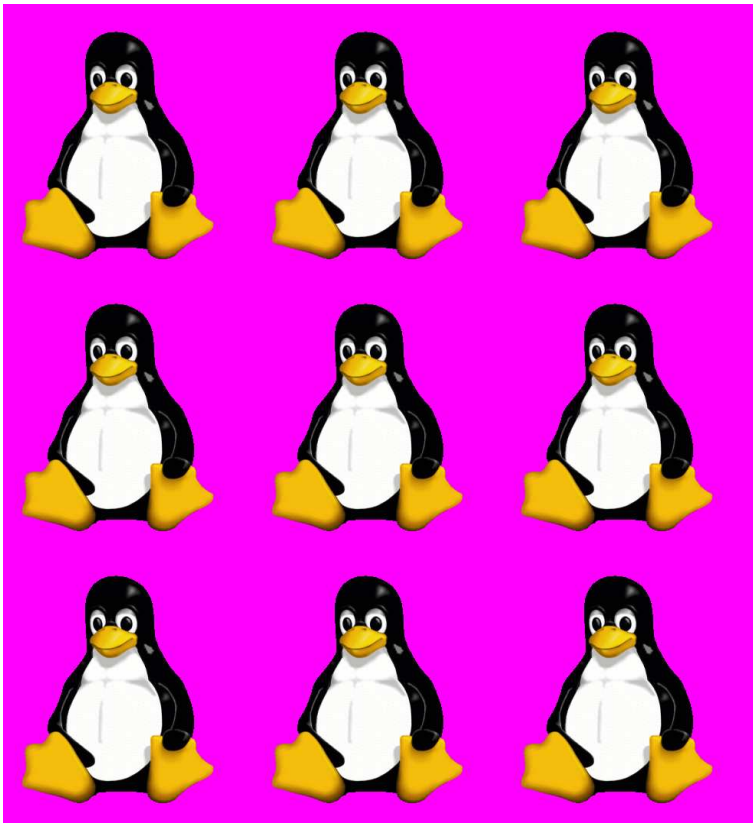
# An evaluation of some Beowulf clusters

Aad J. van der Steen
Dept. of Computational Physics
Utrecht University
P.O. Box 80195
3508 TD Utrecht
The Netherlands
steen@phys.uu.nl
www.phys.uu.nl/~steen

Technical Report **WFI-2000-07**



**Abstract**

We report the results of an evaluation project on three Beowulf type clusters. The purpose of this study was to assess both the performance of the clusters and the availability and quality of the software for cluster management and management of the available resources. For the results we wanted to derive answers whether it would be viable to use a cluster system routinely in a multi-user environment with comparable maintenance cost and effort to that of an integrated parallel machine.

NCF, the National Computing Facilities Foundation, supports and furthers the advancement of technical and scientific research with and into advanced computing facilities and prepares for the Netherlands national supercomputing policy. Advanced computing facilities are multi-processor vectorcomputers, massively parallel computing systems of various architectures and concepts and advanced networking facilities.

# Contents

# 1 Introduction

NCF, the HPC foundation within NWO, the Dutch equivalent of the NSF in the spring of 2000 commissioned a project to assess the conditions required for employing a Beowulf cluster in a multi-user environment as opposed to an environment where a cluster is used by a single group to run one or a few predefined applications. Or, in short, could such a cluster replace an integrated parallel system, like an SGI Origin2000, an IBM SP, or a Compaq Alpha SC? Obviously, in such a case much higher demands have to be made on the cluster and resource management software: It should be possible to install new (system) software on all or a selected group of nodes, to take nodes off line for maintenance without affecting other nodes, and some batch processing system should be available to execute a workload that exceeds the immediate resources of the cluster. In addition, it would be highly desirable to have monitoring software to display the utilisation of the nodes, both for individual jobs and for the entire system.

In High Performance Computing two types of large scale computing occur: capability computing and capacity computing. The first meaning the type of processing to accommodate very large and time consuming computing tasks while the latter means the processing of large amounts of computing jobs, not necessarily large, in a batch environment. Although these two styles of computing are not mutually exclusive, the configuration of a large-scale High Performance Computing system will generally favour one of these styles. The large systems as presently made available by NCF try to accommodate both styles with mixed success. It is well known that most Beowulf clusters are used as capability clusters, i.e., few very demanding programs are run on such clusters without much considerations with respect to throughput. This seems also to hold for the clusters that are presently installed in The Netherlands. The problem as posed by NCF that motivates this study can also be formulated as: Is it possible to operate a Beowulf cluster not only as a capability system but also as a capacity system and, preferably, as both? This should become clear by reviewing the present status of cluster computing.

The last few years the adoption of Beowulf clusters has taken a tremendous flight. This is also partly the reason for this study. The first clusters were demonstration projects which intent it was to show that clusters could be a (much cheaper) alternative for integrated massively parallel machines. These first clusters were built on a strictly do-it-yourself basis and virtually all software that makes life easier: a single system image, centralised software maintenance, and job manipulation software had to be heavily adapted or built from scratch. This situation has much approved as can be found from the IEEE Task Force for Cluster Computing (TFCC) White Paper [14]. A large amount of software in the public domain is now offered to make clusters more usable and easier to build and maintain. In addition, several useful books on building and using clusters, like [1, 13, 10] have been published.

The term Beowulf cluster comprehends a large family of systems, ranging from very homogeneous ones with exactly the same type of nodes and a network in the 100 MB/s range to clusters built from every odd PC around connected by 10 Mb/s Ethernet. Also, various operating systems fall within the definition: Linux (predominantly and in several flavours), Sun's Solaris, Compaq's Thru64 Unix, and Microsoft's Windows NT. In this study we will limit ourselves both with respect to the configurations and the operating systems. This is because the parameter space with respect to the configuration/OS combinations is very large. We confine ourselves in this study to Linux clusters with homogeneous nodes. In addition, we have only looked at cluster products of some vendors that are active in the Beowulf cluster field. This is primarily for practical reasons and does not presume that other, self built clusters, would necessarily be inferior in any respect. We only consider Linux clusters because such systems constitute the vast majority of existing clusters, presently and in the foreseeable future. Also in this respect we do not presume that clusters with other OSes would be of less value.

Three vendors gave us the opportunity to look into the supporting software they provide as well as performing a benchmark on their clusters. The results of both the software inventory and the benchmark are given in

the following sections of this report. It is organised as follows: section 2 describes the hardware as offered by the vendors as cluster systems, section 3 give an overview of the software made available from the vendors for their systems, and section 4 discusses the benchmark which was used to gauge the performance of the systems. Section 5 details the test configurations we used in performing the benchmark and section 6 presents the results of the benchmark. Finally, in section 7 we draw some conclusions from the results of the software inventory and the benchmark results. Initially, also Siemens/Fujitsu had agreed to take part in this test. However, later they had to withdraw their support because they were not able to make available sufficient machine and manpower resources to perform the benchmark associated with with this study. This is in more than one way regrettable because it would have been better to have a still broader basis in terms of configurations for this project but foremost because of the fast SCI ([9]) network that is used in the Siemens/Fujitsu HPCLine clusters. Results from the other clusters indicate that such a fast network is certainly a quite important factor in the overall efficiency of a cluster in capability computing.

Because of the community for which this evaluation was meant only codes from the technical and scientific HPC field were used. Therefore, the findings reported here with respect to performance are limited to this area. The relative virtues and shortcomings of the clusters involved may be different for other workloads.

# 2 Description of hardware

As already remarked, three vendors were willing to make available a cluster configuration to evaluate their hardware and software products. These vendors were: Compaq, IBM, and SGI. Two vendors market Intel-based clusters, viz., IBM and SGI while Compaq offers cluster based on the EV67 Alpha processor and on the Intel PIII-based Proliant systems. In our study we used an Alpha-based cluster. All systems may be coupled with (fast) Ethernet as a communication network, however, most clusters also support much faster networks like Myrinet. Below we give a more detailed description of the configurations as offered by the three vendors.

## 2.1 Compaq cluster hardware

The components in a Compaq cluster are either XP1000 or DS20 nodes, with resp. 1 or 2 CPUs per node. Presently the CPUs are 667 MHz Alpha 21264 (EV67) chips with a theoretical peak performance of 1.33 Gflop/s. The network offered can be (fast) Ethernet or Myrinet (although other networks may be possible). The Alpha EV67 is a superscalar RISC processor which is able to deliver 2 floating point results per clock cycle. The processor further boasts out-of-order execution of instructions and has 2-way set-associative data and instruction L1 caches, a 1–16 MB combined L2 data/instruction cache and a bandwidth to memory of 8 GB/s.
In practice a peak speed of 701 Mflop/s, 52.6% of the theoretical peak speed, was measured in a $10^{\text{th}}$ degree polynomial evaluation algorithm on a 1 CPU XP1000 node. One should bear in mind that in the 2 CPU DS20 nodes the bandwidth from memory to the CPUs has to be shared between them and, consequently, that the performance may be lower per CPU for memory intensive programs.

As mentioned, Myrinet [8] or (fast) Ethernet can be used to connect the nodes. The former has a peak bandwidth approaching 160 MB/s and a latency of 15-20 $\mu$s while a 100 Mb/s Ethernet connection yields a theoretical peak bandwidth 12.5 MB/s. In practice, due to software overhead a peak speed of 11.3 MB/s was observed with a latency of 105 $\mu$s.
The topology of the Myrinet network can be configured as a multistage crossbar or a fat tree in large clusters by using 8-port or 16-port Myrinet switches. The Ethernet-based clusters are usually not very large (16–32 nodes at most) and mostly have a single Ethernet loop as the connecting network which is obviously not an optimal topology for communication intensive programs.

## 2.2 IBM Netfinity 1500 cluster hardware

The IBM Netfinity 1500 cluster is based on the same CPU as used in the SGI systems (although the clock cycle is different in the respective systems). The L2 cache size is 256 KB. It is packaged in rack version 4500R which can contain 1 or 2 CPUs per node. As said before, the theoretical peak performance and the clock speed have the same value for 64-bit floating point computations. In our case this was 600 Mflop/s on the Netfinity nodes [1]. The highest observed speed was 330 Mflop/s.

The interconnection network often is Myrinet [8]. As stated before, a variety of network topologies can be realised with the Myrinet switches: full and hierarchical crossbars and fat trees, e.g. in case of the Los Lobos cluster at the University of New Mexico the network consists of coupling 4 Myrinet 64-way CLOS network

[1] Although the Intel PIII CPUs should be able to deliver 2 flops per cycle for 32-bit operands, they cannot do so 64-bit operands, hence the halved Theoretical Peak Performance(TPP) which makes the TPP in Mflop/s numerically equal to the MHz rate of the CPU.

switches to connect the 256 nodes. Furthermore, of course, (fast) Ethernet or Gigabit Ethernet can be used. In our tests a Myrinet network was used.

## 2.3 SGI L1200 cluster hardware

SGI offers clusters based on its SGI L1200 nodes. These nodes can house 1 or 2 Intel PIII Xeon processors currently at a clock rate of 700 MHz. This means that in our case a theoretical peak speed of 700 Mflop/s per processor is projected. This speed will generally be lower when two processors per node are configured because both processors have to access the same on-board memory. In the results section some evidence is given for this effect which also occurs in the IBM Netfinity cluster nodes. Cache sizes and structure are exactly the same for the IBM and SGI nodes: 32 KB L1 instruction and data cache and 256 KB unified L2 cache. The system bus has a speed of 100 MHz.

Also the network possibilities of the SGI L1200 cluster are identical to those of the IBM cluster: Myrinet, Gigabit Ethernet, and 100/10 Mb/s Ethernet are available as a connecting medium. In the tests performed for this study both Myrinet and fast Ethernet have been used.

# 3 Description of software

Apart from the hardware to run on, the software provided on the cluster is a decisive factor both with respect to the performance and the ease of management and use. In this section we discuss the software repertoire (largely as far as our tests are concerned) that is available on the systems included in this evaluation project. This software comprises the application level tools like the compilers and communication libraries as well as the installed batch system and the cluster management software that is available for installing software, adding or removing nodes, etc.

## 3.1 Compaq

At the Alpha-based cluster of Compaq, we met a mixture of proprietary Compaq software and public domain software. This stems from the fact that Compaq provides its native compilers, Fortran 90, C, and C++, to obtain high performance levels which would be less easy to attain with the compilers as shipped with the usual Linux distributions. For this reason one of the programs in the benchmark, viz. `vasp`, had to be build with the Compaq variant of its `Makefile` to get a functioning executable, rather than the Linux `Makefile` that also was present.
The versions of the Compaq compilers present were Compaq Fortran Version 1.0, C Version 6.2.9.002, and C++ Version 6.3.10. For the programs in the benchmark only the first two compilers were required. For compatibility reasons also the GNU Compiler suite Fortran, C and C++ version 2.91.66 was available. However, we did not use these in our tests. For communication, the public domain MPI implementation MPICH Version 1.2.0 was used. Furthermore, the public domain version of the batch system PBS, Version 2.2, was available to be used in the single-user and throughput parts of the benchmark. All of this software is integrated in the RedHat Linux Kernel 2.2.13-2 on Alpha.

Like most vendors, Compaq has cluster management software to ease the burden of installing software, taking nodes off line for maintenance, monitoring load and performance, etc. Compaq provides CMU, the Cluster Management Utility, to perform these tasks. Although in our actual test situation CMU was not used, we still want to describe it, as it is the official product offered by Compaq.
CMU can be used both for Linux or Compaq own Unix flavour, Tru64 Unix. It is assumed that the nodes in the cluster are (almost) identically configured and also hardware-wise are (almost) similar. CMU uses the Serial Remote Console present on all Alpha workstations or servers. CMU comes with a Graphical User Interface that shows from each node in the system its name, IP address, and status on the screen of a designated monitoring node.
Individual nodes can be selected by mouse-clicking their names of in the displayed panel. After selection a desired action can be specified, like (re)booting or halting a selected node or by setting up a telnet session to such a node. Of course one can also select/unselect all nodes in the cluster. From the console node one can broadcast commands to all selected nodes. This can be verified in the appropriate windows created for each node if desired.
The nodes in the cluster are monitored by polling their TCP/IP echo port. If the status of a node changes, this is noticed and an event is generated which, dependent on the type of event and the configuration of CMU may result in a pop-up window on the console, executing an appropriate script or program, or mail to the cluster manager to inform him/her on the occurrence of the event.
Monitoring tools are rather basic on the Linux clusters: the utilities `xosview`, `xsysinfo`, `xcpustate`, and `top` are present. On the clusters running Tru64 Unix the utility PVIS is available. PVIS can generate bar and polar diagrams for load and CPU activity per node/CPU or display for instance the paging activity in each node. Of course also `top` is present under Tru64 Unix.
The CMU GUI also enables system user administration tasks like adding, and deleting users, changing

passwords, manage groups or update a NIS database.
The simplest way for initially installing the necessary cluster software is by using the cloning feature of CMU: a "clone image" of an installed master disk is made which may reside on any node. This compressed image, however, cannot be made from a booted system disk, so it must be a separate copy. Once this master image is available, CMU can mount it in the file system space. The clone image is downloaded to the local disks of the nodes and NFS mounted in the image server. As soon as a node is mounted the CMU backup script is executed, the local disk is mounted and the compressed image is built. With this first step completed, the clone image can be uploaded to all remaining nodes in the defined group (this does not necessarily have to be all of the nodes in the cluster). The upload procedure is executed in a tree-like fashion: when uploading to node 1 is completed, is starts the same procedure for nodes 2 and 3 which, in turn, will initiate the uploading for nodes 4 and 5 and 6 and 7, respectively, etc. The cloning facility can also be used for backup and update activities.

## 3.2 IBM

On IBM's Netfinity cluster the Portland Group Fortran 77 and 90 compilers are provided. For C and C++ programs the GNU compilers are used. The RedHat distribution of Linux seems to be more or less the standard but IBM also provides SuSe, Caldera, and TurboLinux distributions. For this project we used RedHat version 6.2.
The normal MPI version that is distributed with the cluster is the Argonne MPICH 1.2.0 library which can be used with (Fast) Ethernet. However, when as in our case, Myrinet is the connection network, Myrinet's MPICH-GM is the MPI implementation used. In our case the version was MPICH-GM 1.2..3.
IBM has big plans for porting its own parallel system management software to Linux clusters and to make it available in the public domain. Various product are announced for 2001: System Resource Controller (SRC), Resource Monitoring and Control (RMC), the General Parallel File System (GPFS) supporting MPI-IO, and more.
Presently, the clusters still rely on public domain tools as are available in the cluster computing community and which for instance can be found in [10, 14]. In our particular case, "home grown" on `rsh` based tools were used for the cluster management as the cluster was of small size (8 nodes, 16 CPUs). The batch processing software, however, was more advanced: LSF version 4.0.1 was available for the throughput test in the benchmark.

## 3.3 SGI

On the Intel-based SGI L1200 cluster the Portland Group compiler set with is provided: `pgf77`, `pgf90`, and `pgcc`, respectively. For the benchmark programs `adf` and `vasp`, the appropriate versions of the `Makefiles` were required to obtain correctly functioning executables.
RedHat Linux seems to be the predominant Linux flavour to be provided, although other versions, like SuSe Linux, also are supported. In our tests RedHat Linux 6.2 was used.
The SGI clusters, as most clusters do, rely on a public domain version of MPI. In our case in fact two versions, because for the tests with fast Ethernet the Argonne MPICH version 1.2.0 was used while for the tests with Myrinet we used their version MPICH 1.2..3.

For cluster management SGI distributes the ACE (Advanced Cluster Environment) utility. Conceptually a cluster set up with ACE consists of a head node and number of execution nodes. Although a head node also can be an execution node, the reverse is not true: there can be only one head node. Users log in at the head node and all management tasks are executed by the head node. In this context SGI defines a node to be either a 2-processor L1200 system of a 4-proccesor L1400 system. In our case the 2-processor 1200 nodes were used. ACE consists of various components, all of which are in the public domain:

- `lconsole`. This utility provides a serial link in case the network is not up. It can be used to boot nodes, can add/delete users and passwords via the serial port, start a `telnet` session and power up/down the nodes via this interface.

- PBS 2.2, the public domain batch scheduler that can be used to configure batch queues, to submit jobs to these queues, and to monitor their status.
- Performance Co-Pilot (PCP) that can be used to monitor the system activity. The public domain version distributed with ACE does not have all the GUI interfaces that come with SGI's proprietary IRIX version that was already available for some years on the SGI Origin parallel systems. It can display all kinds of system activity, like CPU usage or network activity which may be useful to identify hot-spots in programs. PCP also includes tools to instrument MPI calls and, in case Myrinet is used as a network, statistics from the Myrinet switches can be gathered and visualised.
- MPICH, the public domain implementation as provided by Argonne National Laboratory, version 1.2.0. This version can be used with (fast) Ethernet and indeed was used so in the fast Ethernet experiments. In addition, when Myrinet is used the Myrinet MPICH 1.2..3 library is required and also the GM drivers to interact with the Myrinet network. In our case the 1.1.3 version of these drivers were used. ACE requires a configuration file that specifies the nodes by name and the ports used to set the configuration for the Myrinet network. When one processor per node is used only port 2 of each node is used; when two CPUs per node are required, ports 2 and 4 have to be specified.
  For Ethernet the `ch_p4` libraries are needed, for Myrinet MPICH the `ch_gm` libraries are required.
- The VA Cluster Manager (VACM). VACM contains cluster administration tools in which also a graphical user interface is included. NEXXUS, a module of VACM, handles all node administration. It for instance adds, removes, and renames nodes, groups, and users, sets and modifies passwords.
  The EMP module monitors the hardware networking status of the nodes and is used to power up/down individual nodes via the network.
  The SYSSTAT module is used to gather node statistics like CPU and memory load, to obtain a list of processes currently running on a node, etc.

ACE can be easily installed by executing `installace` from the head node which can be done automatically or interactively. In the latter case the installer can choose which of the tools described above should be installed and which not. For instance, when no Myrinet network is present, it is not relevant to install the Myrinet MPICH version. ACE contains evaluation software to check whether all installed tools and packages indeed run as intended.

# 4 Description of the benchmark

To measure the performances of the clusters both synthetic programs and real applications were used. The synthetic programs are useful to assess the baseline performance for important operation kernels and to establish system parameters like bandwidth and latency of the connecting network, the maximum attainable 64-bit floating point performance, etc.

The applications were chosen from a set that is on one hand typical for the present use of clusters in the Netherlands and on the other hand would behave reasonably well with respect to porting. Six program(set)s were chosen:

- `adf` — A quantum chemistry code.
- `euroben` — A synthetic benchmark set.
- `maya5` — A neutron star evolution code.
- `primmod` — An MD code for colloid simulation.
- `rboltz` — A Lattice Boltzmann particle MD simulation.
- `vasp` — Another quantum chemistry code.

Of this list the EuroBen set [11] consists of 24 programs that measure system parameters and the speed of basic kernels and standard algorithms. The other 5 programs are frequently-used user applications. All parallel programs within this set use MPI [6, 7]. One program is written in C (`rboltz`) while the others are primarily written in Fortran 77/90 with an odd C routine where convenient.
As can be seen, the number of programs in the benchmark is rather limited. Also no programs were chosen that would take an execution time that would be excessive. In fact, no program took more than $\approx 40$ minutes wallclock time when executed in parallel mode. The limitation in execution time and number of programs was dictated by the cluster configurations we had available for our tests (see section 5): except the Compaq cluster with 18 CPUs all clusters has 16 CPUs. This did not allow for tests on a more elaborate scale.

As we are not only interested in the individual performance of each program but also want to know how the clusters behaved in a multi-user environment also a throughput test was devised. For this part the EuroBen Benchmark Framework [12] was used. Programs or program sets can be inserted in the framework which enables the timing and correctness checking of the individual programs but also their submission by means of a batch system in a prescribed order. In this way a workload for the system can be defined. Executing this workload yields, apart from the total time for this workload, also the waiting and execution times of each of the programs. From this the ability of the system to accommodate a sizeable workload and the mutual influence of the programs on each other's execution times can be assessed. In addition, the framework structure allows to do multiple experiments that are logged separately from which the variability across these experiments can be determined. The EuroBen Framework also contains an interactive benchmark part. In this study, however, we did not consider this part because of its limited relevance for our direct purposes.

## 4.1 Programs in more detail

The parallel programs below, except program `primmmod`, could be run on an arbitrary number of nodes. This was very informative in view of the scalability for the various networks of the clusters we encountered (Fast Ethernet and Myrinet). However, for the throughput benchmark we have fixed the number of processors for each program involved in order to limit the amount of information generated. The number of processors for the programs was:

| Program | No. of CPUs |
|---|---|
| adf | 16 |
| Euroben mod1h | 4, 6, 8 |
| Euroben mod1i | 1–8 |
| Euroben mod1j | 2 |
| maya5 | 6 |
| primmod | 8 |
| rboltz | 4 |
| vasp | 8 |

### 4.1.1 EuroBen

The synthetic programs in EuroBen set are for the main part meant to determine single CPU performance. The set is organised in modules of increasing complexity: module 1 is concerned with the performance of important basic computational kernels like the dotproduct, the `axpy` operation, flat rotation, $2^{nd}$ difference operator, etc., as a function of vector length. Furthermore, the speed and accuracy of the numerical intrinsic functions are tested.
The second module considers basic numerical algorithms that use the operators from module 1. These include matrix-vector multiply, solution of full and sparse linear systems and of full and sparse eigenvalue problems, FFTs, etc. In the $3^{rd}$ module skeleton applications like ODE and PDE solvers are tested that in turn use the algorithms from module 2. This enables us to obtain a systematic view of the performance for various problem areas.
In addition, we added three programs from the distributed memory version of EuroBen that yield information about bandwidth and latency of the network for important communication patterns. One of these programs considers different implementations of a distributed dotproduct that gives an impression of the quality of the MPI implementation of the collective communication functions. These three simple programs all belong to the module 1 level. The first, `mod1h`, was tested on 4, 6, and 8 processors, respectively because of the nearest-neighbour communication patterns to be tested. The second program, `mod1i`, performs 3 implementations of a distributed dotproduct on 1–8 processors. This is particularly interesting when the system has more than 1 CPU per node. In this case, the internal node bandwidth may play a significant role because all processors need to access memory or the L2 cache at the same time. The third program, `mod1j` is a simple ping-pong experiment with very high resolution to determine the latency for point-to-point communication. This program is executed on only 2 processors.

### 4.1.2 `adf`

`adf`, (Amsterdam Density Functional) is a quantum chemistry program for calculations on polyatomic systems. It can be used for the study of such diverse fields as molecular spectroscopy, organic and inorganic chemistry, crystallography, and pharmaco-chemistry.
The underlying theory is the Kohn-Sham approach to the Density Functional Theory (DFT). This implies a one-electron picture of the many-electron systems but yields in principle exact properties such as the electron density (and related properties) and the total energy.
In the current test the total energy of an $H_2O$ complex is computed.
`adf` is written in Fortran 77/90 and uses MPI.

### 4.1.3 `maya5`

Maya5 is an astronomy code that simulates the evolution of pulsars. A large number of pulsars is generated within a range of parameter values, like magnetic field strength, luminosity, distance, etc. Then they are followed during their evolution to assess whether they are likely candidates for detection and how their properties evolve over time. The Maya5 code is almost ideally parallel as the generation and following the evolution of the individual pulsars are virtually independent. Maya5 is written in Fortran 77/90 and uses MPI for communication.

### 4.1.4 primmod

primmod is a program for a Molecular Dynamics simulation of a charged colloidal system for 80 macroparticles and 4800 counterions. It prints out thermodynamic variables and a final configuration for this test case. In contrast to the other programs, primmod is hard-coded for 8 processors, although it is possible to modify the code to run on 2, 4, or 16 processors. Because of the cumbersome procedure to implement this in the tests, this program is only run with 8 processors in the current benchmark. primmod is a Fortran 77 program with MPI for communication.

### 4.1.5 rboltz

The program rboltz is a Lattice Boltzmann particle simulation Molecular Dynamics code written in C and uses MPI for communication. Only a limited number of time steps are done, which makes the execution time quite short (order of seconds wallclock time).

### 4.1.6 vasp

vasp is a Self Consistent Field Quantum Chemistry code and as such iteratively searches for the lowest eigenvalues/vectors of a matrix (using conjugate gradient minimisation procedure in the example). The matrix depends on the eigenvectors, and thus enters into a self-consistency cycle. Main time is spent in matrix-matrix, matrix-vector, and FFT operations.
vasp is mainly written in Fortran 77/90 with some additional C routines and MPI for communication. The C preprocessor is mandatory for building the correct executable for various computing platforms.

### 4.1.7 The throughput test

Because we want to get an impression of how the clusters would behave in a multi-user environment we put together a workload consisting of the programs described above. Multiple instances of these programs were to be submitted according to the following schedule:

```
#
# Schedule for a Throughput Benchmark
#
# This file should NOT be changed!
#
# =======================================================
# Throw in enough jobs to please the system for
# a fair amount of time.
# =======================================================
0:      4 * rboltz_run
0:      2 * primmod_run
0:      1 * adf_run
10:     1 * vasp_run
10:     1 * euroben_run
100:    1 * maya5_run
300:    3 * rboltz_run
# =======================================================
# Now throw in even more. Up to now the system shouldn't
# have idled at all. These jobs (together with the
# previous batch) should keep the system running for
# +/-2 hours (on 16 processors).
# =======================================================
900:    2 * euroben_run
900:    2 * maya5_run
900:    1 * vasp_run
900:    4 * primmod_run
```

```
# ======================================================
# Now we submit some smaller jobs every now and then which
# should be finished within a reasonable time.
# ======================================================
1800:   1 * euroben_run
1800:   4 * rboltz_run
# ======================================================
2400:   1 * euroben_run
# ======================================================
2700:   2 * primmod_run
2700:   3 * rboltz_run
# ======================================================
```

The utility functions in the EuroBen framework [12] cause the programs in the `schedule` file to be scheduled with the multiplicity given in this file, e.g., program `primmod` is submitted 4 times 900 seconds after the start of the throughput benchmark. In this way a multi-user workload can be simulated and the effect on the cluster can be assessed. The wallclock execution times range from roughly 10–$10^4$, while amount of processors per job ranges from 4–16. So, it may be desirable define appropriate queues to process the submitted jobs efficiently. This has, however, not been prescribed because each batch system has its own allocation policies that are not very well comparable. Therefore, we have chosen the queue definition to be at the benchmarker's discretion and just to report what queue definitions were used.

# 5 Summary of testing circumstances

As clusters and their software are even more transient than the integrated HPC systems with their software, it is important to be clear about the exact configurations used together with the software that has been used during the tests. Because the entire cluster field is developing at such a fast rate the results on a different hardware/software configuration can be more or less different from those we report here. In Table 5.1 we display the relevant characteristics of the systems used in this test.

In Table 5.1 we included both the software which is directly relevant for the cluster performance and also the cluster management software for as far as this was present.

As can be seen, for the Compaq cluster no vendor-provided software was used. Instead the cluster is managed with software that has been developed by the team that operates the cluster at the University of Groningen, The Netherlands. The motivation was, that the Compaq product had some features that were not of interest to the cluster managers while the development of scripts for the most frequently used management functions, like installing/updating new software and taking nodes off/on line was relatively easy. They regarded the Compaq Cluster Management software package CMU as "overkill" and consequently use their home grown software. In addition, no serial console server was needed which cut the costs.

Also on the IBM Netfinity cluster no proprietary cluster management software was used. As the experiments were performed remotely on a cluster at the IBM site in Poughkeepsie, it was not possible to get more complete information on what public domain software actually is in use but for (re)installing software some simple `rsh`-based tools are employed and monitoring the load of the system is done by the `top` tool.

Table 5.1: *Hardware and software characteristics of the clusters used in this test.*

| **Hardware** | **Compaq** | **IBM** | **SGI** |
|---|---|---|---|
| Name | XP1000 cl. | Netfinity cl. | SGI L1200 cl. |
| No. of Nodes | 18 | 8 | 8 |
| No. of CPUs | 18 | 16 | 16 |
| CPU Type | 667 MHz Alpha EV67 | 600 MHz PIII | 700 MHz PIII |
| Theor. Peak | 1.3 Gflop/s | 600 Mflops/s | 700 Mflop/s |
| Memory | 256 MB/node | 256 MB/node | 1 GB/node |
| Network type | Fast Ethernet | Myrinet | Myrinet |
| Peak BW | 12.5 MB/s | 160 MB/s | 160 MB/s* |
| **Software** | | | |
| OS | Red HatLinux 6.2 | Red HatLinux 6.2 | Red HatLinux 6.2 |
| | — | Kernel 2.2.16-3 | Kernel 2.2.16-3 |
| Batch system | PBS 2.2 | LSF 4.0.1 | PBS 2.2 |
| Fortran | Compaq Fortran 1.0 | PG F77/90 3.1-3† | PG F77/90 3.1-3† |
| Compiler flags | -O3 | -O3 | -O2 |
| C | Compaq C 1.06.2.9.002 | egcs-2.91.66 | PG C++ 3.1-3† |
| Compiler flags | -O3 | -O3 | -O2 |
| MPI | MPICH 1.2.0 | MPICH 1.2.0/..3 | MPICH 1.2.0/..3 |
| Cluster management | Home grown | Home grown | ACE 1.3 |
| Monitoring | `top` | `top` | PCP (in ACE) |

* In addition also tests with Fast Ethernet were performed.
† PG = Portland Group.

# 6 Results of benchmark tests

In this section we present the test results of the benchmark that was run on each of the clusters detailed in the section 5. We comment on the performance, on porting problems, and other (few) problems encountered as they make up the way an average user is to experience the cluster as his/her computing platform. Although we will report the results of each of the clusters separately, at the end of this section we also will give tables that should make it easier to compare the performance of each the clusters included. One should keep in mind, however, that each cluster has his peculiarities, be it with respect to the CPUs employed, or the difference in the network characteristics. Therefore, it will not be straightforward to compare the outcomes. Still, it might lead one to conclusions with respect to the preferred CPU configuration and/or the network.

For some vendors (Compaq and SGI) results are available from integrated parallel systems they produce. For Compaq this is the AlphaServer SC while for SGI an Origin2000 was used to perform comparisons of the same benchmark with the clusters. The extra information may help in getting straight what type of system should be preferred (if one has the choice).

## 6.1 Compaq test results

The benchmark for the Compaq cluster was done on an 18-node XP1000 cluster with 100 Mb/s Ethernet as the connecting medium. This was the only configuration were there was one CPU per computational node. MPICH was the MPI library used with `p4` as the underlying communication mechanism.

### 6.1.1 Porting considerations

The compiler to be used on the Compaq cluster was `mpif77` which also accepted virtually all Fortran 90 constructs. Surprisingly, `mpif90` could not very well be used because of the restrictions it posed on the source format: it had to be strictly Fortran 90, free format to be accepted. However, `mpif77` worked quite well with all codes in the benchmark and did not generate source related problems.
The default optimisation level used was `-O3`. Although this not gives the highest possible speeds in all cases, it gives reasonable optimisation. In some cases the highest level, `-O5` was tried. The results with respect to speed are often inconsistent: sometimes a somewhat higher performance can be attained, sometimes, however, the performance drops because the code transformations performed (mainly loop blocking) have an adverse effect. Therefore we ultimately chose for the `-O3` optimisation level which gave more consistent results and gave no significant performance loss over the totality of the codes.
On the Compaq XP1000s the timing routine underlying the `MPI_Wtime` function turns out to have a quite course resolution: in the order of milliseconds. This lead to problems in the EuroBen program `mod1h` which attempts to determine the bandwidth of the network. Timings appeared to be 0.0 seconds where this evidently was not the case, disturbing the statistics. To overcome this problem the Compaq specific timing routines `start_timer(itime)` and `stop_timer(itime)` were used, with a resolution of a few $\mu$s. This was also modified in programs `mod1i` and `mod1j` that need a similar resolution of the measured times.

The program `maya5` encountered problems during compilation with some variables that appeared to be doubly declared. On many systems this presents no problem although a warning message may be generated. The Compaq compiler, however, flags it as a fatal error and does not produce an executable. By deleting the double declarations, this problem was easily resolved.
Another problem turned up during communication. In one of the routines the data type `MPI_Char`(C style) was used instead of `MPI_Character` (Fortran style). Although this had gone unnoticed and produced correct results on an SGI Origin2000, this is obviously incorrect and caused problems on the Compaq cluster. A sim-

ilar problem occurred in a routine `mpi_module.f90` where `MPI_Real` and `MPI_Double_Precision` datatypes were unduly mixed. After restoring these inconsistencies the program worked correctly.

In program `vasp` no fatal errors were produced when compiling the code. However, it was necessary to use the `Makefile.dec` versions of the `Makefile` collection that comes with the program to obtain a properly functioning executable. This holds for both the libraries that are generated and the main `vasp` routines proper. In addition the `-i8` in generating the libraries should not used as it introduces incompatible code with respect to the rest of the `vasp` routines, nor should the `-fast` compiler flag not be used. It appears to cause over-optimisation, resulting in incorrect outcomes. The compiler flag `-O2` results in correct code.

Program `rboltz` at first did not seem to give the correct output. However, some values in the test output are the result of the summing $\mathcal{O}(10^4)$ numbers with absolute values of about $10^{-3}$. In 64-bit IEEE arithmetic this turns out to produce deviations in the answers due to rounding errors. The answers turned out to be correct but not identical to the reference values in the check script.

### 6.1.2 Single-user mode results

As remarked before, the benchmark on the cluster of 18 XP1000s the same benchmark was also performed on a Compaq AlphaServer SC. In this system the CPUs are exactly the same as those in the XP1000 (Alpha EV67, 667 MHz) but the packaging is different: four CPUs are housed in an ES40 server which are connected to a shared memory by means of a crossbar. The ES40 nodes are connected to make up an AlphaServer SC by Qnet, a product Quadrics Supercomputing World that can attain a theoretical bandwidth of 267 MB/s for the ES40s (the limiting factor here the dual 64-bit, 33 MHz PCI bus of the nodes, with a 66 MHz PCI bus the theoretical speed increases to 360 MB/s).

In the section 6.4 we will present the relevant results obtained with the AlphaServer SC with those of the XP1000 cluster and of the other system involved.

#### 6.1.2.1 The synthetic programs: Single CPU performance

We will first discuss the single CPU speeds as found from the first module of the EuroBen set. In program `mod1ac` 14 important basic operations are measured, together with variants that implement regular strided access and indirect addressing, 31 kernels in all. It turned out that the strided access and direct addressing variants gave very similar results to that of the contiguously stored operand versions. Therefore we only display the results of the 14 basic kernels in Table 6.1. In this table $r_{\max}$ stand for the maximum observed computation rate for the operations considered.

From Table 6.1 it is clear that the speed of the processor is completely data-bound: only 1 8-byte operand per cycle can be shipped to or from the registers. This results in a an $r_{\max}$ for the triadic operations addition, subtraction, and multiplication of only roughly $\frac{1}{3}$ of the theoretical peak performance as only one floating point operation is done per 3 operand accesses, i.e., the computational intensity $f = \frac{1}{3}$ (see [4] for the definition of $f$). Using the `-O5` optimisation level the dyadic operations can be accelerated by roughly 10% to a speed of about 275 Mflop/s. Using `-O5` on the dotproduct results in the speed loss of almost 50% to 331 Mflop/s.
In kernel 10, representing a flat rotation, 4 loads and a store must be done. This slightly less than 5 cycles per element because some overlap is possible in the computation. The computational intensity $f = \frac{3}{5}$ with an ideal performance of 800 Mflop/s. Because of the bandwidth bottleneck 426 Mflop/s can be realised.
In kernel 14, the evaluation of a $9^{\text{th}}$ degree polynomial, $f = 9$ and, as such, should show a speed close to the theoretical peak performance. However, 17 cycles per element are required here. It reduces $r_{\max}$ to slightly more than half the theoretical peak speed: 701 Mflop/s. This is because an unnecessary copy of the result into the target operand is done. When using `-O5` here close to optimal speed can be reached: 1227 Mflop/s, which shows that under very favourable circumstances the two floating point unit can indeed be kept busy resulting in only 9.72 cycles/operation/element.
The L1 cache latencies range from 13–50 cycles. This means that for instance for a vector multiplication the vector length where half of $r_{\max}$ can be attained is 5–6 elements. It becomes negligible for vector lengths of a few tens or hundreds of elements. However, at a length $> 200$ the speed begins already to degrade because references out of the primary cache begin to occur.

Table 6.1: *Compaq: $r_{\max}$, the number of cycles per operation per element, and the latency for the primary cache operations on a single CPU. Only results of the first 14 kernels are shown. The operations all have unit stride access. The operation latency for the secondary cache is completely hidden by the data access.*

| | **Operation** | L1 cache $r_{\max}$ Mflop/s | L1 cache Cycles per op/element | L1 cache Latency cycles | L2 cache $r_{\max}$ Mflop/s |
|---|---|---|---|---|---|
| 1 | Broadcast | 523 | 1.25 | 24 | 192 |
| 2 | Copy | 254 | 2.00 | 16 | 106 |
| 3 | Addition | 246 | 2.66 | 18 | 86 |
| 4 | Subtraction | 240 | 2.64 | 18 | 86 |
| 5 | Multiplication | 246 | 2.69 | 14 | 86 |
| 6 | Division | 55 | 12.00 | 14 | 55 |
| 7 | Dotproduct | 620 | 2.13 | 16 | 398 |
| 8 | $x = x + \alpha y$ | 513 | 2.55 | 17 | 193 |
| 9 | $z = x + \alpha y$ | 470 | 2.76 | 21 | 173 |
| 10 | $y = x_1 x_2 + x_3 x_4$ | 426 | 4.56 | 23 | 173 |
| 11 | 1st order recurs. | 110 | 5.98 | 13 | 96 |
| 12 | 2nd order recurs. | 136 | 9.75 | 15 | 130 |
| 13 | 2nd difference | 611 | 3.17 | 27 | 315 |
| 14 | 9th Degr. Polynominal | 701 | 17.01 | 50 | 699 |

The speeds for the basic algorithms in module 2 of the EuroBen set are consistent with those found for the basic operations in module 1. Program `mod2a`, a program that compares dotproduct and `axpy` based matrix-vector multiplication shows an $r_{\max}$ of 406.5 Mflop/s for the dotproduct based multiply algorithm. This is well below the 620 Mflop/s observed for the dotproduct in module 1, but the matrix-vector multiplication the matrix elements are accessed with large strides and the elements are not necessarily all in the primary cache when needed. On the other hand, the doubly nested loop yields a opportunity for improved operation scheduling and for some systems gives better performance than the single loops in program `mod1ac` (see [2] and the results in section 6.3). The `axpy` based multiply algorithm attains even only a third of the regular `axpy` operation in `mod1ac`, kernel 8: 186 against 513 Mflop/s. It is evident that more aggressive compiler optimisation is here in order. When using the `-O5` optimisation level here the situation is reversed: the `axpy` based multiply algorithm is now faster and reaches an $r_{\max}$ of 471.6 Mflop/s, 85% of the performance level of the single-loop `axpy`.
The performance of program `mod2b`, the solution of a full linear system using LAPACK routines, shows an $r_{\max}$ which is 73% of that of the `axpy` kernel where the majority of operations are done in a matrix-matrix multiply that is implemented around it. Because also some back substitution has to be done in the solution phase, which is essentially a first order recursion that runs at most at a speed of 110 Mflop/s, this is more or less what can be expected when no improved operation scheduling occurs in the matrix-matrix multiply.
The 1-D FFT in program `mod2f` does relatively well, with a speed ranging from 570–428 Mflop/s for FFTs of size 256–2048. As the radix-4 FFT algorithm used can be seen as a mixture of flat rotation operations and `axpy` operations, the observed speed is very well in line with the speeds of the separate basic operations. For larger problem sizes the speed drop first to about half of that found for the primary cache ($\approx$ 200 Mflop/s) and for problem sizes that address the main memory it decreases to about 65 Mflop/s, $\frac{1}{8}$ of the speed from the L1 cache.
In program `mod2g`, a 2-D Haar wavelet transform, the computational intensity $f = \frac{1}{3}$ on average. In addition, 4 additions/subtractions are done versus 1 multiplication in one analysis/synthesis pass. Therefore the performance may not be expected to be more than $\approx$ 220 Mflop/s at most. We actually find an $r_{\max}$ of 208 Mflop/s when operating from the primary cache. Working from the L2 cache the speed drops by a factor of 2 and from memory by a factor of 8.

The results of the module 3 programs will be discussed in the section that compares the performances of all systems.

Table 6.2: *Compaq: Maximum bandwidth for various important communication patterns as measured with program* `mod1h`.

| | Operation | 4 Processors MB/s | 6 Processors MB/s | 8 Processors MB/s |
|---|---|---|---|---|
| 1 | Pingpong | 12 | 12 | 12 |
| 2 | Pingpong, strided | 13 | 14 | 13 |
| 3 | Broadcast | 11 | 12 | 11 |
| 4 | Collect | 12 | 12 | 12 |
| 5 | Reduction | 22 | 22 | 21 |
| 6 | Transposition | 13 | 14 | 13 |
| 7 | Bisectional BW, Regular | 11 | 11 | 11 |
| 8 | Bisectional BW, Random | 11 | 11 | 11 |
| 9 | Nearest Neighbour 1-D | 10 | 8 | 10 |
| 10 | Nearest Neighbour 2-D | 10 | 9 | 10 |
| 11 | Nearest Neighbour 3-D | 10 | 13 | 10 |

#### 6.1.2.2 The synthetic programs: Communication performance

The programs `mod1h`, `mod1i`, and `mod1j` are used to measure respectively the maximum obtainable bandwidth for a variety of important communication patterns/functions, the performance of 3 implementations of a distributed dotproduct, and a high-resolution point-to-point ping-pong experiment to determine the communication latency. In table 6.2 we display the maximum bandwidths found for the various communication patterns. We ran `mod1h` on 4, 6, and 8 processors to see whether this would affect the bandwidth for any of the communication operations. The amount of processors turns out to have no effect at all as can be from Table 6.2.
In program `mod1i` three different implementations of a distributed dotproduct are executed. In this simple algorithm hardly any communication is involved. Only the partial sums from each processor should be communicated to a 'root processor and the global sum should then be broadcasted again to every processor. So, in the communication virtually only the communication latency should be involved. Furthermore, the first implementation is "naive" in the sense that all processors send their local sums to the root processor directly. This can cause contention at the root processor as all other processors try to reach it at the same time. Similarly, when the root processor has to distribute the global sum it the associated send requests will be posted virtually at the same time. The contention problems of this implementation will increase with an increasing number of processors. The second implementation uses a binary tree send and receive mechanism that avoids the possible contention of the first implementation. The third implementation uses the MPI routines `MPI_Reduce` and `MPI_Bcast`. Ideally, these routines also should make use of tree algorithms between the processors. However, practice has made clear that the collective MPI operations are not always implemented optimally. In Figure 6.1 the results for the three dotproduct implementations are displayed for 1–8 processors. Figure 6.1 shows that there is very little difference between the three implementations. This is due to the large latency of the fast Ethernet network: program `mod1j` finds a latency of almost 105 $\mu$s. In this time over 1000 data items can be sent/received as is shown by the results from program `mod1h` is Table 6.2. Whatever contention may occur during the transfer of the partial and global sums in the dotproduct routines, this is completely drowned in the large latencies of the respective messages.

#### 6.1.2.3 User programs

Apart from the synthetic programs, five user programs were included in the benchmark as described in section 4. In single-user mode each of those were executed on 1–8 processors to assess their scalability on the fast Ethernet connected cluster. The results for all programs are summarised in Table 6.3.
Several remarks are in order here. The first one is that program `maya5` is parallelised using a master-slave approach in which processor 0 only schedules the work to be done, unless the program is executed serially. Therefore is the wallclock time on one processor more or less identical to that on two processors and when

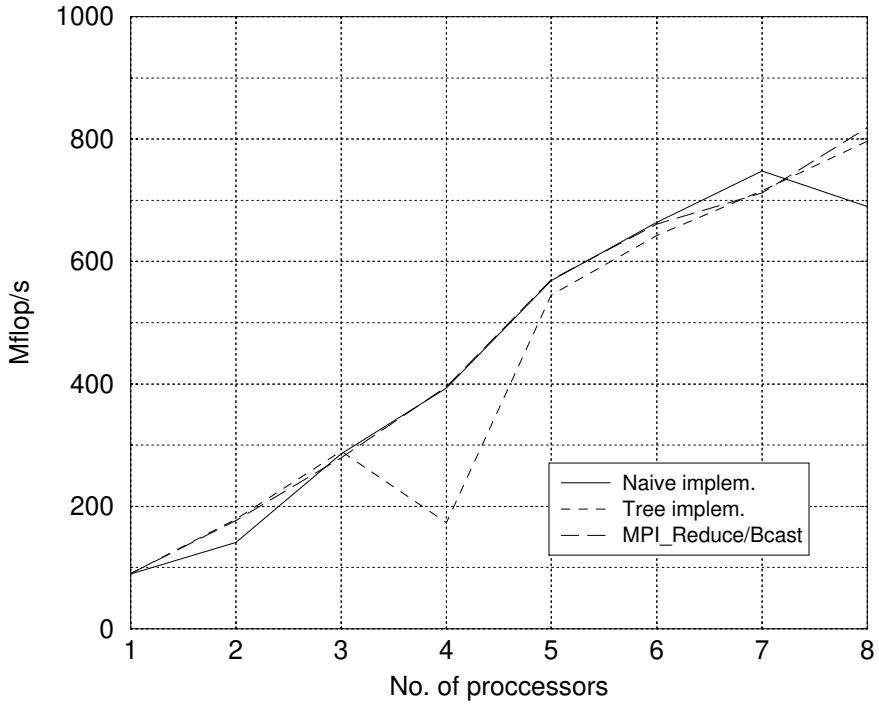



Figure 6.1: *Compaq: Performances for three implementations of the distributed dotproduct using MPI.*

Table 6.3: *Compaq: Single-user wallclock times for the user programs in the benchmarks for 1–8 processors.*

| **Program →** **# of Proc.s ↓** | `adf` Seconds | `maya5` Seconds | `primmod` Seconds | `rboltz` Seconds | `vasp` Seconds |
|---|---|---|---|---|---|
| 1 | 2663 | 5054 | — | 20 | 3573 |
| 2 | 1419 | 5028 | — | 11 | 2951 |
| 3 | 977 | 2627 | — | 8 | 2692 |
| 4 | 767 | 2039 | — | 7 | 2160 |
| 5 | 626 | 1376 | — | 7 | 1714 |
| 6 | 552 | 1434 | — | 7 | 1461 |
| 7 | 476 | 1202 | — | 7 | 1328 |
| 8 | 453 | 1090 | 90.3 | 6 | 1243 |

the program in run on 3 processors the wallclock time is roughly half of that on 2 processors. So, when `maya5` is run on $p > 2$ processors, the ideal speedup would be $p - 1$. We see that on 8 processors the speedup is almost 5 instead of 7. This is caused by an unbalance in the amount of work done by the respective processors. Although each of the processors in first instance generates the same amount of pulsars, not all of them are of a sufficiently realistic nature to follow their evolution. This is the most time consuming part in the code and causes the difference in time per processor (28% on 8 processors).
Both `adf` and `maya5` scale reasonably well. Program `adf` shows a speedup of almost 6 on 8 processors. For both programs holds that the system time is only a few percent of the total elapsed time which indicates that the time spent in the system part of the communication is also not high. As the communication in both programs is altogether small this is consistent with the system times found.
As Table 6.3 shows, program `rboltz` does not scale well. The reason for this is the system time which is more than half of the total elapsed time on 8 processors. When looking at the user time spent in the processors

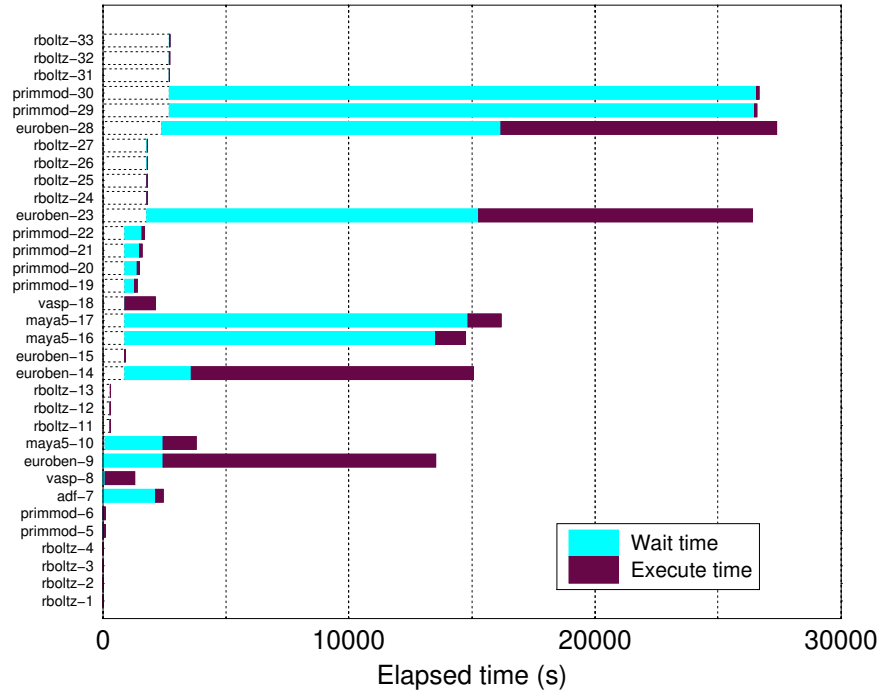



Figure 6.2: *Compaq: Activity graph of the throughput benchmark. The dotted white parts of the bars signify the times after which jobs are submitted*

one finds a total time of 12.85 s on 1 processor and of 2.64 s on 8 processors, a speedup of 4.9. Note that the system time is over 3 s on 8 processors. One can conclude that fast Ethernet should not be the network of choice for this program.
Also `vasp` does not scale very well on a higher number of processors. This is again due to the communication time of which the system time is a good indicator. On 1 processor the system time in only 14 s, 0.4% of the wallclock time. In contrast, on 8 processors the system time is 518 s, 42% of the wallclock time. It is clear that the communication overhead is very high and the figures suggest that the low bandwidth and high latency of fast Ethernet may be the culprit in the low scalability of `vasp`.
As mentioned in section 4, program `primmod` is only run on 8 processors and, consequently, we have no estimate of its scaling behaviour.

### 6.1.3 Throughput test results

The throughput test was performed as described in section 4.1.7. The queue configuration was extremely simple: one dedicated queue was configured and all programs were submitted to this queue according to the `schedule` file. PBS was expected to fit in all the jobs and to maximise the usage of the available processors. This turned out to work just fine: at any time at least 16 of the processors of the 18 available processors were occupied. The activity graph of the scheduled workload is given in figure 6.2. As can be seen from the Figure, the EuroBen jobs take the longest amount of time, about 11,000 seconds. This is almost entirely due to two programs: `mod1h` and `mod1j`. The latter program determines the latency of point-to-point communication and takes 7000 seconds by itself. Figure 6.2 also shows that job 15, an EuroBen job, is never executed. The logfile reports it having been submitted, but it did not show up in the PBS queue log. This has a bearing on the total throughput time: as one EuroBen job would have been left over, the actual throughput time for all jobs should be increased by about 11,000 seconds to a total of approximately 38,500 seconds.
It is evident that the throughput time is completely determined by the EuroBen jobs. Assuming a factor of 4 of speedup when using Myrinet instead of Fast Ethernet (as observed in experiments on the SGI cluster, see

section 6.3), the total throughput time would decrease to about 14,000 seconds. On the other hand, Myrinet in combination with PBS has its problems as is also reported in section 6.3. So, it is not clear whether the same queue configuration would have been possible.

The individual times for the jobs were very much in line with the times for these jobs in single-user mode. The maximum deviation found between single-user times and the times in throughput mode was 14% indicating that the network interference between jobs was moderate to low which is not surprising considering the fact that generally only two jobs were active at any time. For a larger configuration with more jobs active at the same time, this may turn worse when using a single Ethernet loop.

In summary, it is clear that PBS seems to work well with Fast Ethernet, apart from the mysterious disappearance of one job but that Fast Ethernet is definitely a limiting factor when communication intensive jobs are part of the workload.

## 6.2 IBM test results

The tests were performed on an IBM Netfinity cluster with 8 dual CPU nodes. The processors were 600 MHz Pentium III Xeons and, as such, very similar to the ones used in the SGI L1200 cluster. The network was, like in the SGI cluster, Myrinet. On this system only the performance with Myrinet was considered, although also a Fast Ethernet network can be used as a low(er) cost alternative. With the Myrinet network of course also Myrinet's MPICH-GM MPI library was used.

### 6.2.1 Porting considerations

The compilers that were used in the tests were the Portland Group Fortran 90 compiler and the GNU C compiler for the C programs in the benchmark set. As is also evident from section 6.3.1, the Portland Group compiler has its own opinions on how to deal with I/O. However, all user programs could be built without much difficulty.
A further remarkable fact is that where the results of the synthetic benchmark programs do not scale with clock cycle of the otherwise identical processors, the IBM cluster is sometimes slower than could be expected, as the only other difference lies in the optimisation level: on the IBM system the Fortran optimisation flag `-O3` was used where on the SGI cluster the flag was `-O2`. As on the IBM cluster as well as on the SGI cluster the same Fortran compiler was used, the same corrections had to be made to the programs on both systems. These modifications are described fully in section 6.3.1.

In the case of `adf` not the MPI version, but the PVM version was used, because a license for the latter version was available at IBM's and not for the MPI version. Therefore the timings are not completely comparable to those of the other systems. The performance difference appeared to be significant, but because the ratio of communication over computation is small this must be attributed to differences in the computational part.

### 6.2.2 Single-user mode results

As in the Netfinity cluster used each node contains two CPUs, it is of interest to consider the performance of the same programs by using 1 CPU/node and 2 CPUs/node because there may be a noticeable node-internal bandwidth competition when two CPUs have to access the common node memory at the same time. For some programs we have indeed some results for both cases that will be discussed in section 6.2.3 and with the synthetic communication programs where the effects can be quite significant.

#### 6.2.2.1 The synthetic programs: Single CPU performance

As with the preceding system we first discuss the single CPU speeds yielded by the EuroBen set. The $r_{\max}$ values of the first 14 kernels of EuroBen program `mod1ac` are displayed in Table 6.4.

The behaviour of the CPUs on these simple kernels is virtually identical to those in the SGI cluster. Both systems used the same Fortran compiler and the same type of CPUs, be it that in the IBM system 600 MHz processors were employed instead of 700 MHz processors as in the SGI cluster. The speed differences

Table 6.4: *IBM: $r_{\max}$, the number of cycles per operation per element, and the latency for the primary cache operations on a single CPU. Only results of the first 14 kernels are shown. The operations all have unit stride access. The operation latency for the secondary cache is completely hidden by the data access.*

| | **Operation** | L1 cache $r_{\max}$ Mflop/s | L1 cache Cycles per op/element | L1 cache Latency cycles | L2 cache $r_{\max}$ Mflop/s |
|---|---|---|---|---|---|
| 1 | Broadcast | 326 | 1.76 | 57 | 95 |
| 2 | Copy | 286 | 2.00 | 52 | 139 |
| 3 | Addition | 170 | 3.35 | 56 | 37 |
| 4 | Subtraction | 168 | 3.40 | 59 | 37 |
| 5 | Multiplication | 157 | 3.58 | 61 | 37 |
| 6 | Division | 72 | 8.20 | 53 | 37 |
| 7 | Dotproduct | 243 | 4.80 | 58 | 151 |
| 8 | $x = x + \alpha y$ | 273 | 4.34 | 32 | 148 |
| 9 | $z = x + \alpha y$ | 257 | 4.56 | 37 | 75 |
| 10 | $y = x_1 x_2 + x_3 x_4$ | 180 | 9.56 | 43 | 77 |
| 11 | 1st order recurs. | 72 | 8.25 | 31 | 65 |
| 12 | 2nd order recurs. | 105 | 11.30 | 54 | 105 |
| 13 | 2nd difference | 321 | 5.49 | 64 | 210 |
| 14 | 9th Degr. Polynominal | 227 | 47.47 | 22 | 206 |

are therefore within 2.5% when scaled with the clock cycle. It also makes it plausible that for floating-point dominated programs the same scaling factor should be found. As expected, the latencies in cycles and the number of cycles required per operation per element are virtually identical.
Exactly as for the SGI cluster it is clear the Intel processors are far from fulfilling their potential: the best relative performance is found for the second difference operation with a computational intensity of $f = 1$ and a theoretical minimum of 4 cycles/operation/element. Of this 72% can be realised resulting in 53% of the theoretical peak speed. For kernel 14 that should perform very near the Theoretical Peak Performance because of its computational intensity $f = 9$, only reaches 38% of this speed. Further remarks about the kernel performances are made in section 6.3.2.1.

In program `mod2a` two implementations of a matrix-vector multiplication are performed to assess their relative and absolute performance and to relate them to the basic dotproduct and `axpy` operations on which they based. The best performances found for both implementations is consistent with the $r_{\max}$ values for both operations in Table 6.4 in that both are somewhat more than 50% slower in the context of the matrix-vector multiply (MxV) than when implemented as single loops: 156 vs. 243 and 180 vs 273 Mflop/s, respectively. As there is potentially more room for optimisation in the MxV context, this performance is rather disappointing but not unique. Also on the Compaq and SGI clusters the MxV operation was performing worse than the single loop operations they were based on. This is in contrast to what can be achieved on most RISC processors, e.g., on an Origin2000 the `axpy` in an MxV context is more than twice as fast than in the single loop version while the dotproduct is about 15% faster even with the inevitable cache misses that occur for the dotproduct in the MxV context. So, it is clear that compiler-processor combination cannot (yet) realise a fair part of the computational potential of the processor.
This is certainly also true for program `mod2b`, the solution of a full linear system. Here an $r_{\max}$ of only 77 Mflop/s second is found. This is less than 30% of the speed of the `axpy` operation on which it is based. It is also much less than what would be expected as compared with the results for the same program on the SGI cluster. Based on the difference in clock cycle an $r_{\max}$ of roughly 180 Mflop/s would be expected, more than 2 times than what is actually observed. The only difference, apart from the clock cycle, is the compiler optimisation level which was `-O3` on the IBM system and `-O2` on the SGI system, respectively. This might indicate that not all optimisations done are beneficial. Interestingly, in program `mod2d` the evaluation of a dense symmetric eigenvalue problem, the speed difference between the IBM and SGI systems is again largely attributable to the difference in clock cycle, the $r_{\max}$ value found is 140 Mflop/s. This agrees also well with

Table 6.5: *IBM: Maximum bandwidth for various important communication patterns as measured with program* `mod1h` *using Myrinet.*

| | Operation | 4 Processors MB/s | 6 Processors MB/s | 8 Processors MB/s |
|---|---|---|---|---|
| 1 | Pingpong | 95 | 94 | 95 |
| 2 | Pingpong, strided | 55 | 55 | 55 |
| 3 | Broadcast | 105 | 106 | 101 |
| 4 | Collect | 67 | 56 | 45 |
| 5 | Reduction | 86 | 97 | 97 |
| 6 | Transposition | 72 | 54 | 50 |
| 7 | Bisectional BW, Regular | 48 | 47 | 47 |
| 8 | Bisectional BW, Random | 48 | 48 | 47 |
| 9 | Nearest Neighbour 1-D | 62 | 60 | 60 |
| 10 | Nearest Neighbour 2-D | 57 | 55 | 52 |
| 11 | Nearest Neighbour 3-D | 68 | 65 | 42 |

the two main constituting basic operations, the `axpy` and the flat rotation, with respective speeds of 273 and 180 Mflop/s allowing for the non-contiguous addressing and the small vectors on which is operated in a large fraction of the cases.
Program `mod2f`, a 1-D FFT behaves exactly as its counterpart on the SGI cluster. A maximum speed of 183 Mflop/s is found, and where `axpy` and the flat rotation operations again play a major role, the speed that is found is consistent with those of the single-loop operations. This cannot be said from the 2-D Haar wavelet transform in program `mod2g`. A maximum speed of only 62.6 Mflop/s is observed. Based on the computational intensity $f = \frac{1}{3}$ and the fact that the floating point operations cannot be overlapped, the theoretical maximum performance would be about 101 Mflop/s. Even allowing for data accesses with strides at powers of 2, the actual performance is definitely lower than might be expected.

The programs of module 3, are discussed in section 6.4 comparing them with those of the other systems.

#### 6.2.2.2 The synthetic programs: Communication performance

The results of the programs `mod1h`, `mod1i`, and `mod1j` on the IBM cluster are only available for Myrinet. The measurements from `mod1h` for this network is given in Table 6.5. In the case of this IBM cluster only the bandwidth is measured for the configuration of 2 CPUs/node. On the SGI cluster also the case of using 1 CPU/node is considered.
Surprisingly, there are significant differences between the observed bandwidths between the IBM and SGI cluster, although they both use the same Myrinet network hardware and the same software level. There might have been a difference in configuration, but this could not be confirmed. On the IBM system the Broadcast operation was 11–16% faster than that on the SGI system. Likewise, the sum reduction was 18–34% faster on the IBM system. For the other operations the IBM cluster gave virtually the same speeds as found on the SGI L1200 cluster with 2 CPUs/node showing that the performance here is not determined by the processor speed but rather by the network hard and software.
Just as will be seen on the SGI cluster, the bandwidths observed are significantly less than the theoretical bandwidth that is advertised by Myricom. This may be partly due the polling mechanism used but there is certainly room for improvement in this respect. This is especially true for the Collect operation that only realises just over 40% of the theoretical peak. The Reduction operation that should behave similarly is much better in this respect, indicating that the implementation can be improved.

Also program `mod1i` was performed with two CPUs/node. As will become clear from the SGI results where both with 1 CPU/node and 2 CPUs/node was tested, the difference in performance is quite significant. Figure 6.3 shows the performance on 1–8 processors for the three distributed dotproduct implementations. One gets an impression from the node-internal bandwidth competition when considering the performance graph from 2–8 CPUs. The scaling is linear this case, from 74 Mflop/s on 2 CPUs to about 290 Mflop/s

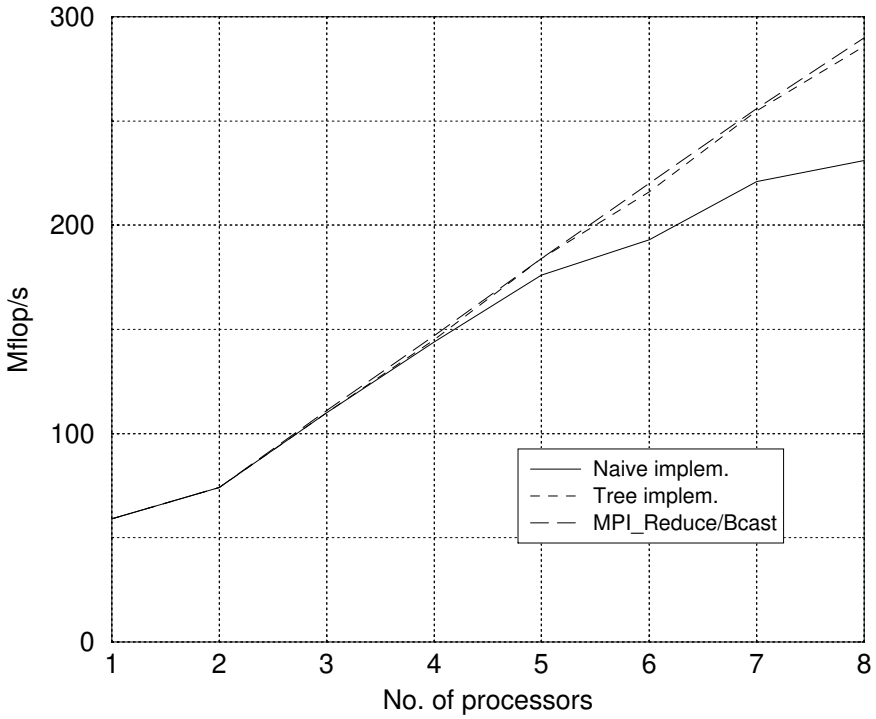



Figure 6.3: *IBM: Performances for three implementations of the distributed dotproduct using MPI.*

on 8 CPUs for the two best implementations. However, the performance on 1 processor is 59 Mflop/s while extrapolating down from 2 to 1 CPU the performance would expected to be 37 Mflop/s. As the experiments with using 1 CPU/node in the SGI cluster shows, this performance difference is *not* due to parallelisation overhead (which is insignificant) but to the fact that the CPUs have to compete for memory bandwidth, resulting in a 60% performance drop relative to having only 1 CPU accessing the node's memory. When using the naive implementation the performance clearly lags behind because all processor try to send their partial sums directly to the same root processor at the same time.

Program `mod1j` was used to assess the asymptotic bandwidth and the latency for point-to-point communication. In this program messages from 4–4,000,000 bytes are sent between processors. The highest bandwidth, $r_{\max} = 132.31$ MB/s was found at a message length of 80,000 bytes. For higher message lengths the performance degrades to about 90 MB/s because cache and buffer sizes are not sufficient to maintain this speed for the larger message sizes. The latency we find is extremely low: 2.74 $\mu$s with an error of 2%. Both with respect to the bandwidth and the latency, the network performs better than that of the SGI cluster although the hard and software is the same. The network configuration may have been tuned better on the IBM cluster but we could not get hard evidence on this point.

### 6.2.3 User programs

In Table 6.6 the results for the user programs in the benchmark are summarised. For the programs `maya5` and `vasp` both on a configuration with 1 CPU/node and with 2 CPUs/node was run. There is no performance difference for `maya5`. This is understandable as the program is almost embarrassingly parallel and the computation to memory access ratio is very large. A more surprising fact is that the absolute speed of `maya5` on the IBM system is much higher than on its SGI counterpart. The IBM system is more than 50% faster, notwithstanding the higher clock frequency of the SGI system. As yet we could not find an explanation for this behaviour.
Program `vasp` behaves as expected: the timings are in accordance with the difference in clock cycle between

Table 6.6: *IBM: Single-user wallclock times for the user programs in the benchmarks for 1–8 processors when using 1 CPU/node and 1–16 processors when using 2 CPUs/node. The network used is Myrinet.*

**1 CPU/node**

| **Program →** `adf` | `maya5` | `primmod` | `rboltz` | `vasp` |
|---|---|---|---|---|
| **# of Proc.s ↓** Seconds | Seconds | Seconds | Seconds | Seconds |
| 1 | — | — | — | — | 13485 |
| 2 | — | 5735 | — | — | — |
| 3 | — | — | — | — | — |
| 4 | — | 2144 | — | 8 | 2989 |
| 6 | — | — | — | — | — |
| 8 | — | 1082 | — | — | 1356 |

**2 CPUs/node**

| **Program →** `adf` | `maya5` | `primmod` | `rboltz` | `vasp` |
|---|---|---|---|---|
| **# of Proc.s ↓** Seconds | Seconds | Seconds | Seconds | Seconds |
| 1 | — | — | — | — | — |
| 2 | — | 5740 | — | — | 7744 |
| 3 | — | — | — | — | — |
| 4 | — | 2148 | — | 9 | 3511 |
| 6 | — | 1435 | — | — | — |
| 8 | — | 1080 | 185.0 | — | 1494 |
| 12 | — | — | — | — | — |
| 16 | 947 | 738 | — | — | 818 |

the SGI and IBM systems and also the speed differences between the 1 CPU/node and 2 CPUs/node configurations are similar. On 4 processors this speed difference is about 15% and on 8 processors about 10%. Programs `adf` and `primmod` have only been run on a 2 CPUs/node configuration. Program `primmod` behaves as expected, but `adf` takes over 50% longer on the IBM system. Unfortunately, it is hard to explain the reason, because on the latter system the PVM version of `adf` was used instead of the MPI version. It is hard to trace back the difference in timing to the differences in the communication. As the computation to communication ratio is quite large one would expect that there may be also slight differences in the computational part but this cannot be checked as only the binary version of the program is distributed.
Lastly, program `rboltz` is doing slightly worse on a 2 CPUs/node configuration just as on the SGI cluster. On the other hand, the absolute performance is better than on the SGI system. The difference here lies in the use of another C compiler: on the IBM cluster the GNU C compiler was used, while on the SGI system the Portland Group C compiler was used; evidently not to the advantage of the performance on the latter system.

### 6.2.4 Throughput test results

The activity graph of the throughput benchmark on the Netfinity cluster is displayed in Figure 6.4.

The graph shows that the jobs are very compactly end regularly scheduled by the bath system, in this case LSF Version 4.0.1. The total time for the entire workload was 9320 seconds, slightly more than 2.5 hours. This is much better than what was achieved on the SGI cluster which needed 4.3 hours to finish the workload. However, after inspection it turned out that in the EuroBen set several programs were not executed, while in the first instance of this job, `euroben9`, also the communication test programs were not performed. This resulted in an average execution time of the `euroben` jobs of about 1800 seconds while the same jobs on the SGI cluster took 2800 seconds. When 1000 seconds per `euroben` job instance is added to the total throughput time, this would become roughly 14,300 seconds. This is somewhat less than the time on the SGI cluster although the individual times of the programs are mostly shorter (except `maya5`). The better throughput time can for a large part be attributed to the better scheduling with LSF. Where there were problems with the combination of PBS 2.2 and Myrinet (see section 6.3.4), such problems are

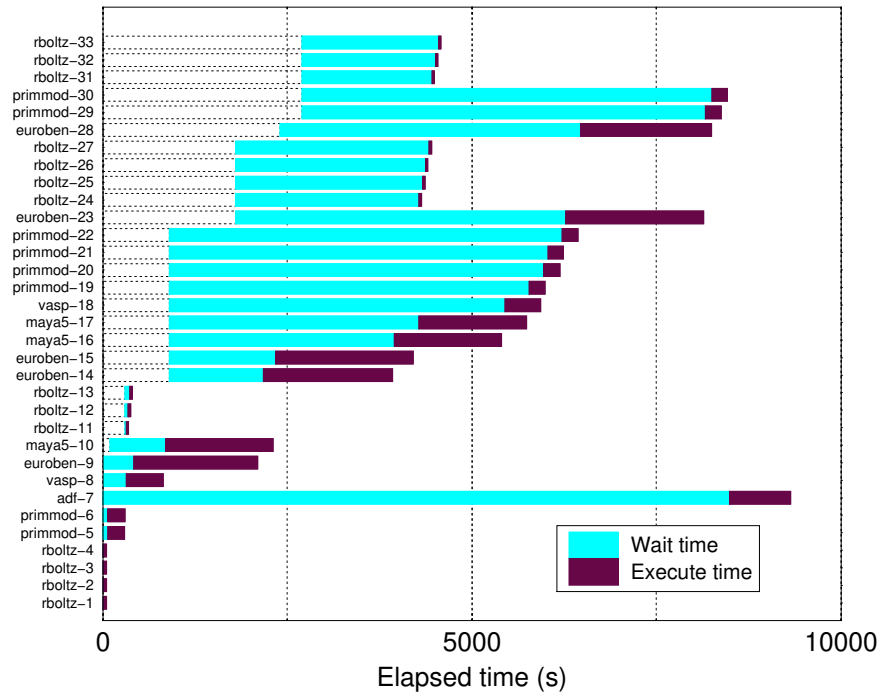



Figure 6.4: *IBM: Activity graph of the throughput benchmark. The dotted white parts of the bars signify the times after which jobs are submitted*

not present with LSF resulting in a more compact schedule.
When compared with the Compaq throughput results, it is clear that the poor results are due to the very long time the `euroben` job instances take on the this cluster. This is due to the fact that the latency of the Fast Ethernet network is more than an order of magnitude larger which slows down the communication in all programs in general and in the communication test programs `mod1h` and `mod1j` of EuroBen in particular.

## 6.3 SGI test results

SGI performed the tests on the SGI L1200 cluster as described in section 5 both with Fast Ethernet and Myrinet. MPICH was used as the MPI library. In the former case the codes were compiled with `-device=ch_p4` while with Myrinet the codes were compiled with `-device=ch_gm`.
In addition, the same benchmark was also done on an Origin2000 integrated parallel machine with 400 MHz MIPS R12000 processors. In this case SGI's own MPI version was used and an environment variable, `MPI_DSM_PPM 1`, was set to use only one CPU per processor board in this system. So, where relevant and available, the results are given for the O2000, the SGI L1200 cluster with Fast Ethernet, and the same cluster with Myrinet, respectively. The results for the Origin2000 are presented in section 6.4 together with those of the other configurations.

### 6.3.1 Porting considerations

On the SGI L1200 cluster we used the Portland Group Fortran 77/90 and C compiler set, version 3.1-3. No serious problems were encountered during the compilation of the programs apart from some bugs that either did not show up with other compilers until now, or did not present problems in the execute phase, or both.

An earlier version of EuroBen program `mod1i` was included in the benchmark which, incorrectly, uses the same parameter as a source and target buffer in an `MPI_Reduce` call. Although this works well on some

systems, it is formally forbidden and caused errors in this particular case. By renaming the target buffers the problem was solved.

In program `mod1j` the Makeframe already included the machine dependent parameters, like compiler name, optimisation level, and library names which must be set via the master Makefile in the root directory of the benchmark. This results in inconsistent definitions in the Makefile. By removing the superfluous definitions from the Makeframe the problem was solved.

For some reason the Portland Group Fortran 90 compiler does not allow scratch files to have a `File` parameter. Consequently, lines like

```
   Open ( lua, File = 'jara', Form = 'UNFORMATTED',
  &       Status = 'SCRATCH' )
```

had to be modified to:

```
   Open ( lua,  Form = 'UNFORMATTED', Status = 'SCRATCH' )
```

Such modifications were necessary in programs `mod3a` and `mod3b` which measure I/O speed. Furthermore, in the latter program the problems size was set such that a 512×512 2-D FFT was performed instead of using the alternative problem size of 8192×8192.

The `run` script of the EuroBen set was modified such that the results of runs for program `mod1h` were accumulated for 4, 6, and 8 processors instead of overwriting the results of former runs. A similar modification was made for program `mod1i` that was run on 1–8 processors.

In program `maya5` a modification of the Makeframe was necessary: a dependency on `*.mod` files had to be removed. As there is no standard for the generation of such files and how these should be named, if present, this is likely to stay a problematic point in making system independent Makefiles.
The Portland Group compiler did not allow a literal `Character` filename to be used in a subroutine call. Hence, such a `Character` literal had first to be assigned to a variable and the call had to be made with this variable as a parameter.
The input file `maya5.in` had to be modified to make the output machine name reflect the actual name of the machine used.
In addition, an optimisation was made in a search routine, `dmdsm.f`. This optimisation can significantly cut the amount of work be done in finding a global minimum. Consequently, the execution time of `maya5` cannot be compared readily with those on the other systems.

With program `primmod` the input file `sp.dat` incorrectly states that the run to be done is a restart run, which causes the program to crash because intermediate result files cannot be found. This was corrected by specifying that the run was not a restart run.

For program `rboltz` it was necessary to specify the correct libraries for C compiler `pgcc` to produce the executable.

For program `vasp` the Linux version of the Makefile was used. Furthermore, the `mpif.h` include file had to be copied to the `vasp.4.4/` directory and processed by running "`convert mpif.h`".
In the `vasp.4.lib/` directory the parameter `DUMMY` in file `diolib.F` had to be set to `.TRUE.` to avoid an "invalid unit number" error to be generated by `pgf90` in three places.

### 6.3.2 Single-user mode results

Most synthetic single-user programs are not meant to measure the influence of having more CPUs on a board. Still, the impact can be quite noticeable because of the competition for the limited bandwidth between CPUs and node memory. For the programs for which we have these results available we present them along with those where only one CPU per board is used.

#### 6.3.2.1 The synthetic programs: Single CPU performance

As with the preceding systems we first discuss the single CPU speeds yielded by the EuroBen set. The $r_{\max}$ values of the first 14 kernels of EuroBen program `mod1ac` are displayed in Table 6.7.

Table 6.7: *SGI: $r_{\max}$, the number of cycles per operation per element, and the latency for the primary cache operations on a single CPU. Only results of the first 14 kernels are shown. The operations all have unit stride access. The operation latency for the secondary cache is completely hidden by the data access.*

| | **Operation** | L1 cache $r_{\max}$ Mflop/s | L1 cache Cycles per op/element | L1 cache Latency cycles | L2 cache $r_{\max}$ Mflop/s |
|---|---|---|---|---|---|
| 1 | Broadcast | 388 | 1.75 | 57 | 219 |
| 2 | Copy | 325 | 1.98 | 56 | 159 |
| 3 | Addition | 197 | 3.30 | 58 | 29 |
| 4 | Subtraction | 197 | 3.32 | 63 | 29 |
| 5 | Multiplication | 185 | 3.55 | 64 | 28 |
| 6 | Division | 83 | 8.26 | 42 | 28 |
| 7 | Dotproduct | 278 | 5.00 | 54 | 140 |
| 8 | $x = x + \alpha y$ | 313 | 4.31 | 32 | 177 |
| 9 | $z = x + \alpha y$ | 298 | 4.44 | 43 | 57 |
| 10 | $y = x_1 x_2 + x_3 x_4$ | 207 | 9.83 | 26 | 32 |
| 11 | 1st order recurs. | 83 | 8.27 | 32 | 75 |
| 12 | 2nd order recurs. | 121 | 11.32 | 38 | 109 |
| 13 | 2nd difference | 371 | 5.50 | 67 | 252 |
| 14 | 9th Degr. Polynominal | 261 | 47.37 | 28 | 254 |

The Table shows that the number of cycles/operation/element for all kernels except division are significantly higher than on RISC processors like the Alpha EV67 (see Table 6.1). This is especially the case for the dotproduct and the flat rotation operation that need no less than 5.00 and 9.83 cycles per operation which results in $r_{\max}$ values of 278 and 207 Mflop/s, respectively. The ideal cycle counts would be 2 and 1.33 based on computational intensities of resp. $f = 1$ and $f = \frac{3}{5}$. Even kernel 14, the evaluation of a $9^{\text{th}}$ degree polynomial, with $f = 9$, where all operations are performed out of the registers shows a speed of 261 Mflop/s, less than 40% of the Theoretical Peak Performance. From the latter kernel it is clear that neither superpipelining, nor superscalar operation are in effect in this case and so a large portion of the computational potential is not realised.
Also the latencies for operation from the primary cache are in all cases but one (evaluation of a $9^{\text{th}}$ polynomial) significantly higher than in a RISC processors like the Alpha processor considered in this report and also for the MIPS processor used in the Origin2000. The $n_{1/2}$ value (i.e., the vector length where half of $r_{\max}$ can be attained, see [3]) is about 18 for a 64-bit floating point multiplication and 28 for a 64-bit vector copy.
The $r_{\max}$ values from the L2 cache show an irregular picture: most kernels show a speed decrease of a factor 5–8, while the recursion kernels, the dotproduct and the `axpy` operation only drop by roughly a factor of 2 in speed.
Unlike in the RISC processors, there is a noticeable effect on the performance when the operations are not done on contiguous elements from outside the primary cache: for instance, the dotproduct $r_{\max}$ from the secondary cache with a stride of 3 shows the same performance level as with stride 1. However, when the stride is 4, the speed drops by almost 25% from 140 to 108 Mflop/s. This due to the latency of the memory banks in the secondary cache and this effect occurs for all basic operations that happen to access the L2 cache with a stride of (a multiple of) 4.

In module 2, the basic numerical algorithms, program `mod2a` performs on the same level as the constituent basic operations. The `axpy` implementation is only 10% slower than the single `axpy` kernel (283 vs. 313 Mflop/s) while the dotproduct implementation almost loses 30% in speed (202 vs. 278 Mflop/s). The latter implementation suffers from the fact that data access is not contiguous as already was noticed analysing the results of Module 1. The compiler does not take advantage of a better scheduling of operations possible with a double loop structure and, as observed in the Module 1 kernels, the superscalar capability of the CPU does not work with 64-bit floating point operands.

Table 6.8: *SGI: Maximum bandwidth for various important communication patterns as measured with program* `mod1h` *using Fast Ethernet.*

| | Operation | 4 Processors MB/s | 6 Processors MB/s | 8 Processors MB/s |
|---|---|---|---|---|
| 1 | Pingpong | 11 | 11 | 11 |
| 2 | Pingpong, strided | 13 | 13 | 13 |
| 3 | Broadcast | 11 | 12 | 11 |
| 4 | Collect | 12 | 12 | 12 |
| 5 | Reduction | 20 | 20 | 20 |
| 6 | Transposition | 13 | 13 | 12 |
| 7 | Bisectional BW, Regular | 11 | 11 | 11 |
| 8 | Bisectional BW, Random | 11 | 11 | 11 |
| 9 | Nearest Neighbour 1-D | 9 | 9 | 9 |
| 10 | Nearest Neighbour 2-D | 10 | 8 | 8 |
| 11 | Nearest Neighbour 3-D | 13 | 11 | 8 |

The same trend is evident from program `mod2b` that solves dense linear algebra systems. The $r_{\max}$ found is 200 Mflop/s, almost 30% slower than the corresponding `axpy` operation. This is consistent with the speed of the latter operations, as also in the solution part a first order recursion at a speed of 83 Mflop/s has to be done together with some data manipulations that do not add to the flop count of the algorithm.
Also program `mod2d`, the solution of dense symmetric eigenvalues performs somewhat slower than the main operations it consists of: the flat rotation and the `axpy` operation. The $r_{\max}$ found for `mod2d` is 188 Mflop/s, where the flat rotation and the `axpy` perform at resp., 207 and 313 Mflop/s. The speed reduction is due to non-contiguous addressing and a fair amount of operations on vectors that are too small to attain full speed: as found from the Module 1 results the operation latencies are rather large. The same basic operations play an important role in program `mod2f`, performing 1-D FFTs. Also in this case, the operand vectors can be small and the access patterns are unfavourable because either the source or target vectors have to be accessed with a stride that is a power of 2. This shows in $r_{\max}$ which is 211 Mflop/s for an $N = 256$ length FFT which can entirely be computed from within the primary cache. For larger problem sizes the speed quickly drops to a level of about 60 Mflop/s.
The best performance to be expected in program `mod2g`, a 2-D Haar transform is 117 Mflop/s as the computational intensity is $f = \frac{1}{3}$ and the floating point operations cannot overlap. $r_{\max} = 95$ Mflop/s, however, which is again partly due to non-unit stride problems in the data access and also to the high latency of the multiply and add operations used.

The Module 3 results will be discussed in section 6.4 along with those of the other systems.

#### 6.3.2.2 The synthetic programs: Communication performance

The results of the programs `mod1h`, `mod1i`, and `mod1j` are available both for Fast Ethernet and Myrinet. The measurements from `mod1h` for these networks are given in Tables 6.8 and 6.9, respectively.

The results found for Fast Ethernet are more or less identical to those found in section 6.1.2.2 for the Compaq cluster. Only for the Nearest Neighbour communication patterns this seems to be slightly lower in the SGI cluster. Generally, on 8 processors the bandwidth is somewhat lower for operations like Reduction, Transposition, and the Nearest Neighbour patterns. For both networks the amount of traffic increases with the number of processors involved, which shows in the bandwidth found.
Comparison of Tables 6.8 and 6.9 makes clear that the maximum bandwidth on the Myrinet-based cluster is 4–8 times higher than that with a Fast Ethernet network. This is somewhat disappointing keeping in mind that the theoretical peak speed of Myrinet is about 160 MB/s. The relatively low speed for the Collect operation is not very well explainable when comparing it to the Reduction operation that in essence should employ a similar gathering technique. This can only be explained by a sub-optimal implementation of the MPI operation involved. This is not the case for the strided point-to-point messages. Here the internal

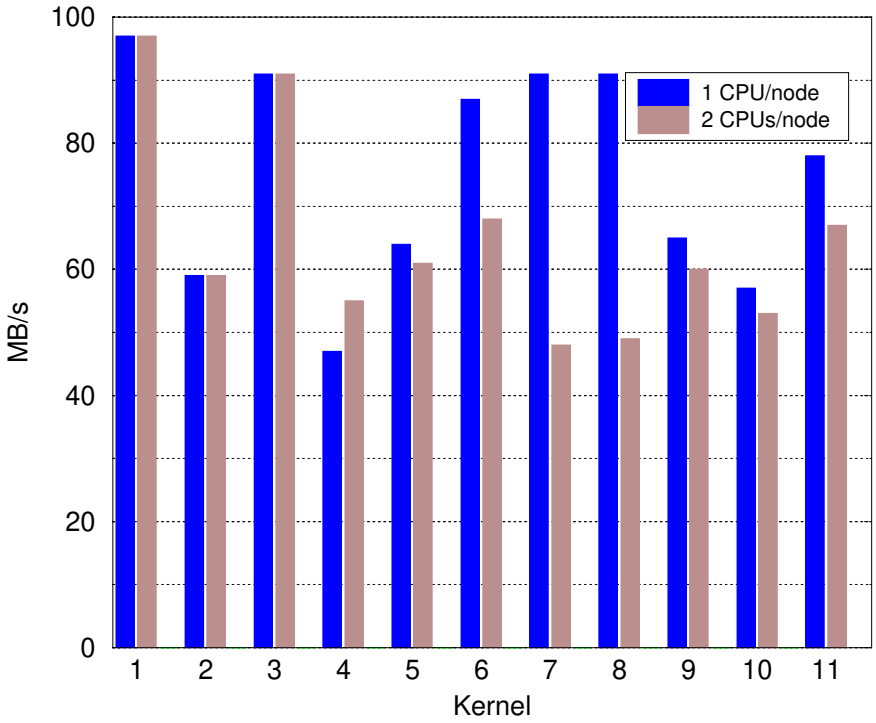



Figure 6.5: *SGI: Influence of internal bandwidth competition when using 1, resp., 2 CPUs/node for the communication patterns in program* `mod1h` *on 4 processors.*

bandwidth within the nodes is a bottleneck that shows in the communication speed. With the slower Fast Ethernet network this effect is not noticeable.
As program `mod1h` was executed on at most 8 processors, there was a choice to use either one processor from each of the 8 nodes in the cluster or to use 2 CPUs per node on 4 nodes. Potential bandwidth problems within the nodes should then show up if the CPUs had to compete for bandwidth. Both modes of operation have been executed and, indeed, when using 2 CPUs per node this internal bandwidth congestion was observed: Figure 6.5 shows the often large influence that occurs when more than 1 CPU per node is used. This is particularly true for the Bisectional Bandwidth kernels and for the Nearest Neighbour kernels. For both types of kernels this is as expected. With the Bisectional Bandwidth kernels one half of the processes in the configured complex communicates with the other half at the same time. When 2 CPUs/node are used, the messages are to be sent or received by these 2 CPUs from the same node memory, resulting in contention on the internal memory bus. A similar effect, though less outspoken happens in the Nearest Neighbour kernels where always a nearest neighbour process will be placed in a CPU in the same node. The effect is less prominent here because first communication in one direction of the ring (toroïd) is done and then in the other direction. The 2-D transposition also suffers from the internal bandwidth competition. It also does All-to-All communication in a less coordinated way than in the former two patterns, however, also local data rearrangement has to be performed here because of the local partial transpositions which too burden the internal busses of the nodes. To a much lesser extent the local data processing in the Reduction kernel (yielding the global maximum of an array) uses the internal bus capacity resulting in a lower observed bandwidth of 5% in the 2CPUs/node configuration. Surprisingly, the Reduction kernel is about 7% *faster* when executed on a 2 CPUs/node configuration. There might be a connection with the fact that the `MPI_Collect` operation in the Myrinet implementation is not very well implemented altogether, but further study is necessary for a good explanation.

The performance difference using 2CPUs/node vs. 1 CPU/node becomes quite evident in observing the

Table 6.9: *SGI: Maximum bandwidth for various important communication patterns as measured with program* `mod1h` *using Myrinet*

| | Operation | 4 Processors MB/s | 6 Processors MB/s | 8 Processors MB/s |
|---|---|---|---|---|
| 1 | Pingpong | 97 | 97 | 97 |
| 2 | Pingpong, strided | 59 | 59 | 59 |
| 3 | Broadcast | 91 | 91 | 91 |
| 4 | Collect | 47 | 47 | 47 |
| 5 | Reduction | 64 | 82 | 74 |
| 6 | Transposition | 87 | 73 | 70 |
| 7 | Bisectional BW, Regular | 91 | 91 | 91 |
| 8 | Bisectional BW, Random | 91 | 91 | 91 |
| 9 | Nearest Neighbour 1-D | 65 | 65 | 65 |
| 10 | Nearest Neighbour 2-D | 63 | 57 | 56 |
| 11 | Nearest Neighbour 3-D | 78 | 71 | 59 |

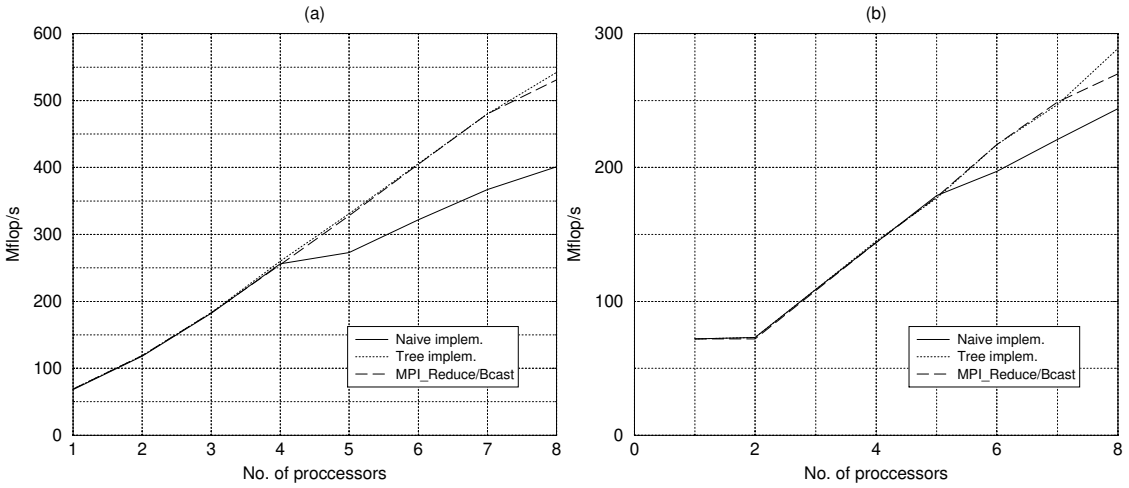



Figure 6.6: *SGI: Influence of internal bandwidth competition when using 1, resp., 2 CPUs/node on the distributed dotproduct (program* `mod1h`*). (a): 1 CPU/node, (b) 2 CPUs/node.*

results of program `mod1i`, implementing a distributed dotproduct in three ways. Figure 6.6 (a) and (b) show the results for 1 CPU/node and 2CPUs/node, respectively, for the three different implementations as explained in section 6.1.2.2.

Three conclusions can be drawn from this Figure: firstly, the performance degradation for this operation is quite significant when 2 CPUs/node are used, for 2 CPUs/node the speed is almost a factor of two lower on 8 processors for the best implementation. Secondly, there is indeed a clear difference in performance between the tree-based implementation and the simple implementation that uses a central root processor. Thirdly, the Myrinet MPI implementation is tree-based: it performs identical to the Fortran tree implementation.
That it is possible to discern between the three implementations is by virtue of the much lower latency of Myrinet in comparison to Fast Ethernet. In the distributed dotproduct the amount of data to be communicated is negligible, so low latency is extremely important. With program `mod1j` we found a latency of 15.9 $\mu$s, almost 10 times lower than that of Fast Ethernet for CPUs on different nodes.

Table 6.10: *SGI: Single-user wallclock times for the user programs in the benchmarks for 1–8 processors when using 1 CPU/node and 1–16 processors when using 2 CPUs/node. The network used is Myrinet.*

**1 CPU/node**

| **Program →** | `adf` | `maya5` | `primmod` | `rboltz` | `vasp` |
|---|---|---|---|---|---|
| **# of Proc.s ↓** | Seconds | Seconds | Seconds | Seconds | Seconds |
| 1 | 6208 | 8033 | — | 41 | 11834 |
| 2 | 3388 | 8263 | — | 25 | 5619 |
| 3 | — | 4216 | — | — | — |
| 4 | 1777 | 3091 | — | 11 | 2613 |
| 6 | — | 2048 | — | — | — |
| 8 | 1002 | 1607 | 152.2 | 7 | 1220 |

**2 CPUs/node**

| **Program →** | `adf` | `maya5` | `primmod` | `rboltz` | `vasp` |
|---|---|---|---|---|---|
| **# of Proc.s ↓** | Seconds | Seconds | Seconds | Seconds | Seconds |
| 1 | — | — | — | 41 | — |
| 2 | 3376 | 7960 | — | 25 | 7505 |
| 3 | — | 4045 | — | — | — |
| 4 | 1785 | 3037 | — | 13 | 3474 |
| 6 | — | 2029 | — | — | — |
| 8 | 1008 | 1596 | 156.1 | 7 | 1369 |
| 12 | — | 1095 | — | — | — |
| 16 | 596 | 1068 | — | 5 | 731 |

### 6.3.3 User programs

In Table 6.10 the results for the user programs in the benchmark are summarised when using Myrinet and in Table 6.11 when using Fast Ethernet.

Executing the programs both with Myrinet and Fast Ethernet on the same hardware gave us a good opportunity to assess the influence of the network speed on the scaling of the programs. For program `adf` the timings when using 1 CPU/node are almost identical, showing that communication bandwidth is not a critical factor. For a 2 CPUs/node configuration the situation is different: on 16 CPUs the speedup with Myrinet is 10.4 while it is 8.34 with Fast Ethernet. This shows that not only the memory bus has to be competed for but also for the network interface within a node.

Program `maya5` uses a master-slave parallel model as explained in section 6.1.2.3. Therefore the ideal speedup is $p-1$ on $p$ CPUs. The computation/communication ratio in this program is extremely high and the amount of communication is also very low in absolute terms. Therefore, `maya5` is not sensitive to either the number of CPUs/node or the type of network used. That the scaling in this program is not perfect (5.00 on 8 CPUs, 7.52 on 16 CPUs) is rather the effect of the load unbalance in the slave tasks than of the lack of inherent scalability.

Program `primmod` can in this form only be executed on 8 processors. There is no decrease in performance when 2 CPUs/node are used, but the slower network has an influence: the execution time is 18% higher when using Fast Ethernet instead of Myrinet on 8 CPUs.

The Lattice Boltzmann code `rboltz` is insensitive for the amount of CPUs/node but there it is sensitive for the type of network used: with Myrinet the speedup on 8 processors is 5.70, with Fast Ethernet 4.26. In the latter case the user-time is only 56% of the elapsed time, indicating that much of this time is due to the communication overhead. User program `vasp` is both influenced by the number of CPUs/node and the speed of the network. The performance loss using Myrinet is 12% when 2 CPUs/node are employed with 8 CPUs. The use of Fast Ethernet also causes a performance loss with respect to the use of Myrinet. With 1 CPU/node on 8 CPUs the execution time the performance decreases by 56%. When using 2 CPUs/node this speed difference is even greater: 71% while on 16 processors the performance loss is almost a factor of 2.

Table 6.11: *SGI: Single-user wallclock times for the user programs in the benchmarks for 1–8 processors when using 1 CPU/node and 1–16 processors when using 2 CPUs/node. The network used is Fast Ethernet.*

**1 CPU/node**

| **Program →** `adf` | `maya5` | `primmod` | `rboltz` | `vasp` |
|---|---|---|---|---|
| **# of Proc.s ↓** Seconds | Seconds | Seconds | Seconds | Seconds |
| 1 | 6163 | 8049 | — | 40 | 11834 |
| 2 | 3466 | 8094 | — | 26 | 6958 |
| 3 | — | 4119 | — | — | — |
| 4 | 1782 | 3073 | — | 13 | 3990 |
| 6 | — | 2049 | — | — | — |
| 8 | 1043 | 1622 | 179.7 | 9 | 1899 |

**2 CPUs/node**

| **Program →** `adf` | `maya5` | `primmod` | `rboltz` | `vasp` |
|---|---|---|---|---|
| **# of Proc.s ↓** Seconds | Seconds | Seconds | Seconds | Seconds |
| 1 | — | — | — | 40 | — |
| 2 | 3401 | 8181 | — | 27 | 8127 |
| 3 | — | 4085 | — | — | — |
| 4 | 1832 | 3030 | — | 13 | 4528 |
| 6 | — | 2157 | — | — | — |
| 8 | 1051 | 1611 | 181.7 | 9 | 2349 |
| 12 | — | 1095 | — | — | — |
| 16 | 739 | 1073 | — | 9 | 1349 |

The user programs show a very different behaviour with respect to their sensibility to network speed and the number of CPUs/node. Except in cases like that of program `maya5` that has minimal communication requirements, it is not straightforward to estimate the influence of these configuration parameters and it is necessary to assess for such programs this behaviour separately to judge whether this behaviour is acceptable.

### 6.3.4 Throughput test results

The Myrinet version of MPI did not agree well with the PBS batch system. Therefore, the throughput results are not as good as they could be without the problems that came up in their combined operation. The activity graph of the throughput benchmark is displayed in Figure 6.7.

It is clear from the Figure that the faster communication has a very large influence on the total throughput. The complete EuroBen suite takes about 5500 seconds as compared to 11,000 seconds on the Compaq cluster, notwithstanding the lower computational performance of the Intel processors. This is mainly due to the much lower latency of the `MPI_Send/Rcev` in the Myrinet implementation which results in an elapsed time of just over 1100 seconds for program `mod1j` as opposed to 7,000 seconds on the Compaq cluster where fast Ethernet was used.

Still, the use of the batch system is not without problems: MPI jobs cannot be run on the same port of the Myrinet switch, so, jobs that are running simultaneously cannot use the same port. The consequence is that one either has to allow only one instance of a job to be active in the system or to configure the job sequence explicitly such that multiple instances of a job do not use the same ports. This is clearly unacceptable in a large and busy system. Furthermore, PBS is not aware of the multiple nodes in the system as it was originally made for SISD systems. So, PBS cannot correctly schedule the available resources because it does not know how many nodes are occupied by a job over the time it is running. Therefore, again, hand scheduling is required to get an optimal workload profile. Of course, this is not possible when jobs are dynamically scheduled by a user community.

Also some specific problems were encountered running this workload. Program `maya5` could not be scheduled with the appropriate number of nodes (`-l nodes=3`, 6 CPUs) but could be scheduled with 4 nodes, although 2 CPUs are idle in this case. Program `primmod` refused to run correctly when it had to

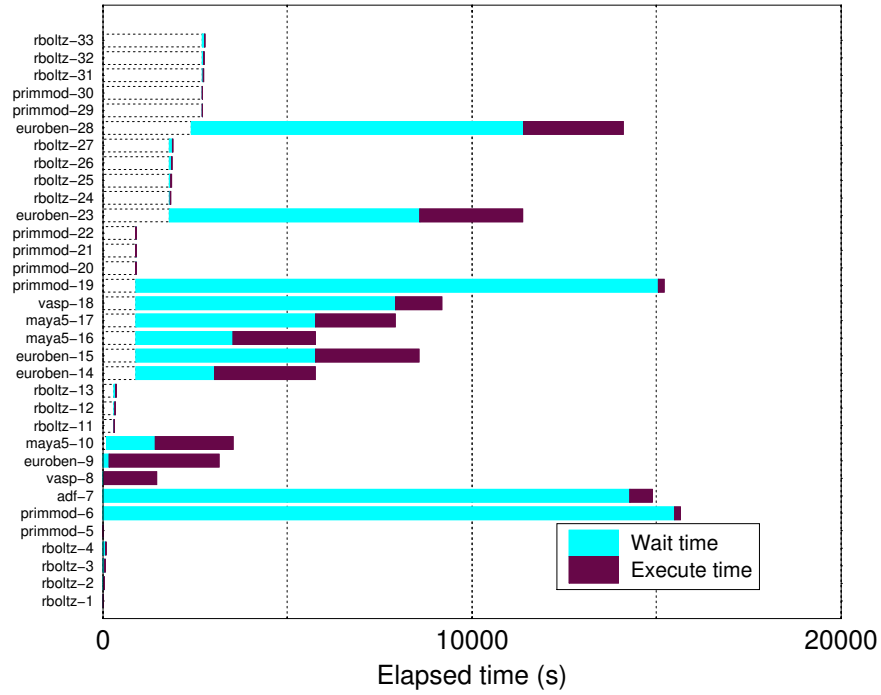



Figure 6.7: *SGI: Activity graph of the throughput benchmark. The dotted white parts of the bars signify the times after which jobs are submitted*

share a node with another program. Therefore, the `primmod` jobs had to be scheduled such that they would run while no other jobs were active. Lastly, Myrinet's MPICH-GM produced a temporary file `/tmp/mpi-gm_smp-b0p5-b0p4.tmp` when running `adf` which stalled all other programs. These files had to be explicitly removed to enable the workload to finish.

Notwithstanding these problems, the total throughput time of the workload was about 16,000 seconds. As can be seen from Figure 6.7, when the jobs `primmod6` and `primmod19` could have been run during the run of `euroben14` the total time could have been under 15,000 seconds. The jobs `primmod29` and `primmod30` were not run altogether because of the problems of the temporary file produced by `adf7`.

## 6.4 Comparisons

In this section we will compare performances as found for the clusters and, in the cases of Compaq and SGI also with their integrated parallel systems, the Compaq AlphaServer SC and the SGI Origin2000. In addition, we present results for the third module of the EuroBen synthetic benchmark. The full performance analysis of each of these programs is beyond the context of this report, so we will simply state the timings in each case.

### 6.4.1 EuroBen Results, Module 1 and 2

Most results of the synthetic programs already have discussed extensively in sections 6.1, 6.2, and 6.3, however, they were not compared with those of the integrated parallel machines. This is no problem for the Compaq AlphaServer SC as the processors are exactly the same as those of the XP1000 nodes in the Compaq cluster, so, ignoring the mostly small differences in performance resulting from the different optimisation levels used in compiling the codes, the outcomes for the single CPU program are virtually the same.
For the Origin2000 we only have results on a 300MHz, MIPS R12000-based machine with a Theoretical Peak Performance of 600 Mflop/s. These results are still of interest because they give an impression of the

Table 6.12: *The $r_{\max}$ values of the first 14 kernels of program* `mod1ac` *from the L1 cache on resp., the Compaq Alpha XP1000, the IBM cluster (Intel PIII, 600 MHz), the SGI Origin 2000 (MIPS R12000, 300 MHz), and the SGI cluster (Intel PIII, 700 MHz).*

| | **Operation** | XP1000 Mflop/s | IBM clust. Mflop/s | SGI O2000 Mflop/s | SGI clust. Mflop/s |
|---|---|---|---|---|---|
| 1 | Broadcast | 523 | 326 | 292 | 388 |
| 2 | Copy | 254 | 286 | 146 | 325 |
| 3 | Addition | 246 | 170 | 98 | 197 |
| 4 | Subtraction | 240 | 168 | 98 | 197 |
| 5 | Multiplication | 246 | 157 | 98 | 185 |
| 6 | Division | 55 | 72 | 14 | 83 |
| 7 | Dotproduct | 620 | 243 | 298 | 278 |
| 8 | $x = x + \alpha y$ | 513 | 273 | 196 | 313 |
| 9 | $z = x + \alpha y$ | 470 | 257 | 196 | 298 |
| 10 | $y = x_1 x_2 + x_3 x_4$ | 426 | 180 | 175 | 207 |
| 11 | 1st order recurs. | 110 | 72 | 147 | 83 |
| 12 | 2nd order recurs. | 136 | 105 | 147 | 121 |
| 13 | 2nd difference | 611 | 321 | 438 | 371 |
| 14 | 9th Degr. Polynominal | 701 | 227 | 596 | 261 |

efficiency with which the processors are used. In Table 6.12 we display the $r_{\max}$ values for this machine along with those for the Alpha XP1000, and the Intel-based clusters.

From this Table we see the different nature of the Intel processors and the RISC processors: for data copies and triadic arithmetic operations the Intel processors do a good job. However, when more complicated arithmetic operations are to be performed, their efficiency lags. This is for instance demonstrated by the dotproduct operation, the recursion operations, and the second difference operation. Here the RISC processors clearly win, even when the clock frequency is much lower, as is the case with the MIPS R12000 processor. So, it depends on the algorithms used whether the Intel processors or RISC processors will be more efficient and, ultimately, will yield the best performance.

Although this is not strictly a performance issue, the first module of the EuroBen Benchmark also checks the accuracy of the mathematical intrinsic functions, as experiences has learned that these are not always are implemented with sufficient accuracy. The outcomes for both the Intel processors, and for the MIPS processor gave in that respect some unpleasant surprises: on the Intel processors (with the Portland Group Compiler set) the maximum deviation in the `exp()` function was almost 3 significant digits while in the `sin()` and `cos()`, respectively, the maximum deviations were 4 and 5 significant digits. The MIPS processor with the SGI Fortran 90 compiler showed the same loss of significant digits for the `exp()` function as the Intel processors and in the same argument range. In addition, the `log()` function was off by no less than 8 significant digits in the range close to 1. On the Compaq Alpha processor with the Compaq Fortran `mpi77` no deviations were found that exceeded 1 significant digit. Clearly, such deviations as found for the Intel and MIPS processors should not occur and one should be cautious with respect to results that draw heavily on these functions or critically depend on them.

A useful first impression of the speed and efficiency of the processors in the context of basic algorithms can be gleaned from the programs in module 2 of EuroBen. We list the results for some of these programs in Table 6.13.

As most of these algorithms contain a large fraction of operations that have a complexity level that at least equals a dotproduct, involving both floating point addition and multiplication, it is not surprising that the RISC processors are doing relatively well, both in efficiency and speed. An exception is the 2-D Haar Wavelet Transform which has a low computational intensity and where the composition of operations is such that additions and multiplications is such that they can only partly overlap. The results in Table 6.13 indicate that in compute intensive the RISC processors will outperform the Intel processors and are utilised with a higher efficiency. As RISC processors are designed for such tasks one can conclude that they meet

Table 6.13: $r_{\max}$ *values for some program in module 2 of the EuroBen Benchmark on resp., the Compaq Alpha XP1000, the IBM cluster (Intel PIII, 600 MHz), the SGI O2000 (MIPS R12000, 300 MHz), and the SGI cluster (Intel PIII, 700 MHz).*

| **Program** | XP1000 cl. Mflop/s | IBM Netfin. Mflop/s | SGI O2000 Mflop/s | SGI L1200 Mflop/s |
|---|---|---|---|---|
| `mod2a`, Matrix-vector mult. | 406.5 | 179.6 | 414.9 | 282.8 |
| `mod2b`, $Ax = b$, (dense) | 373.9 | 101.1 | 176.6 | 200.0 |
| `mod2d`, $Ax = \lambda x$ (dense) | — | 139.3 | 391.9 | 187.9 |
| `mod2f`, 1-D FFT | 570.5 | 183.1 | 263.1 | 210.8 |
| `mod2g`, 2-D Wavelet Transf. | 208.8 | 62.6 | 74.2 | 95.1 |

their design goals, where the Intel processors can provide fair performance levels by virtue of high clock rates and a good ability for moving data between the L1 cache and the processor. For a reason that is not yet clear program `mod2d`, the symmetric eigenvalue problem, experienced a runtime error on the Compaq cluster while it ran excellently on the Compaq Alpha SC with an $r_{\max}$ value of 620.2 Mflop/s.

The comparisons given here are based on $r_{\max}$ values that mostly occur when computing from the L1 cache. When looking at the speed from memory, the difference in efficiency between the RISC processors and the Intel processors is more outspoken because of the lower bandwidth from the memory to the cache/processor in the Intel processors. This is for instance exemplified by program `mod2g`, the 2-D Wavelet Transform: A 512×256 transform attains a Mflop rate of 43.5 Mflop/s on a 300 MHz R12000 MIPS processor, while the speed on the 700 MHz Intel PIII processor is 20.5 Mflop/s in a program where the computational intensity is low and where data manipulation is more prominent.

### 6.4.2 EuroBen Results, Module 3

Module 3 of the EuroBen benchmark contain three different methods for solving PDEs, an ODE solver, a linear and a non-linear optimisation code, and two programs that test I/O speed in the context of a tomography problem and a very large out-of-core multidimensional FFT.
The read and write bandwidth obtained in program `mod3a`, the tomography application, is shown Figure 6.8. Program `mod3a` computes bandwidths for reading and writing for various problem sizes. For the smaller problem sizes the bandwidth varies from 105–170 MB/s for reading and 65–105 MB/s for writing. However, for large problem sizes the Compaq cluster shows a drop in bandwidth to 8 and 5 MB/s for reading and writing, respectively and on the Origin 2000 to 6 and 4 MB/s, while on the IBM and SGI cluster the speed degradation is much less. This is at least partly due to the fact that these systems were also heavily used by other users, so I/O bandwidth had to be shared. For the larger problem sizes the chance of interference is also much larger. For this reason we do not quote the results found on the Compaq Alpha SC, as this system was obviously was heavily loaded and the I/O results from program `mod3a` cannot be trusted.
Figure 6.8 makes clear that the I/O configuration of the clusters is quite good and does not seem a bottleneck when compared with the integrated parallel systems in this test.

The wallclock times of programs `mod3b-h` for the three clusters and the two parallel systems are shown in Table6.14. It must be remarked, however, that on the AlphaServer SC, the Origin 2000, and the Alpha XP1000 cluster there was a non-negligible to heavy load from other users which distorts the wallclock time in program `mod3b`. Both on the AlphaServer SC and the SGI L1200 cluster we have no results for program `mod3h`, presumably because the associated input files were not in the right place. Programs `mod3d` and `mod3e` suffered from over-optimisation on the Compaq AlphaServer but performed quite well with the `-O3` compiler option on the identical processors of the XP1000 cluster.

Like program `mod3a`, `mod3b` suggests that for I/O bound programs the clusters behave quite well on the clusters.

For compute-bound programs the situation is different. The Alpha CPU is 2–4 times faster here than the Intel PIII in the IBM and SGI clusters in accordance to what was observed for several other programs in the

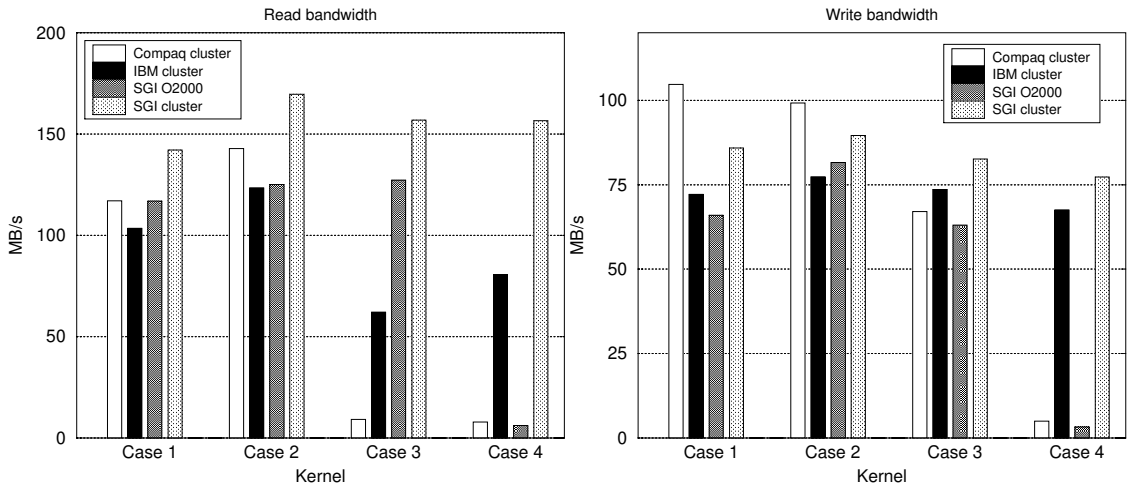



Figure 6.8: *Read and write bandwidth for large out-of-core matrix-vector multiplication (Tomography application, program* `mod3a`*). Size of matrices are 25,000×20,000 in Case 1, 50,000×20,000 in Case 2, 100,000×100,000 in Case3, and 250,000×100,000 in Case 4, respectively. There was interference of other users on the Compaq cluster and on the SGI Origin 2000.*

Table 6.14: *Wallclock times for module 3 programs of the EuroBen Benchmark on resp., the Compaq AlphaServer SC (Alpha EV67, 667 MHz), the Compaq Alpha XP1000 (Alpha EV67, 667 MHz), the IBM cluster (Intel PIII, 600 MHz), the SGI Origin 2000 (MIPS R12000, 300 MHz), and the SGI cluster (Intel PIII, 700 MHz).*

| **Program** | AlphaS. SC Seconds | XP1000 cl. Seconds | IBM Netfin. Seconds | SGI O2000 Seconds | SGI L1200 Seconds |
|---|---|---|---|---|---|
| `mod3b`, I/O, FFT | — | 5.0281 | 0.73253 | 0.99108 | 0.48287 |
| `mod3c`, PDE, Multigrid | 0.419922 | 0.594560 | 2.56111 | 0.880669 | 2.26631 |
| `mod3d`, Linear Optim. | 0.035156 | 0.001163 | 0.004528 | 0.002941 | 0.002752 |
| `mod3e`, Non-linear Optim. | 0.023438 | 0.005076 | 0.017084 | 0.013743 | 0.012645 |
| `mod3f`, ODE, (BVP) | 2.1484 | 2.6440 | 5.6093 | 5.2201 | 4.8015 |
| `mod3g`, PDE, Elliptic solver | 0.033203 | 0.018427 | 0.042914 | 0.040775 | 0.040983 |
| `mod3h`, PDE, Block Gauss † | — | 2.0139 | 9.4809 | 1.4213 | — |

† Grid size 257×257.

previous sections. Even the MIPS R12000 processor with a clock frequency of 300 MHz often does better by a factor of 2 or more (programs `mod3c`, `mod3d`, `mod3h`) when the computational intensity is high. For programs where also an appreciable amount of data manipulation is required, the Intel processors have speed that is comparable to that of the MIPS processor. In all the impression obtained from the former modules is confirmed with respect to the behaviour of the processors.

### 6.4.3 Basic communication

The communication performance of the systems is in many of the programs critical for their success or failure and it may be factor in the choice for a cluster or an integrated parallel machine. We will here consider the basic communication on the three clusters considered and on the Compaq AlphaServer SC and the SGI Origin2000, respectively. The maximum bandwidth for some important communication patterns as measured in EuroBen program `mod1h` the results are displayed in Figure 6.9. For ease of reference we list the communication patterns here again:

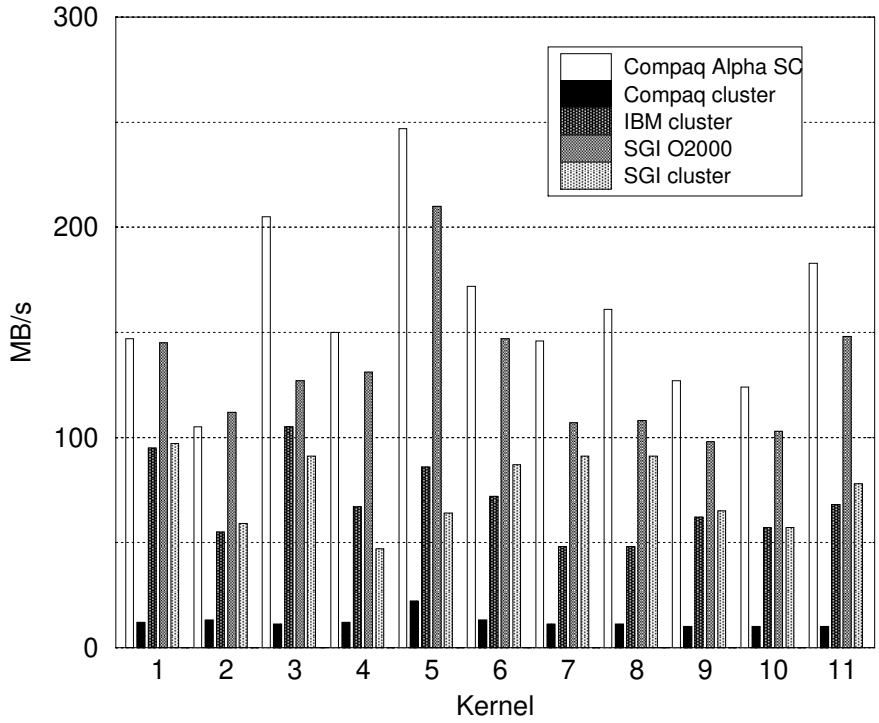



Figure 6.9: *Bandwidth for the 11 communication patterns in program* `mod1h` *on 4 processors for resp. the Compaq AlphaServer SC, the Compaq Alpha XP1000 cluster, the IBM Netfinity cluster, the SGI Origin2000, and the SGI L1200 cluster.*

| | **Communication pattern** |
|---|---|
| 1 | Pingpong |
| 2 | Pingpong, strided |
| 3 | Broadcast |
| 4 | Collect |
| 5 | Reduction |
| 6 | Transposition |
| 7 | Bisectional BW, Regular |
| 8 | Bisectional BW, Random |
| 9 | Nearest Neighbour 1-D |
| 10 | Nearest Neighbour 2-D |
| 11 | Nearest Neighbour 3-D |

As can be seen in the Figure, the network of the Compaq AlphaServer SC, based on QSW's QSNet is in almost all cases the fastest, immediately followed by the network in the SGI Origin2000 as one would hope for on integrated parallel systems. The Myrinet-based systems do not differ a great deal, except for the collect operation (5) and the bisectional bandwidth patterns (7 and 8). However, the latter two patterns could only be measured on a 2 CPUs/node configuration on the IBM cluster. The 2 CPUs/node results for the SGI cluster are very similar. For the difference in the collect operation there is no evident explanation as both the Myrinet hardware and software are identical. Possibly there are differences in the configuration of the Myrinet ports, but this could not be confirmed. For the bisectional bandwidth patterns, the internal bandwidth to memory is quite critical as showing in the observed bandwidths. For similar All-to-All patterns the same type of behaviour may expected, which to a lesser extent can be seen from the Transpose operation

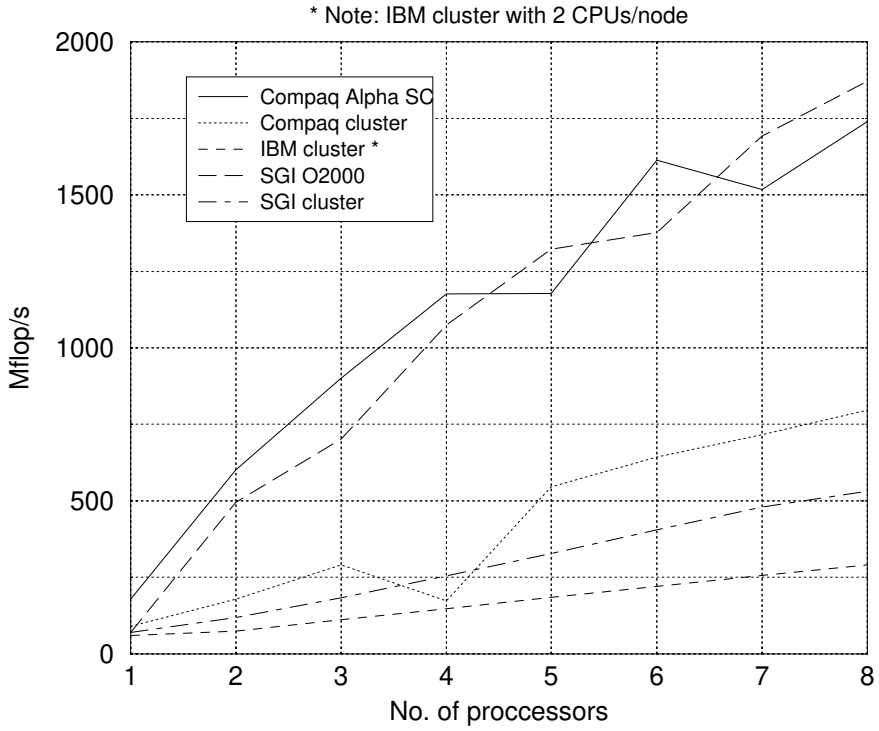



Figure 6.10: *Best implementation of distributed dotproduct on resp. the Compaq AlphaServer SC, the Compaq Alpha XP1000 cluster, the IBM Netfinity cluster, the SGI Origin2000, and the SGI L1200 cluster.*

(6).

For the distributed dotproduct we compare the best implementations for all of the clusters and the AlphaServer SC and the Origin2000. In this case bandwidth is of no consequence but the latency is. On the clusters that can use 2 CPUs/node the internal bandwidth competition is a much more critical factor than the latency: the speed of the computation is halved, although the scaling is virtually linear from 2 CPUs on as can be seen in Figure 6.10.

It is also clear from the Figure that the integrated machines perform significantly better. This comes about by a combination of factors. The first one is the better relative floating point performance of the RISC processors in these systems: per processor a speed of 220–235 Mflop/s or 16–29% of the Theoretical Peak Performance. On the Intel-based clusters the speed is 58–67 Mflop/s per processor which amounts to 9% of the TTP. Furthermore the latency is somewhat higher on the Myrinet network of the of the Intel-based clusters. That the latency plays a non-negligible role can also be seen from the results from the SGI cluster when the same processors are used but Fast Ethernet instead of Myrinet. In that case there is a 20% performance drop on 8 processors with respect to Myrinet. The effect is much more significant when comparing the speeds for the AlphaServer SC and the Compaq XP1000 cluster that has the same processors but uses Fast Ethernet. The performance is halved in this case. However, the different level of optimisation ( `-O5` vs. `-O3`) certainly influences this difference too.

Summarising the maximum observed bandwidths and the latencies of the respective networks through program `mod1j` yields Table 6.15. Especially the network speed and latency on the IBM cluster stand out here. The bandwidth found is as large as that from the Origin 2000 and the latency is unrealistically low. In the similar, but less detailed experiment in program `mod1h` the bandwidth is in fact same as that on the SGI cluster and the same holds for the latency. A possible explanation could be that only internal communication within one node occurred here. But we have not sufficient evidence for this assumption and we have to look further into this interesting behaviour.

Table 6.15: *Maximum observed bandwidth and latency for the five systems involved.*

| **System** | Network type | Bandwidth MB/s | Latency $\mu$s |
|---|---|---|---|
| Compaq AlphaServer SC | QSW QSNet | 164 | 7.2 |
| Compaq XP1000 cluster | Fast Ethernet | 12 | 105.0 |
| IBM Netfinity cluster | Myrinet | 132 | 2.7 |
| SGI Origin2000 | Proprietary | 133 | 6.3 |
| SGI L1200 cluster | Myrinet | 94 | 15.9 |

### 6.4.4 User programs

It is interesting to compare the performance of an integrated parallel system with that of the clusters considered. For, although the synthetic benchmark programs give some indications that the integrated machines may perform better it still remains to be seen whether this is true in practical situations and, if so, to what extent they are doing better. For the SGI Origin2000 we have single-user mode performance results available to make the comparison with the cluster performance (unfortunately we could not get single-user mode results for the Compaq AlphaServer SC in time). We display the best cluster results for each of the clusters, i.e., where possible with 1 CPU/node on the IBM and SGI together with the results for the Origin2000 in Table 6.16. Also on the Origin2000 one can specify that only one of the CPUs of the two on a node board are to be used. This has indeed been done, but on the Origin the effect was negligible so we will disregard the differences in this respect. The times on the IBM and SGI cluster are both with Myrinet as the connecting network. The difference in performance between Myrinet and Fast Ethernet on the SGI cluster are discussed in section 6.3.

For as far as data are available the indications from the synthetic programs are confirmed, except for program `rboltz` that performed roughly similar on the Intel-based clusters and on the Origin2000. The Alpha-based cluster performed 2 times better than the other systems but, on the other hand, the low bandwidth of this cluster frustrates the good scaling of program `vasp`. As far as results reach, the Origin seems to be 2–3 faster than the Intel-based clusters and with respect to scaling this factor seems to be growing rather than decreasing for the program set in this benchmark. This makes sense generally as both the network bandwidth and latency is somewhat better than that of the Myrinet network.

Table 6.16: *Single-user wallclock times for the user programs in the benchmarks for 1–8, 12, and 16 processors. On the IBM and SGI clusters Myrinet was used.*

| **Compaq XP1000 cluster** | | | | | |
|---|---|---|---|---|---|
| **Program →** | `adf` | `maya5` | `primmod` | `rboltz` | `vasp` |
| **# of Proc.s ↓** | Seconds | Seconds | Seconds | Seconds | Seconds |
| 1 | 2663 | 5054 | — | 20 | 3573 |
| 2 | 1419 | 5028 | — | 11 | 2951 |
| 3 | 977 | 2627 | — | 8 | 2692 |
| 4 | 767 | 2039 | — | 7 | 2160 |
| 5 | 626 | 1376 | — | 7 | 1714 |
| 6 | 552 | 1434 | — | 7 | 1461 |
| 7 | 476 | 1202 | — | 7 | 1328 |
| 8 | 453 | 1090 | 90.3 | 6 | 1243 |
| 16 | 314 | — | — | — | — |
| **IBM Netfinity cluster** | | | | | |
| 1 | — | — | — | — | 13485 |
| 2 | — | 5735 | — | — | 7744 |
| 4 | — | 2144 | — | 8 | 2989 |
| 6 | — | 1435 | — | — | — |
| 8 | — | 1080 | 185.0 | — | 1356 |
| 16 | 947 | 738 | — | — | 818 |
| **SGI Origin2000** | | | | | |
| 1 | — | 2320 | — | 47 | 4085 |
| 2 | — | 2280 | — | 27 | 2088 |
| 3 | — | 1179 | — | — | — |
| 4 | — | 864 | — | 13 | 965 |
| 6 | — | 568 | — | — | — |
| 8 | — | 390 | 69.7 | 7 | 462 |
| 12 | — | 276 | — | — | — |
| 16 | — | 268 | — | 5 | 261 |
| **SGI L1200 cluster** | | | | | |
| 1 | 6188 | 8033 | — | 41 | 11834 |
| 2 | 3363 | 7960 | — | 25 | 5619 |
| 3 | — | 4216 | — | — | — |
| 4 | 1768 | 3037 | — | 11 | 2613 |
| 6 | — | 2029 | — | — | — |
| 8 | 993 | 1596 | 152.2 | 7 | 1220 |
| 12 | — | 1095 | — | — | — |
| 16 | 589 | 1068 | — | 5 | 731 |

# 7 Conclusions

The evaluation of the clusters has been an interesting experience and we have the impression that the information gathered, both generally and by means of this rather simple benchmark has greatly improved our understanding of what can be done with clusters and how well it can be done. This leads us to the following conclusions:

1. There is still an appreciable difference in the amount and maturity of the software for managing the clusters and for managing the workload: For cluster management the SGI ACE software is presently the most comprehensive, followed by the CMU package of Compaq, while on the IBM cluster publicly available cluster management tools are used. However, IBM expects to have available several of its sophisticated parallel system tools available in 2001, including SRC, RMC, and GPFS.
   With respect to batch processing, Compaq and SGI provide the public domain package PBS 2.2. The functionality of this package is limited which could potentially cause under-usage of the system (for instance, there is no back-fill functionality). In addition, the benchmark suggested that the combination of PBS and Myrinet is not a happy one (see section 6.3.4). For the IBM cluster LSF is available which does a better job but which is not in the public domain.

2. With respect to performance the RISC processor based systems often outperform the Intel processor based systems by a factor 2–4 for compute intensive programs based on the processor performance. For programs that operate from memory the efficiency of the Intel processors is more heavily affected due to the lower bandwidth from the memory to the processor (see section 6.4.1).
   In the integrated parallel systems are doing somewhat better for communication intensive programs, because of a higher bandwidth and lower latency.
   For programs that have a larger fraction of general data manipulation and are not particularly communication intensive, the Intel based clusters can perform at the same level as the integrated parallel systems (see the detailed discussions in section 6 and in section 6.4).

3. With respect to I/O, the clusters showed a performance that was comparable to that of the integrated systems (see section 6.4).

4. For experiments on the IBM Netfinity cluster and the SGI L1200 cluster it became evident that having more CPUs in a node, sharing the node memory has in many cases a detrimental effect on the performance and one should avoid such nodes if cost and density arguments allow it (see sections 6.2 and 6.3.

5. The total outcome of this project leads us to the conclusion that clusters can be useful in an environment where a limited amount of parallel applications is to be run in a user environment that does not change often and/or drastically. This follows from the (still) limited amount of management software and its (still) relatively immature state. However, this situation might be overcome in a time frame of a couple of years.
   The price/performance ratio is in favour of clusters for solutions on a working group or department level (see paragraph above) but this will not hold for larger user communities especially when a larger variety of third party software is required that in many cases is not available for Beowulf clusters.

## Acknowledgments

We like to thank NCF and SARA (Stichting Academisch Rekencentrum Amsterdam) for their financial support. Furthermore, we thank the Compaq, IBM, and SGI for providing both machine and human resources to make this study possible. In particular we want to thank Enda O'Brien and Michael Lough at the Compaq Support Centre in Galway, Ireland for their porting suggestions and executing the benchmark on the AlphaServer SC, Kees Visser from the Computing Centre from Groningen University, The Netherlands for his support when running the benchmark on the Compaq cluster at their site, Nicky Hekster from IBM, The Netherlands for his support and information, Colin Dumontier from the IBM Support Centre in Montpellier for porting and running the benchmark there and in Poughkeepsie, Jan Thorbecke from the Dutch SGI branch for porting and executing the complete benchmark on the SGI L1200 and cluster and also running it on the Origin2000 and also Peter Michielse from SGI, the Netherlands for his support.

# References


[1] R. Buyya (ed.) *High Performance Cluster Computing, Vol. 1 System and Architecture, Vol. 2 Programming and Applications*, Prentice Hall PTR, Upper Saddle River, 1998.

[2] Directory with EuroBen results: `www.euroben.nl/results`.

[3] R. W. Hockney, C. R. Jesshope, *Parallel Computers II*, Bristol: Adam Hilger, 1987.

[4] R.W. Hockney, $f_{1/2}$*: A parameter to characterize memory and communication bottlenecks*, Parallel Computing, **10** (1989) 277-286.

[5] D.V. James, A.T. Laundrie, S. Gjessing, G.S. Sohi, *Scalable Coherent Interface*, IEEE Computer, **23**, 6, (1990),74–77. See also:
Scalable Coherent Interface: `http://sunrise.scu.edu/`

[6] M. Snir, S. Otto, S. Huss-Lederman, D. Walker, J. Dongarra, *MPI: The Complete Reference Vol. 1, The MPI Core*, MIT Press, Boston, 1998.

[7] W. Gropp, S. Huss-Ledermann, A. Lumsdaine, E. Lusk, B. Nitzberg, W. Saphir, M. Snir *MPI: The Complete Reference, Vol. 2, The MPI Extensions*, MIT Press, Boston, 1998.

[8] `http://www.myrinet.com`

[9] IEEE Standard 1596-1992, *IEEE Standard for Scalable Coherent Interface (SCI)*, Inst. of Electrical and Electronics Engineers Inc., New York, USA, August 1993.

[10] D.H.M. Spector, *Building Unix Clusters*, O'Reilly, Sebastopol, CA, USA, July 2000.

[11] A.J. van der Steen, *The benchmark of the EuroBen Group*, Parallel Computing **17** (1991) 1211–1221.

[12] A.J. van der Steen,*The EuroBen Throughput Benchmark Framework*, Technical Report WFI-99-8, Utrecht University, Dept. of Computational Physics, August 1999. (Also available through `www.euroben.nl`, directory `reports/`.)

[13] T.L. Sterling, J. Salmon, D.J. Becker, D.F. Savaresse, *How to Build a Beowulf*, The MIT Press, Boston, 1999.

[14] Mark Baker (ed.) *Cluster Computing White Paper*, July 2000, to be downloaded from `www.dcs.port.ac.uk/ mab/tfcc/WhitePaper/`.