Res. Lett. Inf. Math. Sci., (2000) **1**, 113-118
Available online at http://www.massey.ac.nz/~wwiims/rlims/

Building an Inexpensive Parallel Computer

Lutz Grosz and Andre Barczak
I.I.M.S., Massey University Albany Campus, Auckland, N.Z.
l.grosz@massey.ac.nz

Introduction

PCs linked through a network are used for parallel computing. These systems are often called 'Beowulf clusters'. The definition of a 'Beowulf cluster' is very blurred. It is applied to systems which use PCs in a student lab or in a department, typically when they are idle at nighttime, but also to systems of connected PC dedicated to run parallel applications. The common factor is building a parallel computer using inexpensive, standard hardware for nodes and network, as they can be bought in every computer shop. This concept of using a large number of cheap, mass-produced components is contrary to the classical high-performance computing approach, which relies on expensive and highly specialised components. In addition to convenient prices for hardware and software (most software components are available in the public domain) Beowulf clusters are more flexible regarding their applicability and adaptation to improvements of the technology.

We have assembled a prototype cluster of four Intel based PCs linked through a 4-port Ethernet switch. The operating system on each node is LINUX. MPICH [3] provides the interface to the network for application programs. The parallel finite element code VECFEM [5] has successfully been tested on the cluster. In this paper we give a brief overview on the architecture and discuss a few basic performance issues.

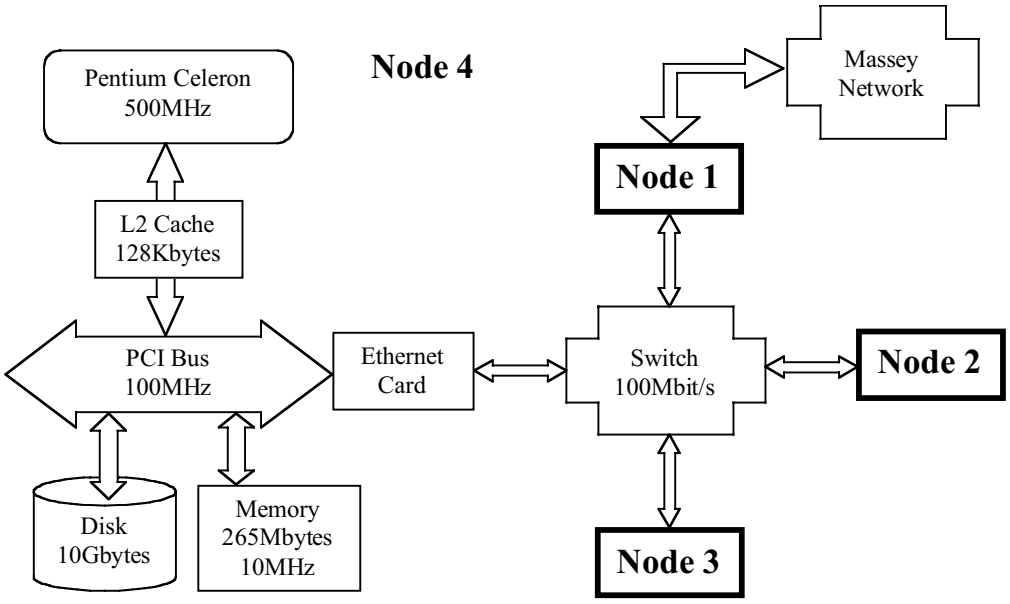

Figure 1: Cluster architecture basing on four PCs and a 4-port Ethernet switch.

www.manaraa.com

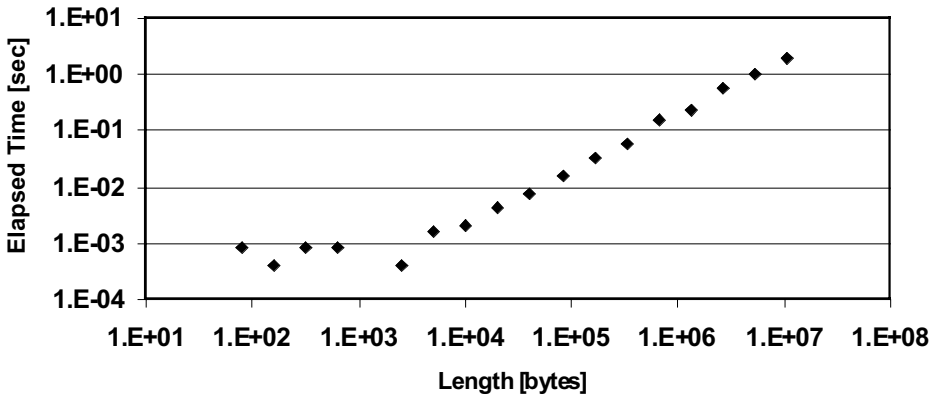

Figure 2: Transfer times versus message length for four processors and a 100Mbits switch

Network

Figure 1 shows the configuration of the installed prototype with four compute nodes. They are linked through a 4-port Ethernet switch, which has a theoretical peak transfer rate of 100Mbits per second. This transfer rate is provided for communication between every pair of nodes simultaneously. For instance node 1 can send a message to node 3 at transfer rate 100Mbit/s and at the same time node 2 can send a message to node 4 at the same transfer rate. The network between the nodes is isolated from the Massey Network in order to keep away undesirable loads. One node (node 1 in Figure 1) acts as a gateway and establishes the connection to the outside in order to allow users to work on the machine.

Figure 2 shows the timing for exchanging messages of various lengths between pairs of nodes in a four-processor configuration. All processors are sending and receiving messages at the same time. This simulates a situation, as it is typical for data parallel applications. The test program uses the MPICH implementation for LINUX [3] of the MPI message passing library [2]. The communication is non-blocking but synchronized. That means that neither send nor receive calls wait for finalization before return but the send of a message is not set up before the corresponding receive has been called. This allows exchanging data without expensive allocation of message buffers. The timings in Figure 2 are elapsed times on a dedicated cluster. For message length less than 3000 Bytes the timing is independent from the message length. This indicates the start-up costs of about 0.8msec for any data transfer. For longer messages (>3000Bytes) a transfer rate of about 5.5Mbytes/sec is reached. As the ports of the switch are used in two directions simultaneously, the test shows that a sustainable, bi-directional transfer rate of 88Mbits/sec is achieved at least for long messages.

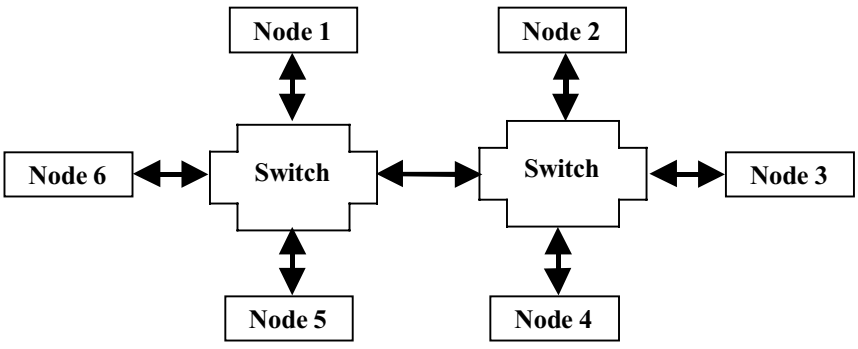

Figure 3: Configuration of six nodes linked trough two 4-port switches.

www.manaraa.com

It is very important that the nodes are linked through one switch, only. Unfortunately, the number of ports in a switch is limited (usual 4, 8 or 16 ports can be bought). Figure 3 shows this situation when using two 4-port switches to link six nodes. In this configuration simultaneous data transfers between the node pairs 1 and 2, 6 and 3 and 5 and 4 have to share the only link between the switches. One has to expect that the transfer rate is reduced by 1/3 for each of the three data transfer (=33Mbits/sec). When linking two groups of 15 nodes through two 16-port switches the available transfer rate is 100/15=6.7Mbits/sec between nodes in a worst case scenario. So in theory, the network is scalable as new nodes can be added through additional switches. However, this produces an inhomogeneous system, where the costs for data transfers between nodes depend on the nodes involved in the communication process. The programmer has to address this problem when he wants to develop scalable program codes without massive performance drops. One possible solution is the introduction of a logical ring topology where only data transfers between direct neighbors are used. When using a suitable mapping of this ring topology onto the physical network halve of transfer rate which is available on the ports is achieved independent from the number of nodes. In the configuration of Figure 3 only data transfer between node pairs 1 and 2, 2 and 3, 3 and 4, 4 and 5, 5 and 6 and 6 and 1 are used.

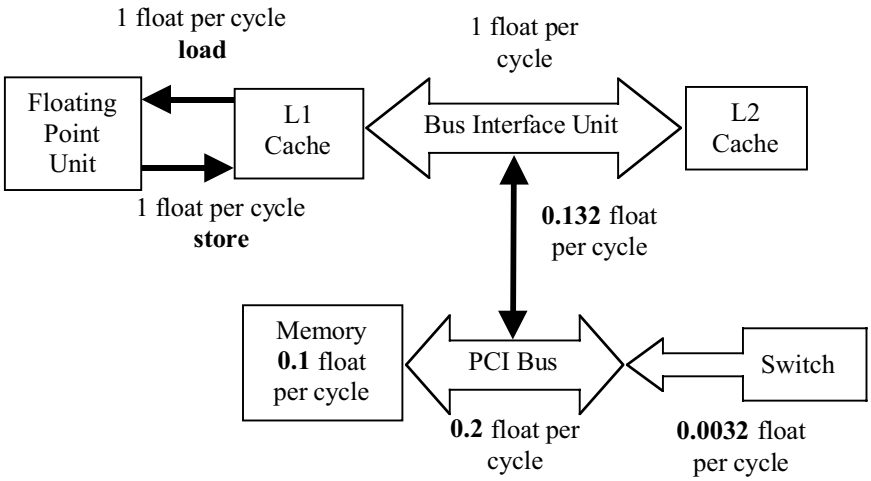

Figure 4: Data Transfer Rates on a Node.

Node Architecture

Each node is standard PC technology, see Figure 1. The 64-bits PCI bus, which is running at 100MHz, transfers the data between the processor, the memory unit, the disk and the Ethernet card, which holds the connection to the Ethernet switch. The processor is an Intel Celeron at 500MHz [4]. Figure 4 shows more details on the data flow in the node. In the following all transfer rates are expressed in 64-bit packages (called one ***float***) per machine cycle (=2nsec) as numerical applications, which are the major application area we are interested in, use operations on 64-bit floating-point numbers.

The floating-point unit is pipelined and can deliver one addition result per machine cycle, if enough data are available. That means that in an optimal situation the processor can manage to deliver 500Mflops (i.e. 500 Million additions per second). As shown in Figure 4 data from the level-1 (L1) data cache (16Kbytes of size) can be loaded and stored simultaneously each with 1 float/cycle and a latency of 3 cycles [1]. As an addition of two values needs two load and one store operation, the maximal achievable performance is 2/3 of the theoretical peak performance (=333Mflops) if all data are in the level-1 cache. Data, which are

www.manaraa.com

not available in the L1 cache, are loaded from the level-2 cache (128Kbytes of size) via the bus interface unit, which transfers one float per cycle. In this case the maximal achievable performance for the addition is reduced by the factor 1/3 (=167Mflops) as two floats have to be load from and one float has to be stored to the L2 cache. In a general situation, the data are located in the main memory and have to be transferred through the PCI bus. The connection from the bus interface unit to the PCI bus is running on 66MHz with 64 bits, which does not scale with the processor clock. So only 0.132 floats per cycle are transferred from the bus interface unit to the PCI bus although the PCI bus itself as well as the 64-bit memory running on 100MHz provides 0.2 float per cycle. This slows down the achievable performance for adding two vectors by the factor 0.132/3=0.044 (=22Mflops).

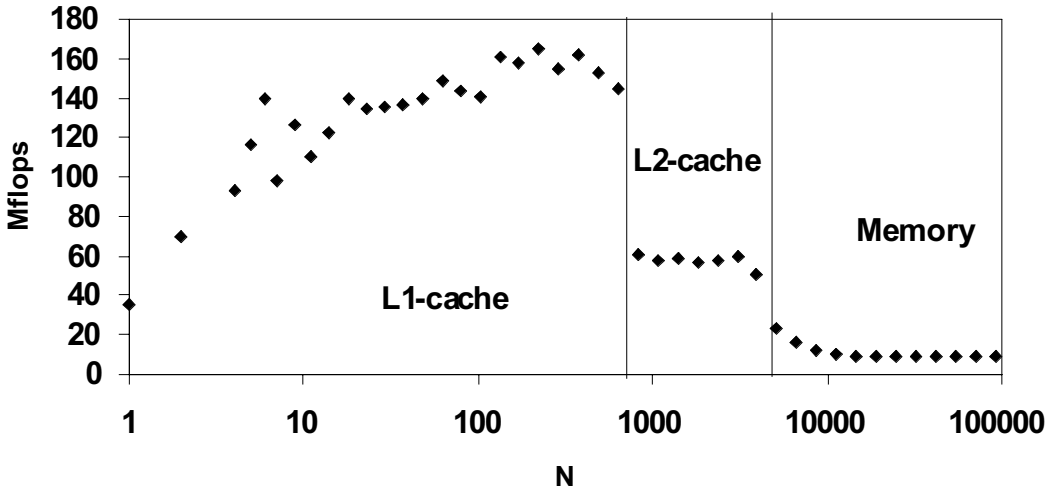

Figure 5: Performance of vector addition.

Figure 5 shows the performances measured for the vector addition operation

```
REAL*8 X(N),Y(N),Z(N)
DO I=1,N
      X(I)=Y(I)+Z(I)
ENDDO .
```

The code is compiled using the g77 compiler (see http://gcc.gnu.org/onlinedocs/g77_toc.html) under LINUX with the option `-ffloat-store -ffast-math -funroll-all-loops -fno-f2c -O3`. The loop is repeated several times in order to ensure a reliable time measurement. The performances in Figure 5 show three regions: For N less than 650 there is an increase of performance up to about 160Mflops. If N is increased above 650 the performance drops down to 55Mflops and for N above 5200 down to about 12Mflops. If the vectors X, Y and Z are longer than 630 they do not fit into the 16Kbytes L1 cache at the same time and have to be stored into the L2 cache. For this situation we have expected a performance drop by a factor 2/3 but the measured performance drop from 160Mflops to 53Mflops is much larger (about 1/3). For N larger than 5200 the data have to be stored into the memory. The measured performance 12Mflops for this case is less than the predicted performance of 22Mflop. It is very difficult to give a clear explanation for the gaps between the predicted and measured performances without a detailed analysis of the instructions produced by the g77 compiler. In fact, the efficient usage of pipeline capabilities of the Celeron processor requires firing load, store and addition instructions in the right order and at the right time.

www.manaraa.com

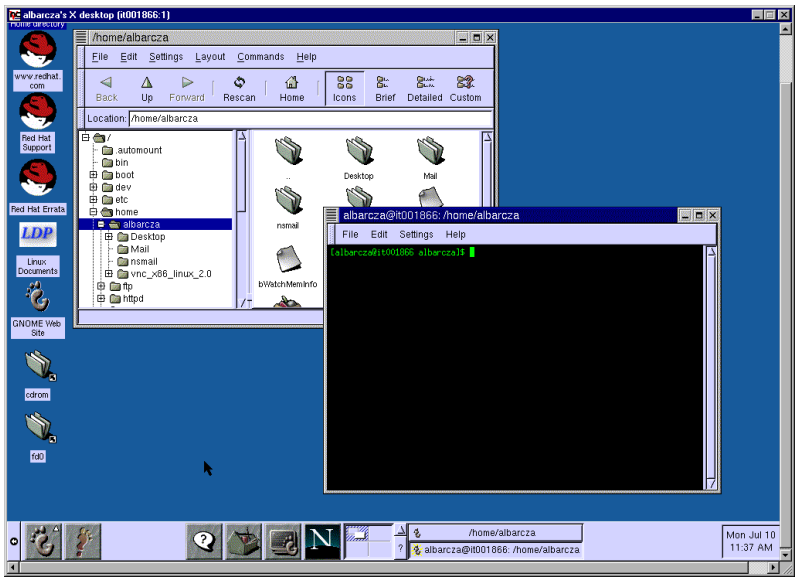

Figure 6: The Gnome interface window.

Software

Our prototype installation of a Beowulf cluster runs the Red Hat distribution of LINUX (see http://www.redhat.com). The kernel is version 2.2.14, one of the latest available. LINUX is the preferred operating system on Beowulf clusters because it is very reliable, the sources are publicly available, and it can be used without paying license fees. Several numerical software packages, including VECFEM [5] and SCALAPCK [9], have been installed and tested using the MPICH installation of MPI [3] and the GNU version of the Fortran 77 compiler (see http://gcc.gnu.org/onlinedocs/g77_toc.html).

The Red Hat distribution comes with the powerful graphical user interfaces, Gnome and KDE [7]. They provide a productive environment for both the system administrator and the user. Figure 6 shows a snapshot of the Gnome user interface. The cluster is accessed by VNC [8]. Through a very light protocol it allows remote users viewing a computing desktop environment, like Gnome, from anywhere on the Massey network and from a wide variety of machine architectures including Win98, WinNT or even MacOS.

Summary

We are very pleased with the achieved transfer rates for the switch-based network. Nevertheless, the achievable performance for the Celeron-based nodes is not as good as expected (only 12Mflops compared to 500Mflops theoretical peak performance). This is mainly caused by the architecture of the Celeron processor but also by poor compiler efficiency. Higher clock rate on the Celeron will not change this, as transfer speed of the system bus does not scale with the clock rate of the processor. In addition to the parallelization aspects, the user has to consider cache-aware algorithms in order to get a reasonably good performance on each node. A larger L2-cache will certainly help to speed-up most applications. Nevertheless, data location, if in the cache, the local memory or the memory of another node, has to be a major concern of the programmer.

References

[1] *Intel Architecture Optimization Reference Manual*, Intel Corporation, Order Number: 243706-012, 1999.

[2] M. Snir and S. W. Otto and S. Huss-Ledermann and D. W. Walker and J. Dongarra: *MPI-The Complete Reference*, MIT Press, Cambridge, 1994.

www.manaraa.com

[3] W. Gropp and E. Lusk and N. Doss and A. Skjellum: *A high-performance, portable implementation of the MPI message passing interfaces standard*, Parallel Computing, 22(6), 1996, pp 789-828.
[4] *Intel® Celeron™ Processor and Intel® 810E Performance Brief*, Intel Corporation, Order Number: 243706-012, 2000.
[5] L. Grosz and C. Roll and W. Schoenauer: *Nonlinear Finite Element Problems on Parallel Computers*, in: J. Dongarra and J. Wasniewski: Parallel Scientific Computing, First Int. Workshop, PARA'94, Denmark, 1994, pp 247-261
[6] *Intel Architecture Software Developer's Manual, Intel Corporation Volume 1: Basic Architecture*, Intel Corporation, Order Number 243190, 1999.
[7] *The Official Red Hat Linux Getting Started Guide*, Red Hat Inc, Durham NC, 2000.
[8] T. Richardson and Q. Stafford-Fraser and K. R. Wood and A. Hopper: *Virtual Network Computing*, IEEE Internet Computing, Vol.2 No.1, Jan/Feb 1998 pp33-38.
[9] L. S. Blackford, J. Choi, A. Cleary, E. D'Azevedo, J. Demmel, I. Dhillon, J. Dongarra, S. Hammarling, G. Henry, A. Petitet, K. Stanley, D. Walker, R. C. Whaley: *ScaLAPACK Users' Guide*, SIAM, Philadelphia, PA, 1997.

www.manaraa.com