Cluster Configuration Aided by Simulation

Dieter F. Kvasnicka[1], Helmut Hlavacs[2], and Christoph W. Ueberhuber[3]

[1] Institute for Physical and Theoretical Chemistry,
Vienna University of Technology,
dieter@hermes.theochem.tuwien.ac.at
[2] Institute for Computer Science and Business Informatics,
University of Vienna
hlavacs@ani.univie.ac.at
[3] Institute for Applied and Numerical Mathematics,
Vienna University of Technology,
christoph@aurora.anum.tuwien.ac.at

Abstract. The acquisition of a PC cluster is often limited by financial restrictions. A person planning to buy such a cluster must choose between numerous configurations possible due to the large number of different PC components a cluster may be built of. Even if some of the applications that will be run on the planned cluster are known, it is generally difficult if not impossible to identify the one configuration yielding the optimum price/performance ratio for these applications a priori.
In this paper it is demonstrated how to use the newly developed simulation tool CLUE to decide which configuration of the components of a cluster yields the best price/performance ratio for a particular software package from computational chemistry. Due to the simulation based approach, even the impact of components available in the future only can be evaluated.

1 Introduction

Due to the dramatic price decrease of standard PC components and the exponential performance growth as described by Moore's law, traditional supercomputers are gradually being substituted by PC clusters consisting of several standard PCs containing between one and four processors and being interconnected by either Fast Ethernet or gigabit networks.

As there are numerous companies offering off-the-shelf PC components at very different prices and capabilities, a potential cluster buyer is usually faced with a large number of different possible cluster configurations. However, there is usually an upper limit to the overall cluster price, rendering some configurations as being too expensive. An example for a critical and difficult decision is, whether to use the available budget for buying more nodes, more processors per node, more main memory per node or a faster node interconnection network. For selecting one particular configuration it is necessary to know the number of users that are projected to use the cluster and the type of applications that will be run on it.

Although there is usually a general understanding of the influence of each component, it is often impossible to judge the total impact the chosen configuration will have on the applications that are foreseen to be run on the cluster. This is even more difficult if the projected applications are parallel applications consisting of several processes run on more than one processor or node. At this point, usually rules of thumb are applied, leaving room for unpleasant surprises once the cluster has been bought and installed.

In this paper it will be demonstrated how to apply the newly developed *CLUster Evaluator* CLUE to assess the performance of various cluster configurations for given parallel software, in this case the software package WIEN 97 [3], an application code from computational chemistry. This software 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.

2 Related Work

In the past, several attempts have been made to simulate the performance of parallel programs.

In trace-driven approaches as carried out for example by the PVM Simulator PS [1], TAU [9] or DIP [8], it is assumed that the interprocess communication patterns, i.e., the number and directions of sent messages, 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.

In case the simulation kernel is execution driven, however, as provided for example by SIMOS [10], changing communication patterns may also be taken into account at the expense of increasing the simulation time drastically.

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. Approaches like this are primarily meant for rapid prototyping, the main drawback being the need for creating program models in addition to the actual implementation.

3 The Simulation Tool CLUE

Being based on the *Machine Independent Simulation System for PVM 3* (MISS-PVM [7], the simulation tool 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.

CLUE allows the simulator user to model cluster configurations by specifying important parameters concerning the communication network and the computational nodes. These parameters are easily obtained either by carrying out measurements on real systems or by taking known or extrapolated parameters (see Section 5.1). The simulator assumes that the simulated parallel applications use the message passing library *Parallel Virtual Machine* (PVM) [6] for communication between the processes. The software models are most easily constructed by taking the original source code as input for the simulator. Thus it is not necessary to rewrite existing C or Fortran code or to create additional code. PVM based code can be used without modification.

The simulator then is driven by actually executing the original parallel program, the simulator routines being activated by catching all calls to PVM and redirecting them to special routines implemented in CLUE. The structure of CLUE is shown in Fig. 1. At the virtual layer, for each running process one

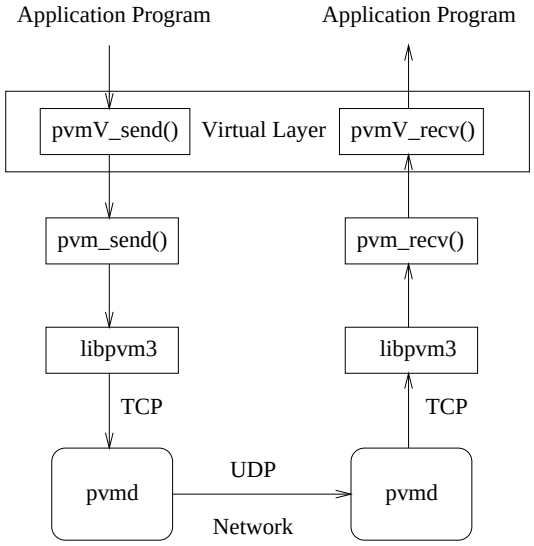

Fig. 1. Structure of CLUE.

instance of the simulator is created as well. The simulator itself thus consists of distributed instances communicating with other instances by exchanging PVM messages. Each simulator instance maintains its own local virtual time, representing the simulated computing time of this process. The virtual time, however, must not be confused with the real simulation runtime. Instead, it is a variable controlled by the simulator instances and increased if the respective attached application process has carried out some computation. This is detected by measuring the CPU time consumed by the attached processes between two adjacent calls to PVM. The local virtual time is then increased by this measured CPU time multiplied by the processing time factor of the node hosting the process as specified in the configuration file.

Virtual time is also increased when sending or receiving application messages, as carried out by the executed application program. It must be assured, however, that the reception of messages is in correct order with respect to the global virtual time, i. e., if at virtual time t_1 process P_1 sends a message to P_2 waiting for messages to arrive, it must be impossible that after being woken up and receiving this message, another process P_3 being at virtual time $t_2 < t_1$ sends a message to P_1. Thus, the simulator instances must use a protocol for distributed simulation in order to synchronize their local virtual times and guarantee message delivery in the right order. Therefore, the used protocol relies on

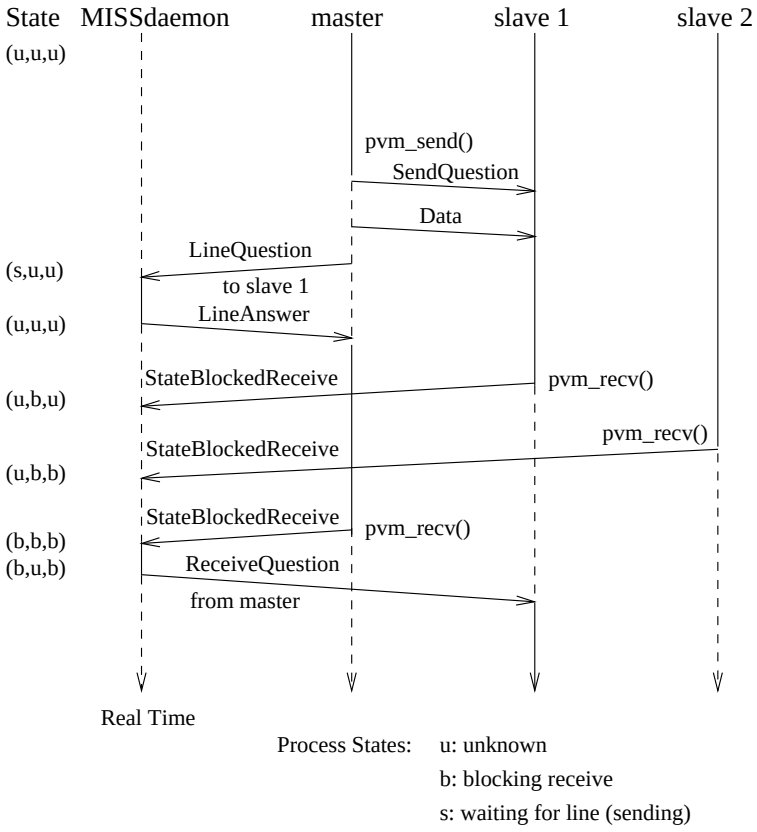

Fig. 2. Distributed simulation protocol.

an additionally spawned process called "MISSdaemon" for synchronization. The daemon keeps track of all processes, marking each process to be either in state *unknown*, *blocking receive*, *waiting for line*, *waiting for probe* or *killed*. Fig. 2 shows sample protocol messages in case a master process at the application level sends a message to a slave process. Upon calling a PVM routine, each simulator instance sends its new state to the daemon. As soon as it can be assured that the

global virtual time can not be violated, the receiving slave is allowed to receive the message.

4 Hardware Configuration Parameters

The cluster configuration model used by CLUE is called a "virtual machine". Virtual machines are defined by an input file being read in at simulation start. 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 Time. This is the time needed for `pvm_spawn` to be called. It is measured in multiples of 100 microseconds.

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

Send Time. 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 time $s(m)$ depending on the message length m is given by $s = k \times m + d$, or may be specified as tuples $(m_x, s(m_x))$, where the actual send time is interpolated linearly between these points.

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

Packing Time. 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 times 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. If the actual performance model turns out not to be exact enough, it can easily be changed by modifying the configuration file.

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

In this case study it is demonstrated hoy to apply CLUE to find an optimum hardware configuration for a particular parallel application, in this case by taking the well known computational chemistry package WIEN 97 as an example. It is assumed that an institute is planning to purchase a PC cluster to run the computationally expensive WIEN 97 simulations. Furthermore it is assumed that there is a tight limit on the budget that the institute can spend.

The most time consuming part of WIEN 97 is spent in a routine called `lapw1`, basically solving a generalized symmetric eigenproblem by applying Cholesky factorization and transforming the problem to a (simpler) tridiagonal eigenproblem. Thus, a cluster running WIEN 97 should be optimized for running parallel versions of the Cholesky factorization and for tridiagonalization, preferably as provided by the basic linear algebra subpackage BLAS [5], being a part of the parallel linear algebra package SCALAPACK [2].

Also, two application cases have been chosen for simulation:

- A small case with matrix sizes of 2500×2500 which can be solved on one processor on a standard PC.
- A larger case with matrix sizes 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.

5.1 Communication Models

Though CLUE can apply various communication models, in this case study, piecewise linear models are used for the send and transmission times (see Fig. 3). All model parameters have been measured on existing networks by running custom programs for measuring parameters like latency, bandwidth and processing speed.

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.

Alternative approaches for estimating configuration parameters include for instance taking published values from benchmarks or parameters provided by companies for their products (often specified, for example, for gigabit networks). Also, a popular rule of thumb specifies that when increasing the processor clock rate by $N\%$, then the performance gain will not exceed $N/2\%$. This rule may then be applied for extrapolating the performance of systems, even if the respective processors are not available yet. However, it does not apply if new processor cores or SIMD instructions are introduced or if other possible performance bottlenecks like caches or main memory are changed.

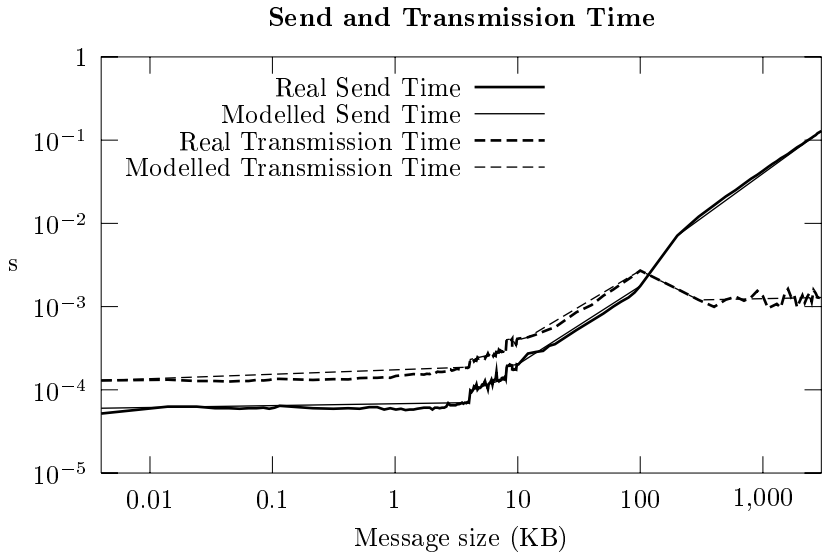

Fig. 3. Send and transmission time for a Fast Ethernet PC cluster. Sender and receiver are on the same node. Although they share the same memory, they communicate via a message passing library.

5.2 Hardware Configurations

To choose the configuration of PC clusters with highest performance for WIEN 97, several configurations are examined. The budget limit being set to $20.000,– 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.

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. Cluster configurations under consideration.

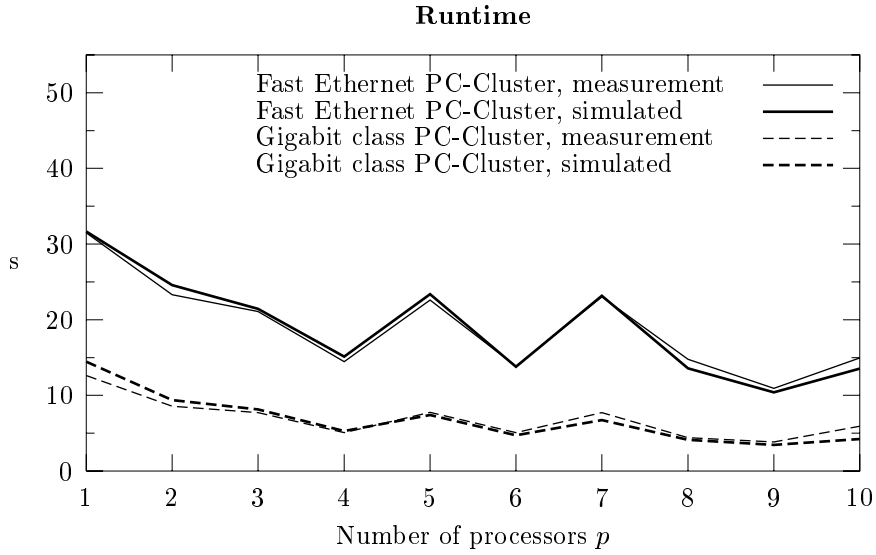

Fig. 4. Simulation of the Cholesky factorization using CLUE ($n = 2000$).

6 Simulation Results

Preliminary experiments simulating the BLAS Level 3 based Cholesky factorization have been performed with (i) a model for Fast Ethernet communication and (ii) a model assuming gigabit class communication. The communication model was constructed and validated using the PC cluster of the RWTH Aachen using an SCI network. Results with increasing number of processors are given in Fig. 4. It can be seen that for both network types, best results are achieved with a rectangular (e. g., 2×2, 2×3, 2×4 or 3×3) processor grid. In general, empirical observation 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.

Furthermore, for the small application 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, the influence of the network or the memory size being negligible. Thus, instead of applying simulation, an analytical model was chosen to evaluate the configuration performance. The performance $P(C)$ of configuration C is given by $P(C) = N \times f$, where N denotes the number of processors and f denotes the memory contention factor for *dual* processor nodes. This factor is further defined to be $f = 0.5 \times f_c + 0.5 \times f_t$, where $f_c = 1.0$ is the memory contention factor for the Cholesky factorization and f_t being the memory contention factor for the tridiagonalization, which has been measured on a Pentium II system to be 0.77 for *dual* processor and 1.0 for *single* processor nodes.

Results of the final simulation experiments are shown 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 in total (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 (*Kernel 1*) and the tridiagonalization (*Kernel 2*).

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.

7 Choosing the Optimum Configuration

In order to find the optimum cluster configuration for WIEN 97, the simulation results shown in Table 2 and the results given by the analytical model must further be investigated. This is done by distributing points to each result in such a way that the fasted configuration is rewarded 100 points. The points given to the remaining configurations then show how many percent of the fastest performance they achieved. Furthermore, the points for the large test case are split equally to the two kernels, each being assigned at most 50 points. The result is shown in Table 3, for configurations having two entries in Table 2, only the better entry has been considered.

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.

Name	$P(C)$	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.

8 Conclusion

In this paper it has been demonstrated how to use the cluster evaluator CLUE for finding optimum PC cluster configurations for given parallel applications and budget limits. By applying this technique, a priori performance evaluations of PC clusters to be bought may be carried out, thus being able to assess different cluster configurations by a quantitative procedure rather than a rule of thumb.

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. L.S. Blackford et al., SCALAPACK *Users' Guide*, SIAM Press, Philadelphia, 1997.
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 et al., *A Graphical Toolset for Simulation Modelling of Parallel Systems*, Parallel Computing 22–13 (1997).
5. J.J. Dongarra, J. Du Croz, S. Hammarling, R.J. Hanson, *An Extended Set of* BLAS, ACM Trans. Math. Software 14 (1988), pp. 18–32.
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. D.F. Kvasnicka, C.W. Ueberhuber, *Developing Architecture Adaptive Algorithms using Simulation with MISS-PVM for Performance Prediction*, 1997.
8. J. Labarta et al., DIP: *A parallel program development environment.*
9. W. Mohr, A. Malony, K. Shanmugam, SPEEDY: *An integrated performance extrapolation tool for* PC++, Proc. Joint Conf. Performance Tools 95 and MMB 95, Springer-Verlag, Berlin, 1995.
10. M. Rosenblum et al., *Using the* SIMOS *Machine Simulator to Study Complex Computer Systems*, ACM TOMACS Special Issue on Computer Simulation (1997).