EPCC-SS98-13

Benchmarking and Related Work on the Hitachi SR2201

Krzysztof Wiktor Zelazowski

Abstract

The Hitachi SR2201 machine is relatively new among the resources at EPCC, which include the Cray T3D, Cray T3E as well as a Sun Enterprise 3000. The aim of the project was to investigate the execution time variation of several standard benchmark codes with respect to a variety of optimisation flags and processor number, determine the sustainable memory bandwidth of SR2201 as well as to verify the efficiency of Parallel FORTRAN translator.

Contents

1 Introduction

The following benchmark codes were utilised to compare execution times on SR2201 with respect to both Crays.

- ANGUS
- DL_POLY
- CETEP

These are real working codes which consist of existing Cray T3D user codes written in FORTRAN and utilising the MPI message passing library. The codes were run on all three machines, in order to compare their parallel efficiencies. It's important to appreciate, that due to the fact that the number of processors must be a power of two on Cray T3D, it's impossible to make a direct comparison.

In order to measure the sustainable memory bandwidth, a program called STREAM was employed as there are many results obtained from this program for different machines. Despite several drawbacks of this simple benchmark, which are going to be discussed later, it provides quite a good estimate of the real memory bandwidth.

Finally, the Parallel FORTRAN translator was tested on a simple code: "The Game of Life" and the results compared with those obtained from Cray machines. Due to considerable differences in compilation process between translators on the SR2201 and Crays, the exact procedure is explained so that a potentially new user might learn how to compile his/her program written in Parallel Fortran on the Hitachi machine. Also several drawbacks of this translator are investigated.

1.1 Hardware

The following three machines were used:

Hitachi SR2201 Comprised of 8 processors with 256 Mb per processor. Operates the HI-UX/MPP parallel system that does not share memory among nodes, ie. a distributed memory machine. Several benchmark exercises were carried out on 16 processors utilising the Hitachi from Maidenhead.

Cray T3D Comprised of 512 Alpha processors , each running at 150MHz with 64 Mb local memory.

Cray T3E-900 Comprised of 368 Digital EV5.6 processors, each running at 450 MHz with either 64,128 or 256 Mb of memory

1.2 Optimisation Flags

Before carrying out the benchmarking exercises, the following assumptions were made:

i. There would be no insertion of compiler directives into the codes on any of the machines

ii. Apart from the use of optimisation flags there would be no attempt to modify or optimise by hand any of the codes.

In the section below, the optimisation flags will be briefly summarised.

1.2.1 Hitachi

The compiler utilised was the HI-UX/MPP optimising Fortran90 compiler, OFORT90.

With regards to code optimisation on the Hitachi, the flags **-W0,'opt(o(s))** and **-W0,'opt(o(ss))** were utilised.

-W0,'opt(o(s))-Specifies the following range of optimisation options set up to enhance the execution speed.

- O(4) - performs a thorough optimisation, including control structure transformation and operation sequence change.
- ISCHEDULE(3) - involves changing the sequence of instructions in consecutive loops and changing the instruction sequence that references the elements of the array for the same iteration of the loop.
- APPROX(1) - allows division operations to be performed by inverse number multiplication.
- DISBRACKET(1) - changes the explicit sequence of operations indicated by the statement order and parenthesis use.
- DIVMODE(1) - moves loop invariants outside a loop even when a divide interrupt may occur as a result of evaluating a conditional expression within the loop.
- EXPAND(1) - specifies loop expansion.
- PROD(1) - allows the replacement of a multiplication with a DPROD or QPROD intrinsic function reference when the result of a single-p recision multiplication is converted to double-precision type or the result of a double precision multiplication is converted to quadruple-precision type.
- RPAIDCALL(1) - prevents checking of the arguments when a call is made.

-W0,'opt(o(ss)) - performs all the optimisations of -W0,'opt(o(s))' and the following :

- PVEC(DIAG(0)) specifies pseudo vectorisation, this is a pipeli ning technique which allows data to be loaded directly from main memory (i.e. by passing the cache) into floating point registers without blocking the execution of subsequent instructions

- OPT(FOLD(2)) allows the addition or subtraction of constants for the base register when refer encing data. The processing of saving or restoring data into the space register is also deleted

1.2.2 Cray T3D

The compiler utilised was the Cray CF90 (Version 3.0.0.2) compiler. When codes were compiled on T3D the following options were enabled :

- **-O3** : provides aggressive optimisation, where characteristics include a potentially larger compile size, longer compile time, global array optimisations and possible loop nest restructuring.
- **-OUNROLL2** : provides the highest level of loop unrolling carried out by the compiler.
- **RDAHEAD=ON** : provides an improvement in the on-chip memory bandwidth.

1.2.3 Cray T3E

The compiler utilised was the Cray CF90 (Version 3.0.0.1) compiler.

Optimisation flags **-O3** and **-Ounroll2** were also used on the T3E, however **streams=on** replaced read-ahead buffer in order to improve the memory bandwidth. Note that **streams** is enabled by default.

2 Measuring Performance of Codes

To compare efficiency of the three massive parallel machines we have used two standard metrics: **Parallel Speedup** and **Parallel Efficiency**. These are defined as follows:

- Parallel Speedup - ratio of the execution time of the parallel code on a single processor to the execution time of the same code on P processors
- Parallel Efficiency - ratio of the parallel speed-up to the number of P processors used.

2.1 ANGUS

ANGUS performs Direct Numerical Simulation of Turbulent Flames by solving the reacting Navier-Stokes equations. A cubic grid is used in the code and is parallelised via regular domain decomposition. Using a conjugate gradient algorithm, the majority of the time is spent solving a Poisson like equation which involves parallel matrix-vector and vector-vector operations. A great deal of optimisation has been carried out at CLRC (Council for the Central Laboratory of the Research Councils) and Edinburgh HPCI (High Performance Computing Initiative) Centres and the code now is highly efficient. Further details can be obtained from [5]. Production runs of the code have been carried out using a grid of 128x128x128 points, with up to 1500 time steps per run. Each run occupied 64 processors of the T3D for a total of 12 hours. For this reason, we have altered the lattice size for the program to be suitable as a benchmark exercise. In this work, we have used a grid of 96x96x48.

In this benchmark exercise different times were measured. These are summarised below:

- tBLAS - time taken in BLAS routines
- tmess - total time for doing boundary swaps
- t(p=Aq) - time taken during matrix-vector multiplication

Combination	total time	tBLAS	tmess	t(p=Aq)
1x1x1	1685	1050	4	401
4x1x1	640	413	28	142
2x2x1	508	406	13	96
8x1x1	378	224	48	72
1x8x1	337	223	14	66
1x1x8	381	249	25	74
4x2x1	328	203	25	68
2x2x2	321	203	17	67
16x1x1	245	126	47	41
8x2x1	190	102	30	32
4x4x1	162	95	16	30
4x2x2	166	95	16	32

Table 1: Timings obtained for ANGUS with various processor arrangement, with no optimisation.

It should be firstly noted that with regards to the results shown in Table 1, CRAY optimised BLAS routines were used. As it can be seen from the results shown in table 1, the execution times depend strongly on the processor arrangement. Considering the fact that we have used a grid of 96x96x48 points, it is clear that a processor arrangement of 2x2x1 will give better results than 4x1x1 as the lattice is more evenly distributed across the processor grid. As was anticipated, for 8 and 16 processors, the optimal arrangement was 2x2x2 and 4x4x1 respectively. It is also interesting to note how processor arrangement affects time taken for boundary swaps. Again, better results were obtained when the lattice was distributed evenly accross the processor grid.

In order to compare the results obtained on the Hitachi with the Cray T3D and T3E we have chosen the optimal processor's arrangement of 2x2x2, although Cray optimised BLAS have been used for these runs. Time variation with different optimisation flags were investigated with the results summarised in Tables 2 and 3 for 8 and 16 processors respectively.

Machine	Compile Option	total	tBLAS	tmess	t(p=Aq)
T3D	no opt	287	92	37	120
T3D	03	287	92	37	120
T3D	03,unroll2	268	92	36	102
T3D	03,unroll2,readahead=on	261	92	34	97
Hitachi	no opt	321	203	17	67
Hitachi	W0,'opt(o(s))'	233	155	17	37
Hitachi	W0,'opt(o(ss))'	113	55	17	21
T3E	no opt	102	43	16	25
T3E	03	102	43	16	25
T3E	03,unroll2	99	43	16	22
T3E	03,unroll2,streams=on	99	43	16	22

Table 2: Timings obtained for ANGUS with various opt flags using Cray optimised BLAS: 8 PE's

Machine	Compile Option	total	tBLAS	tmess	t(p=Aq)
T3D	no opt	158	49	28	59
T3D	03	158	49	28	59
T3D	03,unroll2	146	49	28	47
T3D	03,unroll2,readahead=on	143	49	27	45
Hitachi	no opt	162	95	16	30
Hitachi	W0,'opt(o(s))'	127	74	16	19
Hitachi	W0,'opt(o(ss))'	71	28	16	13
T3E	no opt	61	23	15	11
T3E	03	60	23	15	11
T3E	03,unroll2	60	23	16	9
T3E	03,unroll2,streams=on	60	23	16	9

Table 3: Timings obtained for ANGUS with various opt flags using Cray optimised BLAS: 16 PE's

Table 4 on page 8, present the figures obtained from runs using the same optimal processor arrangement, however, this time using the SR2201's optimised BLAS. 8 processors results are only provided.

Analyzing figures obtained using Cray's optimised BLAS, it is clear that the SR2201 performs

Machine	Compile Option	total	tBLAS	tmess	t(p=Aq)
Hitachi	no opt	148	41	20	65
Hitachi	W0,'opt(o(s))'	114	41	20	35
Hitachi	W0,'opt(o(ss))'	98	41	17	22

Table 4: Timings obtained for ANGUS using SR2201's optimised BLAS with various opt flags: 8 PE's

somewhere between both Cray machines. It is also evident, that most benefit from the BLAS has been achieved through the use of pseudo vectorisation. Furthermore, using optimised BLAS gives an additional 13 % , as shown in Table 4 above.

On pages 9 and 10, parallel speedup and parallel efficiency curves are shown for ANGUS over 16 processors. [1]

[1]Note that these represent results using Cray optimised BLAS with o(ss) as results for hand optimised BLAS were not available for 16 processors. However, we would not expect this to have too many qualitative differences, as scaling will be the same.

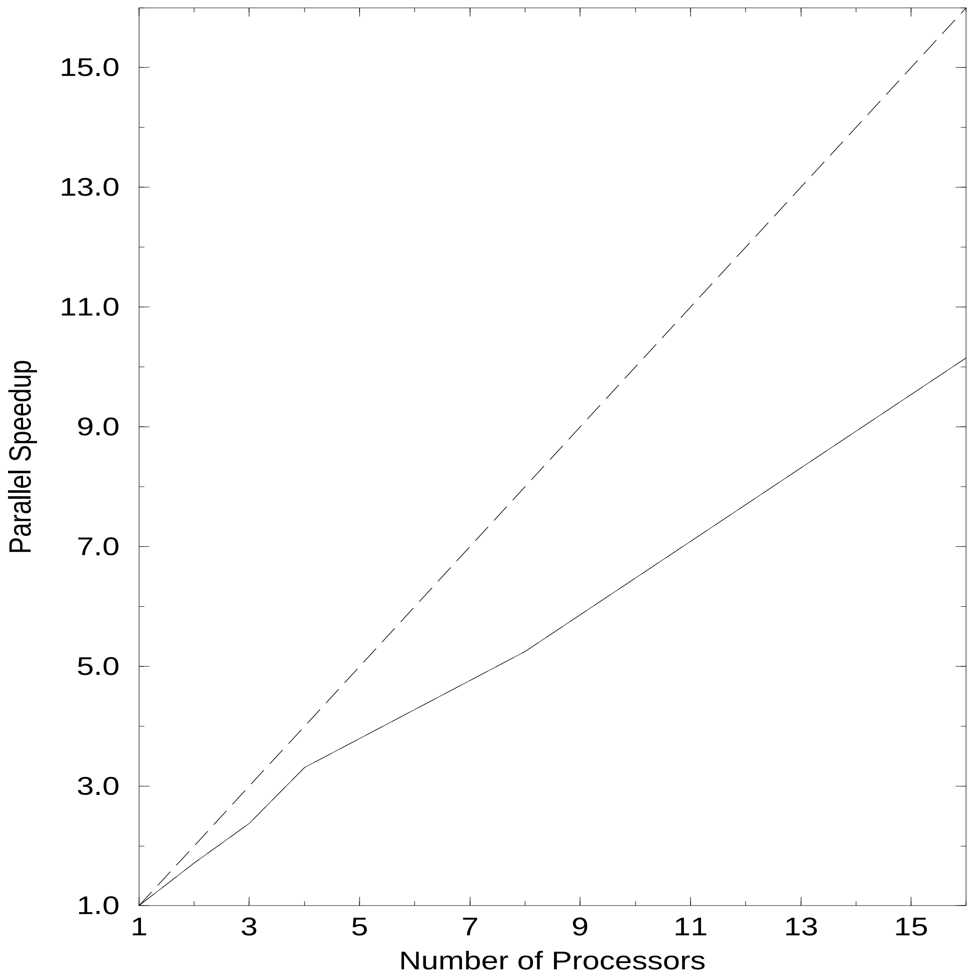


Figure 1: Parallel speedup over 16 processors for ANGUS. The dashed line represents linear speedup, whereas the solid line represents the actual speedup.

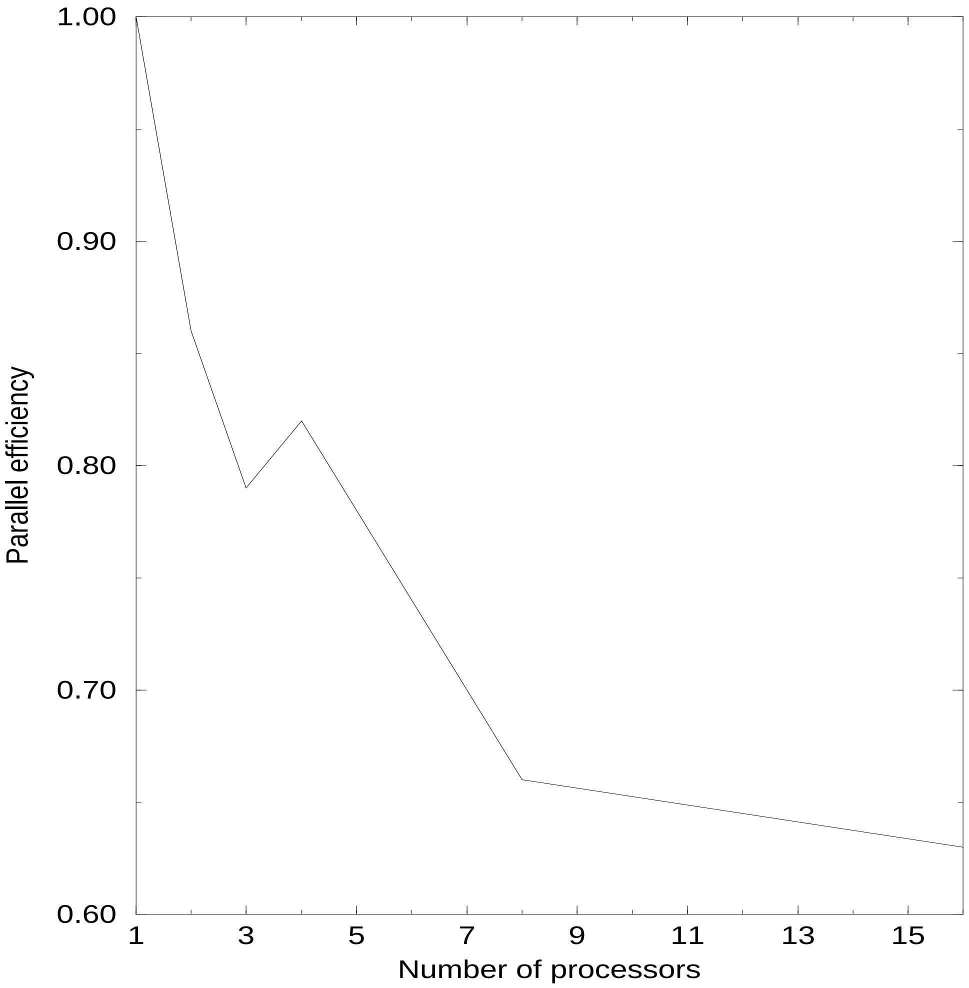


Figure 2: Parallel efficiency over 16 processors for ANGUS.

There is a slight increase in parallel efficiency between 3 and 4 processors, which can be observed on the parallel speedup curve, and which is even more noticeable on the parallel efficiency curve above. The reason behind this is that a three dimensional lattice of the type used in the exercise (96x96x48) can not be evenly distributed across a processor grid of 3 processors[1] [2] , as the only possible configuration is 3x1x1. When using 4 processors, one can use an arrangement of 2x2x1 which clearly allows for a more even distribution.

[1]The results for 3 processors have been adapted from the work of Lorna Smith and Connor Mulholland (EPCC-PROJ-HEL-BM 1.5)

2.2 CETEP

CASTEP (Cambridge Serial Total Energy Package) is a code for the solution of the electronic ground state of periodic systems with the wave functions expanded in a PW basis using a technique based on density functional theory. The CETEP (Cambridge Edinburgh Total Energy Package)[6] code is similar to CASTEP but designed to run on massively parallel supercomputers. This latter code was developed as part of the UKCP (The UK Car-Parrinello Consortium) collaboration. More information about this code may be found in M.C. Payne et. al., Rev. Mod. Phys. (1992) 64 p. 1045-

The task which requires most computation in this program is a parallel Fast Fourier Transform, which involves all-to-all communications and global sums.

The parallel version of the code (the CETEP) requires the number of processors to be a power of 2 - in this benchmark exercise this did not pose a problem as we have used 8 and 16 processors. Because of very large memory requirements, a very simple data set was used for this exercise (si8), with 100 iterations to obtain a reasonable timing. Furthermore, the CETEP version used for this benchmarking exercise makes no use of a SHMEM library for communications but a MPI implementation only.

In the tables below, the results from all three machines are shown.

Machine	Compile Option	total
T3D	no opt	91
T3D	03	91
T3D	03,unroll2	88
T3D	03,unroll2,readahead=on	86
Hitachi	no opt	80
Hitachi	W0,'opt(o(s))'	78
Hitachi	W0,'opt(o(ss))'	67
T3E	no opt	37
T3E	03	33
T3E	03,unroll2	34
T3E	03,unroll2,streams=on	34

Table 5: Timings obtained for CETEP with various opt flags: 4 PE's

As it can be seen from table 5, on 4 processors, the flag o(s) had very little effect on execution times on Hitachi SR2201, whereas o(ss) has reduced timing to a greater extent. This was not true for 8 and 16 processors on which virtually no variation was observed (see tables nos. 6 and 7). It is clear from Tables 5 and 6 that CETEP performs significantly better on the Cray T3D and even more so on the T3E for 8 and 16 processors. The reason for this improvement in scaling on the T3D and T3E is due to the global MPI routines being significantly faster on the Crays than those used on the SR2201.

Machine	Compile Option	total
T3D	no opt	55
T3D	03	55
T3D	03,unroll2	54
T3D	03,unroll2,readahead=on	52
Hitachi	no opt	65
Hitachi	W0,'opt(o(s))'	63
Hitachi	W0,'opt(o(ss))'	64
T3E	no opt	20
T3E	03	20
T3E	03,unroll2	19
T3E	03,unroll2,streams=on	19

Table 6: Timings obtained for CETEP with various opt flags: 8 PE's

Machine	Compile Option	total
T3D	no opt	38
T3D	03	38
T3D	03,unroll2	38
T3D	03,unroll2,readahead=on	37
Hitachi	no opt	59
Hitachi	W0,'opt(o(s))'	61
Hitachi	W0,'opt(o(ss))'	57
T3E	no opt	13
T3E	03	12
T3E	03,unroll2	12
T3E	03,unroll2,streams=on	11

Table 7: Timings obtained for CETEP with various opt flags: 16 PE's

2.3 DL_POLY

DL_POLY is a parallel molecular dynamics simulation package which is very portable and runs on a variety of different platforms. These include, vector machines such as the Cray-J90, parallel distributed machines such as the Cray T3D and T3E as well as the SP2 and pseudo vector machines like the SR2201. It was developed at Daresbury Laboratory by W. Smith and T.R. Forester under the auspices of the Engineering and Physical Sciences Research Council (EPSRC) for the EPSRC's Collaborative Computational Project for the Computer Simulation of Condensed Phases (CCP5) and the Molecular Simulation Group (MSG) at Daresbury Laboratory. It uses replicated data and therefore requires efficient all-to-all communications and global sums of small arrays. Further information can be found in [7].

The code for DL_POLY came with 3 different data sets, but only Bench 1 (a 19652 Al atom system) was used for these set of exercises. However for implementation purposes, adjustments were made with regards to the size of the problem in order to gain a reasonable completion time.

In tables 8 and 9, results obtained for 8 and 16 processors are illustrated.

Machine	Compile Option	total
T3D	no opt	346
T3D	03	348
T3D	03,unroll2	327
T3D	03,unroll2,readahead=on	326
Hitachi	no opt	172
Hitachi	W0,'opt(o(s))'	132
Hitachi	W0,'opt(o(ss))'	138
T3E	no opt	68
T3E	03	69
T3E	03,unroll2	68
T3E	03,unroll2,streams=on	66

Table 8: Timings obtained for DL_POLY with various opt flags: 8 PE's

Machine	Compile Option	total
T3D	no opt	215
T3D	03	216
T3D	03,unroll2	207
T3D	03,unroll2,readahead=on	206
Hitachi	no opt	165
Hitachi	W0,'opt(o(s))'	110
Hitachi	W0,'opt(o(ss))'	111
T3E	no opt	51
T3E	03	51
T3E	03,unroll2	50
T3E	03,unroll2,streams=on	50

Table 9: Timings obtained for DL_POLY with various opt flags: 16 PE's

In this set of results we have obtained a quite considerable speed gain when using any of the optimisation levels on SR2201. Comparing the two optimisation levels however, no benefit was observed with pseudo vectorisation, reflecting the fact that DL_POLY is less vectorisable than ANGUS. The same was true on both Cray machines.

If we compare just the best results from all the machines, it is clear that the Hitachi SR2201 performed somewhere between the T3D and T3E. In the first case, i.e. for 8 processors, the difference between the T3D and Hitachi is almost three fold (see table 8).

Figures 3,4,5 and 6 illustrate parallel speedup and efficiency for the SR2201 [1] and both Crays [2] over 16 processors.

With regards to the speedup figures (ie. Figs 3 and 5), the dotted line represent the optimal speedup curve, while the other one or two in the case of figure 5, represent the curves obtained

[1] The timings obtained for 1,2,3 and 6 processors were adapted from the work done by Lorna Smith and Connor-Mulholland (EPCC-PROJ-HEL-BM 1.5)

[2] Timings retrieved from the findings of this investigation as well as those quoted in (EPCC-PROJ-HEL-BM 1.5) have been used to present figures 5 and 6.

for DL_POLY for the machines in question.

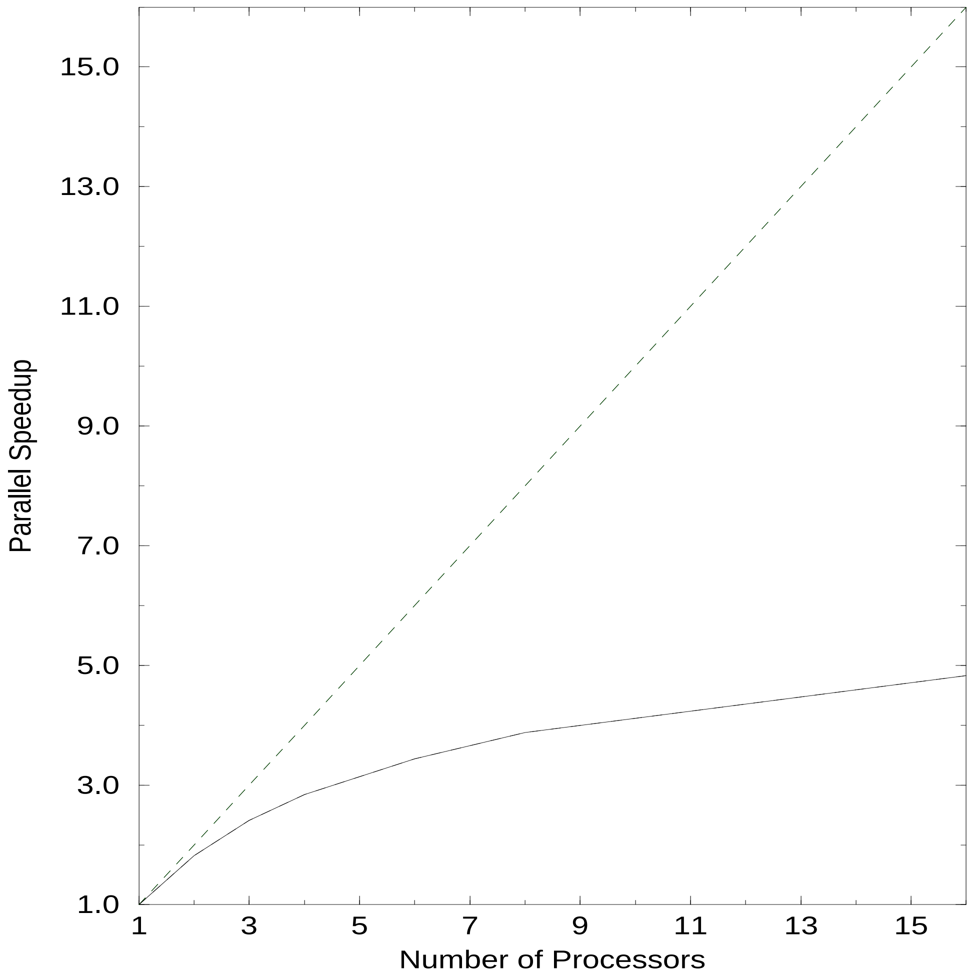


Figure 3: Parallel speedup on SR2201 over 16 processors for DL_POLY.

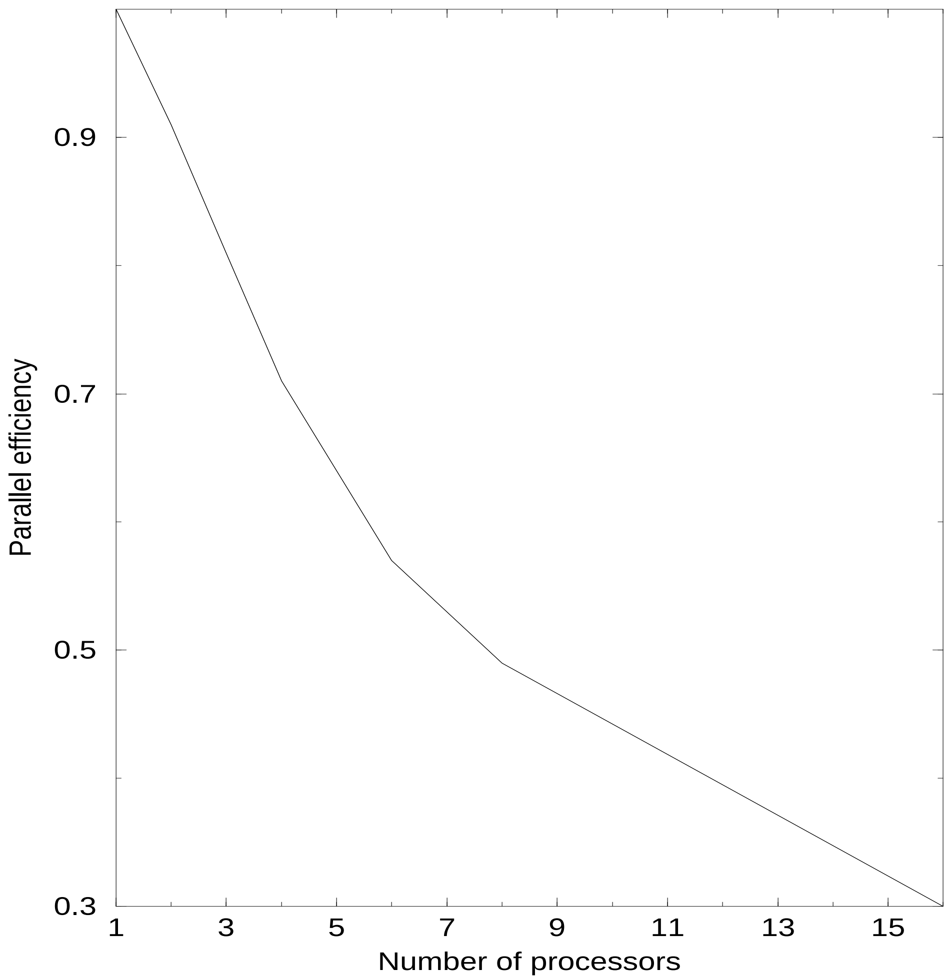


Figure 4: Parallel efficiency on SR2201 over 16 processors for DL_POLY.

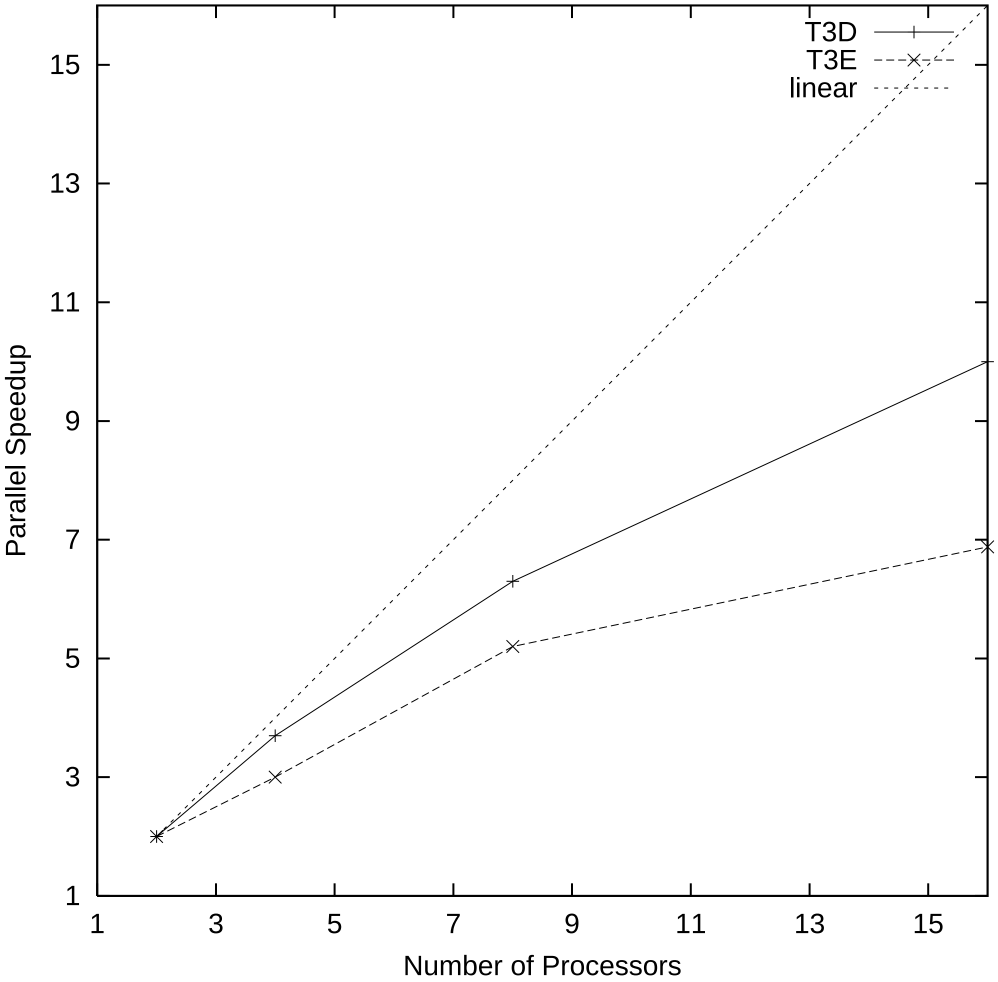


Figure 5: Parallel speedup for both T3D and T3E over 16 processors.

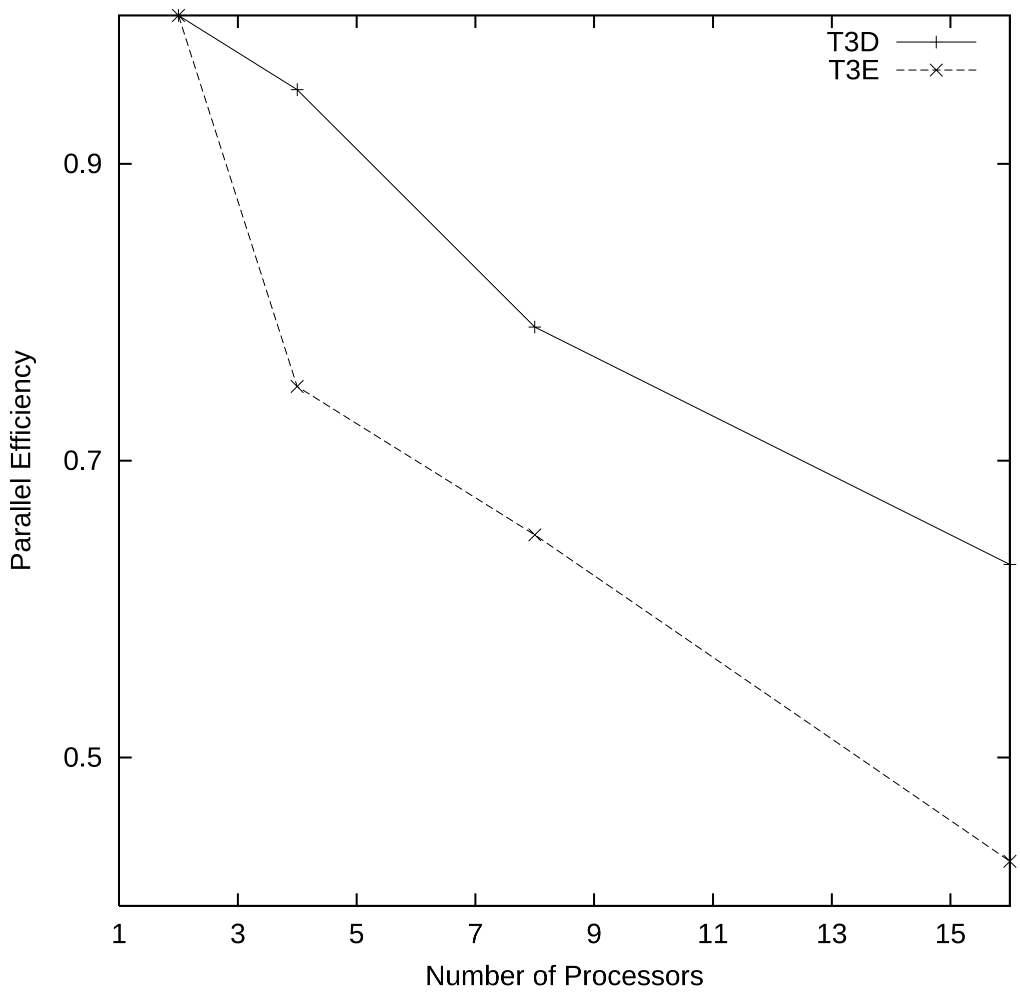


Figure 6: Parallel efficiency over 16 processors for both T3D and T3E.

Analyzing figure 3, it is clear that DL_POLY does not scale all that well on the SR2201. In fact, the curve almost flattens once over 4 processors. This becomes even more apparent, when considering its parallel efficiency curve in figure 4, where the efficiency falls below 70 % once we use more than 4 processors. This is to be expected, as it is well known that molecular dynamics with long range forces do not scale particularly well on MPPs, as it requires global communications. However, comparing this scalability and efficiency with those results obtained for the Cray T3D and T3E, it is clear that DL_POLY appears to scale a little better on the Crays than on the SR2201. Furthermore, the efficiency decrease as shown in figure 6 for both Crays is not as dramatic as that illustrated in figure 4.

2.4 Discussion

The results obtained during this benchmark therefore indicate firstly, that the code appears to scale better on the T3D and T3E rather than the SR2201. This is reflected in the figures given. However, using the timings given in the results and with regards to absolute performance, the SR2201 lies between T3D and T3E, performing approximately to a factor 2 faster than the T3D.

3 Measuring Sustainable Memory Bandwidth

3.1 Introduction

Computer CPUs are getting faster much more quickly than computer memory systems. It has been recently estimated that on average, CPUs are getting faster by about 80 % per year, while memory systems are increasing by about 7 % per year. As an extreme example, several current high-end machines run simple arithmetic kernels for out-of-cache operands at 4-5% of their rated peak speeds — that means that they are spending 95-96% of their time idle and waiting for cache misses to be satisfied.

Moreover, vendors are usually seldom to publish enough details about the computer architecture for the memory bandwidth to be calculated explicitly. Therefore it is clear that the bandwidth needs to be measured.

The STREAM [4] benchmark is a simple synthetic benchmark program that measures the sustainable memory bandwidth (in MB/s) and the corresponding computation rate for simple vector kernels. The STREAM remains an independent academic project, which will not be influenced or directed by commercial concerns. In order to maintain this independence, the STREAM benchmark is hosted at University of Virginia under the sponsorhip of Professor Alan Batson and Professor William Wulf.

It is specifically designed to work with datasets much larger than the available cache on any given system, so that the results are (presumably) more indicative of the performance of very large, vector style applications.

It needs to be pointed out that the intent of STREAM is not to suggest that “real” applications have no data re-use, but rather to decouple the measurement of the memory subsystem from the hypothetical “peak” performance of the machine. In this respect the test is quite complementary to the LINPACK benchmark test, which is typically optimised to the point that a very large fraction of full speed is obtained on modern machines, independent of the performance of their memory systems.

Information relating to the four kernels mentioned above are given in the table below:

Name	Kernel	Bytes/iteration	FLOPS/iteration
COPY	a(i)=b(i)	16	0
SCALE	a(i)=q*b(i)	16	1
ADD	a(i)=b(i)+c(i)	24	1
TRIAD	a(i)=b(i)+q*c(i)	24	2

Table 10: STREAM - technical information relating to the four kernels.

It should be noted that in the memory bandwidth estimates, the STREAM benchmark gives “credit” for both memory reads and memory writes. Also worthwhile noticing is the fact, that STREAM does not take into account different memory hierarchies - it has been suggested that it should in the future for more objective results.

3.2 Overview of the runs

The STREAM benchmark was run on the 8 processor machine at EPCC as well as the 16 processor machine in Maidenhead, although we have only illustrated the results for the SR2201, at EPCC. For this exercise, we have used an almost 8 million element vector test case. However, it was only enough to obtain precision of about 10%. For the recommended precision (less than 5%), one would need to use a 12 million element vector, but this would occupy about 240 Mb of RAM. Due to system configuration, at the time of this benchmark exercise, only about 179 Mb were available per node, even though the theoretical maximum is 256 Mb per node.

3.3 Results

The output from the benchmark is presented below:

```
This system uses 8 bytes per DOUBLE PRECISION word.
Array size = 7800000, Offset = 0
Total memory required = 178.5 MB.
Each test is run 10 times, but only
the *best* time for each is used.

Your clock granularity/precision appears to be 9993 microseconds.
Each test below will take on the order of 189998 microseconds.

   (= 18 clock ticks)

Increase the size of the arrays if this shows that
you are not getting at least 20 clock ticks per test.
WARNING -- The above is only a rough guideline.
For best results, please be sure you know the
precision of your system timer.

Function  Rate (MB/s)  RMS time   Min time   Max time
Copy:     567.2806     0.2241     0.2200     0.2300
Scale:    594.2829     0.2190     0.2100     0.2200
Add:      603.8769     0.3120     0.3100     0.3200
Triad:    603.8769     0.3180     0.3100     0.3200
```

It is clear from the above, we have only obtained 18 clock ticks, due to memory shortage, as explained earlier. Also, worthwhile noticing is the fact that these results represent transfer rates between one node and corresponding memory (the program was run only on one processor). But, considering that Hitachi SR2201 is a distributed memory machine, the overall bandwidth for all 8 processors is simply 8 times the "results obtained for one processor".

A similar approach is undertaken when benchmarking systems like T3D or T3E. There, copies of STREAM need to be run concurrently on N processors, and the overall bandwidth is the sum of the results obtained from each process.

Although we have not achieved the required accuracy, using the current results on the STREAM site for a 1 processor SR2201, T3D and T3E, a fairly reasonable comparison can be made. Table 10 below illustrates the measurements obtained from the SR2201 based at EPCC, as well as presenting the STREAM site's results for SR2201, T3D and T3E in columns 3,4 and 5 respectively.

name	SR2201 (EPCC) (MB/s)	SR2201 (MB/s)	T3D (MB/s)	T3E (MB/s)
COPY	567	775	384	484
SCALE	594	726	329	482
ADD	604	829	187	571
TRIAD	604	827	177	568

Table 11: STREAM - sustainable memory bandwidth for 1 processor machines.

3.4 Conclusions

Using the 1 processor figures in Table 10, it is fair to conclude that the SR2201 performs better than the 1 processor T3D and T3E published on STREAM's performance pages, with respect to sustainable memory bandwidth. Furthermore, although the figures obtained from the machine based here at EPCC do not match those of column 3, ie best results for SR2201 as published on STREAM's website, I would expect similar results if a more higher resolution timer were available.

One of the goals of this benchmark was to publish the results, however, because we were not able to obtain 5% accuracy that was not possible.

There is a wide range of results for machines from different vendors published on the web. These may be found at:

http://www.cs.virginia.edu/stream/

4 Parallel FORTRAN

Parallel FORTRAN [1] is a translator for converting a program consisting of statements and directives for parallel execution into a Fortran 90 source program.

The language specifications of Parallel FORTRAN include the language specifications of High Performance Fortran (HPF) Language Specification Version 1.0, May 3,1993, which is a widely adapted standard that enables a simple and relatively painless method for creating portable data-parallel programs, which can be used on all types of HPC systems.

The features provided by Parallel FORTRAN are as follows:

- Data distribution
- Data parallelization
- Extended intrinsic functions and standard library
- EXTRINSIC procedure
- Handling by storage sequence and sequence association

The statements and directives with these features are translated into a Fortran 90 source program. At translation, a communication feature for handling data distributed to different processors' memory is included in the program.

4.1 How does it work ?

The Parallel FORTRAN source code is translated into Fortran 90 code by using the following command:

```
pf90 [ options ] files
```

Below, some of the options are summarized .

Options:

```
-Wt,'Translate-Options'
```

Specifies the translate options for controlling Parallel FORTRAN translator. Each translate option must be separated by a comma. The list of translate options must be enclosed in single or double quotations.

Translate-Options

- comlib(SubOptions) Specifies the execution model for compiling and running a translated program. The communication library option can be one of Remote DMA, PARALLELWARE, MPI or PVM.
  - rdma[(block|spin[(nnn)])] Generates a Fortran90 source program that can be executed by using the Remote DMA library. Block or spin can be specified for the receive acknowledge method. For details about Remote DMA, see "Remote DMA transfer FORTRAN USER'S GUIDE". [1]

    block Specifies the block method for the receive acknowledgment.

    spin(nnn) Specifies the spin method for the receive acknowledgment. nnn is the maximum number of communication fields used for the spin method. The default value of nnn is 1.
  - e[(0|1)] Uses PARALLELWARE and generates a Fortran90 source program that is executed on the host node model.

    0 Generates a source program with no synchronous termination.

    1 Generates a source program with synchronous termination. If this option is specified, the program must be run with a host program which includes a PF_HOST_FINAL function call. Parallel FORTRAN provides the host program. See the "Parallel FORTRAN USER'S GUIDE" for details. [1]

    If only e is specified, e(1) is assumed.
  - mpi Generates a Fortran90 source program that can be executed by using MPI.
  - pvm Generates a Fortran90 source program that can be executed by using PVM.

  The default is comlib(rdma(spin(1))). The translated output file (Fortran 90 source code) is saved in a directory called **/pf_node** At this point, this source file should be compiled to obtain the actual executable. What type of compiler should be used depends on the communication library specified during translation. The example below shows how to compile a source code with mpi communication library.

4.2 Example

The example below shows how to translate a Parallel FORTRAN file and then how to compile it to obtain the executable, on the Hitachi SR2201.

- Translation

  ```
  pf90 -Wt,'comlib(mpi)','form(free)' ./example.f
  ```

  where :

  - **-Wt,'comlib(mpi)'** specifies MPI as the communication library.
  - **-Wt,'form(free)'** option needed when compiling a free format file.

- Compilation

  ```
  cd ./pf_node/
  mpif90 -W0,"'FORM(FREE)'" /usr/ccs/lib/libhpf.a ./example.f
  ```

  where :

  - **/usr/ccs/lib/libhpf.a** Specifies a path for HPF library files.

- Running

  ```
  mpirun -n 8 -part p8 ./a.out
  ```

  where :

  - **-n 8** Specifies 8 processors for the program to be run on.
  - **-part p8** Specifies partition "p8" as the destination.

4.3 The Game of Life

The Parallel Fortran translator was finally tested on a very simple code - "The game of life", and results compared with those obtained on the Cray T3D. Because the results obtained on the SR2201 deviate strongly from the T3D, the exact procedure followed during benchmarking is shown on page 25, so that these results may be verified at some later stage.

4.3.1 pf90 translator on the Hitachi

translation:

```
pf90 -Wt,'comlib(mpi)','form(free)' ./life.f
```

compilation:

```
mpif90 -W0,"'FORM(FREE)'" /usr/ccs/lib/libhpf.a ./life.f -o life
```

execution :

```
mpirun -n [#PEs] -p [partition] ./life
```

size: 500x500, 10 iterations

No. PE's	Distribution	partition	Optimisation	Time
1	N/A	'p2b'	no opt	43.4 sec
2	[BLOCK|BLOCK]	'p2b'	no opt	151.3 sec
4	[BLOCK|BLOCK]	'p4'	no opt	244.2 sec
4	[BLOCK|BLOCK]	'p4'	o(s)	231.0 sec
4	[BLOCK|BLOCK]	'p4'	o(ss)	236.1 sec

Table 12: Timings obtained for The Game of Life on the SR2201

Results in the table above indicate that optimisation makes little difference.

4.3.2 pghpf on Cray T3D

Environment variables:

```
> module load pghpf
```

```
> export DISPLAY=heliodor.epcc.ed.ac.uk:0.0
```

```
> export TARGET=cray-t3d
```

compilation:

```
> pghpf -Mautopar -o life life.hpf
```

execution :

```
> ./life -npes [#PEs]
```

size : 500x500, 10 iterations

No. PE's	Distribution	Time
1	N/A	3.675 sec
2	[BLOCK|BLOCK]	1.773 sec
4	[BLOCK|BLOCK]	0.965 sec

Table 13: Timings obtained for The Game of Life on the T3D

Table 13 above, which illustrate timings obtained on the Cray T3D, show significantly better performance and good parallel speedup.

4.3.3 Summary

Table 12 illustrates an increase in the execution time for the SR2201 as the number of processors increase. Even the use of full optimisation does not improve the performance that significantly. These figures contrast those obtained for the Cray T3D in Table 13, which indicate a considerable speed up being observed. Preliminary investigation to this shows that it is probably due to inefficient implementation of the **cshift( )** routine. Further study would need to be done to fully assess the efficiency of the Paral lel Translator on the SR2201.

References

[1] Parallel FORTRAN Reference . Technical report, EPCC, www.epcc.ed.ac.uk/h̃itachi/SR2201onLineDocs/index.html, December 1997.

[2] Lorna Smith and Connor Mulholland. *"Benchmarking the Hitachi SR2201"*, March 1997. EPCC-PROJ-HEL-BM 1.5.

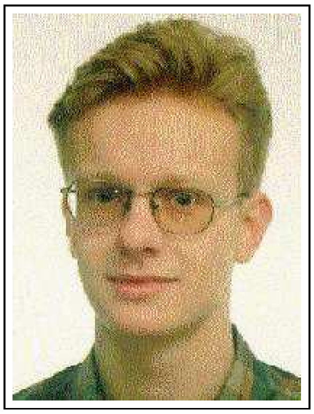
[3] Gavin J. Pringle. "Evaluating PGHPF on the Cray T3D/T3E". Technical report, EPCC-TR98-02, September 1998.

[4] John D. McCalpin. "Measuring Sustainable Memory Bandwidth in High Performance Computers". Technical report, Advanced Systems Division Silicon Graphics,Inc., http://reality.sgi.com/mccalpin/papers/bandwidth/bandwidth.html, October 1995.

[5] R.S.Cant, R.Prosser, K.W.Jenkins, L.Leboucher, W.K.Bushe . "Direct Numerical Simulation of Turbulent Flames". Technical report, http://wserv1.dl.ac.uk/TCSC/HPCI/HPCProfile/mar97/, March 1997.

[6] Matthew Segall. "Cambridge Edinburgh Total Energy Package". Technical report, University of Cambridge, www.tcm.phy.cam.ac.uk/castep/, 1997.

[7] W. Smith. "The DL_POLY Molecular Dynamics Simulation Package". Technical report, Central Laboratory of the Research Councils, http://wserv1.dl.ac.uk/TCSC/Software/DL_POLY/, 1998.



About the author:
I have just completed 3rd year of my undergraduate studies at Warsaw University of Technology, at the faculty of Electronics and Information Techniques. My major is: Telecommunication Networks and Services Management.

Acknowledgements: I would like to thank my supervisor, Connor Mulholland who helped me over the course of this project. Thanks also to the rest of the EPCC staff who I asked for ideas and assistance at several points during this programme.