CLUE
A Tool for Cluster Evaluation

Helmut Hlavacs* Dieter F. Kvasnicka† Christoph W. Ueberhuber‡

Abstract

This paper describes the simulation tool CLUE which makes possible the highly accurate performance assessment and performance prediction of clusters of symmetric multiprocessors (SMPs). Using CLUE, reliable information can be obtained to reach the optimum decision on hardware configurations (processing elements and communication networks) before actually purchasing this hardware. Thus, hardware can be adapted to individual software features, reversing the currently used techniques of adapting high-performance software to hardware features (as used, for instance, in FFTW, PHIPAC, or the ATLAS tool).

Keywords: cluster computing, high-performance computing, simulation, parallel computing, execution driven simulation.

1 Introduction

In the last few years high-performance hardware has changed dramatically. Until quite recently, shared memory parallel vector processor (PVP) computers and distributed massively parallel processing (MPP) machines dominated the field of supercomputing. Nowadays high-performance computing has turned towards clusters of shared memory symmetric multiprocessors (SMPs). Most of these clusters are built using custom processors, as well as custom interconnection devices and memory. Accordingly, clusters of SMPs appear in a multitude of different configurations, ranging from teraflops machines to small clusters, comprising several PCs or workstations. The large number of possible configurations makes it difficult to decide which particular structure is suitable to solve a particular class

*Institute for Computer Science and Business Informatics, University of Vienna.
†Institute for Physical and Theoretical Chemistry, University of Technology, Vienna.
‡Institute for Applied and Numerical Mathematics, University of Technology, Vienna.
The work described in this paper was supported by the Special Research Program SFB F011 "AURORA" of the Austrian Science Fund.

of problems under certain performance requirements, and of course under given financial constraints. To aid this process of evaluating computer clusters the simulation and assessment tool CLUE (*clu*ster *e*valuator) has been developed. This paper describes CLUE and gives some results gathered in numerical experiments.

Synopsis. Section 1.1 describes current PC clusters, used in this paper as examples of SMPs. Sections 1.2 and 1.3 give a description of the specific clusters and algorithms, used in the experimental sections of this paper. Section 2 describes CLUE, the machine independent simulation system for clusters of SMPs. Section 3 shows the experimental scenarios and their results. Finally, Section 4 gives the conclusions of this work and provides an outlook on future work.

1.1 PC Clusters

As the performance of commodity computer and network hardware increases, and prices decrease, it becomes more and more attractive to build parallel computer systems from off-the-shelf components. The current price / performance ratio of a PC cluster is often ten times better than that of traditional supercomputers. PC clusters scale reasonably well, are easy to construct and only the hardware has to be paid for as most of the software is free. Meanwhile several entries of the top 500 supercomputer list[1] are clusters of off-the-shelf SMPs.[2]

Usually PC clusters do not contain any proprietary hardware components and thus are trivially reproducible. Commodity software like the Linux operating system, PVM (Geist et al. [6]) and MPI (Gropp et al. [7]) are used for developing parallel programs.

A server node controls the whole cluster and serves files to the client nodes. These nodes can be thought of as a (multi) CPU plus a memory package which can be plugged into the cluster, just like a CPU or memory module can be plugged into a motherboard. PC clusters used for high-performance computing are often referred to as *Beowulf clusters*.[3] Typical configurations are, for example, a compound of several computers connected via Fast Ethernet or faster networks like Gigabit Ethernet, Myrinet or SCI (Scalable Coherent Interface).

As a form of parallel computers PC clusters fall somewhere in between MPPs (Massively Parallel Processors) and NOWs (Networks of Workstations). They profit from developments in both architectures. MPPs are usually larger than PC clusters and use faster networks. Typical problems when programming MPPs are load and data distribution, granularity, and minimizing the communication overhead. Programs, which are not too fine grained, can be ported from MPPs

[1] `http://www.netlib.org/benchmark/top500.html`

[2] `http://www.cs.sandia.gov/cplant/`
`http://cnls.lanl.gov/avalon/`
`http://wwwwissrech.iam.uni-bonn.de/research/projects/parnass2/`

[3] `http://www.beowulf.org`

to PC clusters easily and achieve there a satisfactory performance. When programming NOWs, in most cases algorithms are designed to utilize unused cycles of powerful workstations. These algorithms have to be tolerant of load variations and have to use dynamic load balancing. All programs running on NOWs run at least as good on dedicated PC clusters.

Often PC clusters are used as a (relatively cheap) platform for large production codes which were running for many years on Unix workstations. It would be difficult to adapt these codes to the requirements of the hardware, thus it is important to configure the cluster according to the requirements of the code. This was one of the reasons for the development of CLUE.

1.2 Hardware

In this paper two specific PC clusters are investigated.

The Vienna Cluster was built and operated by the authors. It consists of five dual 350 MHz Pentium II systems ($p = 10$) with 256 MB main memory, 512 KB Level 2 cache and local 4.5 GB hard discs. The nodes are connected via a switched Fast Ethernet network (measured bandwidth: 12.5 MB/s).

The Aachen Cluster, a Siemens hpcLine PC cluster, consists of 16 dual processor boards equipped with 400 MHz Pentium II processors ($p = 32$), 512 KB Level 2 cache, 512 MB main memory and local 4 GB hard discs. The nodes communicate either via switched Fast Ethernet or SCI (Scalable Coherent Interface; measured bandwidth: 80 MB/s). The SCI network is configured as a two-dimensional torus.

For more details see Hlavacs et al. [8].

1.3 Algorithms

Algorithms for distributed memory parallel computers see three scenarios:

Computation bound algorithms achieve near-linear speedup (provided the work load is distributed evenly). Simulation of such algorithms does not depend significantly on the accuracy of the network simulation, since communication time is negligible. In this case the accurate simulation of computing time is important for reliable simulation results.

Communication bound algorithms do not speed up at all, since additional processors increase the amount of data to be communicated. Simulation of such algorithms does not require an accurate processor simulation, since computation time is insignificant. Accurate network simulation is crucial in this case, including (in many cases) contention analysis and simulation.

Transition algorithms which are computation bound when running on a few processors and communication bound when running on many processors.

To find the transition point between the computation bound mode and the communication bound mode reliably, both processors and network have to be modeled accurately.

Considering these three modes, ScaLapack routines were chosen to demonstrate the usefulness and the reliability of CLUE. With respect to an increasing ratio of floating-point operations to data movement the following algorithms have been selected.

Cholesky Factorization. `SCALAPACK/pdpotrf` is a routine which computes the Cholesky factorization $U^T U$ of a symmetric, positive definite matrix A. As the computation proceeds, elements of A are overwritten with elements of U. A and U are stored in a block-cyclic two-dimensional way using identical block sizes in both dimensions.

LU Factorization. `SCALAPACK/pdgetrf` computes the LU factorization PLU of a general matrix A. The output matrices U and L are stored in place of the input matrix A which is distributed in a two-dimensional block-cyclic manner using identical block sizes in both dimensions.

Matrix Multiplication. The routine `PBLAS/pdgemm` multiplies two matrices: $C = AB$. All three $n \times n$ matrices are distributed in a two-dimensional block-cyclic way. Identical block sizes are used in both dimensions and for all matrices.

2 The Simulation Tool CLUE

CLUE is based on MISS-PVM, the *Machine Independent Simulation System for PVM 3* (Kvasnicka, Ueberhuber [9]). CLUE is meant to support the development of software for parallel computers which are not yet available and to carry out reproducible performance assessments in environments with constantly changing load characteristics (like NOWs). CLUE also makes the debugging of parallel programs easier. Using CLUE, reliable information can be obtained to reach the optimum decision on hardware configurations (processing elements and communication networks) before actually purchasing this hardware. Thus, hardware can be adapted to individual software features, reversing the currently used techniques of adapting high-performance software to hardware features (as used, for instance, in FFTW [4, 5], Phipac [1], or the Atlas tool [11]). To exploit these features of CLUE, it is not necessary to rewrite existing code or create additional one (neither in C nor in Fortran). PVM based code can be used without modification.

CLUE's favorable properties are achieved by establishing a virtual layer which does not have to be tampered with when developing software. The *Virtual Layer for PVM 3* is situated between the user program and PVM 3 (see Fig. 1). All calls to PVM 3 are redirected to CLUE subroutines performing virtual timing

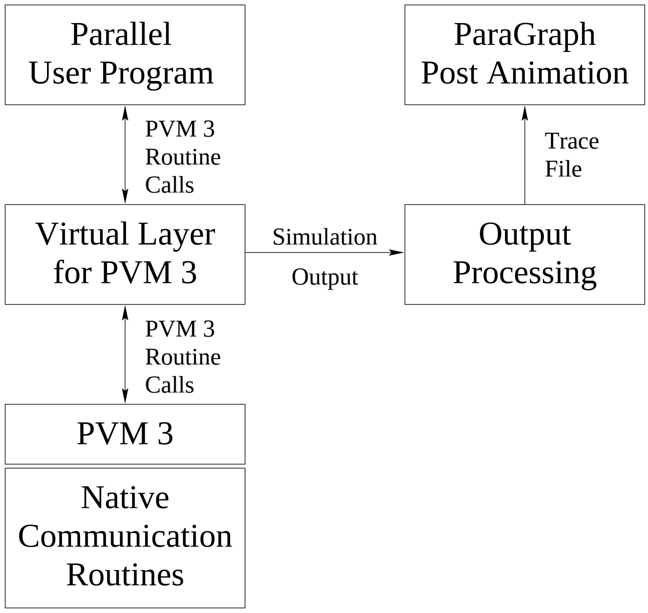

Figure 1: Position of the Virtual Layer for PVM.

and virtual machine adaptation. They eventually pass on the calls (in a modified form) to PVM 3.

Use of the virtual layer is possible without the necessity to modify the user program. It is sufficient to use different include files and to link the user program to additional libraries. The virtual layer generates output files which trace calls to communication subroutines. These trace files are the input of post-mortem visualization.

This tracing technique has two major advantages over conventional trace file writing. The *Virtual Layer for PVM 3* (i) uses its own simulated *system time*, and (ii) makes a *virtual machine* available to the user.

The virtual machine may represent a wide variety of real machines which may even be non-existent or not available at the moment. Machine parameters are read from a file when CLUE is started. These parameters may be changed dynamically during simulation.

The virtual layer makes it possible to compare program runs on computers having different communication latency and computation speed (independent of actual load characteristics). For instance, reproducible experiments for the assessment of load balancing strategies on irregularly loaded NOWs can be made easily and quickly.

High level communication libraries like the BLACS (Dongarra et al. [2, 3]) can be simulated very comfortably. Once the high level library routines (based on PVM) have been recompiled, no further modifications are needed. The user program is linked with the new library. The result is a program that performs the same task as before, except that it writes an output file (if required) and can be simulated on virtual machines.

For small programs usually the whole simulation runs on one processor. For large programs (especially those performing compute intensive tasks) the overall execution time is often a prohibitive factor in simulation. Therefore, in cases which require much time or main memory, the simulation can be distributed to several processors.

CLUE was designed to have a very low simulation overhead by using execution driven simulation and compiled communication models, which are parametrized at the beginning of each simulation run.

3 Numerical Experiments

This chapter contains two sections: the first one briefly describes the simulation scenario and shows how to get performance data with respect to existing clusters, the second one validates these results in a comparative study of simulated and measured program runs.

3.1 Measurement, Modeling, and Simulation

Up to now, two Beowulf clusters have been examined (see Section 1.2). On both clusters, measurements have been carried out to obtain the following communication related parameters. (i) *Send time*: The time the sender spends in the send call. (ii) *Transmission time*: The difference between the send time and the time the message needs to be received by the receiver.

Fig. 2 shows the send and transmission times observed on the Vienna cluster. Both sender and receiver run on the same node, thus the message is not sent over the network. The piecewise linear model used for the simulation in shown as well.

Additionally, for the case of sending messages from one sender to several receivers at the same time, contention has been observed that increases both the send and transmission time. This is important for those cases, where all tasks synchronize, and one of the tasks then sends out messages to one or several others to redistribute work or results.

Each cluster was simulated by using two communication modes (intra- and inter-node) and by using a contention model. This method achieves sufficiently accurate results even on the torus-shaped network of the Aachen cluster.

Measured computation time is modified by a computational factor. For example, each processor of the Aachen cluster was measured to be 10 % faster than one of the processors of the Vienna cluster.

The SCI network of the Aachen cluster was only available for the MPI version of the Blacs. Thus, communication parameters and performance data of the real

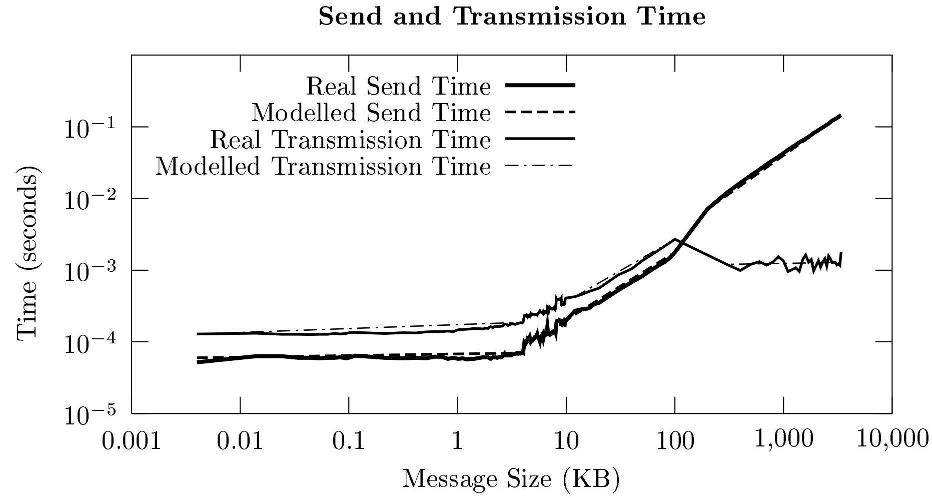

Figure 2: Send and transmission time for the Vienna cluster. Sender and receiver are on the same node.

runs were measured on the Aachen cluster using the MPI version of the Blacs, whereas the simulation runs were carried out on the Vienna cluster using the PVM version of the Blacs.

3.2 Results

In order to obtain widely applicable results, three important algorithms from the field of numerical linear algebra (see Section 1.3) have been chosen to be simulated on the two clusters. 2000×2000 matrices were chosen to set up problems of a sizeable computational complexity. Moreover, these problems are imposing substantial main memory requirements on the simulating node.

The simulation runs were carried out to answer the following questions. (i) Do the real observations and the simulated runs have the same qualitative properties? (ii) Do the real observations and the simulated runs have the same quantitative properties? (iii) Can the simulation results be used to evaluate the performance of workstations clusters a priori?

In Figs. 3 to 8, the observed and simulated wall times are plotted against the processor grid used. Such a grid or 2-dimensional mesh is always assumed to define the topology of the parallel computer, even if in reality this is a cluster of SMPs connected over a bus, star or ring topology. Each processor is assigned a place in the virtual mesh topology. Basically, an $N \times M$ grid means that $N \times M$ processors were used to compute the task. The relation of N to M defines the communication pattern used, which has a strong influence on efficiency as the diagrams show.

The diagrams reveal the answers to the above questions. (i) The real observations and the simulated runs have the same qualitative properties. (ii) The real

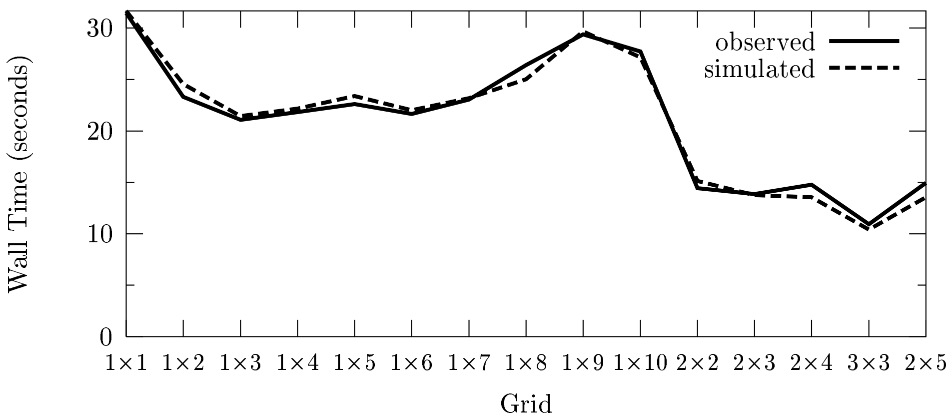

Figure 3: Cholesky factorization of a 2000×2000 matrix on the Vienna cluster. Blocksize 100.

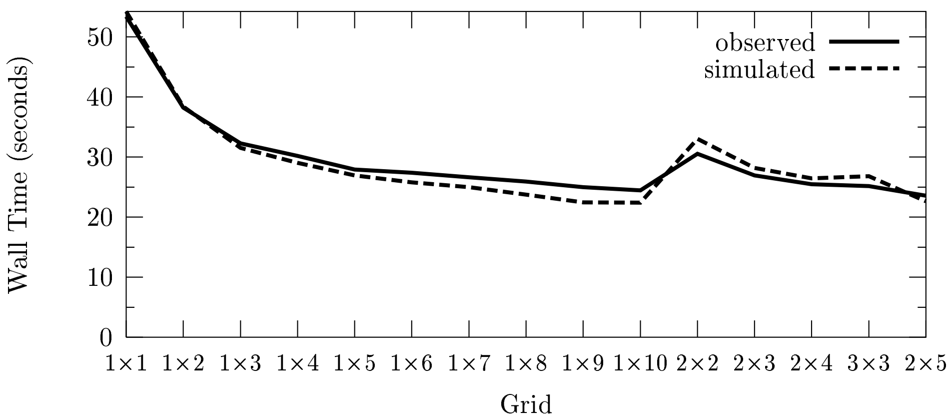

Figure 4: LU factorization of a 2000×2000 matrix on the Vienna cluster. Blocksize 200.

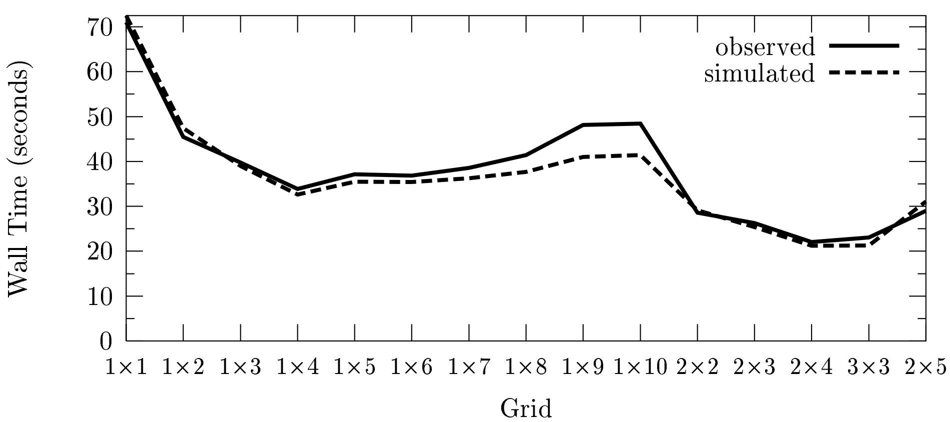

Figure 5: Multiplication of a 2000×2000 matrix on the Vienna cluster. Blocksize 500.

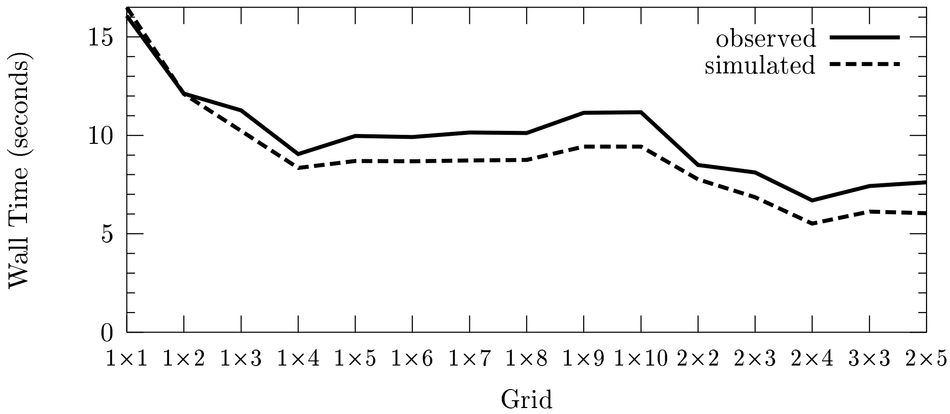

Figure 6: Cholesky factorization of a 2000×2000 matrix on the Aachen cluster. Blocksize 400.

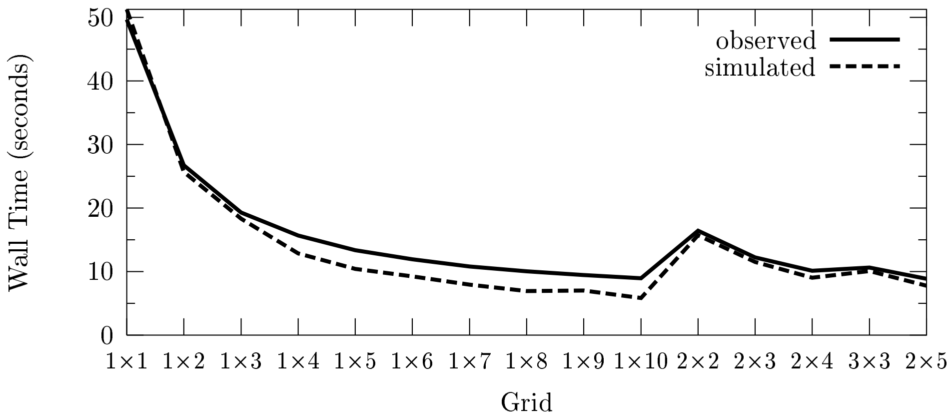

Figure 7: LU factorization of a 2000×2000 matrix on the Aachen cluster. Blocksize 50.

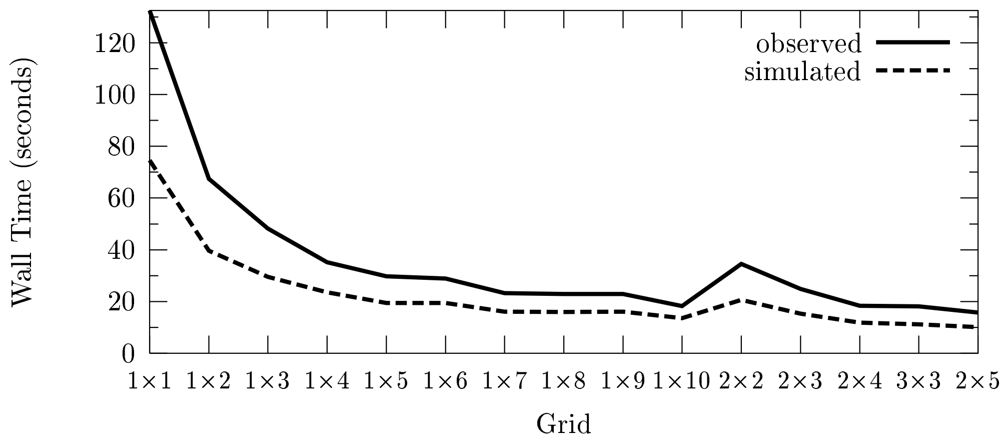

Figure 8: Multiplication of a 2000×2000 matrix on the Aachen cluster. Blocksize 100.

observations and the simulated runs have the same quantitative properties, with a few exceptions. Compared with the real run (Fig. 7), the simulation predicts a slightly stronger decrease in execution time with increasing number of processors, and the performance factor chosen for the Aachen cluster was not appropriate in some cases (cf. Fig. 8). This deviation must be due to some internal differences of the processors. (iii) The simulation results can be used to evaluate the performance of workstations clusters a priori, since the important variatons in execution time are at the correct places.

4 Summary and Conclusions

In this paper the new simulation tool CLUE is described. This tool is meant to investigate the performance behavior of clusters of SMPs.

Simulation results obtained for a PC cluster with slow communication (using Fast Ethernet) turned out to be very accurate. Both qualitative and quantitative performance behavior can be simulated highly satisfactory.

Simulating the performance of PVM versions of ScaLAPACK and the PBLAS does not require any modification of the PVM code. Simulating MPI versions by using PVM versions on a different type of node is more demanding. Still, performance diagrams indicate that the qualitative behavior of the parallel programs is described accurately whereas quantitative results are sometimes a little misleading.

It may be concluded, that performance comparisons between different clusters of SMPs are possible, though experiments must be carefully designed and results have to be interpreted cautiously. The qualitative behavior of parallel programs running on clusters of SMPs can be simulated accurately.

Using CLUE it is possible to analyze the behavior of parallel programs and predict their performance, depending on cluster parameters. Simulation results can be used to investigate the influence of different parameters of the simulated workstation or PC cluster, in order to plan new hardware configurations or make an educated choice between several alternatives.

Outlook. Future work will include the development of improved network models to simulate various topologies. Also, classical network simulators like OPNET or NS will be used to develop new ways of estimating simulation parameters for workstation or PC cluster configurations that are not (yet) available.

Furthermore, the performance of several other clusters will be investigated in order to improve and stabilize the simulation methodology.

Acknowledgements

We wish to express our gratitude to the computing center of the RWTH Aachen for the access to the hpcLine cluster and to the Austrian Science Fund (FWF) for financial support.

References

[1] J. Bilmes, K. Asanovic, C.-W. Chin, J. Demmel, *Optimizing Matrix Multiply using* PHIPAC*: a Portable, High-Performance, ANSI C Coding Methodology*, Proceedings of the 1997 International Conference on Supercomputing in Vienna, Austria, ACM Press, New York, 1997, pp. 340–347.

[2] J. J. Dongarra, R. van de Geijn, *Two-dimensional Basic Linear Algebra Communication Subprograms*, LAPACK Working Note 37, 1991.

[3] J. J. Dongarra, R. C. Whaley, *A User's Guide to the* BLACS *Version 1.1*, LAPACK Working Note 94, 1995.

[4] M. Frigo, *A Fast Fourier Transform Compiler*, Proceedings of the ACM SIGPLAN 1999 Conference on Programming Language Design and Implementation in Atlanta, Georgia, ACM Press, New York, 1999, pp. 169–180.

[5] M. Frigo, S. G. Johnson, *The Fastest Fourier Transform in the West*, Technical Report MIT-LCS-TR-728, MIT Laboratory for Computer Science, 1997.

[6] A. Geist, A. Beguelin, J. J. Dongarra, W. Jiang, R. Manchek, V. Sunderam, *PVM: Parallel Virtual Machine—A Users' Guide and Tutorial for Networked Parallel Computing*, MIT Press, Cambridge London, 1994.

[7] W. Gropp, E. Lusk, A. Skjelum, *Using MPI, 2nd ed.*, MIT Press, Cambridge London, 1999.

[8] H. Hlavacs, D. F. Kvasnicka, C. W. Ueberhuber, *CLUE—Cluster Evaluation*, Technical Report AURORA TR 2000-05, Vienna University of Technology, 2000, `www.vcpc.univie.ac.at/aurora/publications/`.

[9] D. F. Kvasnicka, C. W. Ueberhuber, *Developing Architecture Adaptive Algorithms using Simulation with MISS-PVM for Performance Prediction*, Proceedings of the 1997 International Conference on Supercomputing in Vienna, Austria, ACM Press, New York, 1997, pp. 333–339.

[10] C. W. Ueberhuber, *Numerical Computation*, Springer-Verlag, Berlin Heidelberg New York Tokyo, 1997.

[11] R. C. Whaley, J. J. Dongarra, *Automatically Tuned Linear Algebra Software*, LAPACK Working Note 131, 1997.