Cluster Configuration by Simulation

October 13, 2000

Abstract

In many cases the acquisition of a PC cluster is limited by financial restrictions. Given a fixed amount of money, the newly developed simulation tool CLUE can be used to decide which configuration of the components of the cluster gives the best price/performance ratio. Due to the simulation based approach, even the impact of components can be evaluated that are projected to be available in the future.

As a case study, it will be demonstrated how to use CLUE for finding the cluster configuration yielding the optimum price/performance ratio for WIEN 97, a package for computational chemistry.

Keywords: Cluster Computing, Simulation, High-Performance Computing, Parallel Computing, Execution Driven Simulation.

Introduction

Many customers of high performance computers intend to run their parallel applications as fast as possible under given restrictions. These restrictions include, for example, a limited budget or high hardware availability. Usually, there is a confusing multitude of hardware alternatives meeting these restrictions, which customers must choose from.

One popular way of exploiting high performance computers is to use them as centralized resource within computing centers, being accessible by a large number of users. These computers are most often classical multiprocessors/multicomputers. Customers using centralized computers usually pay only for the cycles they actually consume.

An alternative option is to buy computers exclusively dedicated to the customers applications. In this case customers have to pay for the hardware and software maintenance, even if the computer systems are not used permanently. Dedicated PC clusters consisting of standard off-the-shelf components have gained high popularity over the last years, as these configurations yield a favorable price/performance ratio.

Customers of PC clusters have to choose between various hardware configurations. However, it is difficult to estimate the impact of these decisions on the performance of their parallel software. The newly developed *CLUster Evaluator* CLUE described in this paper is a tool for evaluating the performance of various configurations of PC clusters. It's usability is demonstrated by a case study for finding an optimum PC cluster configuration for the WIEN 97 ([3]) package, an application code from computational chemistry. WIEN 97 is available in two different parallel implementations, one requiring a large amount of memory for each processor node, the other one relying on a fast communication network. Both of these requirements nearly double the price of each node.

The limit imposed by budget restrictions renders many cluster configurations impossible. Still, a large number of feasibly configurations remain, and it is impossible to find the optimum configuration without applying appropriate performance estimation techniques, like

- analyzing published benchmark results;
- obtaining accounts on existing PC clusters and running own benchmarks (synthetic or real applications);
- using analytical models to evaluate the theoretical performance of different configurations; or
- creating a model of the feasible configurations and using a simulator to gain performance information.

Utilizing published benchmark results is the quickest and most straight forward way. If results from standard benchmarks are available, the probability of faulty results is low. However, standard benchmarks do not provide useful information for all sequential applications. This is even more true for parallel applications, especially if these applications have irregular communication patterns and/or different instruction mixes. For example, the popular LINPACK benchmark shows a highly regular communication pattern. Since benchmark results are available only for a subset of possible configurations, several preconditions have to be satisfied: the computer configurations must exist, standard benchmarks must have been executed on them, and the results must have been published. Existing benchmark results might reflect older hardware, because the gap between existing clusters and hardware that is available at the moment of decision making is about one processor generation.

Running your own benchmarks is an approach to obtain very accurate performance metrics if the PC cluster can be accessed exclusively. However, these benchmarks or application runs are usually limited to a rather small number of existing PC cluster configurations. Moreover, it is not always easy to obtain accounts, especially on production systems where it might be impossible to get an exclusive access to the system even for a short time. Also, as in the standard benchmark approach, components of existing clusters are usually older than the ones to be purchased.

Using analytical models is one of the fastest methods to obtain results, provided the analytical models already exist. There is no limit for configuration parameters that may be modelled and there is no measurement error. However, accurate analytical models are extremely complicated to create, usually, only few components of the real hardware configurations and application benchmarks can be included. These restrictions often necessitate oversimplifying assumptions, which may decrease the model's accuracy in an intolerable way.

Simulation makes it possible to run experiments for many different configurations, including existing and non-existing (future) hardware. In this approach, the cluster under consideration is represented by a mathematical model (a "virtual cluster") that drives the simulation kernel. Basically, models of arbitrary complexity can be created, limited only by the modelling effort spent. Simulation models are potentially more accurate than analytical models. For example, when simulating a queuing network instead of solving it analytically, any distribution for the (stochastic) interarrival and service times can be used. Also, it is possible to obtain insight why the used benchmark achieves better performance on certain configurations. Furthermore, the virtual cluster is owned completely by the user, so there is no interference with a production system or other users. However, there are drawbacks inherent to this technique. The creation of models for hardware configurations and applications may be difficult and time-consuming. The complexity (and thus accuracy) of the hardware and software models will be limited by the simulation environment and the simulation run times. Also, the simulation models must be validated with care. It is usually not possible to include "back ground utilization" of clusters, i. e., load created by competing users.

In this paper CLUE is presented, a cluster evaluation tool possessing all the mentioned advantages of the simulation approach and eliminating as many of the disadvantages as possible. The hardware model parameters are easily obtained either by real measurements or by extrapolation of known parameters. The software models are most easily constructed by taking the original source code as input to the simulator. Furthermore, CLUE allows taking into account back ground utilization created by competing cluster users.

1 Related Work

In the past, several attempts have been made to simulate the performance of parallel programs. N-MAP [7], for example, allows to predict the performance of parallel programs by specifying code fragments only. The PVM Simulator PS [1] follows an approach similar to ours, accepting full PVM programs or program prototypes. PS, however, does not use execution driven simulation but produces trace files describing the communication pattern of the application under observation. These trace files then drive the simulation kernel to derive simulation results. SPEEDY [12] is part of the performance extrapolation tool TAU, that also collects run-time trace information of parallel programs written in pC++, which is later used to drive the trace-driven simulation engine. The same trace-driven approach is used in DIP [11].

In general trace-driven approaches assume that the communication patterns are fixed and do not depend on the run-time situation. This assumption is valid, for example, for routines from the SCALAPACK library. In cases where the communication depends on the run-time situation, however, for example when simulating the effect of load balancing mechanisms, in contrast to execution driven simulation, this approach cannot be used.

More sophisticated execution driven simulators include, for example, the simulation platform SIMOS [13]. The simulation kernels of this platform allow the simulation of various processor models and hardware features with varying degree of complexity. Simulators for different processor models include EMBRA, MIPSY, and MXS. Users of SIMOS simply start the binary executable of their code which then drives the simulator. This approach, however, relies on the availability of a processor model for the processor family the executable was compiled for. Other platforms cannot be simulated. The processor families being modelled so far include the MIPS R10000 and the Compaq Alpha family.

Other parallel programming tools like AIMS [15] observe real program executions and provide sophisticated tools for post-mortem trace file analysis. In this approach, only the performance of parallel programs which are executed on available platforms can be evaluated.

Yet another approach is taken in the EDPEPPS [4] tool. Here, users may construct application models for PVM programs by using the graphical program representation language PVMGraph. This representation then drives the simulation kernel. This approach is meant for rapid prototyping but cannot be used for the simulation of sophisticated load balancing techniques.

2 The Simulation Tool CLUE

CLUE is based on MISS-PVM [17], the *Machine Independent Simulation System for PVM 3*. CLUE is meant to support (i) configuration decisions concerning clusters of SMPs, (ii) the development of software for parallel computers which are not yet available, (iii) reproducible performance assessments in environments with constantly changing load characteristics (like NOWs), and (iv) the debugging of parallel programs.

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, the acquisition of 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 [8, 9], PHIPAC [2], or the ATLAS tool [14]). To exploit these features of CLUE, it is not necessary to rewrite existing C or Fortran code or to create additional code. 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). Calls to PVM 3 are redirected to CLUE subroutines performing virtual timing and virtual machine adaptation. Afterwards, the calls are passed to PVM 3 in a modified manner.

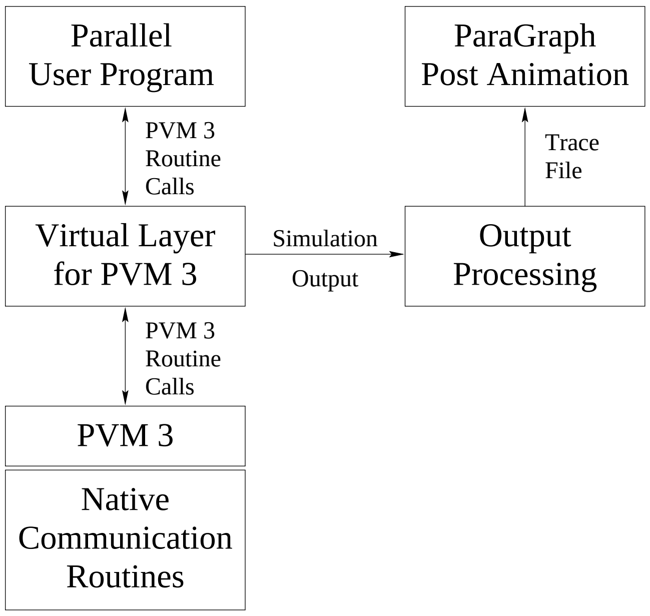

Figure 1: Position of CLUE's Virtual Layer.

Using the virtual layer is possible without the necessity to modify the user program. Instead, it is sufficient to use different include files and to link the user program to to the CLUE library. The virtual layer generates output files tracing all calls to communication subroutines, which can be used for 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. The parameters defining the machine characteristics are read from a configuration file at simulation start and may be changed dynamically during the simulation run.

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. [5, 6])

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.

The simulation of MPI programs using MISS-PVM requires some caution. For example, the communication of ScaLapack routines is completely hidden in the communication library Blacs, which can be based on either MPI, PVM, or other message passing libraries. There are major differences in the program start and communication setup, but during the algorithm the communication patterns are essentially the same. It has been demonstrated that the simulation of MPI programs using MISS-PVM is sufficiently accurate (see XXX [16]).

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. Simulation with Clue is only accurate if enough memory is available for all parallel program instances. If swapping occurs, inaccuracies amounting to more than 10 % may occur.

The simulation performed in this paper is only valid for an exclusively accessed PC cluster. It is also assumed that there is only one process per processor. Clue was designed to produce only a very low simulation overhead by using execution driven simulation and compiled communication models, which are parameterized at the beginning of each simulation run.

3 Hardware Configuration Parameters

Virtual machines are defined by an input file. This file contains the specification of a machine used for the master program and for additional machines or host types. For each record possible parameters are:

Name of the machine or host type. This name is used in `pvm_spawn`. If the machine *performance factor* (described next) is 0, the machine is assumed to be *real*, and the program is started on this machine. Otherwise the machine has a *virtual* name, and PVM 3 is asked to look for a suitable machine.

Performance factor. This is a floating-point multiplier p for calculating the computation time. If this parameter is 0, the computation timing results are not changed. Otherwise, if a process of a parallel application consumes n CPU seconds between two adjacent calls to the virtual layer, the virtual time is increased by $p \times n$ seconds.

Initialization Delay. This is the time needed for `pvm_spawn` to be called. It is measured in multiples of 100 microseconds.

Spawn Delay. This is the time spent in `pvm_spawn`. It is measured in multiples of 100 microseconds.

Send Delay. This is the time used for sending a message using `pvm_send` or `pvm_mcast`. This time contains packing the message, resolving the address of the host and starting the transmission (as far as the sending process is involved). This time is measured in 100 microseconds per KB. The parameters may be specified as k, d, where the send delay $s(m)$ depending on the message length m is given by $s = k \times m + d$, or may be specified as tupels $(m_x, s(m_x))$, where the actual send delay is interpolated linearly between these points.

Receive Delay. This is the time used in calling the receive routines `pvm_recv`, `pvm_nrecv` and `pvm_probe`. This time is always the same whether these routines succeed or fail. It is measured in multiples of 100 microseconds.

Transmission Delay. This is the time used to transfer a message minor the send delay. As with the send delay, the transmission delay may be specified as linear model or linearly interpolated data points. This time is measured in 100 microseconds per KB.

Packing Delay. This is the time used to pack the message into the PVM 3 send buffer. This time is measured in 100 microseconds per KB.

Send and transmission delay may be specified for any pair of hosts, they may also be specified for the send and transmission of messages from one host to itself, in case multiprocessor machines are to be modelled. Fig. 2 shows the assumed model for the send and transmission delay. If the actual performance model turns out not to be exact enough, it can easily be changed by modifying the configuration file.

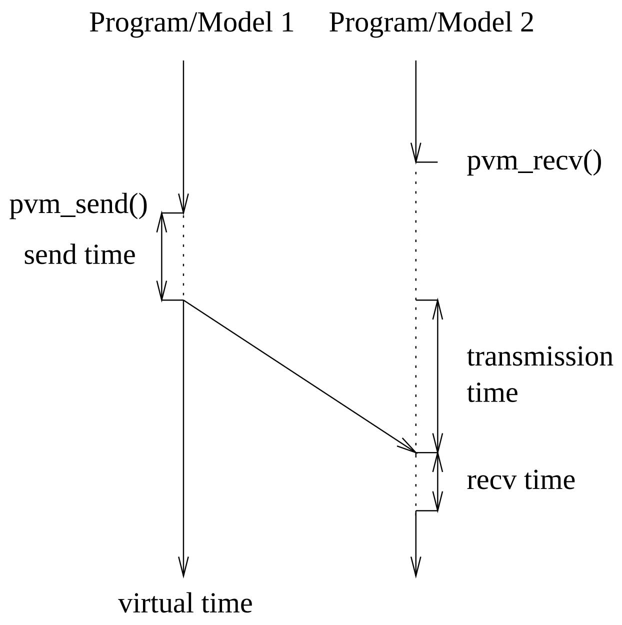

Figure 2: CLUE communication model.

4 Case Study: Finding an Optimum Cluster Configuration for WIEN 97

In this case study we demonstrate the applicability of CLUE for finding an optimum hardware configuration for a particular parallel application, in this case by taking a well known package from computational chemistry, WIEN 97[1], as an example. We assume that an institute is planning to purchase a PC cluster to run the computationally expensive WIEN 97 simulations. Furthermore we assume that there is a tight limit on the budget that the institute can spend.

WIEN 97 uses the linearized augmented plane wave (LAPW) method for solving the Kohn-Sham equations for the ground state electron density ρ, the total energy $E[\rho]$, and the (Kohn-Sham) eigenvalues ϵ_i. This program package is written in Fortran 77 and requires a UNIX operating system since several programs are linked together via C-shell scripts. WIEN 97 has been

[1] WIEN 97 is used by over 400 academic and industrial user groups worldwide; see `http://www.tuwien.ac.at/theochem/wien97/`

implemented successfully on many mainframes and workstations of the major computer companies as well as on PCs running Linux.

The execution time of one particular WIEN 97 run is presented in Fig. 3 which shows that `lapw1` is the most time consuming program of the WIEN 97 package (see also Haunschmid, Kvasnicka [10]). `lapw1` is used in a self-consistency cycle, which is repeated until convergence criteria are met.

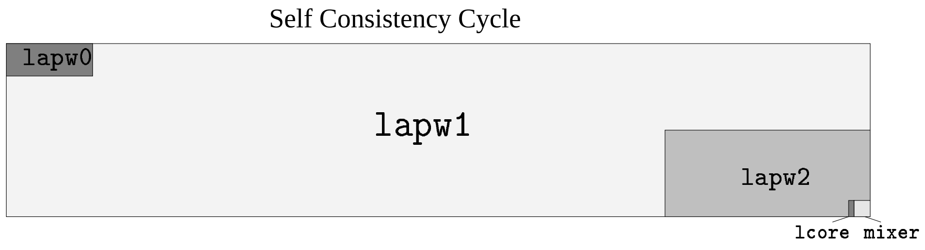

Figure 3: Execution time subdivision of the self-consistency cycle of WIEN 97 (material: rutile, TiO_2). The area of each box is proportional to the (sequential) execution time of the respective routine.

Since `lapw1` requires most of the sequential execution time, it is also most critical in a parallel execution on just a few processors. Furthermore, with increasing problem size, it remains the critical program also with an increasing number of processors.

The most important subroutines of `lapw1` are (i) setting up the Hamiltonian H and the overlap matrix S, and (ii) solving a generalized symmetric eigenproblem for about 10 % of the eigenpairs (eigenvalues ϵ_i and eigenvectors c_i):

$$Hc_i = \epsilon_i S c_i \ . \tag{1}$$

Typical problem sizes range from $n = 1000$ to $n = 10\,000$.

The standard method for solving the eigenproblem (1) for dense matrices, as implemented, for example, in SCALAPACK, is to reduce the generalized eigenproblem to a standard eigenproblem by using Cholesky factorization and furthermore to reduce this standard problem to a tridiagonal eigenproblem, where it can be solved easily. These two steps consume the majority of the computing time spent in `lapw1` . Further analysis of the used linear algebra calls also reveals that for parts of the code it is necessary to run the simulation on a computer with a similar cache and memory hierarchy as the one that is modelled.

4.1 Software Models

Two application cases have been chosen:

- A small case with a matrix size of 2500×2500 which can be solved on one processor on a standard PC.
- A larger case with a matrix size of 6000×6000. In this case it is necessary to either have more memory for each processor, or to distribute each matrix to several processors.

Each case is responsible for 50 % of the overall workload of the PC cluster.

Two kernels, which are representative for the majority of the execution time, have been extracted:

- Cholesky factorization, and
- Tridiagonalization.

Both of these kernels represent 50 % of the overall execution time.

Library routines such as the BLAS, LAPACK, SCALAPACK and BLACS are utilized as much as possible. The simulation is performed with CLUE which is based on PVM.

4.2 Communication Models

CLUE can apply various communication models. In this paper piecewise linear models are used (see Fig. 4). Here, the time for sending a message of particular size is represented by two models: (i) the *send time*, which is the time the sending process is blocked, and (ii) the *transmission time*, which is the additional time until the message is received. Latency and network throughput are modelled automatically. The model parameters for the send and transmission times have been measured on existing networks.

Additionally, a simple contention model is used which increases the communication time (both send and transmission time) by a factor depending on the number of simultaneous communication operations carried out.

4.3 Hardware Configurations

To choose the configuration of PC clusters with highest performance for WIEN 97, several configurations are examined (see Table 1). It is assumed

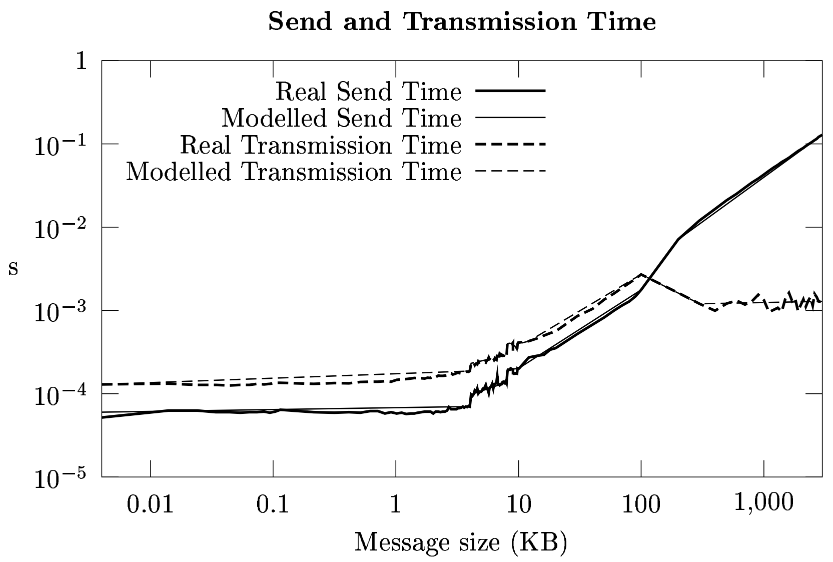

Figure 4: Send and transmission time for a Fast Ethernet PC cluster. Sender and receiver are on the same node. Although they share the same memory, they communicate via a message passing library.

that the budget limit is set to ATS 300.000,–.[2] This price reflects only pure hardware investments. Costs for the cluster installation and configuration as well as for software are not included and would be similar for all configurations.

Table 1 shows the configurations affordable according to the budget limit. In this table, information on the clock rate and the manufacturer were omitted, due to the fact that updates on clock rates occur too often and customers will always aim at buying the fastest available version of a processor at the time of purchase. Thus, this decision is delayed to the final stage of the evaluation.

4.4 Preliminary Experiments

Preliminary experiments simulating the Level 3 based Cholesky factorization have been performed with (i) a model for Fast Ethernet communication and

[2]approximately Euro 22.000,– or $ 20.000,–

Name	Number of Nodes	Memory per Node	Processors per Node	Interconnection Network
Network	6	256 MB	2	Gigabit Class
Memory	6	1024 MB	2	Fast Ethernet
Fine	7	128 MB	1	Gigabit Class
Coarse	7	1024 MB	1	Fast Ethernet
Cheap	10	256 MB	2	Fast Ethernet
Cheapest	15	128 MB	1	Fast Ethernet

Table 1: Configurations of clusters of PCs.

(ii) a model assuming gigabit class communication.[3] Results with increasing number of processors are given in Fig. 5.

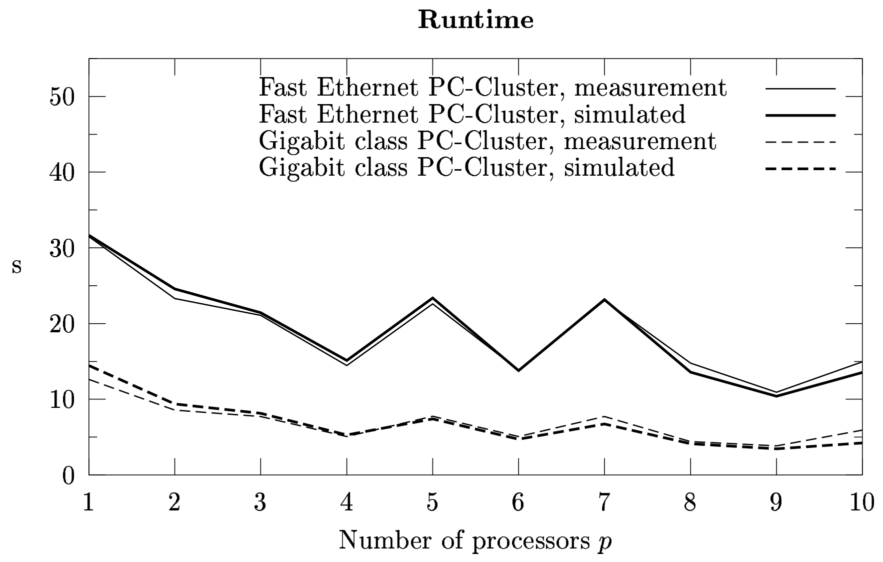

Figure 5: Simulation of the Cholesky factorization using CLUE ($n = 2000$).

It can be seen that for both network types, best results are achieved with a square (e. g., 2×2 or 3×3) processor grid. In general, empirical observa-

[3]The communication model was constructed and validated using the PC cluster of the RWTH Aachen, which uses an SCI network. A model for Myrinet communication networks is available, but not used for the work described in this paper.

tion shows that linear algebra algorithms work best if the processors can be mapped onto a $N \times M, N > 1, M > 1$ processor grid. Thus, most cluster configurations under consideration (with two exceptions) hold this property.

4.5 Final Experiments

For the small application case, an analytical model was chosen to evaluate the configuration performance. In this case it turned out that only the number of processors, their performance, and in the dual processor nodes memory contention have a significant effect on the overall performance. In contrast, the influence of the network or the memory size can be neglected.

The performance x of the configurations was estimated using the analytical model

$$x = (\#\text{processors}) \times (\text{memory\_contention\_factor}),$$

where the

$$\text{memory\_contention\_factor} = 0.5 \times \text{mcf}_{\text{Cholesky}} + 0.5 \times \text{mcf}_{\text{tridiag}},$$

with

$$\text{mcf}_{\text{Cholesky}} = 1.0$$

and

$$\text{mcf}_{\text{tridiag}} = \begin{cases} 0.77 & \text{for } \textit{dual} \text{ nodes,} \\ 1.0 & \text{for } \textit{single} \text{ nodes.} \end{cases}$$

The parameters $\text{mcf}_{\text{Cholesky}}$ and $\text{mcf}_{\text{tridiag}}$ for *dual* nodes were measured at a Pentium II system. The parameter $\text{mcf}_{\text{tridiag}}$ is expected to vary depending on the processor and memory bus system.

For the large test case, experiments using CLUE were made. Results of the experiments are given in Table 2. In some cases more than one parallelization strategy per configuration is shown. The column *high level* parallelization tells how many instances of `lapw1` are solved concurrently. The column *low level* parallelization tells how many processors are used for each instance of `lapw1` (including shared-memory parallelism). The column *parallelism* tells how many processors are engaged (the product of *high level* and *low level*). The column *empty* tells how many processors are not used in the program run. The columns *kernel 1* and *2* tell the overall floating-point performance in Mflop/s for the Cholesky factorization and the tridiagonalization, respectively.

Name	High Level	Low Level	Parallelism	Empty	Kernel 1	Kernel 2
Network	3	4	12	0	5194	1578
Memory	3	4	12	0	4153	1501
Memory	3	2	6	6	3066	884
Fine	1	6	6	1	2225	814
Fine	1	7	7	0	1852	702
Coarse	3	2	6	1	2283	1025
Coarse	3	1	3	4	1552	587
Cheap	3	6	18	2	4666	1913
Cheapest	3	4	12	3	3284	1763
Cheapest	3	5	15	0	3248	1543

Table 2: Floating-point performance (in Mflop/s) for the large test case.

5 Configuration Optimization

Table 3 evaluates both the small and the large test cases by points. Only the better result for each configuration (from Table 2) is included. The points are distributed for each test case such that the fastest configuration is awarded 100 points. The points for the large test case are split equally to the two kernels, each being assigned at most 50 points.

Name	Small Case	Large Case Kernel 1	Large Case Kernel 2	Sum
Cheap	100	45	50	195
Cheapest	85	32	46	163
Network	60	50	41	151
Memory	60	40	39	139
Coarse	40	22	27	89
Fine	40	22	21	83

Table 3: Assessment by points.

It turns out that the *Cheap* configuration has to be considered for the final decision, whereas the *Cheapest* configuration being second best. The *Network* configuration performs also well for the large test case and might even win the competition (for the large test case) if communication becomes more important. This may happen either

- if the processor speed increases,
- if less "high level" parallelism is available, or
- if smaller problems have to be solved. In this case the ratio of communication to computation is increased.

The other configurations perform rather poor, mainly because of a lack of raw compute power, since they have fewer processors.

Conclusion

The cluster evaluator CLUE has been demonstrated to be an efficient tool for finding optimum PC cluster configurations for given parallel applications and budget limits.

References

[1] R. Aversa, A. Mazzeo, N. Mazzocca, U. Villano, *Heterogeneous system performance prediction and analysis using* PS. IEEE Concurrency 6-3 (1998), pp. 20–29.

[2] 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.

[3] P. Blaha, K. Schwarz, P. Sorantin, S. B. Trickey, *Full-Potential, Linearized Augmented Plane Wave Programs for Crystalline Systems*, Comp. Phys. Commun. 59 (1990), pp. 399–415.

[4] T. Delaitre, G. R. Justo, F. Spies, S. Winter, *Simulation Modelling of Parallel Systems.* In Proceedings of the 1st Austrian-Hungarian Workshop on Distributed and Parallel Systems. Technical Report KFKI-1996-09/M,N, Hungarian Academy of Sciences, 1996.

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

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

[7] A. Ferscha, J. Johnson, *Performance prototyping of parallel applications in* N-MAP, Proceedings of the IEEE Second Int. Conference on Algorithms and Architectures for Parallel Processing, IEEE CS Press 1996, pp. 84–91.

[8] 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.

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

[10] E. Haunschmid, D. Kvasnicka, *High Performance Computing in Material Sciences. Maximizing Cache Utilization without Increasing Memory Requirements*, Technical Report AURORA TR1998-17, Vienna University of Technology, 1998.

[11] J. Labarta et al., DIP*: A parallel program development environment*, Proceedings of Euro-Par '96, Vol. II, Springer-Verlag, Berlin Heidelberg New York 1996, pp. 665–674.

[12] W. Mohr, A. Malony, K. Shanmugam, SPEEDY*: An integrated performance extrapolation tool for* PC++. Proceedings of the Joint Conf. Performance Tools 95 and MMB 95, Springer-Verlag, Berlin Heidelberg New York 1995.

[13] M. Rosenblum, E. Bugnion, S. Devine, S. Herrod, *Using the* SIMOS *Machine Simulator to Study Complex Computer Systems.* ACM TOMACS Special Issue on Computer Simulation, 1997.

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

[15] J. Yan, S. Sarukkai, P. Mehra, *Performance measurement, visualization and modeling of parallel and distributed programs using the* AIMS *toolkit.* Software Practice and Experience 25-4 (1995), pp. 429–461.

[16] XXX, *CLUE—Cluster Evaluation*, Technical Report, 2000.

[17] XXX, *Developing Architecture Adaptive Algorithms using Simulation with MISS-PVM for Performance Prediction*, 1997.