

Comparative Performance of a Commodity Alpha Cluster running Linux and Windows NT

David Lancaster
Electronics & Computer Science
University of Southampton
Southampton SO17 1BJ, U.K.
djl@ecs.soton.ac.uk

Kenji Takeda
Aeronautics & Astronautics
University of Southampton
Southampton SO17 1BJ, U.K.
ktakeda@soton.ac.uk

Abstract

Using a cluster of commodity Alpha processors we compare two software platforms based on Linux and Windows NT and intended to support intensive scientific computations. Networking and compiler performance are separately analysed and then results for NAS parallel benchmarks are given. We find that a compiler able to make good use of the cache is more important than low network latency in obtaining high performance. We argue that for all types of cluster, the choice of compiler is critical in selecting a cost effective platform for computationally intensive scientific applications.

1. Introduction

Over the last few years several institutions have constructed high-performance clusters of commodity processors, often known as Beowulf systems [1] and it has become clear that powerful systems can be put together at a fraction of the cost of proprietary supercomputers. Many of the systems so far constructed have been based on personal computers using Intel processors. Southampton University has recently taken a different route by installing a cluster based on eight Compaq/DEC Alpha workstations. The Alpha is a commodity processor priced to compete with Intel processors and was selected because of its good price/performance for computationally intensive scientific and engineering codes. Each node of the cluster contains a 500MHz AXP21164A processor on a PC164 motherboard with 256MB of main memory. The nodes are connected with 100Mb/s switched fast Ethernet.

The policy of using cheap commodity parts is not limited to the hardware, the operating system and compilers must also come under a strict budget. This has generally meant that Beowulf systems run Unix-like operating sys-

tems such as Linux with little or no cost. Cost has not been the only issue in this choice, source code availability has been important to enable code modifications which facilitate parallel computation on these systems. In an industrial setting many other considerations come into the choice of operating system but in this article we shall only consider purchase cost and performance.

For the Alpha platform, we propose that Windows NT Workstation could be a cost effective choice. Certainly, Windows NT is not an expensive operating system and is often supplied as standard with new systems, but the reason that it is a serious contender comes from a consideration of the compiler. Our aim is to run computationally intensive scientific codes in Fortran with message passing (we shall only consider MPI in this article) so a good Fortran compiler is essential. The GNU f77 Fortran compiler that is usually installed with a Linux distribution is a front end to the GNU C compiler and is unable to take advantage of certain aspects of Fortran. Other commercial compilers are available on the Linux/Alpha platform [2] but we shall only consider the EGCS Fortran compiler [3]. Without any doubt, the best compiler for the Alpha is Digital Visual Fortran, made by the manufactures of the chip. Digital Fortran is not cheap on the Tru64 UNIX operating system which itself is expensive, however this compiler is available under Windows NT for a modest cost.

The Southampton system has been set up with the Digital Visual Fortran compiler on Windows Workstation NT 4.0. It is the purpose of this article to assess the effectiveness of this choice in comparison with the more usual choice of Linux and the EGCS compiler. We have set up a diskless system using Linux kernel 2.0.36 for this purpose. We compare the most important influences on overall performance, that is, the compiler and message passing behavior for both Windows NT and Linux systems. We present careful measurements of the Fortran compiler performance using Digital Visual Fortran in the case of Win-

dows NT and the latest version of EGCS for Linux. We use the standard benchmarks of Linpack [4] and Livermore Loops [5] and also compare the instructions generated for a simple dot product code in order to analyse the differences. For message passing we first compare the low-level performance of the network and protocol stack using TCP stream tests. We then consider the MPI message passing library built on top of this layer. For Windows NT we employ the MPI Software Technology Inc. version known as MPI/Pro [6] whereas with Linux we use the standard MPICH distribution based on the `ch_p4` device. We finally consider the NAS parallel benchmark suite [7] and investigate the influence of the operating system/compiler combination on overall performance.

2 Network Performance

We start by reporting on the low-level performance of the TCP/IP networking that is provided with each operating system. The test we have employed is based on sending a stream of TCP packets from one machine to another. This test, called “netperf”, is freely available [8] and measures the bandwidth for various size packets. In figure 1 we show a plot of the performance for both operating systems. Although the Linux networking is clearly superior, the Windows NT performance is adequate over most of the range of packet sizes. This is a well known problem that WNT 2000 is expected to address (though we have not seen any improvement in the beta 2 release). In the tests used for the figure, socket sizes were set to their default values: 64kB for Linux and 8kB for Windows NT. When the measurements are repeated with a common socket size of 32kB, the Linux curve hardly changes whereas the Windows NT curve is noticeably worse.

In order to obtain the performance shown above for Linux, it was important to set up the network card driver (tulip) by disabling the autonegotiate feature and setting 100baseTx FD protocol by hand. No such technical procedures were needed for Windows NT.

For the scientific codes we are considering, MPI provides a suitable API for message passing, so we now turn to consider the network performance at this level. For Windows NT on Alphas there is only one choice of implementation at the time of writing (Q1 1999), known as MPI/Pro [6] commercially produced by MPI Software Technology Inc. We have used various releases, but the figures reported here are given for version 1.2.3. With Linux we use the standard MPICH distribution [9] based on the `ch_p4` device. Many benchmark tests are available to test the performance at this level. The well-known “PingPong” tests are all quite similar and we choose to use the latest version of the Genesis tests written in Southampton [10]. The tests

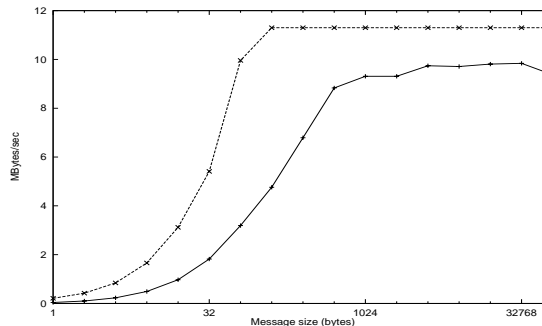


Figure 1. Bandwidth for TCP stream test (netperf) between two different hosts. Solid curve is for Windows NT, dashed for Linux.

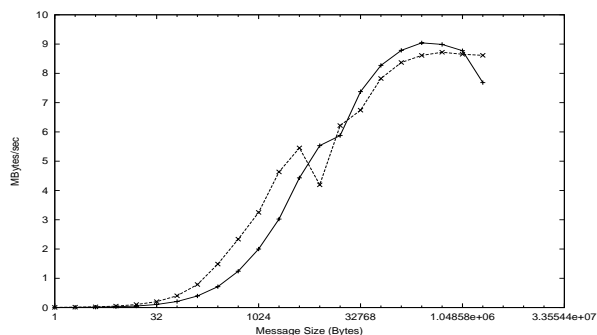


Figure 2. Bandwidth for MPI PingPong test between two different hosts. Solid curve is for Windows NT, dashed for Linux.

perform a non-interleaved (messages only travel in one direction at a given instant) send and receive pattern with simple `MPI_Send` and `MPI_Receive` calls and provides rates based on the time for a single message interchange. This type of measurement has the effect of averaging over the behaviour reported in [11] in which every 35th packet was substantially slower. We have tested performance with the patch of the TCP stack recommended in [11], but find that on Alpha this has the opposite effect, improving the large message bandwidth at the expense of latency.

The bandwidth is shown for each operating system in figure 2. The measurements obtained from these tests also allow us to determine the latency by plotting the messages times against packet size for small packets and this is shown in figure 3.

Although the present implementation of MPI for Win-

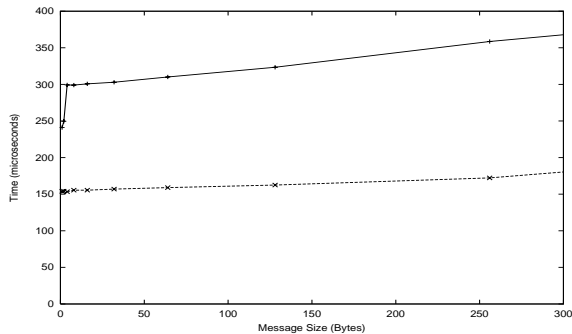


Figure 3. Latency for MPI PingPong test between two different hosts. Solid curve is for Windows NT, dashed for Linux.

Windows NT gives a bandwidth that is very similar to that of MPICH on Linux, the latency is considerably worse. The vendors of MPI/Pro have recognised this problem, and through reducing the number of buffer copies, this product has already improved over the past versions. Now they have an implementation based on VIA technology[12] that has substantially lower latency[13], and their TCP/IP versions are expected to reflect some of the optimisations from the VIA work.

Further tests of MPI find that the Windows implementation does not support bidirectional messages very well. Although the Linux based MPICH appears good, improvements have been reported by the Avalon group [14]. Also a version of MPICH based on a faster channel device called “fast messages” is available [15]

3 Compiler Performance

As was explained in the introduction, a motivating factor to use the Windows NT operating system was the availability of Digital Fortran at a reasonable price. In this section we compare the performance of Digital Fortran with the EGCS Fortran compiler [3] running on Linux. We consider various single node benchmarks that test performance for scientific and engineering codes. The compiler versions we use are Digital Fortran 5.0c and EGCS release 1.1.2.

The LINPACK [4] benchmark provides a good baseline performance figure with which to start. In table 1 we show the Mflops rates for problems with matrix size 1000×1000 which is sufficiently large to completely overwhelm the cache. It was important to use the EGCS `-funroll-loops` compiler optimisation flag to obtain this performance on Linux. Under Windows NT, we use the “full optimisation” choice in the Developer Studio interface

| | |
|----------------------|-------------|
| EGCS g77, Linux | 32.4 Mflops |
| Digital Fortran, WNT | 32.3 Mflops |

Table 1. Linpack performance for a 1000×1000 matrix using EGCS g77 under Linux and Digital Visual Fortran under Windows NT.

| | WNT | Linux |
|----------------|-----|-------|
| Maximum Rate | 476 | 191 |
| Quartile Q3 | 133 | 98 |
| Average Rate | 126 | 74 |
| GEOMETRIC MEAN | 98 | 58 |
| Median Q2 | 84 | 66 |
| Harmonic Mean | 80 | 41 |
| Quartile Q1 | 57 | 33 |
| Minimum Rate | 30 | 7 |

Table 2. Livermore Loop performance using EGCS g77 under Linux and Digital Visual Fortran under Windows NT. Figures are in MFlops.

and tune for the ev5 Alpha architecture.

The performance is essentially the same, and indeed it would be strange if the compilers produced very different code for this simple benchmark. This test merely assures us that we are using the compilers correctly and a more useful test of the compilers requires more complex code. A well-known example is provided by the Livermore Loops [5] benchmark which is based on 24 kernels representing different typical code sequences. We use the same optimisations as for the LINPACK tests and show the results in table 2.

The behaviour of the EGCS compiler now appears much worse. Some insight into this result may be obtained from studying the timings for each of the 24 kernels. In the majority of cases, the timings are quite similar, but for a small number of kernels the Digital compiler gives considerably faster times. The kernels for which this occurs are: Hydro, Equation of State, ADI Integration, Integral Predictors, First Differential, 2D Hydro, and Planck Distribution.

To clarify the origins of the different compiler performance, we have looked carefully at a simple dot product code that can be analysed in depth. The code performs a dot product of two vectors of various lengths over a range that covers the L1-D cache size (8192 bytes) [16]. We find two broad classes of behavior depending on whether the vector length fits in cache. The computation is separately timed for four implementations of the central loop of the Fortran code including the naive loop and three versions hand un-

| 2048 bytes | Digital | EGCS |
|-------------|---------|------|
| Simple loop | 170 | 210 |
| 2× unrolled | 280 | 340 |
| 4× unrolled | 470 | 400 |
| 8× unrolled | 500 | 310 |
| 16384 bytes | | |
| Simple loop | 150 | 120 |
| 2× unrolled | 290 | 180 |
| 4× unrolled | 450 | 200 |
| 8× unrolled | 360 | 180 |

Table 3. Performance (in MFlops) for four versions of a dot product: the simple original code and hand modified versions unrolled by 2, 4 and 8 times respectively. Shown for vector lengths of 2048 bytes and 16384 bytes to illustrate typical below and above cache size (8192 bytes) behaviour.

rolled by 2, 4 and 8 times. Higher degrees of unrolling require more registers for the dot product sum and allow more freedom in rearranging and optimising the code. The compilers perform further unrolling steps in generating assembler code and we find that each compiler implements a different policy.

After experimentation we have chosen the optimisation flags: `-O3 -funroll-loops` for the EGCS and `/tune:ev5 /optimize:5` for the Digital compiler. With these compiler flags we find the timings shown in table 3 for two vector lengths chosen to display typical above and below cache behaviour.

For orientation, recall that the clock cycle is 2ns, and that the Alpha chip is potentially able to initiate two floating point operations in one cycle, so peak performance is about 1000 MFlops. The dot product cannot be fully optimized and the best we approach is 500 MFlops for the Digital compiler on the 8 times unrolled code for vectors that are able to fit into cache.

Some insight into these performance figures comes from an inspection of the assembler code generated in each case. Each compiler has a different policy: the Digital compiler always unrolls by a further 4 times whereas the EGCS compiler only unrolls to give a maximum of 8 iterations (it unrolls the simple and 2× hand unrolled code by 4, the 4× hand unrolled code by an additional 2 times, but does not perform any unrolling on the 8× hand unrolled code)[17]. The reason for this difference in policy may come from the more efficient (in terms of register use) addressing algorithm present in the Digital compiler, which could ultimately arise from the fact that it was initially designed for

Fortran rather than C.

A more important difference is apparent when comparing the timings for vectors that either fit in or do not fit in cache. Whereas the figures are not too dissimilar for small problems indicating differences in pipelining efficiency, Digital Fortran is significantly better than EGCS for large problems. It is clear that the Digital compiler is taking advantage of detailed knowledge of the system to the extent of inserting padding operations and improved operator ordering that allow better use of the cache.

Although it is dangerous to extrapolate from tests on such simple codes [18], this study has pointed to some of the possible causes of the difference in compiler performance. It appears to be common knowledge that while the gcc compiler usually generates the same floating operations as commercial compilers, its management of registers is not as sophisticated and therefore the address calculations tend to be less well optimized. We have also learned from this test that the cache usage is much less sophisticated than with the Digital compiler. A discussion of some of these issues for the Intel platform is given in [19].

4 NAS Parallel Benchmarks

To see how the effects of compiler and networking combine in influencing the performance of more substantial codes we have considered the NAS parallel benchmarks version 2.3[7]. Our setup only has two of the eight machines equipped with extra hard disks supporting Linux, so for the measurements reported here an nfs-root system is employed for the additional nodes. In comparing bandwidths, the nfs-root system did not appear to give rise to undue bandwidth contention with the MPI.

We show the number of (million)operations/sec/processor in the table 4. We consider problem size A and find that the problem sometimes does not fit into the memory of a single processor (256 MB). All timings vary slightly on repeating the measurement, and the results, on both operating systems, typically have a spread of ± 0.1 MOp/s/proc. In the case of LU with Linux, these fluctuations were noticeably larger amounting to errors up to the order of 5%. Some of the codes, notably BT, SP only run on a number of processes that is a perfect square, and unfortunately we do not have nine nodes. Results for EP are only shown for one processor because this test scales almost perfectly. The FT benchmark cannot be compiled using the EGCS compiler as it uses some F90 constructs. The IS benchmark is written in C and we use the EGCS gcc compiler on Linux and Visual C++ v5.0 on Windows NT. We use a version of CG that is explicitly unrolled twice for reasons that are clear from the previous section. The results of two kernels, CG and EP depend on the choice of random number generator and we have used “randdp”.

Results for the simple tests EP and CG can be interpreted in terms of the behaviour of compiler and messaging that we have already considered: the compiler on WNT is better, but the messaging is better on Linux. This is apparent in the good scaling under Linux of the communication intensive CG code and the difference in performance for the EP code which requires no communication.

For all the other more complicated codes it would appear that the compiler is the dominating influence. LU is particularly interesting. On Linux the performance scales almost linearly reflecting the fact that the latency is low since this code passes many short messages (results for LU on Linux suffer from anomalously large fluctuations when repeated). On WNT, the raw performance is higher, but the scaling is noticeably super-linear up to 4 nodes. This indicates that the Digital compiler is making much better use of the cache than EGCS is able to. This would appear to fit with the observations made on the dot product test code. A similar effect is observed with MG.

IS is an integer sorting code written in C and scales very badly on both operating systems, for WNT the scaling appears to be quite erratic.

5 Conclusion

We have presented an argument for using the Windows NT operating system on a cluster of Alpha workstations to be used as a cheap scientific computing platform: the availability of the efficient Digital Fortran compiler at a reasonable price. The results we have shown indicate that although the Windows NT platform suffers from worse messaging performance, particularly at the level of the latency, this does not outweigh the advantages of the compiler on clusters of the size we have considered. This is an interesting observation in the light of the frequent insistence that low latency is the key to good parallel performance. The historical reason for this emphasis on low latency is that in the past commercial distributed processor machines came with compilers tuned for those architectures. Nowadays, when attempting to build cost effective commodity clusters based on either Alpha or other processors, the choice of compiler is critical. It is not surprising that the Digital compiler is better able to take advantage of the architectural features of the Alpha chip, but our study of assembler points out two specific areas in which it outperforms the EGCS compiler. Firstly the Digital compiler has a better addressing mode which allows for more aggressive unrolling and secondly the operator ordering and padding generated by the Digital compiler allow for better cache usage. It is this second feature which appears to have nullified the effect of worse latency in realistic parallel codes, even leading to superlinear scaling.

| | Number of Nodes | Mop/s/process | |
|-----------|-----------------|---------------|------|
| | | LINUX | WNT |
| EP | 1 | 1.8 | 2.4 |
| CG | 1 | 18.7 | 19.7 |
| | 2 | 19.0 | 18.5 |
| | 4 | 12.2 | 12.4 |
| | 8 | 11.4 | 8.6 |
| MG | 4 | 31.0 | 34.0 |
| | 8 | 31.4 | 38.8 |
| LU | 1 | 48.5 | 60.8 |
| | 2 | 48.7 | 62.1 |
| | 4 | 48.1 | 65.2 |
| | 8 | 46.9 | 59.5 |
| SP | 1 | 31.6 | 36.6 |
| | 4 | 26.1 | 28.5 |
| BT | 4 | 38.5 | 46.3 |
| IS | 1 | 3.72 | 3.9 |
| | 2 | 1.64 | 0.96 |
| | 4 | 0.88 | 1.16 |
| | 8 | 0.60 | 0.72 |

Table 4. Performance of the NAS parallel benchmarks (NPB2.3) using EGCS g77 under Linux and Digital Visual Fortran under Windows NT. Figures are given in MOps/s per processor and are subject to measurement fluctuations of order ± 0.1 MOps/s except as mentioned in the text.

| | Linux |
|----------------|-------|
| Maximum Rate | 491 |
| Quartile Q3 | 186 |
| Average Rate | 142 |
| GEOMETRIC MEAN | 111 |
| Median Q2 | 91 |
| Harmonic Mean | 91 |
| Quartile Q1 | 62 |
| Minimum Rate | 35 |

Table 5. Livermore Loop performance using Compaq Fortran under Linux. Figures are in MFlops.

As we insisted in the introduction, we do not treat all the other considerations that come into deciding which operating system to use on a cluster. In the restricted terms of performance and purchase price, it appears that at present the advantages of the Digital compiler available with WNT outweigh the problems with the MPI for parallel applications. However this is not a static situation, and the measurements only represent the capabilities of the software on the system at this time. Indeed, in earlier versions of this report the conclusions were the opposite! In the longer term we expect that both commodity message passing (both software and hardware) and compilers will improve. For message passing, improvements will come from VIA [12]. Compaq has announced that it will be supplying a compiler tuned for Alpha under Linux. Please see the note added concerning results obtained with the Compaq Fortran compiler under beta release.

6 Compaq Fortran

The availability of the Compaq Fortran compiler for Linux under a beta release program [20] has again changed the picture. In tables 5 and 6 we present some preliminary results. We use the optimisation flags `-O -fast` and were obliged to recompile the MPI library. Shared libraries are not used for parallel applications.

Serial performance measured by the Livermore Loops benchmark exceeds that achieved with Windows NT. Results for NAS parallel benchmarks indicate slightly better node performance and generally have better scaling than for Windows NT. However, LU continues to suffer from the anomalous fluctuations in measured performance noted earlier and still does not match the performance seen under Windows NT. We are still investigating this problem.

| | Number of Nodes | Mop/s/process LINUX |
|-----------|-----------------|---------------------|
| EP | 1 | 2.5 |
| CG | 1 | 20.7 |
| | 2 | 20.4 |
| | 4 | 13.8 |
| MG | 8 | 10.8 |
| | 4 | 35.0 |
| LU | 8 | 37.3 |
| | 1 | 56.3 |
| SP | 2 | 57.4 |
| | 4 | 55.2 |
| | 8 | 55.0 |
| BT | 1 | 36.8 |
| | 4 | 29.0 |
| BT | 4 | 47.3 |

Table 6. Performance of the NAS parallel benchmarks (NPB2.3) using Compaq Fortran under Linux. Figures are given in MOps/s per processor and are subject to measurement fluctuations of order ± 0.1 MOps/s except as mentioned in the text.

Acknowledgments

We would like to acknowledge support from MPI Software Technology Inc and Microsoft Research. We are also grateful for assistance from Simon Cox, Jonathan Hardwick and Denis Nicole.

References

- [1] D. Ridge, D. Becker, P. Merkey and T. Sterling. *Beowulf: Harnessing the Power of Parallelism in a Pile-Of-PC's*. Proc. 1997 IEE Aerospace Conference.
See the Beowulf Project page at CESDIS:
<http://cesdis.gsfc.nasa.gov/linux/beowulf/beowulf.html>
- [2] We are aware of products by NAG and Microway. Compaq has announced that it will be supplying a commercial compiler for Linux/Alpha in the near future.
- [3] EGCS is step in the development of GCC, the GNU C compiler. The EGCS steering committee has now been appointed the official GNU maintainer for GCC. The compiler includes a front end to Fortran.

<http://egcs.cygnus.com/>

- [4] J.J. Dongarra, *Performance of Various Computers Using Standard Linear Equation Software*, Report CS-89-85, Univ. of Tennessee, Knoxville, Nov. 1996. The Linpack benchmark is available from:
<http://www.netlib.org/>
- [5] F.K.McMahon, *The Livermore Fortran Kernels: a Computer Test of Numerical Performance Range*, Lawrence Livermore National Lab., Technical Report UCRL-53745, 1986. The Livermore Loops benchmark is available from:
<http://www.llnl.gov/>
- [6] MPI/Pro is available from MPI Software Technology Inc.
<http://www.mpi-softtech.com>
- [7] D. Bailey, T. Harris, W. Saphir, R. Wijngaart, A. Woo and M. Yarrow. *The NAS Parallel Benchmarks 2.0*. NAS Report NAS-95-020, December 1995. NAS Parallel Benchmarks are available from:
<http://science.nas.nasa.gov/Software/NPB>
- [8] Information on Netperf, a Network Performance Benchmark, is available at:
<http://www.cup.hp.com/netperf/NetperfPage.html>.
- [9] Message Passing Interface Forum. *MPI: A Message-passing Interface Standard*, 1994.
<http://www.mpi-forum.org/docs>
MPICH, a portable and freely available implementation of MPI may be found at:
<http://www.mcs.anl.gov/mpi/mpich/>
- [10] A.J.G. Hey and D. Lancaster, *The Development of Parkbench and Performance Prediction*. Proceedings of the first NASA Workshop on Performance-Engineered Information Systems, October 1998. To be published in International Journal of High-Performance Computing. The Southampton Lowlevel communications benchmarks using MPI and Fortran 77 are available at:
<http://gather.ecs.soton.ac.uk>.
- [11] Work on the Beowulf cluster at ICASE based on Intel processors has resulted in a report: *Linux TCP Performance Fix for Short Messages*, available at:
<http://www.icas.edu/>
- [12] The VIA protocol is described at:
<http://www.viarch.org/>
- [13] L.S. Herbert, W. Seefeld, A. Skjellum, C.D. Taylor and R. Dimitrov. *MPI for NT: Two Generations of Implementations and Experience with the Message Passing Interface for Clusters and SMP Environments*. Proc. 1998 International Conf. on Parallel and Distributed Processing Techniques and Applications, Las Vegas, July 1998.
- [14] The 70 processor Alpha cluster known as Avalon is at Los Alamos:
<http://cnls.lanl.gov/avalon/>.
- [15] M. Lauria and A. Chien. *MPI-FM: High Performance MPI on Workstation Clusters*. J. of Parallel and Distributed Computing, **40**, 1, January 1997. The fast messages package is produced by the High Performance Virtual Machines group:
<http://www-csag.cs.uiuc.edu/projects/hpvm.html>.
- [16] See also: D. Emerson, K. Maguire, K. Takeda and D.A. Nicole. *An Evaluation of Commodity Supercomputers for CFD Applications*. Proc. Parallel CFD '98, Taiwan, May 1998.
- [17] Timings using EGCS on this simple dot product code have improved quite substantially from the older version 1.0.3 to the newer version 1.1.2 mainly due to improved operator ordering and more unrolling. It is unclear whether this improvement has carried through to realistic codes.
- [18] J.L. Hennessy and D.A. Patterson. *Computer Architecture: a Quantitative Approach*. Morgan Kaufman, San Francisco, 1996.
- [19] A. Maranda. *Tuning the GNU compiler for performance on the Intel platform*.
<http://members.xoom.com/AlexMaranda/gnuintel/GNUintel.htm>
- [20] Information about the Compaq Fortran beta release program is available at:
<http://www.digital.com/fortran/linux>