DL/SCI/TM96T

technical memorandum

Daresbury Laborator

DL/SCI/TM96T

HOMOGENEOUS WORKSTATION CLUSTERS FOR PARALLEL CFD

by

R.J. ALLAN, R.J. BLAKE and D.R. EMERSON, SERC Daresbury Laboratory

MAY, 1993

G93/101

Science and Engineering Research Council

DARESBURY LABORATORY

Daresbury, Warrington WA4 4AD

ISSN 0144-5677

Homogeneous Workstation Clusters for Parallel CFD

R. J. Allan, R. J. Blake and D. R. Emerson

SERC, Daresbury Laboratory,
Daresbury, Warrington, WA4 4AD, U.K.

A typical CFD application has been implemented on clusters of identical workstations connected by various local area networks, such as Ethernet, FDDI, Ultranet and SOCC. Performance measurements were made of the code and results on workstation clusters were compared to similar results from loosely coupled multi-processor systems communicating via shared memory. The programming methodology is briefly described, followed by measured system performance parameters and a discussion of the results. It is shown that performance can be predicted even without exclusive access to the processors and network. Some recommendations are made for those considering purchase of workstation clusters.

keywords: workstation clusters, local area network, CFD, domain decomposition, parallel performance, message passing

Introduction

A number of parallel applications in science and engineering are now being developed worldwide and are being run on UNIX workstations coupled by local area networks (LAN). In general, the interconnect is either copper Ethernet or Copper Distributed Data Interface (CDDI). However, optical fibre interconnects, such as Ultranet, Fibre Distributed Data Interface (FDDI) or the IBM Serial to Optical Channel Converter (SOCC), are now beginning to appear more frequently. This is a potentially attractive prospect since workstation cpus are often only partly used interactively, and may sometimes be entirely unused during night-time periods, especially in educational environments. A parallel background job can utilise wasted cpu cycles and return high effective performance, allowing useful research to be undertaken with existing equipment. Manufacturers such as Silicon Graphics, IBM and Hewlett-Packard are now offering clustered workstation solutions to high-performance computing requirements. A further incentive is to make use of different capabilities of networked UNIX hosts, perhaps combining a compute server, a disk server and a high performance graphics engine (e.g. Allan et al. 1991). A number of *ad hoc* methods involving UNIX sockets have been used in the past to achieve this.

A networked workstation cluster is however an inherently dynamical multi-user system and the code developed faces several difficulties if good performance is to be obtained. Extra problems arise moreover in a heterogeneous assembly. Important factors are:

1. Network traffic - which is dynamically varying and can slow down communication between some or all processors
2. Differences in network bandwidth - on a multi-branch LAN
3. Differences in processor speed - in a heterogeneous or quasi-heterogeneous cluster
4. Local cpu load - may change dynamically, e.g. another user may run a big sequential job at high priority
5. Local memory load - may change dynamically, e.g. another user may start a big edit or backup job leading to disk paging and loss of performance
6. A workstation may be switched off or become inaccessible for some reason, requiring checkpointing and program restart with a different processor configuration. We suggest how the checkpoint file can be stored.
7. Different number representations - in a heterogeneous cluster translation between binary representations takes extra time

We have carried out a project designed to study these effects (Garner et al., 1993). In the present paper a typical CFD engineering application code, which was already being used on parallel multicomputers, was ported to homogeneous workstation clusters on the Daresbury Laboratory LAN. Some new performance measures are derived which enable workstation clusters to be evaluated in a systematic way. An attempt is especially made to separate the parallel overhead into two components, which reflect intrinsic network parameters and load. We are thus able to comment on how items 1, 2, 4 and 5 from the above list impact on useful parallel performance, as well as making predictions for larger systems and different interconnects.

The FLOW code and numerical techniques are described by Emerson and Poll (1990), Blake et al. (1992), and Emerson et al. (1992a, b). The parallel programming environment used in this work is an optimised implementation of the Fortnet harness (Allan et al., 1990, Cooper and Allan, 1992 and Allan, 1992). Whilst Fortnet has been specifically developed as an interface for the needs of this kind of portable and dynamical programming, it is possible to use the methods presented here directly with other software such as tcgmsg (Harrison 1991), PVM (Geist and Sunderam, 1992) or PARMACS (Bomans and Hempel 1990, and Hempel 1991).

Example CFD Application: The FLOW Benchmark Code

FLOW is one of a set of reduced application code benchmarks, or kernels, that can be used to assess the suitability of a range of parallel processing systems for solving demanding application problems. The kernels have their origin in production codes that are used in fields such as computational fluid dynamics (CFD), financial transactions, image processing, process control and neural networks.

The CFD benchmark is controlled by a single parameter, NMAX, scaleable in powers of 2, that sets the problem size which can be changed so that breakpoints or threshold points under which the behaviour of the architecture changes can be tested. The code was implemented for concurrent execution from the start, using subdomain partitioning and the communicating sequential process (csp) model of programming (Hoare, 1978). It is implemented in Fortnet to be portable.

Preliminary results have been obtained on a number of multicomputers. The results confirm the expected behaviour that as more processors are thrown at a fixed size problem, the communication costs increase relative to fixed total computation costs (Amdahl, 1967). On the other hand, increasing the job size for a given number of processors increases the efficiency (Gustafson 1988).

The test case concerns the high-speed flow of air over a rectangular cutout (cavity) set in a flat plate (Emerson and Poll, 1990). On a flat plate the flow is characterised by a boundary layer where the velocities of the flow change rapidly from zero at the surface to the free stream values. The presence of a cavity causes the boundary layer to separate and reattach at some point downstream. For the case under consideration the flow will reattach on the downstream cavity face. This type of flow is called "open" cavity flow and produces significant aerodynamic effects, such as fluctuating pressures, noise, heat transfer and increased drag. The study of flow in and around cavities is therefore of fundamental interest to aeronautical engineers.

Numerical schemes must be used to solve the complex set of equations governing this model. The usual prescription is to impose a grid of points on the geometry and then seek a solution in terms of values of the quantities which control the flow, e.g. local velocities, temperature and pressure, at the grid points. The partial differential equations can be translated into sparse algebraic equations using a suitable discretisation scheme. In the FLOW code we use the finite-volume, operator-split approach of MacCormack and Paullay (1972) for discretisation of the 2D Navier-Stokes equations. The field values

between neighbouring grid points are coupled and the system is solved by an explicit time-marching procedure once the initial conditions have been specified. Such time-marching schemes are numerically unstable when regions of high gradients, such as shocks, are encountered. To stabilise the solution a Total Variation Diminishing (TVD) scheme is therefore incorporated into the corrector stage. Further details are described by Emerson and Poll (1990). In the benchmark we march for one hundred time steps, whereas real calculations would require in excess of 10,000 time steps for periodic motion to be set up in the cavity.

In the numerical model the physical geometry is inscribed within a logical rectangle of points. To ease the computations a buffer of two grid points is added to the rectangular domain to accommodate the finite-volume computational stencils on all boundaries. This buffer is referred to as "halo" data. In the code we use a grid which is refined in the regions where we expect the flow variables to change rapidly, e.g. near solid boundaries and in the cavity shear layer. This is topologically equivalent to a regular grid.

The computational process involves: establishing a grid of points fitted to the cavity geometry; initialising the flow field and then calculating the new allowable time step; applying the prediction, stabilisation and correction steps along the y and then x directions. These procedures are explicit in that the calculations of the new value at each grid point can proceed independently. The computations are highly vectorisable and highly parallelisable. Synchronisation of the computation is required before starting another iteration either to calculate global norms of the solutions or to calculate the time step for the next iteration.

The code is parallelised using a grid partitioning strategy. The computational grid contains NMAX active points (excluding the buffer region) along both the x and y axes. The grid is initially cut into equal area squares of size NMAX/NPROCX points along the x axis and NMAX/NPROCY points along the y axis. The total number of processors available, p, is given by the product NPROCX*NPROCY. The grid partitioning is further discussed below.

Implementing a 2D domain partitioning scheme is more difficult than employing just a vertical or horizontal decomposition approach. There are however potential advantages. The amount of data to be transferred in a 1D layer decomposition over p processors is $\propto NMAX * (p-1)$. In a 2D regular block decomposition however it is only $\propto 2 * NMAX * \left(\sqrt{p}-1\right)$ in the symmetric case. This is important since current workstation speeds greatly exceed network speed. Nevertheless, the high-speed networks we have studied benefit from using long message sizes and the effect is reduced for small numbers of processors making a layer decomposition preferable.

The computations require details of both global and local data. The time step for the explicit scheme is calculated by determining the largest flow velocity at any of the grid points in the computational domain. The largest permissible time step is first calculated for each of the subdomains. These values are then circulated to the other subdomains, compared, and the minimum permitted global time step adopted on all subdomains. In a similar manner, an estimation of the norms is made which requires calculation on a

single subdomain before being circulated to the other subdomains and accumulated into global values. This would be used as a test for convergence a steady-state flow.

The stencils used in the computations span two neighbouring grid points. Application of these stencils along the boundaries of the subdomains couples in values at two grid points located in an adjacent subdomain. The subdomains are each padded with two grid points to allow the application of the same stencils at the interior and surface points of the subdomain. This halo data has to be communicated between the edges of the subdomains prior to the finite volume stencils being used. The communications of the halo data is effected by a subroutine which circulates data in the x and y directions independently.

The subdomains are labelled according to two conventions which depend on whether grid based or global data are being exchanged. For the halo exchange the subdomains are mapped onto a two dimensional logical mesh of size NPROCX by NPROCY with indices NPIX and NPIY which describe their relative positions in the global geometry. Exchange of halo data along the x direction takes place between subdomains whose NPIX indices differ by one. Similarly, exchange of halo data along the y direction takes place between subdomains whose NPIY indices differ by one in the initial decomposition. There is no halo data exchange between the left hand and right hand subdomains or the top and bottom subdomains in the two dimensional logical mesh. For global data circulation the subdomains map well onto a logical ring of processors. In passing the halo data for the finite-volume template we attempt to do parallel communication in two steps for each direction. This works if the underlying network has sufficient bandwidth.

Parallel Implementation

The Computation Model. The benchmark codes used in this report are written using the Fortnet portable message-passing interface (see Allan, 1992). The Fortnet abstract model divides a parallel task into a number of computational jobs (concurrent sequential threads which send and receive data to other threads). The communicating sequential process (csp) paradigm (Hoare 1978) is used to transfer data between jobs. The current implementation of Fortnet is classed as a blocked asynchronous harness.

Jobs are logically connected to all other jobs and no fixed topology is assumed in the programming interface. Single or multiple jobs may be placed on each workstation processor. Some jobs which run on one (possibly multiprocessor) workstation may communicate via UNIX shared-memory constructs (e.g. Silicon Graphics or Apollo systems observed in this work). Others communicate using UNIX 4.2BSD sockets. Job placement can be controlled by a configuration file which is read at task startup time. The configuration file also indicates if the standard Ethernet driver, or faster FDDI, SOCC or Ultranet drivers, are to be used. A buffer size can be specified when using socket communication, although it is not accessible from the configuration file. A standard size of 32k bytes was used for the Ethernet and FDDI networks, but it was found to be beneficial to increase this size to 1M byte on both the sending and receiving job when using the Ultranet with long messages.

The UNIX software used in this work as a communication layer below the Fortnet interface is tcgmsg (Harrison, 1991). An alternative, such as PVM (Geist and Sunderam, 1992), could be used. On IBM RS/6000 systems an optimised implementation, called PVMe, using the SOCC is provided (Richelli, 1992). Whilst a version of Fortnet was written as a thin layer interface to PVMe, an alternative to tcgmsg, we did not find any advantage in using it for the present application.

Results

A Measure of Performance. In this application the execution time of the sequential code scales with the size of the grid, NMAX, so $t \propto NMAX^2$. In the ideal parallel implementation, on p identical processors, $t \propto 1/p$ thus in the absence of parallel overhead $\overline{M_p} = NMAX^2/(t_p * p)$ where $\overline{M_p}$ is a constant proportional to the single processor performance. With parallel overhead $\overline{M_p}$ decreases as p increases, and increases as NMAX increases, reflecting Amdahl's and Gustafson's Laws respectively.

The best that we can do therefore is to compare $M = p*\overline{M_p}$ for different combinations of workstations and look for the highest value for a given size of problem.

In considering the times for parallel execution the best performance (i.e. speedup with 100% efficiency) should correspond to using the measured value of M for the single processors as a measure of the speed, s, of that processor. Thus the measured value is

$$\overline{M_p} \leq \sum_{i=1}^{p} M_i/p.$$

and the actual efficiency is obtained by comparing the measured performance (computed as above) to this ideal best performance.

Homogeneous Workstation Clusters. We have observed behaviour of the code on three types of workstation cluster: a system with three IBM RS/6000 model 520s; a system with four HP/9000 model 720s and one HP/9000 model 735; and a system of two Apollo DN10000s having a total of five processors. All systems are linked by the Laboratory local area network (Ethernet) which has high Network File System (NFS) traffic. The IBM 520 workstations are, in addition, coupled by dedicated optical fibre SOCC links and Ultranet. The HP model 720 systems are coupled by FDDI.

It has been possible to measure both the cpu time used by each UNIX process, and the elapsed time from the system clock on which the process is running. Measurements of elapsed time may include interference from other users and network traffic which will cause the process to be interrupted, unless we can guarantee single-user access. This was not possible and we in fact aimed to study a more typical situation in this work. The measured cpu time does not however reflect time spent waiting for system resources, such as sockets, semaphores, i/o etc. We measured this independently in ping-pong tests, and took the best results approximating single user conditions. These are reported in table I using Fortnet with the indicated LAN and interface, together with the ratio of

elapsed time to cpu time for short messages (of relevance in this application) and the asymptotic message passing rates (c.f. manufacturers' literature). These parameters will be used in a system scalability model.

Overall time for message transfer in one direction is given by:

$t = a + bn + c + dn$ microseconds

where n is the number of bytes in the message, a and b parameterise the cpu time, and c and d the elapsed time. This assumes a straight line fit to the data. A straight line is however not strictly correct, see figure 1, and the best performance peaks at between 32k and 1M bytes, roughly suggesting a useful upper limit to the size of system buffers. In the current application messages are less than 4k bytes, and a straight line fit is optimistic, the curve being quite flat until the data terms exceed the latency.

On a busy sysem it is impossible to get an exact picture of the time spent in message passing in a real application, because of the variable elapsed time component. The above results were a best estimate. We therefore quote both average cpu time and total elapsed time in table 3 for the FLOW program to show the state of system load. Typically the elapsed time in communications should be related to the cpu time used by the ratios given in table 1 if there was no other network traffic. The relationship is not however a direct one due to the fact that average cpu times are reported. For a layer decomposition, outer layers pass data only to one other processor, whereas inner layers pass to two, thus average cpu time is proportional to 2*(p-1). The elapsed time however is the maximum rather than the average result and is therefore related to twice the two-processor time for p>2. Here we assume that pairs of processors can interact independently on small systems with no collisions between messages. This assumes that packets are moving very fast on the network and all the overheads are in the device drivers and software protocol. Similar arguments apply to the 2D block decomposition and are summarised in table 2.

In column 5 of table 3 the cpu component of the communication is multiplied by the ratios in tables 1 and 2 to represent the true elapsed time and then the computation time added to give an estimated total run time. This represents the best that can be expected from the given application in a "single user" environment. The percentage utilisation is then the estimated run time compared to the measured run time, and reflects other system or network useage if all our assumptions are true. This is justified by the close agreement between some of the values in columns 4 and 5 of table 3. Only the values for the FDDI are difficult to explain.

In a real system the effect of elapsed time can, in principle, be reduced by allowing the processors to work on other tasks but, if these use the network the situation is made worse. The estimated run time was used to compute the performance measure M reflecting single user and therefore best results.

Column 7 of table 3 contains an estimate of parallel efficiency calculated for this application. It decreases very rapidly as more processors are added due to high apparent network latency.

Multi-Processor Workstations. A number of manufacturers offer multi-processor

workstations to increase potential cpu power. Performance is achieved by using the shared memory to transfer data between processors, which can be done at effective speeds up to 10Mb/s, allowing for software overhead and internal bus contention, or by using parallel compiler technology to unroll the program structure, typically by distributing (tiling) loops.

We have observed behaviour of the code on two systems: a three-processor Apollo DN10000; and a four-processor Silicon Graphics 4D/420 GTX.

Results are presented in table 4 in a similar way to those in table 3. Fortnet is used again as an interface to the tcgmsg software which controls the UNIX shared memory data transfer to simulate message passing between separate jobs in the parallel task. Scaling of such systems is typically better than the network solution due to apparent improved communication performance.

Scaling Laws. If the scaling behaviour of the code is known, as it is in the present case (see table 2), speedup can be estimated based on observation of a two-processor system. The computation time of the code must be known, together with the communication time and communication parameters for the appropriate message sizes, as presented in table 1. In the FLOW code the amount of computation is approximately constant, but the message passing scales in a known way. Based on measured parameters for two processor systems it is therefore possible to predict the performance of systems with more processors, and for larger problem sizes. The predictions can be compared to results of tables 3 and 4. For the HP systems with FDDI with 3, 4 and 5 processors predictions yield M values of 142.7, 173.6 and 197.9 respectively. This is quite close to the best estimated values in table 3. Since the computational work in the code scales as NMAX*NMAX, whereas the communication scales as NMAX, clearly performance is better for larger problem sizes. It is estimated for instance to yield M values of 160, 193, 242, 283 on 2, 3, 4, and 5 HP processors connected with FDDI for a problem of size NMAX=128.

Some differences between the predicted and measured performance arise from imperfect load balancing due to boundary and geometry effects. This is the subject of other work (Garner et al., 1993).

Conclusions

We have reported some results of a project which was designed to study aspects of parallel programming on homogeneous and heterogeneous workstation clusters connected by a local area network. It was found that in the FLOW code, even with a small grid size, useful performance could be obtained from a small number of identical workstations. Performance is limited by both the ultimate speed and the latency of the connecting network and better speedup was found in multiprocessor systems with shared memory. Some systematic measures of performance have been given for this code which can be adapted to others with a similar data decomposition and message passing scheme. Such parameters allow performance of workstation clusters to be predicted from a knowledge

of processor performance and simple message-passing 'ping-pong' tests between two processors.

Whilst it is tempting to make comments about the relative performance of workstation clusters of different types this would not be entirely fair to the manufacturers concerned, due to the hidden effects of the loaded network and multiple users discussed in the text. We therefore leave it to the readers to draw their own conclusions.

Some comments are however relevant to the so-called high-speed networks offered at not negligible expense. Whilst it is true that asymptotic performance of such a network can be good (several times better than Ethernet), there is a large class of application which transfer many relatively short data packets between the parallel components, say less than 2k bytes. For these applications the high message latency, or startup cost, of the higher speed network can mean little or no improvement over the standard Ethernet connection. A large part of this latency arises from interaction with the UNIX operating system, and a solution needs to be found using light-weight processes.

As far as absolute performance goes, only a comparison of the run times with contemporary supercomputers is meaningful. The FLOW code, taking the same parameters as in table 2, delivers a flow mark of M=1591.7 on a single Cray Y-MP/8I processor. This is equivalent to 129.3 Cray MFlops, a ratio of 13:1. It is expected that on workstations a ratio of around 10:1 is likely due to the lack of square root hardware etc. The Cray is delivering only around half its peak performance, probably because of indirect addressing in certain areas of the code which is not fully optimised for vector hardware. Nevertheless the single processor workstation performance is disappointing by comparison.

Workstations are designed as interactive desktop machines for multi-user access. They should be connected only to share files or run facilities such as X-windows connecting a graphical server to a high-performance computational device. Use of such a resource as a parallel compute engine is unlikely to be cost-effective except in trivial coarse-grained cases and embarassingly parallel methods, such as Monte-Carlo, and can only be justified to absorb otherwise unused cycles, and even then the performance may be disappointing. For high-speed parallel computation a closely coupled multi-processor system is currently preferable.

Acknowledgements

We acknowledge help from the vendors, in particular loans of HP and IBM workstation clusters.

We thank Dr. S. Zurek, Mr. W. Purvis, and Mr. J. Garner for their work on development of Fortnet, and Dr. R.J.Harrison for allowing us to use and adapt the tcgmsg software.

The FLOW code is a CFD benchmark developed under contract for NPL, and forms part of their PEPS suite (EC Esprit III funded project 6942).

References

Allan, R.J. (June 1992) "Fortnet v4.0: the Parallel Programming Software" Daresbury Laboratory

Allan, R.J., Durham, P.J. and Guest, M.F. (1991) "Networked Computer Power" Physics World 4 51-4

Allan, R.J., Heck, L. and Zurek, S. (1990) "Parallel Fortran in scientific computing: a new occam harness called Fortnet" Computer Physics Communications. 59 325-44

Amdahl, G. (1967) "Validity of the single-processor approach to achieving large-scale computer capabilities" AFIPS Conference Proc. vol. 30

Blake, R.J., Emerson, D.R. and Allan, R.J. (1992) "FLOW: a Parallel Benchmark Code for High-speed Air Flow: version 1" Daresbury Laboratory. Documentation produced under contract to NPL.

Bomans, L. and Hempel, R. (1990) "The Argonne/GMD macros in Fortran for portable parallel programming and their implementation on the Intel iPSC/2" Parallel Computing 15 119-32

Cooper, R.K. and Allan, R.J. (1992) "Fortnet (3L) v1.0: Implementation and extensions of a message passing harness for transputers using 3L Parallel Fortran" Computer Physics Communications 70 521-43

Emerson, D.R., Blake, R.J. and Allan, R.J. (1992) "Implementation of a Navier-Stokes Solver using Fortnet: Implementations and Results" in 'Parallel CFD '91' (Elsevier Science Publishers bv)

Emerson, D.R., Blake, R.J. and Allan, R.J. (1992) "CFD Experiences on a Range of Novel Architecture Systems: Implementations and Results" in 'Parallel CFD '92' (Elsevier Science Publishers bv) submitted

Emerson, D.R. and Poll, D.I.A., (1991) "High-Speed Laminar Flow over Cavities" in 'Applications of Supercomputers in Engineering II' (Elsevier, Applied Science Series)

Garner, J., Allan, R.J., Blake, R.J. and Emerson, D.R. (1993) "Dynamic Load Balancing in heterogeneous workstations clusters" for submission to Journal of Scientific Computing

Geist, G.A. and Sunderam, V.S. (1992) "Network-based Concurrent Computing on the PVM System" Concurrency 4 293-311

Gustafson, J.L. (1988) "Re-evaluating Amdahl's Law" ACM 31 number 5 532-3

Harrison, R.J. (1991) "documentation of tcgmsg harness" Argonne National Laboratory and private communication

Hempel, R. (November 1991) "The ANL/GMD Macros (PARMACS) in Fortran for Portable Parallel Programming using the Message Passing Programming Model: User's Guide and Reference Manual, v5.1" GMD mbH, Sankt Augustin, W.Germany

Hoare, A.C.R. (1978) "Communicating Sequential Processes" Comm. ACM 21 (8) 666–77

MacCormack, R. W. and Paullay,A. J. (1972) "Computational Efficiency Achieved by Time Splitting of Finite Difference Operators", A.I.A.A. Paper 72–154.

Leffler, S.J., Fabry, R.S. and Joy, W.N. (19??) "A 4.2BSD Interprocess Communication Primer" Computer Systems Research Group, Department of Electrical Engineering and Computer Science, University opf Berkley, California

Richelli, G. (1992) "PVMe User Guide, Aix 3.2 version" ECSEC Rome

Tables

Table 1. Performance of a ping-pong test between two nodes of the given homogeneous systems. All times are in microseconds unless stated otherwise.

system	a	b	c	d	asymptotic rate [Mb/s]	ratio elapsed/ cpu
IBM system 520s						
ethernet	1540	0.469	1571	0.794	0.792	1.02
SOCC	2010	0.096	1595	0.289	2.60	0.793
PVMe setxmode(1)	630	0.186	8000	1.004	0.840	1.64 *
PVMe setxmode(2)	1620	0.259	1220	0.272	1.88	0.753 *
PVMe setxmode(4)	1300	0.219	2650	0.463	1.466	2.04 *
Ultranet	4996	0.0371	3444	0.0803	8.74	0.689
HP system 720s						
ethernet	361	0.149	375	0.917	0.939	1.04
FDDI	348	0.147	1135	0.206	2.83	3.26
other systems						
apollos	2010	0.548	7370	2.46	0.33	2.82
multi-processor systems using UNIX shared memory						
iris2	995	0.128	410	0.0999	4.39	0.78
apollo2	905	0.0838	285	0.0861	5.88	1.03

*) The SETXMODE command was introduced into PVMe (Richelli, 1992) to bypass PVM daemon processes and control the synchronicity of access to the SOCC driver in IBM installations.

Table 2. Proportionality factors for FLOW.

p	comm cpu	best elapsed
2x1	1	1
3x1	4/3	2
4x1	3/2	2
5x1	8/5	2
6x1	5/3	2
2x2	1	2
3x2	7/6	3

Table 3. Average performance measure and efficiency for homogeneous workstation clusters in runs of 100 iterations of a problem of size nmax=64. Key to machine types is given below.

Columns are comp: average computational cpu time; comm: average comunication cpu time; elaps(m): measured total elapsed time; elaps(e): total run time estimated using tables 1 and 2 (see text); %util: ratio elaps(e)/elaps(m); %eff: parallel efficiency; $\overline{M}$ average performance value; M total perfomance value as described in the text. All times are in seconds.

	comp	comm	elaps (m)	elaps (e)	% util	% eff	$\overline{M}$	M
HP/9000 systems with Ethernet								
tcs-hp1	41.34	0.04	80.30	41.4	51.6			98.99
2x1	20.56	2.76	32.05	26.2	81.2	79.0	78.2	156.4
3x1	13.34	3.5	29.87	20.5	68.6	67.3	66.7	200.0
4x1	9.94	3.68	35.1	20.2	57.5	51.2	50.7	202.8
2x2	9.44	4.1	26.4	17.8	67.3	58.1	57.3	230.1
HP/9000 model 720 workstations with FDDI								
2x1	20.46	2.33	29.79	30.39		68.1	67.4	134.8
3x1	13.36	3.2	22.7	26.9		51.3	50.8	152.3
4x1	9.9	3.26	25.3	23.79	94.0	43.5	43.0	172.2
2x2	9.59	3.93	21.5	26.3		39.3	38.9	155.7
IBM RS/6000 model 520 workstations with ethernet link								
tcs-ibm2	61.57	0.18	92.65	61.75	66.6			66.33
2x1	29.31	6.72	59.2	42.9	72.4	72.0	47.8	95.52
3x1	19.95	9.79	59.2	39.7	67.1	51.9	34.4	103.1
IBM 520 workstations with Serial to Optical Channel Converter								
2x1	29.26	5.83	59.0	39.7	67.3	77.7	51.6	103.2
3x1	19.24	8.64	92.6	34.7	37.5	59.3	39.4	118.0
IBM 520 workstations with Ultranet								
2x1	29.16	12.84	52.07	50.9	97.7	60.7	40.3	80.6
3x1	19.76	16.99	70.37	48.5	68.9	42.5	28.2	84.5
Apollo DN10000 multi-processor workstations with ethernet link								

apollo1	56.46	0.14	56.6		100			72.37
apollos 1+1	29.57	10.07	75.88	68.0	89.7	41.6	30.1	60.2
apollos 2+1	19.17	9.87	67.58	56.9	84.2	33.2	24.0	72.0
apollos 3+1	15.52	10.79	71.27	56.7	79.6	29.4	18.0	72.2
apollos 2+2	14.87	9.58	66.09	57.5	77.9	27.5	19.9	79.6
apollos 3+2	14.82	10.73	84.28	55.8	66.2	20.3	14.7	73.4

Key to systems:

tcs-hp0 — Hewlett Packard model 750
tcs-hp1 to 4 — Hewlett Packard models 720
HPs — cluster of one Hewlett Packard model 750 and the four model 720 machines linked by Ethernet and FDDI. All i/o goes to a RAID disc system on tcs-hp0 resulting in heavy ethernet traffic under typical conditions
apollo1 — Apollo DN10000 3 processor model
apollo2 — Apollo DN10000 2 processor model
Apollos — cluster of the two DN10000 systems linked by Ethernet, with a total of 5 processors
tcs-ibm2 to 4 — IBM models 520
IBMs — cluster of the three IBM model 520 machines linked either by Ethernet, SOCC optical connection or Ultranet

Code on all IBMs was compiled using xlf -O. Code on HPs uses f77 +O3 +OP4 +OPunroll. Code on Apollos f77 –O.

Table 4. Average performance measure and efficiency for multi-processor workstations as in table 3.

	comp	comm	elaps (m)	elaps (e)	% util	% eff	$\overline{M}$	M
Silicon Graphics system, iris2								
1x1	90.08	0.14		90.22				45.47
2x1	49.1	5.84	63.19	59.5	94.2	75.7	34.4	68.8
3x1	36.21	6.94	54.17	48.6	89.7	61.8	28.1	84.3
2x2	27.38	4.12	93.07	34.7	37.3	64.9	29.5	118.0
4x1	32.60	9.38	58.35	49.3	84.5	45.7	20.8	83.1
Apollo DN10000 system apollo1								
1x1	56.46	0.14		56.6				72.37
2x1	29.94	4.70	39.78	39.5	99.2	71.7	51.9	103.7
3x1	18.97	5.24	31.54	29.6	93.9	63.7	46.1	138.4

Key to systems as in table 2 except:

1. iris2 — Silicon Graphics 4D/420 GTX four-processor model

Code on SUNs uses f77 –O3 and on SGIs f77 –O2 and on Apollos f77 –O.

Comment: The SGI operating system automtically balances processor load by swapping all jobs. The system has four processors of differing speed, but this cannot be seen from any tests we made.

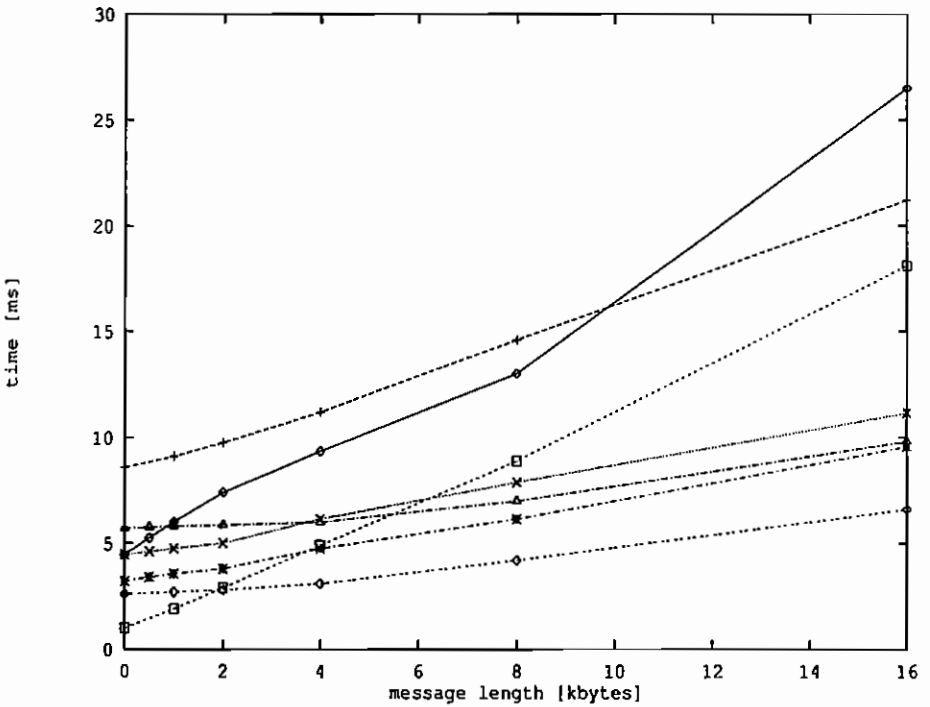

Figure 1. Time to transfer data packets of different sizes between two processors on a Local Area Network.

Key to networks and workstations

◇ IBM model 520 using tcgmsg and Ethernet
\+ IBM model 520 using PVMe with setxmode(1) and SOCC
⊡ HP/9000 model 720 using tcgmsg and Ethernet
× IBM model 520 using PVMe with setxmode(4) and SOCC
△ IBM model 520 using tcgmsg and SOCC
* IBM model 520 using PVMe with setxmode(2) and SOCC
♦ HP/9000 model 720 using tcgmsg and FDDI

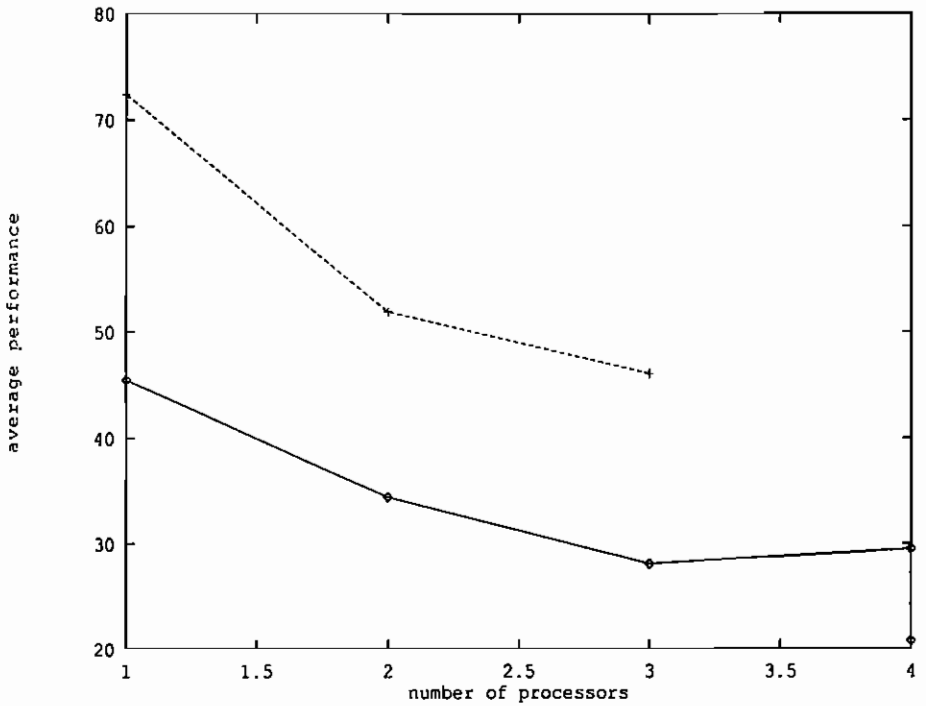

Figure 2. $\overline{M}$ value for the different problem sizes and processor combinations using shared memory as reported in table 3.

Key to workstations
+ Apollo DN10000
◇ Silicon Graphics model 4D/420 GTX

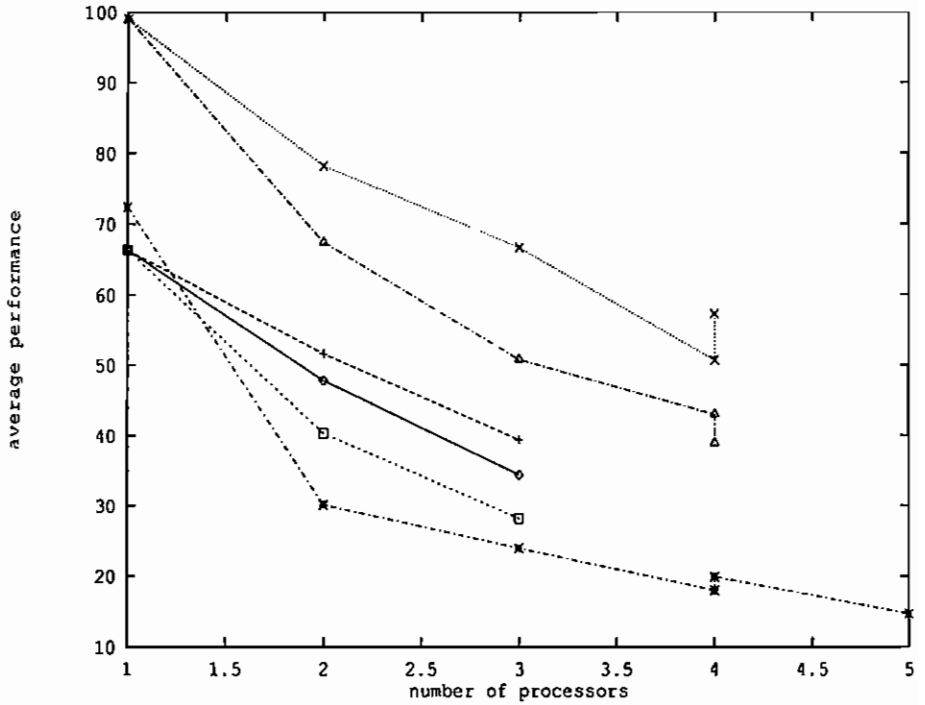

Figure 3. $\overline{M}$ value for the different problem sizes and processor combinations as reported in table 4.

Key to networks and workstations
× HP/9000 model 720 using Ethernet
△ HP/9000 model 720 using FDDI
\+ IBM model 520 using SOCC
◇ IBM model 520 using Ethernet
⊡ IBM model 520 using Ultranet
* Apollo DN10000 using Ethernet