation

One main difference between simulation and reality turned out to be that the simulation results were often more easily obtained than reliable measurements on available parallel computers. The reason for this effect is that exclusive access on parallel computers often is difficult to obtain. In this sense the simulation results present a 'best case' result which can not always be reached with actual parallel program runs.

Comparison between Reality and Simulation The timing measurements used a QR factorization with the Householder algorithm for matrix triangularization without manual performance tuning.

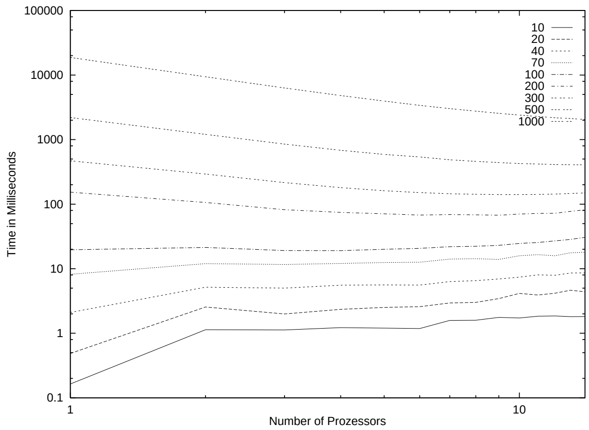

Figure 7: The Householder Algorithm on an IBM SP2

Figures 7 and 8 show the results of the simulation of the Householder Algorithm. For small matrices the time estimation mainly results from communication time; for big matrices computation time is more important. The results were satisfactory in both cases. Communication in the Householder Algorithm uses a lot of multicasting, which is prone to create contention on the network. Contention is not simulated, so the rise of the processing time at more than ten processors is not reported correctly (with problem sizes under 100×100). For all other cases the relative error is nearly always lower than 10%.

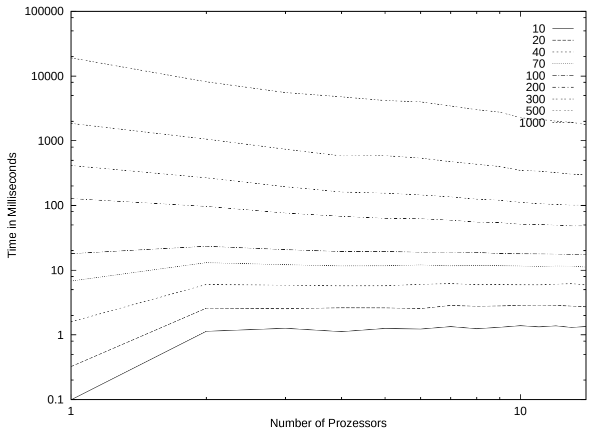

Figure 8: The Householder Algorithm on an IBM SP2 (Simulated on an SGI Power Challenge XL)

8 Conclusion

In this paper has been demonstrated that the simulation of a parallel computer can be established with satisfactory accuracy by use of a model based on simple performance measurements. The execution time information obtained by simulation runs is accurate enough by far to make quantitative evaluations of different parallel computers running numerical applications possible. The simulation engine MISS-PVM is thus an important tool for developing Architecture Adaptive Algorithms.

References

[1] C. A. Addison, V. S. Getov, A. J. G. Hey, R. W. Hockney, I. C. Wolton, *Benchmarking for Distributed Memory Parallel Systems: Gaining Insight From Numbers*, Parallel Comput. 20–10-11 (1994), pp. 1653–1668.

[2] L. A. Barroso, S. Iman, M. Dubois, K. Ramamurthy, *RPM: a Rapid Prototyping Engine for Multiprocessor Systems*, IEEE Computer 28–2 (1995), pp. 26–34.

[3] R. C. Bedichek, *Talisman: Fast and Accurate Multicomputer Simulation*, Performance Evaluation Rev. 23–1, special issue (1995), pp. 14–24.

[4] E. A. Brewer, W. E. Weihl, *Developing parallel applications using high-performance simulation*, SIGPLAN Notices 28–12 (1993), pp. 158–168.

[5] E. A. Brewer, C. N. Dellarocas, A. Colbrook, W. E. Weihl, *Proteus: A High-Performance Parallel-Architecture Simulator*, Technical Report MIT/LCS/TR-516, MIT, 1992.

[6] D. C. Burger, D. A. Wood, *Accuracy vs. Performance in Parallel Simulation of Interconnection Networks*, Proceedings 9th International Parallel Processing Symposium, IEEE Press, Los Alamitos, 1995, pp. 22–31.

[7] R. Candlin, J. Phillips, *A Statistical Study of the Factors that Affect the Performance of a Class of Parallel Programs on a MIMD Computer*, IFIP Trans. A Comput. Sci. Tech., Vol. 39, 1993, pp. 175–186.

[8] H. Davis, S. R. Goldschmidt, J. Hennessy, *Multiprocessor Simulation and Tracing Using* TANGO, International Conference on Parallel Processing, Vol. 2, (H. D. Schwetman, Ed.), Pennsylvania State University Press, Los Angeles, 1991, pp. 99–107.

[9] P. M. Dickens, P. Heidelberger, D. M. Nicol, *A Distributed Memory LAPSE: Parallel Simulation of Message-Passing Programs*, Proceedings 8th Workshop on Parallel and Distributed Simulation (PADS '94), (D. K. Arvind, R. Bagrodia, J. Y. B. Lin, Eds.), SCS, San Diego, CA, USA, 1994, pp. 32–38.

[10] P. M. Dickens, P. Heidelberger, D. M. Nicol, *Parallelized Network Simulators for Message-Passing Parallel Programs*, Proceedings of the Third International Workshop on Modeling, Analysis, and Simulation of Computer and Telecommunication Systems, (P. Dowd, E. Gelenbe, Eds.), IEEE Press, Los Alamitos, 1995, pp. 72–76.

[11] C. Figueira, E. Hernandez, *Benchmarks Specification and Generation for Performance Estimation on MIMD Machines*, IFIP Trans. A Comput. Sci. Tech., Vol. 44, 1994, pp. 215–223.

[12] S. R. Goldschmidt, J. L. Hennessy, *The Accuracy of Trace-Driven Simulations of Multiprocessors*, Performance Evaluation Rev. 21–1, special issue (1993), pp. 146–157.

[13] A. R. Krommer, C. W. Ueberhuber, *Architecture Adaptive Algorithms*, Parallel Comput. 19 (1993), pp. 409–435.

[14] J. Labarta, S. Girona, V. Pillet, T. Cortes, J. M. Cela, *A Parallel Program Development Environment*, Technical Report CEPBA/UPC RR-95/02, University of Technology Barcelona, 1995.

[15] O. Naim, A. J. G. Hey, *Invasiveness of Performance Instrumentation Measurements on Multiprocessors*, IFIP Trans. A Comput. Sci. Tech., (C. Girauld, Ed.), Vol. 44, Amsterdam, 1994, pp. 319–328.

[16] P. Naur, *Machine Dependent Programming in Common Languages*, BIT 7 (1967), pp. 123–131.

[17] G. D. Peterson, R. D. Chamberlain, *Beyond Execution Time: Expanding the Use of Performance Models*, IEEE Parallel & Distrib. Technology: Systems & Applications 2–2 (1994), pp. 37–49.

[18] S. K. Reinhardt, M. D. Hill, J. R. Larus, A. R. Lebeck, J. C. Lewis, D. A. Wood, *The Wisconsin Wind Tunnel: Virtual Prototyping of Parallel Computers*, Performance Evaluation Rev. 21–1, special issue (1993), pp. 48–60.

[19] A. Sivasubramaniam, A. Singla, U. Ramachandran, H. Venkateswar an, *A Simulation-Based Scalability Study of Parallel Systems*, J. Parallel Distrib. Comput. 22–3 (1994), pp. 411–426.

[20] H. Wabnig, G. Haring, *Performance Prediction of Parallel Systems with Scalable Specifications – Methodology and Case Study*, Performance Evaluation Rev. 22 (1994), pp. 288–289.

[21] D. W. Walker, *The Design of a Standard Message Passing Interface for Distributed Memory Concurrent Computers*, Parallel Comput. 20–4 (1994), pp. 657–673.