|epcc|

THE UNIVERSITY OF EDINBURGH

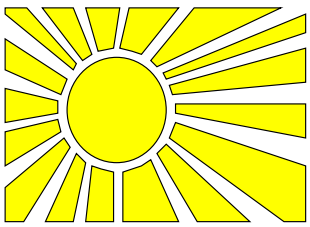
EPCC-SS-2000-01

Benchmarking the Sun HPC3500, Sun HPC6500 and the Wildfire system

Michael A. Clark

Abstract

High performance computers are analysed using simple MPI, OpenMP and mixed mode Game of Life codes. Direct comparison between the HPC 3500 and HPC 6500 is used to approximate the operating system overhead present, and it is found that operating system uses up to 5% of the machine's processing power. Vast performance increases are discovered when using OpenMP on the HPC 6500, which are due to caching effects caused by the shared memory nature of OpenMP. The scalability of the Wildfire system is investigated, it is found that the interconnect latency is negligible, though the operating system overhead can be witnessed.



Contents

1 Introduction

Two recent additions have been made to EPCC facilities. These are the eighteen processor Sun HPC 6500 and the thirty four processor Sun Wildfire system. These machines represent the largest shared memory machines which EPCC possess. This project is concerned with benchmarking these machines together with the older eight processor HPC 3500.

Through benchmarking analysis, otherwise anonymous performance issues, can be explained and justified. The operating system (O/S) of any computer will always present an extra overhead, but by quantifying this overhead better predictions about code performance can be made. The size of this overhead is important to clarify also because dependent of its size, the performance of an HPC machine could benefit if the O/S is given sole rights to a single processor. The O/S overhead should only really become apparent when a parallel machine is operating at peak capacity, and a dip in the performance will be witnessed. A direct comparison between the performance of the eight processor HPC 3500 and eighteen processor HPC 6500 is used as the method of quantifying this overhead. By looking at the scaling up to eighteen processors on the HPC 6500 it is investigated whether the O/S overhead is apparent.

The Wildfire machine is an experimental system, which is made up of three shared memory boxes which are connected via the Wildfire interconnect. To the user, the machine appears as a single thirty four processor system. Wildfire is seen as a possible solution to constructing large scale (> 50 processors) shared memory machines, because true uniform access shared memory machines become increasingly economically inefficient as the number of processors is increased. This investigation investigated the potential of Wildfire through benchmarking analysis.

Almost all of the performance analysis is performed using the supplied Game of Life code, which came in three forms: OpenMP, MPI and mixed mode. MPI can operate on shared or distributed memory architectures, OpenMP on shared memory architectures only, while mixed mode programmes implement both MPI and OpenMP calls. Mixed mode programming is a relatively new concept, and is particularly aimed at SMP clusters, e.g. Wildfire. In theory, within the SMP OpenMP should give the most efficient parallelisation strategy, while MPI should be optimal between the SMP boxes. This theory can be put to the test using the Wildfire machine.

2 Background

2.1 Systems Overview

The three machines which were bench-marked operate using Sun OS and appear as shared memory machines.

The Sun HPC 3500 This is an 8-processor Sun HPC 3500 UltraSPARC II based system. The HPC 3500 and HPC 6500 both have a queueing system installed, and so it possible to ensure the necessary exclusive access when benchmarking.

Specification:

- 8 x 400MHz UltraSPARC II processors
- 16-KB instruction, and 16-KB data on chip primary cache per processor
- 4-MB external cache per processor
- 8 Gbytes of shared memory
- 54 Gbytes of disc space
- 100Mhz Main memory bandwidth
- 2.6 GB/sec sustained (at 84 MHz), 2.7 GB/sec peak (at 84 MHz) system interconnect (internal)

The Sun HPC 6500 This is an 18-processor Sun HPC 6500 UltraSPARC II based system, with a similar architecture to the HPC 3500.

Specification:

- 18 x 400MHz UltraSPARC II processors
- 16-KB instruction, and 16-KB data on chip primary cache per processor
- 4-MB external cache per processor
- 18 Gbytes of shared memory
- 108 Gbytes of disc space
- 80Mhz Main memory bandwidth

- 2.6 GB/sec sustained (at 100 MHz), 2.7 GB/sec peak (at 100 MHz) system interconnect (internal)

The Wildfire This the collective name given to three Sun machines which appear as a single machine. Since Wildfire is an experimental machine, it does not have the level of support that the HPC 3500 and HPC 6500 have. Wildfire has no queueing system installed so care is needed when benchmarking, to ensure no processes are clashing.

Specification [3]:

- 34 x 250Mhz UltraSPARC II processors, 1 x e6000 (18 processors) and 2 x e4000 (8 processors each)
- 16-KB instruction, and 16-KB data on chip primary cache per processor
- Mixture of 4-MB and 1-MB external cache per processor
- (approximately) 7.5Gbytes of shared memory
- 3.2 GB/sec (peak) backplane
- 1.6 GB/sec system interconnect (Wildfire)

2.2 Game of Life

The Game of Life is a simple rather contrived example of how parallelism can be achieved. This is a simple two dimensional grid based problem, of size $N \times N$ from which complex behaviour arises. Within the Game of Life every cell is either dead or alive, with the state of each cell determined by its nearest neighbours on the previous iteration [5].

1. Initialise the 2D cell.
2. Carry out boundary swaps.
3. Loop over the 2D grid, to determine the number of alive neighbours.
4. Update the 2D grid, based on the number of alive neighbours and calculate the number of alive cells.
5. Iterate steps 2-4 for the required number of iterations.
6. Write out the final 2D grid.

The Game of Life codes which were used came in OpenMP, MPI and mixed mode forms. The MPI code has the inherent problem that the grid dimensions used must be a multiple of the number of processors used if an unbiased performance measure is required, though this is less of a problem if large dimension sizes are used. This meant that the grid dimensions came into consideration when deciding how many processors the benchmarking should be run on (see Table 1).

	Number of Processors																
N	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18
720	X	X	X	X	X		X	X	X		X			X	X		X
840	X	X	X	X	X	X	X		X		X		X	X			
1920	X	X	X	X	X		X		X		X			X	X		
2448	X	X	X		X		X	X			X				X	X	X
4896	X	X	X		X		X	X			X				X	X	X

Table 1: The grid dimensions used, and the corresponding numbers of processors to which they apply.

The MPI code, as it was given, implemented a two dimensional decomposition strategy. This works extremely well for processor numbers which can be split into two dimensions quite evenly, however, if this is not the case, e.g. five and seven, a one dimensional decomposition is used which increases the communication latency. In order to perform unbiased analysis between processors the decomposition strategy was re-implemented as a one dimensional technique.

All codes produced the same format of output, that is the timing information as printed from the High Resolution Timer (HRT) programme [1]. These times are required to produce the speedup and efficiency figures. The initialisation of variables at the beginning of the code is purely serial, and is not affected by altering the number of processors so only the parallel region is timed. In addition to the execution times, the number of processors used is also required to make the necessary calculations.

Running parallel code on Wildfire can be unreliable because tasks are not bound to processors. Extra libraries must be run in order to guarantee that MPI processes [4] and/ or OpenMP threads [2] are bound the same processor throughout.

2.3 Peak Performance Measure

The Game of Life requires considerable memory access, and so it is expected that the speedup will diverge away from the perfect result significantly. Evidence for the presence of the O/S overhead can again be investigated using code which requires negligible memory access, but significant processing power. A simple loop was used to perform this function, this loop sums the squares of a series of numbers between one and N, a

previously set parameter. This was performed on the HPC 3500 and the HPC 6500 in both OpenMP and in MPI forms (see Appendix B).

2.4 Perl Script

Before the benchmarking began, a Perl script was written to aid in the analysis. This script takes the output files from the Game of Life codes and automatically converts these into meaningful figures, e.g. speedup and efficiency.

The script that was written scans through all files in the current directory with the suffix $.log$. Each of these $.log$ files are output from runs of the code. From these files the number of processors and the loop execution time is read in. When the script finds more than one result with the same number of processors, it will produce the mean loop execution time. It was decided that four runs of each result were needed to ensure reliable results. To perform this averaging, the current loop time average figures are stored in an array which is indexed by the number of processors corresponding to a given loop time. Another array, also indexed by the number of processors contains the number of counts used to make that average. This is needed so that the mean is correctly weighted when it is re-evaluated.

A later revision of the Perl script saw further additions. As well as the mean time, speedup and efficiency, the maximum and minimum times were also deemed to be necessary. Also, to keep track of the execution count, the number of runs forming the mean, is output to STDOUT. The method of job submission was changed so that multiple outputs were written to a single output file. This meant that the statistical calculations had to be calculated from within the initial loop.

When performing the final results analysis it became apparent that, in order to better quantify the results, Amdahl curves could be used to predict what shape the speedup curves should take. Amdahl's Law states that the proportion of serial (overhead) code in a programme will limit the possible speedup gain. The formula for Amdahl's speedup is

$$S(\alpha, p) = \frac{1}{\alpha + (1-\alpha)\frac{1}{p}}, \tag{1}$$

where α is the overhead proportion associated with the parallel code and p is the number of processors. By calculating a value of α from the results generated, a prediction for the shape of the speedup curve can be made. The calculation of α for each result was implemented into the Perl script. In order to calculate a prediction curve a single value of α must be used, after some experimentation, it was found that the mean value of α generally gave the closest fitting curves.

A complete copy of the Perl script can be found in Appendix A.

3 Results and Analysis

3.1 Operating System Overhead

In order to quantify the O/S overhead on the HPC 3500 a direct comparison between the HPC 3500 and HPC 6500 is required. The dimension size of $N = 840$ was chosen for this analysis because this is the lowest dimension possible which will result in equal load distribution between one and eight processors (see Section 2.2). These comparisons were performed on all versions of the code (OpenMP, MPI, mixed mode) in order to draw more valid conclusions.

3.1.1 Results Obtained

All the results obtained are given as raw data to three significant figures in tabular form and plotted in graphical plots. The time shown is the mean time with the max and min labelled in the tables are the maximum and minimum bounds which were obtained from the fours runs performed.

Machine	Processors	Time	Max	Min	Speedup	Efficiency
HPC 3500	1	324	325	324	1	1
	2	169	169	169	1.91	0.956
	3	116	116	116	2.78	0.928
	4	89.7	89.8	89.6	3.62	0.905
	5	73.4	73.6	73.3	4.42	0.884
	6	62.9	63.0	62.8	5.15	0.859
	7	55.7	56.0	55.6	5.82	0.832
	8	51.3	51.4	51.2	6.32	0.790
HPC 6500	1	326	326	325	1	1
	2	173	175	172	1.87	0.937
	3	121	122	121	2.67	0.893
	4	93.9	94.3	93.4	3.47	0.867
	5	77.4	77.7	77.2	4.20	0.841
	6	66.5	66.6	66.3	4.90	0.816
	7	58.7	59.0	58.4	5.55	0.793
	8	52.9	53.1	52.8	6.15	0.768

Table 2: The times obtained for the OpenMP Game of life code on the HPC 3500 and HPC 6500 on one to eight processors (N=840), together with the resulting speedup and efficiency figures.

The $N = 840$ data from the OpenMP runs of the code are shown in Table 2, with the speedup plotted on Figure 1 and the efficiency plotted on Figure 2.

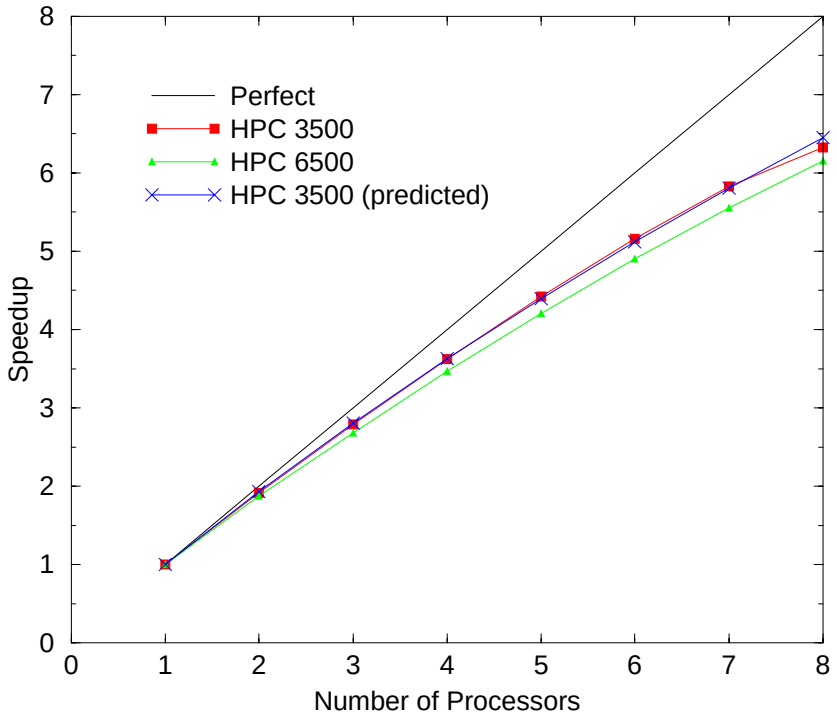


Figure 1: A graph illustrating the speedup of the OpenMP version of the game of life code (N=840).

It can be seen that the speedup and efficiency for the HPC 3500 is always better that the HPC 6500. This is initially surprising considering that both machines have identical processors. However, the memory access rate is lower on the HPC 6500 by twenty percent, so the performance suffers in a direct comparison with the HPC 3500.

The difference between the HPC 3500 and the HPC 6500 increases except when eight processors are being used, where the HPC 3500 appears to show a dip in performance. On Figure 1 there is also an Amdahl curve plotted for the HPC 3500 results. The fit of the curve is extremely good, and illustrates the unexpected drop in speedup at eight processors.

The performance drop caused by the O/S overhead is not as great as the memory bandwidth bottleneck of the HPC 6500. The HPC 3500 as a result retains its superior performance.

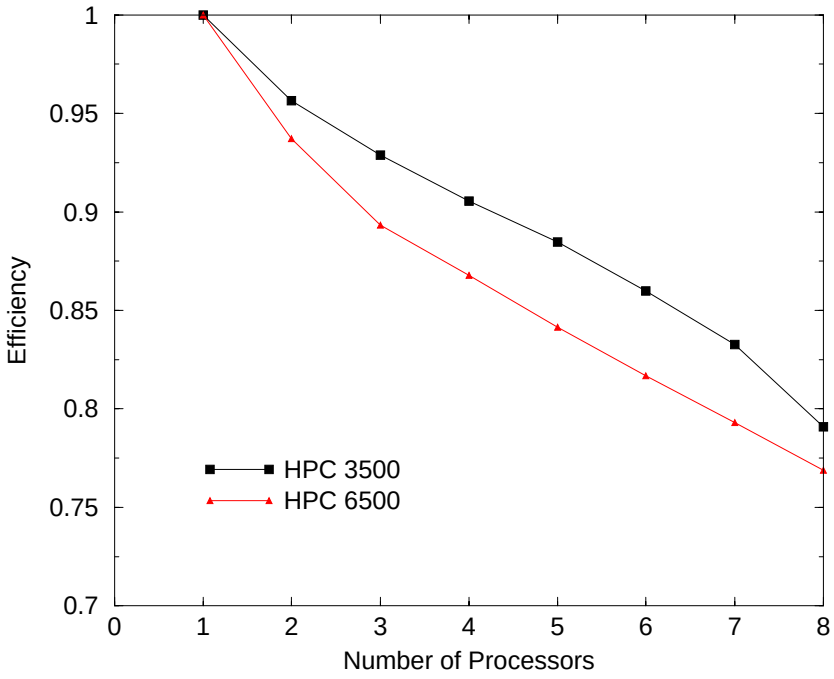


Figure 2: A graph illustrating the efficiency of the OpenMP version of the game of life code (N=840).

Machine	Processors	Time	Max	Min	Speedup	Efficiency
HPC 3500	1	379	380	378	1	1
	2	198	199	198	1.90	0.953
	3	141	141	140	2.68	0.894
	4	100	101	100	3.75	0.939
	5	99.1	99.4	98.7	3.82	0.765
	6	71.5	71.8	71.3	5.29	0.883
	7	88.2	90.1	87.4	4.29	0.613
	8	61.1	61.3	61.0	6.19	0.774
HPC 6500	1	381	381	381	1	1
	2	200	200	199	1.90	0.952
	3	141	141	140	2.70	0.900
	4	100	101	100	3.78	0.947
	5	99.3	100	98.9	3.83	0.767
	6	71.5	71.8	71.3	5.32	0.887
	7	86.5	86.8	86.3	4.40	0.629
	8	57.9	57.9	57.9	6.58	0.822

Table 3: The times obtained from running the MPI (2D) Game of life of the HPC 3500 and HPC 6500 on one to eight processors, together with the resulting speedup and efficiency figures.

The $N = 840$ data from the two dimensional decomposition MPI runs of the code are shown in Table 3, with the speedup plotted on Figure 3 and the efficiency plotted on Figure 4.

Between one and seven processors the there is very little difference between the two machines. At eight processors it can clearly be seen that the HPC 6500 has the superior performance advantage. This result is significant considering that the upper bound on the HPC 6500 is three seconds less than the lower bound from the HPC 3500, see Table 3. At five and seven processors the drop in performance discussed in Section 2.2 is witnessed.

The superior performance of the HPC 3500 between one and seven processors witnessed with the OpenMP code isn't visible with the MPI code. The MPI code seems to be less affected by the memory bottleneck than the OpenMP code. This is perhaps not surprising since the performance of a parallel code exploiting a shared memory architecture will be more susceptible to bandwidth limitations than an explicit message passing model.

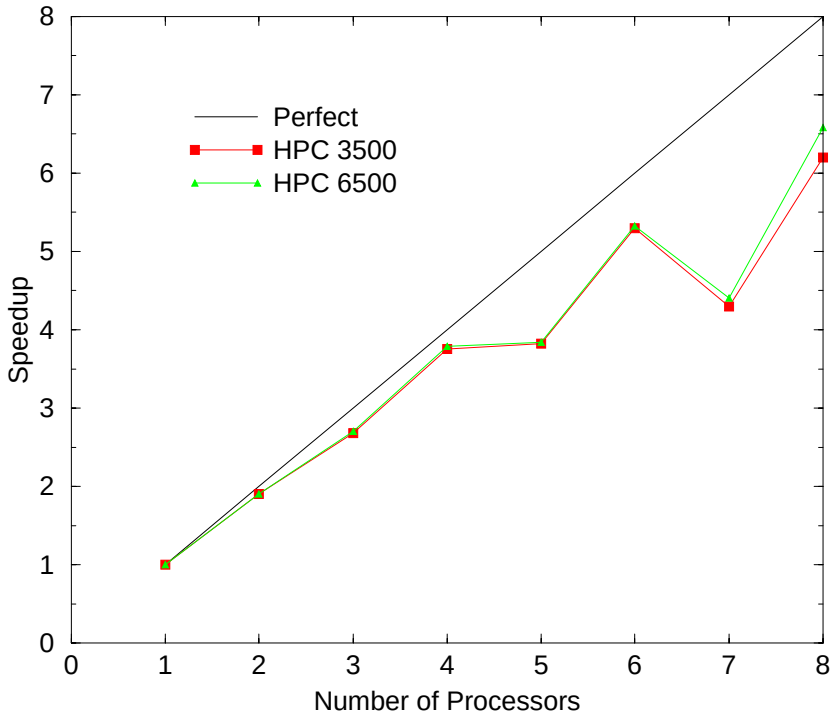


Figure 3: A graph illustrating the speedup of the MPI version of the game of life code (N=840) (2D).

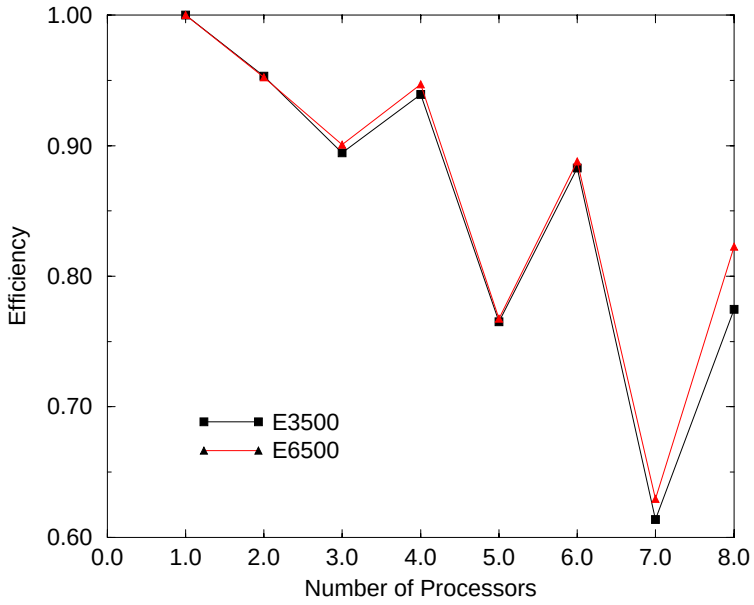


Figure 4: A graph illustrating the efficiency of the MPI version of the game of life code (N=840) (2D).

Machine	Processors	Time	Max	Min	Speedup	Efficiency
HPC 3500	1	379	380	379	1	1
	2	199	199	198	1.90	0.953
	3	141	141	140	2.69	0.897
	4	114	114	113	3.33	0.832
	5	99.4	99.6	99.1	3.81	0.763
	6	91.2	91.5	91.1	4.16	0.693
	7	87.4	87.7	87.0	4.34	0.620
	8	89.0	89.6	88.2	4.26	0.533
HPC 6500	1	379	381	379	1	1
	2	199	199	198	1.90	0.951
	3	140	141	140	2.69	0.898
	4	113	114	113	3.33	0.832
	5	99.0	99.6	98.4	3.83	0.766
	6	91.0	91.3	90.6	4.17	0.695
	7	86.8	88.7	86.0	4.37	0.624
	8	84.3	84.6	84.0	4.50	0.562

Table 4: The times obtained from running the MPI (1D) Game of life of the HPC 3500 and HPC 6500 on one to eight processors (N=840), together with the resulting speedup and efficiency figures.

The $N = 840$ data from the one dimensional decomposition MPI runs of the code are shown in Table 3.1.1, with the speedup plotted on Figure 5 and the efficiency plotted on Figure 6.

The pattern with the one dimensional code is similar in shape to the OpenMP results, with tailing off in the speedup. Like the two dimensional MPI code the difference between the two machines on one to seven processors is negligible. It can also be seen that the one dimensional decomposition is generally far less efficient than the equivalent two dimensional decomposition (compare Figures 4 and 6).

An initial visual comparison between the plots reveals the HPC 3500 unexpectedly dips at eight processors while the HPC 6500 continues upon its expected trajectory. An Amdahl curve has been included on the plot. The curve is a good fit up until six processors, where it begins to diverge from the actual results and over estimates the speedup. This highlights the inability of Amdahl's law to properly model the increasing communication overhead.

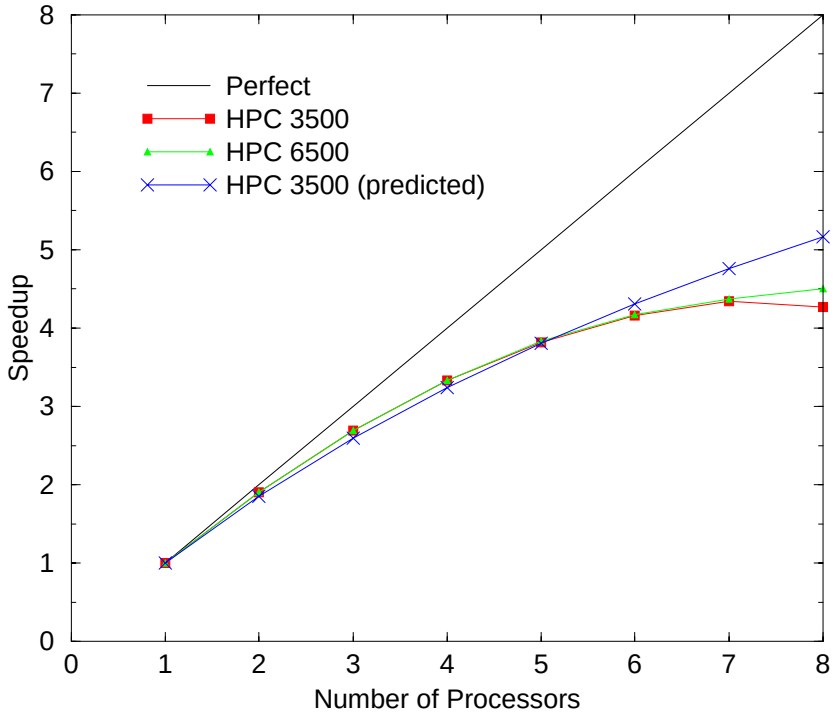


Figure 5: A graph illustrating the speedup of the MPI version of the game of life code (N=840) (1D).

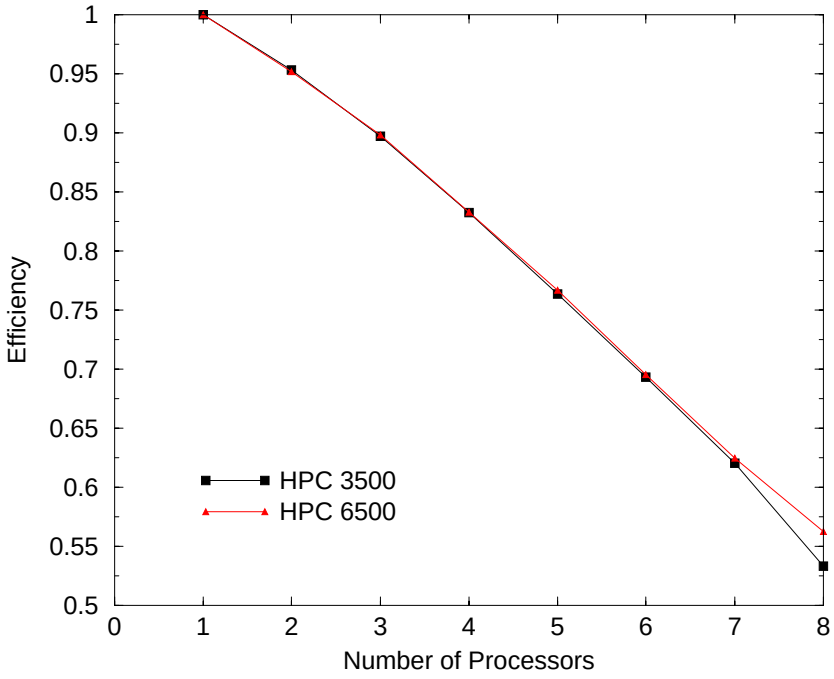


Figure 6: A graph illustrating the efficiency of the MPI version of the game of life code (N=840) (1D).

Machine	Processors	MPI	OMP	Time	Max	Min	Speedup	Efficiency
HPC 3500	1	1	1	327	328	326	1	1
	2	2	1	172	173	172	1.89	0.946
	2	1	2	171	172	171	1.90	0.952
	4	4	1	88.0	88.4	87.4	3.71	0.929
	4	2	2	100	101	100	3.24	0.810
	4	1	4	92.3	92.4	92.2	3.54	0.885
	6	6	1	63.0	63.1	62.8	5.19	0.865
	6	3	2	78.1	78.2	77.9	4.18	0.698
	6	2	3	74.1	74.2	73.8	4.41	0.736
	6	1	6	66.1	66.1	65.9	4.95	0.825
	8	8	1	55.0	55.2	54.9	5.94	0.742
	8	4	2	54.8	55.1	54.7	5.96	0.745
	8	2	4	63.8	64.2	63.4	5.12	0.641
	8	1	8	55.3	55.3	55.1	5.91	0.739
HPC 6500	1	1	1	328	329	328	1	1
	2	2	1	173	174	173	1.89	0.946
	2	1	2	177	179	175	1.85	0.925
	4	4	1	88.4	88.6	88.0	3.71	0.929
	4	2	2	107	108	106	3.04	0.762
	4	1	4	97.5	97.9	97.2	3.37	0.842
	6	6	1	63.4	63.7	63.3	5.17	0.863
	6	3	2	79.5	79.7	79.2	4.13	0.689
	6	2	3	79.5	80.2	79.0	4.13	0.689
	6	1	6	69.6	70.1	69.4	4.72	0.786
	8	8	1	52.1	52.3	51.9	6.30	0.788
	8	4	2	55.0	55.2	54.8	5.97	0.746
	8	2	4	65.8	65.9	65.6	4.99	0.624
	8	1	8	56.2	56.3	56.1	5.84	0.730

Table 5: The times obtained from running the mixed mode Game of life of the HPC 3500 and HPC 6500 on one to eight processors (N=840), together with the resulting speedup and efficiency figures.

The $N = 840$ data from the mixed mode runs of the code are shown in Table 5, with the speedup plotted on Figures 7, 8, 9 and 10.

Figure 7 is a plot of the results where the number of MPI processes have been varied between one and eight, using one thread per process. Here the HPC 3500 and HPC 6500 have near identical performance, with the exception being when the HPC 3500 is using eight processors. These results are very similar to the equivalent MPI results (Table 3). The Amdahl plot is a very close match to the actual results except at eight processors. Again the prediction is better than the actual HPC 3500 result, with the HPC 6500 almost exactly in between the two. This is evidence of an extra overhead at eight processors.

Figure 8 is a plot of the results where the number of MPI processes have been varied between one and four, using two threads per process. The two machines have near identical performance at eight processors, however, before this point the HPC 3500 has better performance. The interpretation is that the overhead of the O/S on the HPC 3500 will be balanced by the overhead of the memory bandwidth bottleneck on the HPC 6500. The Amdahl plot is a very poor fit, though this is because at eight processors a 2×2 MPI decomposition is being performed, which will give a significant jump over the 3×1 decomposition at six processors

Figure 9 is a plot of the results where the number of OpenMP threads have been varied between one and four, using two MPI processes. Figure 10 is a plot of the results where the number of OpenMP threads have been varied between one and eight, using one MPI process. In both of these examples the HPC 3500 maintains its performance advantage even at eight processors. At eight processors there is a slump in performance of the HPC 3500. In both of these plots the Amdahl curve is a very good match except at eight processors where it overestimates the speedup.

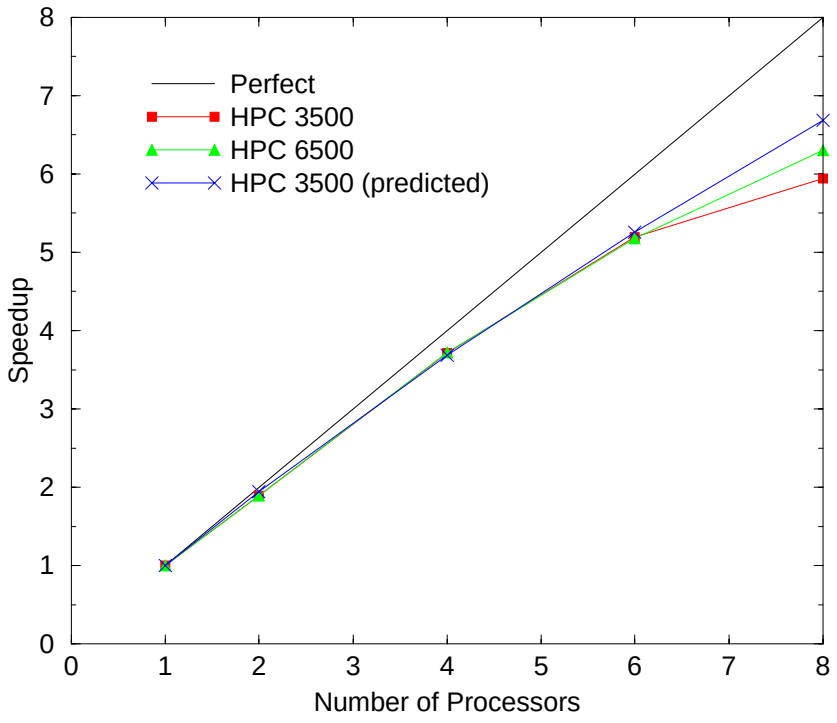


Figure 7: A graph illustrating the speedup of the mixed mode version of the Game of life code (N=840) (MPI=1-8,OMP=1).

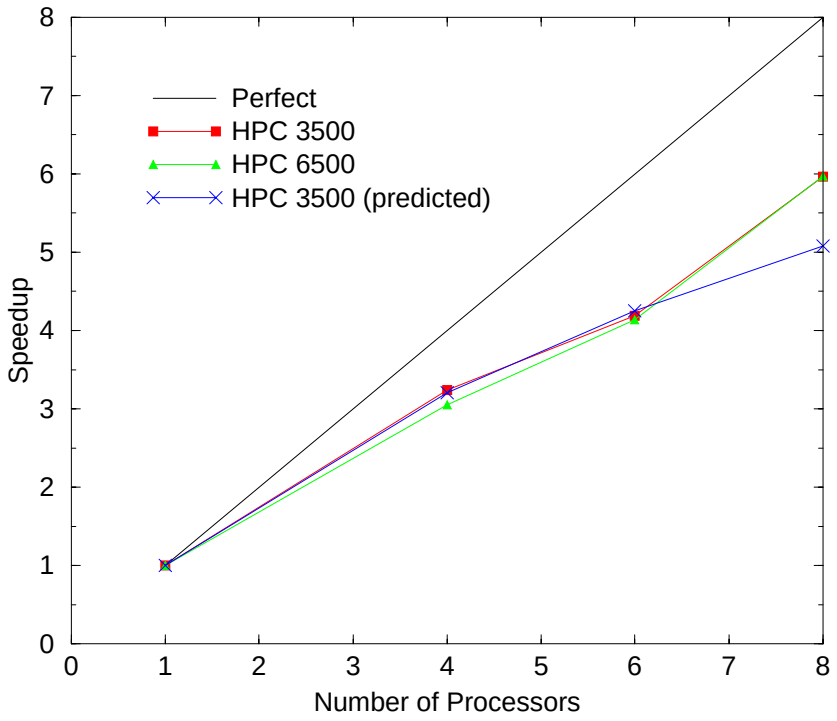


Figure 8: A graph illustrating the speedup of the mixed mode version of the Game of life code (N=840) (MPI=1-4,OMP=2).

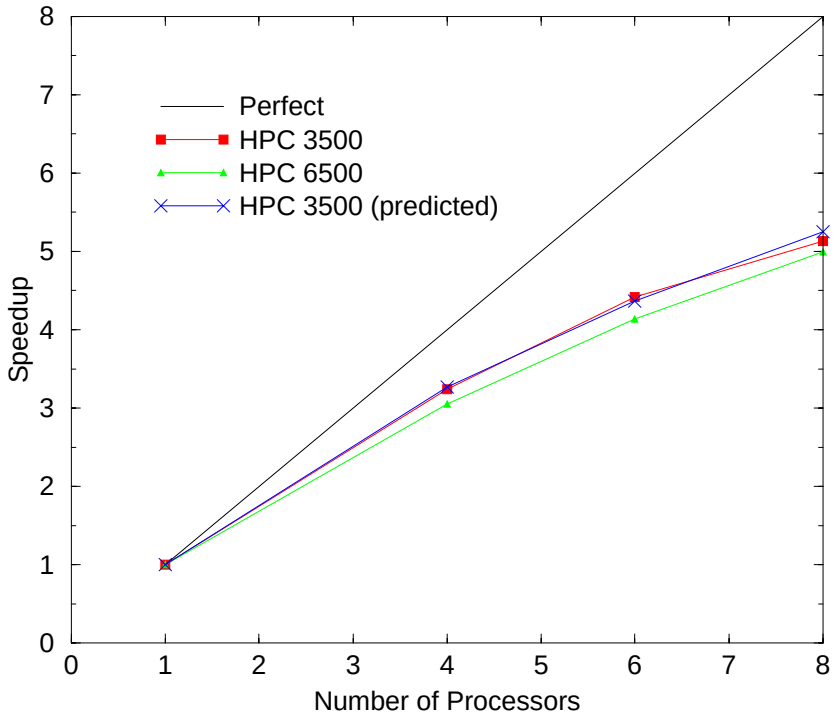


Figure 9: A graph illustrating the speedup of the mixed mode version of the Game of life code (N=840) (MPI=2,OMP=1-4).

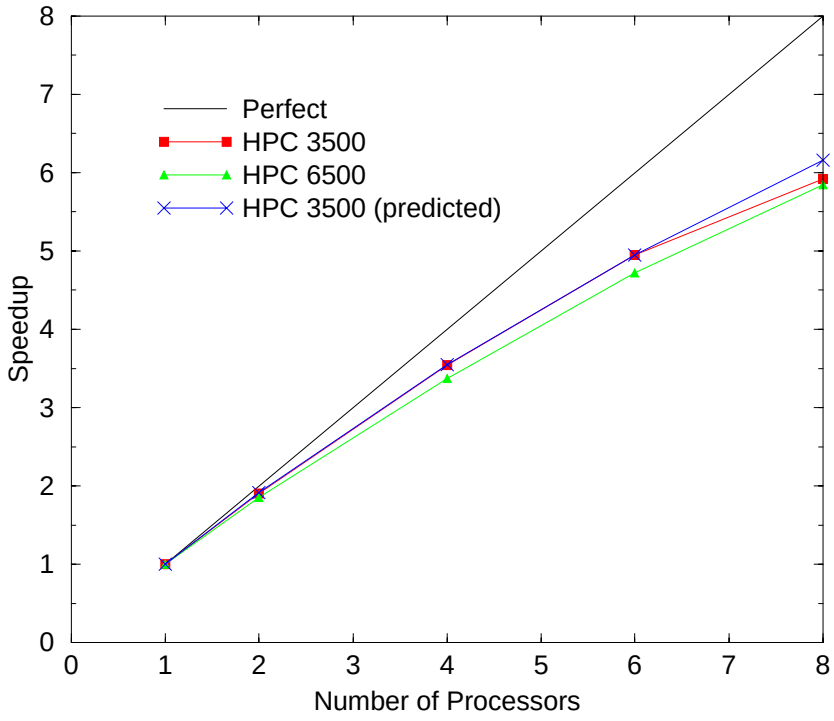


Figure 10: A graph illustrating the speedup of the mixed mode version of the Game of life code (N=840) (MPI=1,OMP=1-8).

3.2 Peak Performance Measure

Machine	Processors	Time	Max	Min	Speedup	Efficiency
HPC 3500	1	5.13	5.1	5.1	1	1
	2	2.62	2.6	2.6	1.95	0.978
	3	1.75	1.7	1.7	2.93	0.977
	4	1.31	1.3	1.3	3.91	0.977
	5	1.05	1.0	1.0	4.87	0.975
	6	0.87	0.8	0.8	5.86	0.978
	7	0.75	0.7	0.7	6.83	0.976
	8	0.65	0.6	0.6	7.80	0.975
HPC 6500	1	5.13	5.1	5.1	1	1
	2	2.63	2.6	2.6	1.95	0.976
	3	1.75	1.7	1.7	2.93	0.976
	4	1.31	1.3	1.3	3.91	0.979
	5	1.05	1.0	1.0	4.89	0.978
	6	0.875	0.8	0.8	5.87	0.978
	7	0.750	0.7	0.7	6.85	0.978
	8	0.656	0.6	0.6	7.82	0.978
	9	0.583	0.5	0.5	8.81	0.978
	10	0.525	0.5	0.5	9.77	0.977
	11	0.477	0.4	0.4	10.7	0.979
	12	0.438	0.4	0.4	11.7	0.975
	13	0.404	0.4	0.4	12.7	0.977
	14	0.377	0.3	0.3	13.6	0.972
	15	0.352	0.3	0.3	14.5	0.972
	16	0.330	0.3	0.3	15.5	0.972
	17	0.311	0.3	0.3	16.5	0.971
	18	0.294	0.2	0.2	17.4	0.970

Table 6: The times obtained from running of the HPC 3500 and HPC 6500 up to eighteen processors (where available) using the OpenMP loop code.

The peak performance data from the OpenMP runs are shown in Table 3.2, with the speedup plotted on Figure 11 and the efficiency plotted on Figure 12.

The results from the OpenMP code show a very good linear speedup curve. There is very little visible tailing off of the speedup curve as would be expected from such a simple code. The efficiency profiles are visibly different, with an apparent tailing off at eight processors. However, because of the scales involved, the differences are negligible. This tends to suggest that the O/S overhead only becomes apparent when it is competing for memory access.

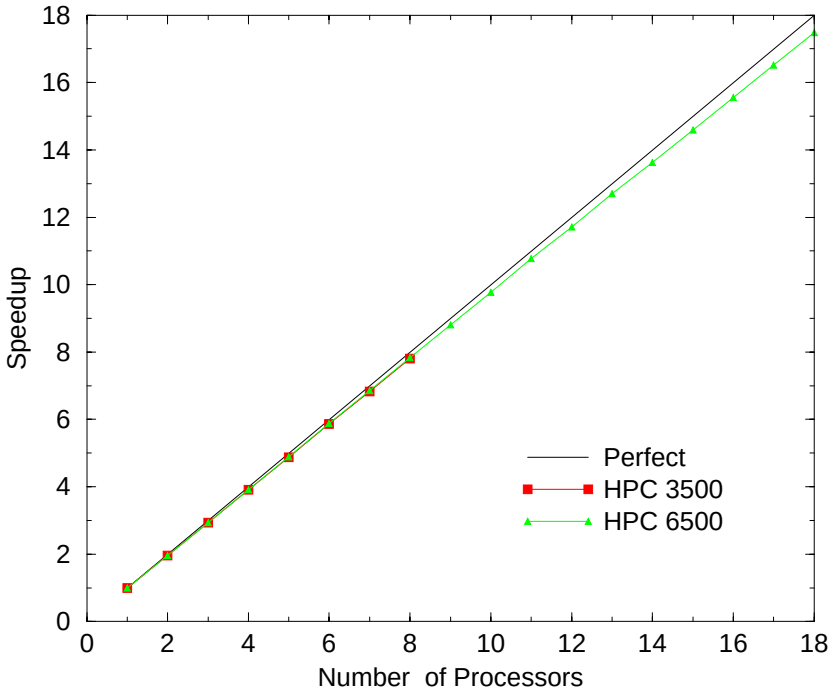


Figure 11: A graph illustrating the speedup of the OpenMP version of the peak performance code between one and eighteen processors.

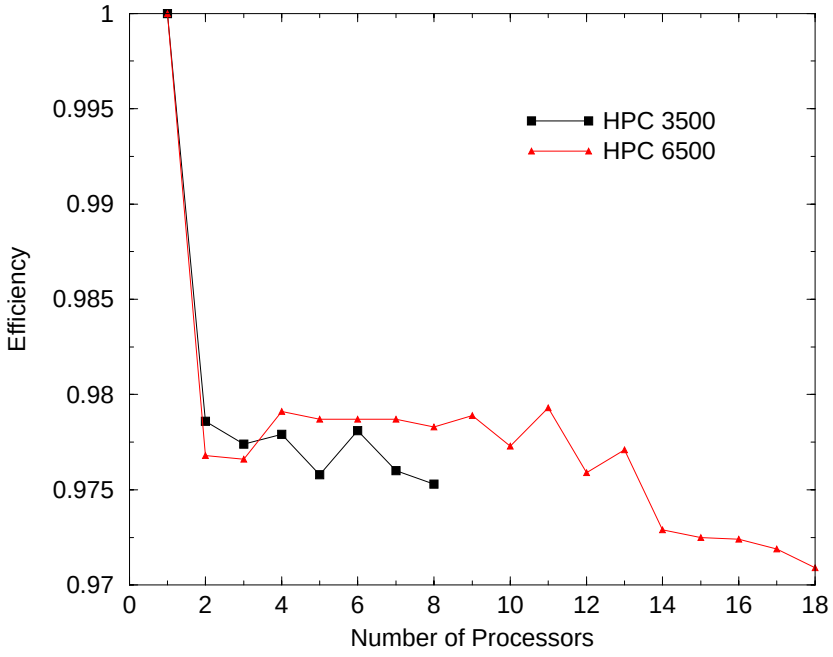


Figure 12: A graph illustrating the efficiency of the OpenMP version of the peak performance code between one and eighteen processors.

Machine	Processors	Time	Max	Min	Speedup	Efficiency
HPC 3500	1	5.13	5.1	5.1	1	1
	2	2.62	2.6	2.6	1.95	0.978
	3	1.75	1.7	1.7	2.93	0.978
	4	1.31	1.3	1.3	3.91	0.979
	5	1.05	1.0	1.0	4.88	0.977
	6	0.876	0.8	0.8	5.86	0.976
	7	0.752	0.7	0.7	6.83	0.976
	8	0.703	0.8	0.6	7.30	0.913
HPC 6500	1	5.14	5.1	5.1	1	1
	2	2.62	2.6	2.6	1.95	0.978
	3	1.75	1.7	1.7	2.93	0.979
	4	1.31	1.3	1.3	3.91	0.979
	5	1.05	1.0	1.0	4.89	0.978
	6	0.875	0.8	0.8	5.87	0.979
	7	0.750	0.7	0.7	6.85	0.979
	8	0.656	0.6	0.6	7.82	0.978
	9	0.584	0.5	0.5	8.79	0.977
	10	0.525	0.5	0.5	9.77	0.977
	11	0.479	0.4	0.4	10.7	0.975
	12	0.438	0.4	0.4	11.7	0.978
	13	0.404	0.4	0.4	12.7	0.978
	14	0.375	0.3	0.3	13.7	0.979
	15	0.351	0.3	0.3	14.6	0.975
	16	0.329	0.3	0.3	15.5	0.974
	17	0.309	0.3	0.3	16.5	0.976
	18	0.293	0.2	0.2	17.5	0.973

Table 7: The times obtained from running of the HPC 3500 and HPC 6500 up to eighteen processors (where available) using the MPI loop code.

The peak performance data from the MPI runs are shown in Table 3.2, with the speedup plotted on Figure 13 and the efficiency plotted on Figure 14.

Similar to the OpenMP code, there is a very linear speedup with little tailing off found when benchmarking with the MPI loop code. At eight processors there is no apparent performance dip due to the O/S.

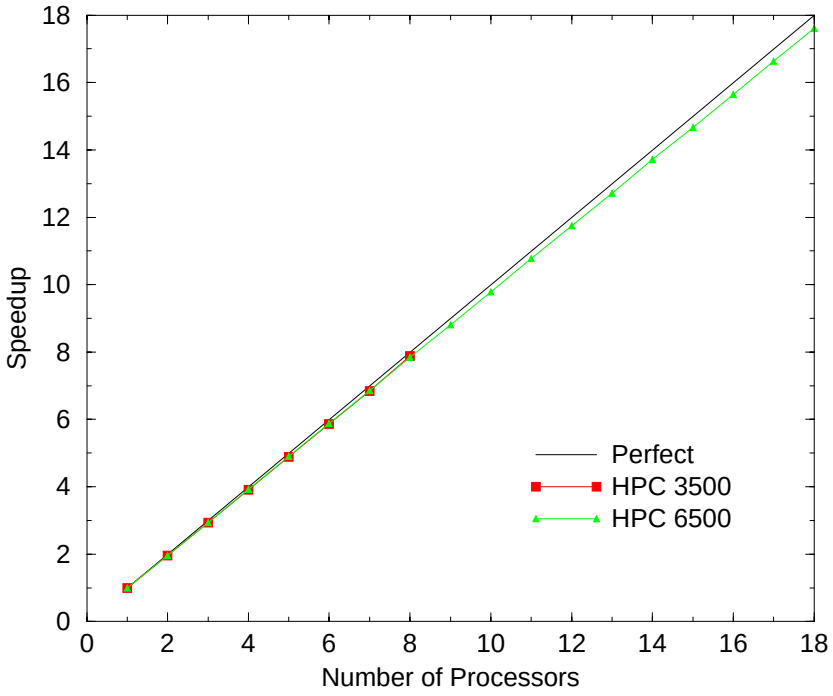


Figure 13: A graph illustrating the speedup of the MPI version of the peak performance code between one and eighteen processors.

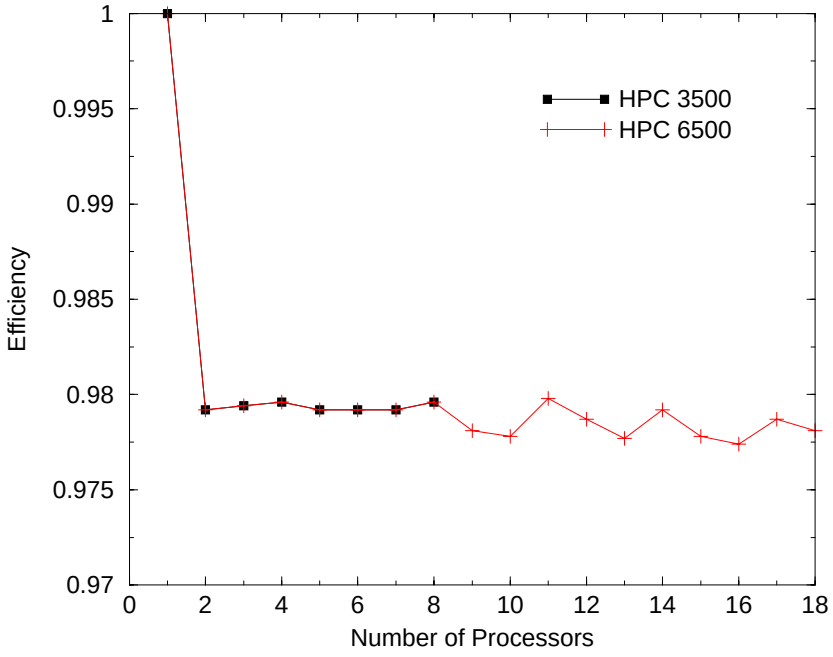


Figure 14: A graph illustrating the efficiency of the MPI version of the peak performance code between one and eighteen processors.

3.2.1 Quantifying the O/S Overhead

After the comparison results were completed it was apparent that the O/S overhead does have an affect on code performance. Part of the aim of this project was to assess whether the overhead caused the O/S warrants the sole allocation of a processor for the O/S.

In order to try and obtain some estimate of the overhead size, a direct comparison between the HPC 3500 and HPC 6500 times has to be performed. A method of doing this is to look at the ratio of the times between the machines. This ratio should remain approximately constant, except when the O/S overhead becomes apparent. Ideally the ratio should remain constant until it reaches eight processors, where a deviation from this should be seen. If we call this deviation difference δ, then δ is a measure of the overhead of the O/S.

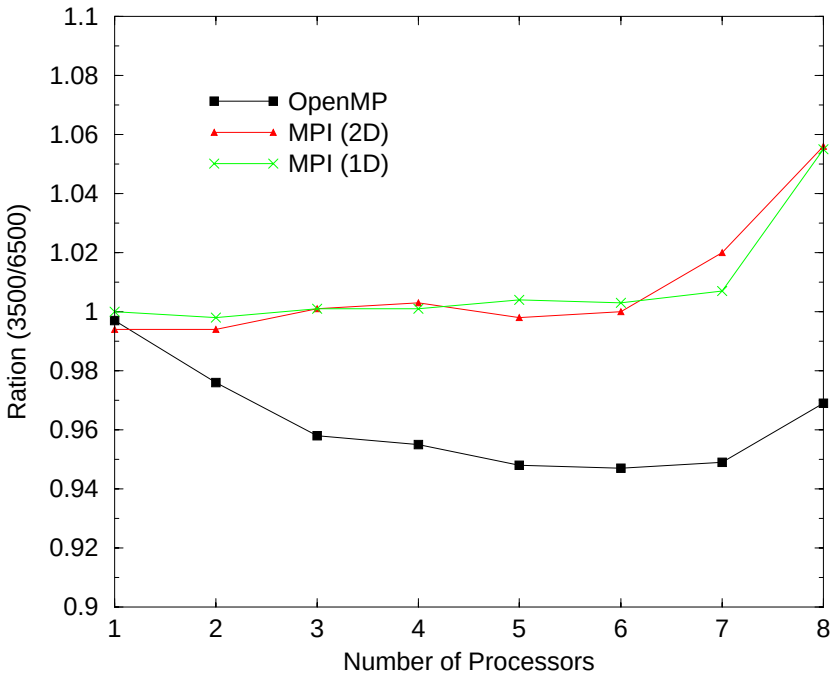


Figure 15: A plot of the HPC 3500 and HPC 6500 time ratio (N=840) using the OpenMP and MPI versions of the Game of Life code.

The ratios from the OpenMP and MPI runs of the code are plotted plotted on Figure 15. This reveals the change in ratio at eight processors. The MPI ratios both remain approximately constant at around unity until this point. For both MPI codes $\delta \approx 0.05$, with any further accuracy not possible. The OpenMP ratio has more variation, however, if the processor numbers one and two are ignored, then the variation is much reduced. This is valid because the first two results are the most likely to be unreliable. If we now take a measure of δ as the deviation from processors three to seven ratio we find $\delta \approx 0.02$.

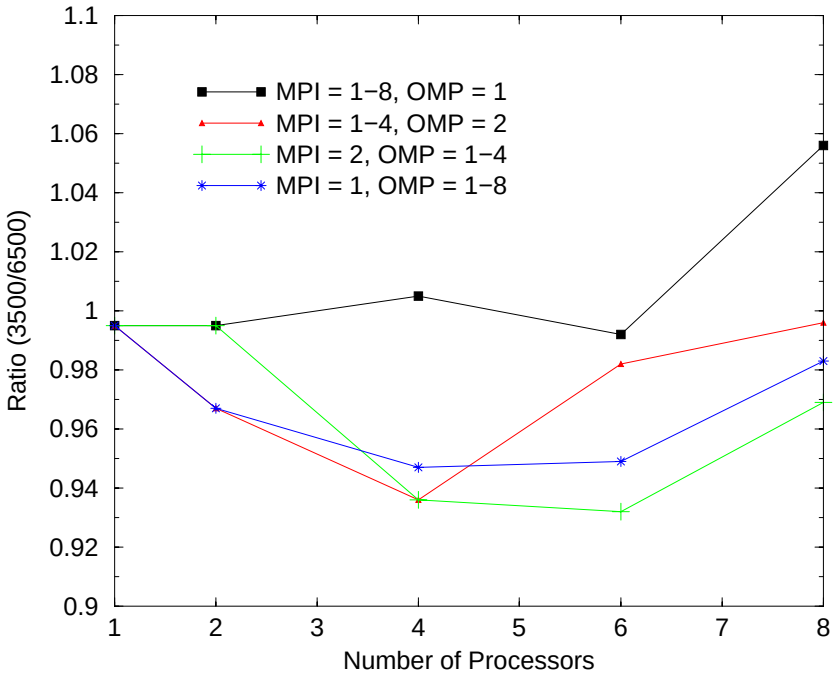


Figure 16: A plot of the HPC 3500 and HPC 6500 time ratio (N=840) using the mixed mode version of the Game of Life code.

The ratios from the OpenMP and MPI runs of the code are plotted plotted on Figure 15. The ratios seem to have much more variation than when using pure OpenMP or MPI. The results where only one OpenMP thread has been used, is the most consistent set of results, this being similar to the equivalent MPI result. Again here the value of $\delta \approx 0.05$. The introduction of multiple OpenMP threads complicates the analysis somewhat with such a variation in the ratios. What can be seen however, is that at no point does the value of δ exceed the estimation obtained from the single threaded result of $\delta \approx 0.05$.

Unfortunately due to the variation in the results, an absolute value of the O/S overhead cannot be given, though a bound can be given. The most reliable estimates seem to come from the MPI based results, and it is these results which also give the maximum value of δ. Therefore is can be said with a degree of certainty that $\delta < 0.06$. With an eight processor machine this represents nearly 50% of a single processor's CPU time. While this might seem significant, the actual affect the overhead has on the runtime does not warrant sole allocation to a processor.

3.3 Scaling on the HPC 6500

The aim of this section was to look at how well the HPC 6500 scales up to eighteen processors. As before with the HPC 3500 at eight processors, the HPC 6500 should reveal a dip in performance at eighteen processors, which can be attributed to the O/S overhead. Using increasing numbers of processors can also give otherwise unexpected performance benefits due to caching issues.

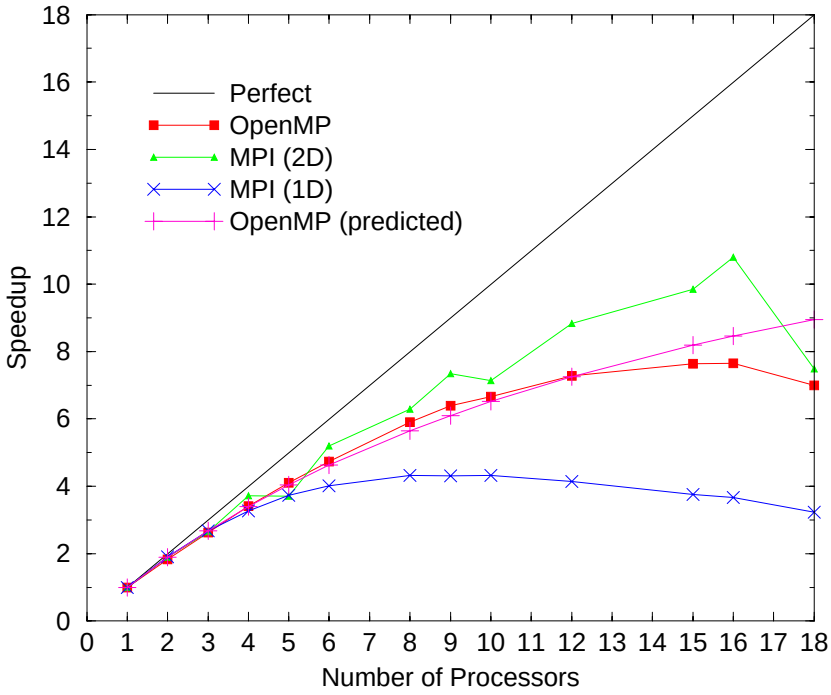


Figure 17: A graph illustrating the speedup of the Game of Life code on the HPC 6500

The $N = 720$ data from the OpenMP and MPI runs of the code are shown in Table 8, with the speedup plotted on Figure 17.

Firstly looking at the OpenMP results, it can be seen that when plotted the increasing communication overhead can clearly be witnessed. This fits well with the plotted Amdahl curve, though as before with the HPC 3500 the Amdahl curve over estimates the speedup significantly at the maximum number of processors. This final dip in performance at eighteen processors will be partly associated with the O/S overhead, but also with the communications overhead.

The two dimensional MPI benchmarking reveals particularly that the best performance is always when a square decomposition strategy is implemented, e.g. 2×2, 3×3 etc. The one dimensional decomposition strategy scales very poorly, with the speedup decreasing around ten processors. This curve does not fit with Amdahl's Law and again highlights its inability to model the increasing communications overhead.

Type	Processors	Time	Max	Min	Speedup	Efficiency
OpenMP	1	240	240	239	1	1
	2	130	131	128	1.83	0.917
	3	91.2	91.8	90.8	2.63	0.877
	4	70.5	70.8	70.1	3.40	0.851
	5	58.5	58.7	58.2	4.10	0.820
	6	50.6	50.8	50.3	4.73	0.789
	8	40.6	40.9	40.5	5.90	0.737
	9	37.5	37.8	37.4	6.39	0.710
	10	36.0	36.3	35.5	6.66	0.666
	12	33.0	33.5	32.5	7.27	0.606
	15	31.4	31.6	31.0	7.63	0.509
	16	31.3	31.5	31.2	7.65	0.478
	18	34.3	34.6	34.0	6.99	0.388
MPI (2D)	1	279	279	279	1	1
	2	147	147	147	1.89	0.946
	3	105	105	105	2.65	0.884
	4	75.2	75.4	74.9	3.71	0.928
	5	75.4	76.0	75.0	3.70	0.741
	6	53.8	54.1	53.7	5.19	0.865
	8	44.4	44.9	44.1	6.28	0.785
	9	38.0	38.2	37.9	7.34	0.815
	10	39.1	39.3	38.9	7.14	0.714
	12	31.6	31.7	31.5	8.82	0.735
	15	28.3	28.5	28.3	9.84	0.656
	16	25.8	26.1	25.7	10.7	0.674
	18	37.3	37.6	37.0	7.48	0.415
MPI (1D)	1	280	281	280	1	1
	2	147	147	147	1.90	0.953
	3	105	105	104	2.67	0.890
	4	85.9	86.9	85.2	3.26	0.816
	5	75.2	75.7	74.5	3.73	0.746
	6	69.9	70.8	69.4	4.01	0.668
	8	65.0	65.3	64.8	4.31	0.539
	9	65.0	65.4	64.7	4.31	0.479
	10	64.9	65.6	64.6	4.32	0.432
	12	67.8	68.5	67.1	4.13	0.344
	15	74.6	75.1	74.0	3.76	0.250
	16	76.5	77.0	76.0	3.66	0.229
	18	87.0	88.1	85.9	3.22	0.179

Table 8: The times obtained from running the OpenMP and MPI Game of Life on the HPC 6500 between one and eighteen processors (N=720), together with the resulting speedup and efficiency figures.

Type	Processors	Time	Max	Min	Speedup	Efficiency
OpenMP	1	5740	5760	5710	1	1
	2	2910	2910	2910	1.97	0.987
	4	1540	1550	1540	3.72	0.931
	5	1260	1260	1250	4.55	0.911
	6	1070	1070	1070	5.34	0.890
	8	892	903	886	6.43	0.804
	9	816	820	805	7.03	0.782
	10	753	758	745	7.62	0.762
	11	713	721	704	8.05	0.732
	12	469	469	469	12.2	1.019
	16	369	372	363	15.5	0.973
	17	350	369	342	16.4	0.964
	18	330	333	325	17.3	0.964
MPI 1D	1	6550	6550	6550	1	1
	8	865	873	860	7.57	0.946
	9	811	821	806	8.07	0.897
	12	732	741	726	8.94	0.745
	16	702	711	694	9.32	0.582
	17	749	752	742	8.74	0.514
	18	757	765	746	8.65	0.480

Table 9: The times obtained from running the OpenMP and MPI (1D) Game of life on the HPC 6500 between one and eighteen processors (N=2448), together with the resulting speedup and efficiency figures.

The $N = 2448$ data from the OpenMP and MPI runs of the code are shown in Table 8, with the speedup plotted on Figure 17.

This larger N size was initially investigated to give more reliable scaling up to eighteen processors, as the dimension increases the calculation:communication ratio decreases.

The MPI plot is as one would expect, the usual tailing off in the results, with this tailing off occurring much higher up than with the lower $N = 720$ dimension.

The OpenMP results are initially puzzling, up until eleven processors there is the usual tailing off in the speedup, which appears to be in accordance to Amdahl's law. At twelve processors there is a sudden performance jump which is super-linear before again beginning to tail off. This performance jump can be attributed to caching effects. Since OpenMP operates on a shared memory principle the combined cache of the processors appear as a single large cache. Up until eleven processors the large array size of 2448^2 will always cause cache misses. At twelve processors the combined cache is large enough to contain all arrays within the programme, and no performance hit from the cache misses means a substantial benefit.

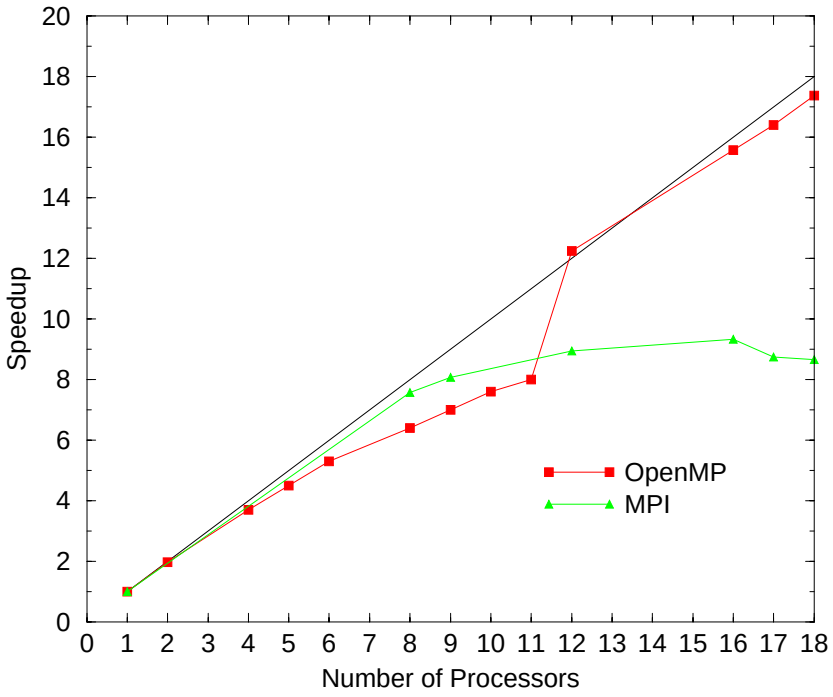


Figure 18: A graph illustrating the speedup of the OpenMP and MPI (1D) version of the game of life code (N=2448).

In order to test this hypothesis, two more sets of results were taken. These were performed on OpenMP only, with array sizes, $N = 1200$ and $N = 1600$. At these sizes it is possible to predict in advance where the performance jumps should take place. The $N = 1200$ code should see a performance jump between two and three processors, while the $N = 1600$ should see a jump between four and five processors.

Dimension	Prediction	Greatest Jump	Maximum Efficiency
1200	2-3	2-3	3
1600	4-5	4-5	6
2448	11-12	11-12	12

Table 10: The locations where the predicted jumps in performance are expected and there actual locations.

The numerical results can be seen in Table 11 and are plotted in Figure 19. The largest performance jumps are seen at their predicted locations (see Table 10). From the plot it can be seen that super linear speedup is witnessed both before and after these locations. For the $N = 1600$ result, the greatest efficiency figures do not concur with the predictions. This can be explained by the fact the code in fact uses two arrays of size N, and as a result the point of cache miss or no cache miss becomes clouded. These results do confirm the hypothesis.

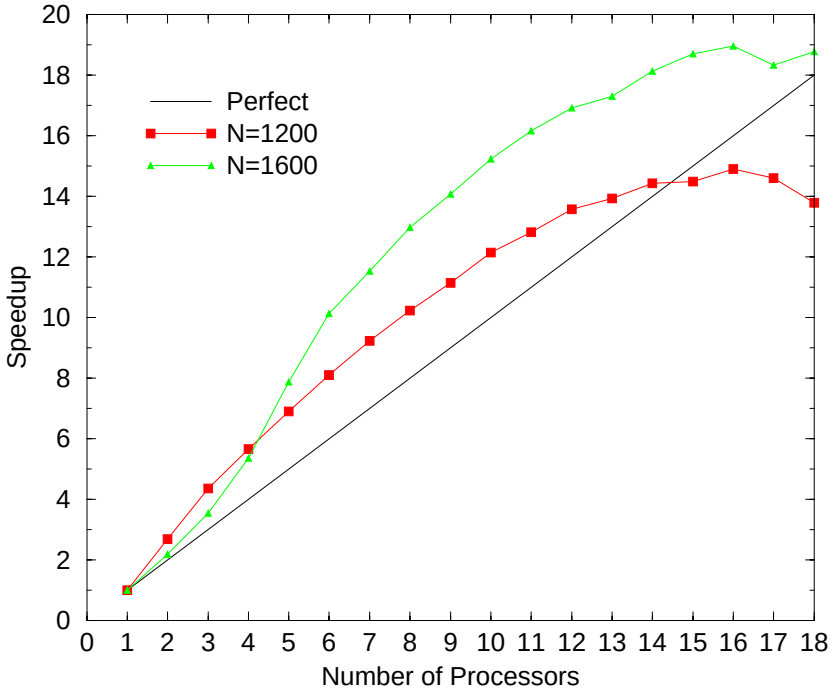


Figure 19: A graph illustrating the speedup of the OpenMP version of the Game of Life code (N=1200 and N=1600).

Dimension	Processors	Time	Max	Min	Speedup	Efficiency
N=1200	1	1050	1050	1050	1	1
	2	390	391	390	2.68	1.34
	3	240	241	238	4.35	1.45
	4	185	187	184	5.65	1.41
	5	152	153	150	6.90	1.38
	6	129	130	128	8.10	1.35
	7	113	114	112	9.22	1.31
	8	102	103	101	10.2	1.27
	9	94.2	95.3	93.3	11.1	1.23
	10	86.4	86.7	85.8	12.1	1.21
	11	81.9	83.2	81.2	12.8	1.16
	12	77.3	77.8	76.9	13.5	1.13
	13	75.3	77.1	74.1	13.9	1.07
	14	72.8	73.3	72.0	14.4	1.03
	15	72.4	73.0	71.8	14.4	0.96
	16	70.4	72.2	68.6	14.9	0.93
	17	71.9	72.8	70.8	14.5	0.85
	18	76.1	78.2	72.7	13.7	0.76
N=1600	1	2250	2240	2240	1	1
	2	1020	1020	1020	2.18	1.09
	3	633	639	627	3.54	1.18
	4	418	422	414	5.36	1.34
	5	285	286	284	7.86	1.57
	6	221	222	221	10.1	1.68
	7	194	195	193	11.5	1.64
	8	173	174	171	12.9	1.62
	9	159	160	158	14.0	1.56
	10	147	148	146	15.2	1.52
	11	138	139	138	16.1	1.46
	12	132	135	130	16.9	1.40
	13	129	130	128	17.2	1.33
	14	123	126	121	18.1	1.29
	15	120	122	117	18.7	1.24
	16	118	120	117	18.9	1.18
	17	122	124	120	18.3	1.07
	18	119	123	117	18.7	1.04

Table 11: The times obtained from running the OpenMP Game of Life on the HPC 6500 between one and eighteen processors (N=1200 and N=1600), together with the resulting speedup and efficiency figures.

The $N = 4896$ results were performed in addition to the previous dimensions in an attempt to obtain a better measure of the overhead at eighteen processors. The size of $N = 4896$ is large enough so that it will not fit into any combined cache size until 46 processors are used so in theory the caching effects should be avoided.

Type	Processors	Time	Max	Min	Speedup	Efficiency
OpenMP	1	22300	22300	22300	1	1
	2	11700	11700	11700	1.90	0.951
	4	6220	6250	6190	3.58	0.896
	6	4380	4380	4380	5.09	0.848
	8	3410	3410	3410	6.53	0.816
	10	2900	2910	2900	7.68	0.768
	11	2730	2740	2730	8.15	0.741
	12	2500	2500	2490	8.92	0.743
	13	2270	2280	2270	9.80	0.754
	14	2120	2120	2110	10.5	0.751
	15	1970	1980	1970	11.2	0.752
	16	1870	1880	1860	11.8	0.742
	17	1880	1900	1840	11.8	0.697
	18	1650	1650	1640	13.5	0.751
MPI (1D)	1	25900	25900	25900	1	1
	10	3450	34600	3440	7.51	0.751
	11	3310	3320	3300	7.81	0.710
	12	3170	3170	3170	8.17	0.680
	13	3090	3100	3090	8.37	0.643
	14	3070	3070	3070	8.43	0.602
	15	3000	3000	3000	8.62	0.574
	16	3010	3020	3010	8.59	0.537
	17	3030	3040	3030	8.54	0.502
	18	3310	3320	3310	7.81	0.434

Table 12: The times obtained from running the OpenMP and MPI Game of Life on the HPC 6500 between one and eighteen processors (N=4896), together with the resulting speedup and efficiency figures.

The $N = 4896$ data from the OpenMP and one dimensional MPI runs of the code are shown in Table 3.3, with the speedup plotted on Figure 20 and the efficiency plotted on Figure 21.

The MPI result shown in Figure 20 is exactly as expected. The tailing off in the speedup occurs around seventeen processors, it is only eighteen processors which is significantly reduced in performance.

Unfortunately the OpenMP is puzzling. Up until ten processors there is an expected Amdahl type tail off, but proceeding this is an almost linear increase in the speedup.

This is confirmed by seeing that the efficiency numbers remain almost identical. This can only be attributed to an unforeseen caching affect.

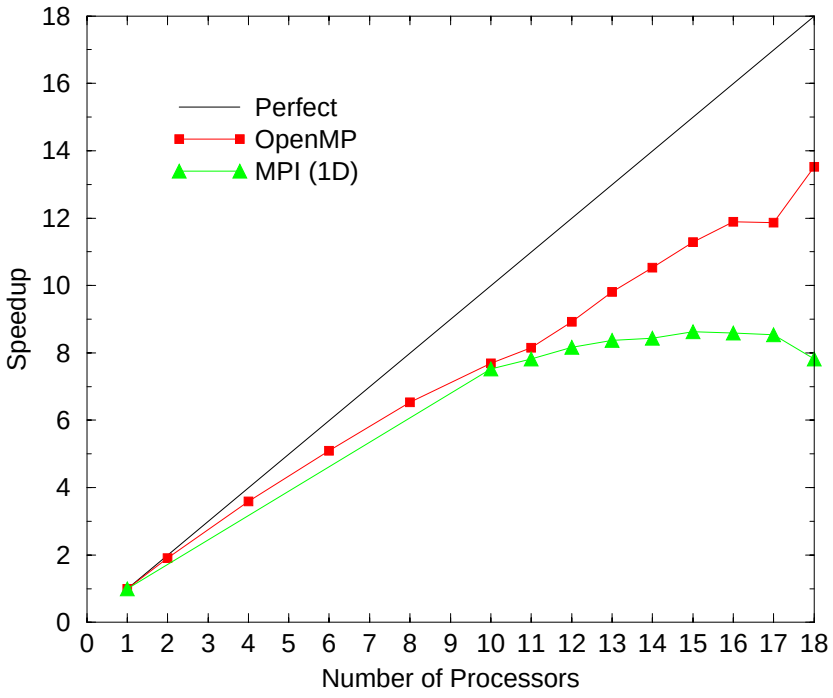


Figure 20: A graph illustrating the speedup of the OpenMP and MPI version of the Game of Life code (N=4896).

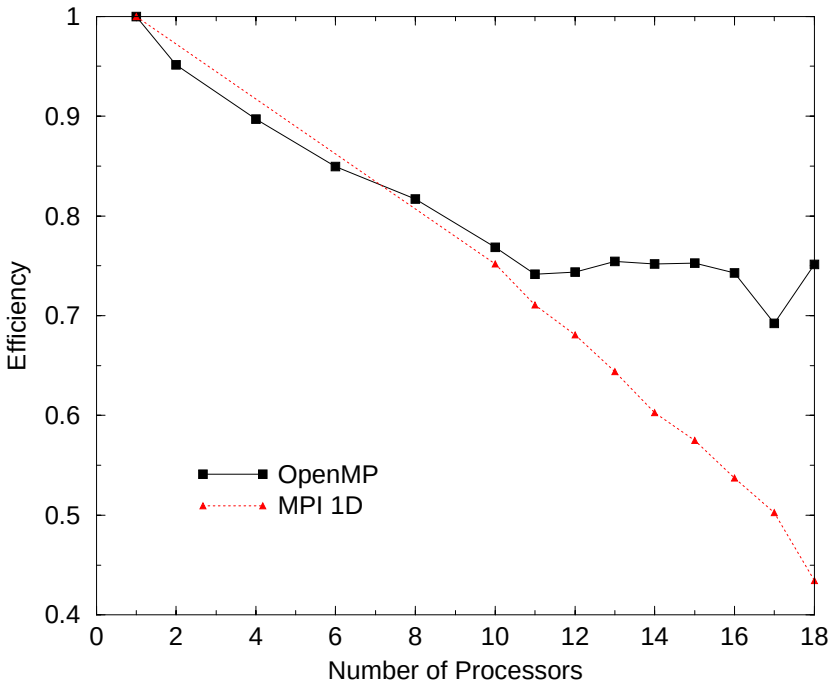


Figure 21: A graph illustrating the efficiency of the OpenMP and MPI version of the Game of Life code (N=4896).

3.4 Wildfire

When performing benchmarking on the Wildfire, the order of processor allocation is important, because it must be known where the Wildfire latency occurs. When performing pure OpenMP or MPI, the threads or processes were allocated to a single box until it full, then another box filled up, before the final box was filled. The notation used in this report states the order of box allocation, e.g. 18-8-8 means that the e6000 is filled first followed by one e4000, and then the final e4000.

The mixed mode code was only performed on the final dimension size, since any benefit would be mostly witnessed here. Each box was allocated as a single MPI process, with up to eight OpenMP threads within each box. The number of threads was always kept equal between the boxes, to keep a balanced decomposition.

The $N = 720$ data from the OpenMP and one dimensional MPI runs of the code are shown in Table 3.4, with the speedup plotted on Figure 22.

The MPI results here scale extreme badly, with a much larger array size being needed. There is a drop in speedup between eight and nine processors and again between sixteen and seventeen processors. This could be because of Wildfire latency, however, considering that there is no evidence of an O/S overhead when the boxes are at full capacity, this is unlikely. The OpenMP results scale much better than MPI, but the tailing off occurs before first box is filled, again meaning that no conclusions can be drawn.

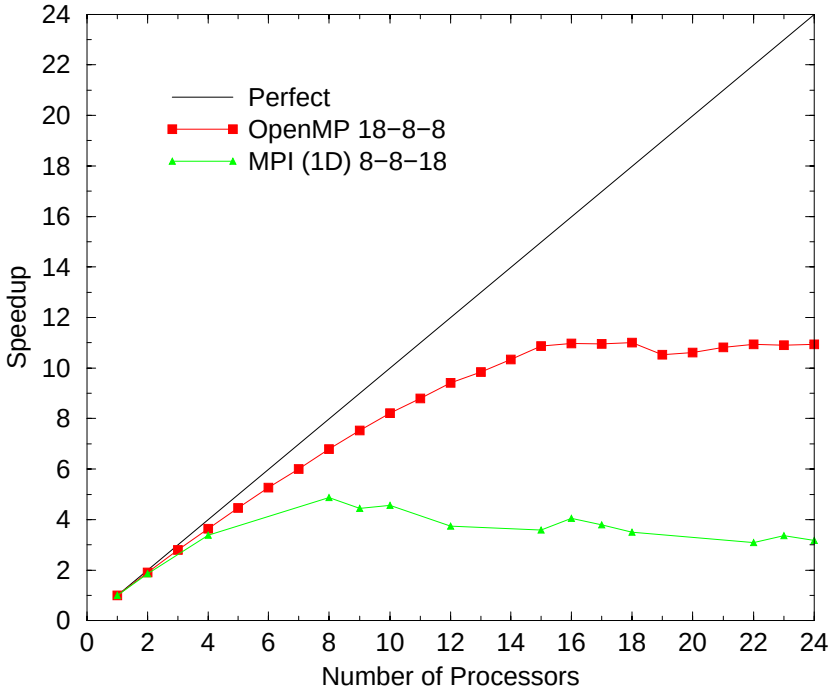


Figure 22: A graph illustrating the speedup the OpenMP and MPI versions of the Game of Life on the Wildfire (N=720).

Machine	Processors	Time	Max	Min	Speedup	Efficiency
OpenMP	1	642	642	642	1	1
	2	336	336	336	1.90	0.953
	3	229	229	229	2.79	0.931
	4	176	176	176	3.64	0.910
	5	143	143	143	4.46	0.892
	6	121	121	121	5.27	0.878
	7	106	106	106	6.01	0.859
	8	94.7	94.7	94.7	6.77	0.847
	9	85.3	85.3	85.3	7.52	0.835
	10	78.3	78.4	78.3	8.19	0.819
	11	73.0	73.0	73.0	8.79	0.799
	12	68.1	68.2	68.1	9.42	0.785
	13	65.2	65.2	65.2	9.84	0.757
	14	62.1	62.2	62.1	10.3	0.737
	15	59.1	59.1	59.1	10.8	0.724
	16	58.0	58.0	57.9	11.0	0.692
	17	56.9	56.9	56.8	11.2	0.663
	18	58.4	58.4	58.4	10.9	0.610
	19	61.0	61.2	60.8	10.5	0.553
	20	60.5	60.7	60.3	10.6	0.530
	21	59.2	59.3	59.2	10.8	0.515
	22	58.6	58.9	58.3	10.9	0.497
	23	58.6	58.8	58.3	10.9	0.476
	24	59.2	59.4	58.9	10.8	0.451
MPI (1D)	1	72.2	72.2	72.1	1	1
	2	38.7	38.9	38.6	1.86	0.931
	4	21.3	21.5	21.1	3.38	0.845
	8	14.8	14.9	14.7	4.86	0.608
	9	16.2	16.4	16.0	4.44	0.493
	10	15.8	16.1	15.5	4.55	0.455
	12	19.3	19.3	19.3	3.73	0.311
	15	20.1	20.7	19.6	3.57	0.238
	16	17.7	18.1	17.4	4.05	0.253
	17	19.0	19.3	18.7	3.78	0.222
	18	20.6	21.1	20.0	3.49	0.194
	22	23.3	23.6	23.0	3.09	0.140
	23	21.4	21.6	21.3	3.36	0.146
	24	22.7	22.8	22.7	3.16	0.132

Table 13: The times obtained from running the Game of Life with Wildfire (N=720).

Type	Processors	Time	Max	Min	Speedup	Efficiency
OpenMP	1	4890	4890	4890	1	1
	2	2540	2540	2540	1.92	0.961
	4	1310	1320	1310	3.71	0.929
	8	701	702	701	6.97	0.871
	9	637	637	637	7.66	0.852
	10	583	584	582	8.38	0.838
	12	516	516	516	9.46	0.788
	15	455	455	455	10.7	0.715
	16	430	430	429	11.3	0.710
	17	416	417	414	11.7	0.691
	18	420	420	420	11.6	0.645
	22	370	371	369	13.1	0.599
	23	357	358	355	13.6	0.595
	24	347	348	346	14.0	0.586
MPI (1D)	1	5050	5060	5050	1	1
	2	2740.	2740	2740	1.84	0.923
	4	1610	1610	1600	3.13	0.784
	8	1260	1260	1250	4.01	0.501
	9	1160	1150	1150	4.37	0.485
	10	1140	1150	1130	4.42	0.442
	12	1080	1090	1070	4.66	0.388
	15	1020	1020	1020	4.93	0.329
	16	1020	1040	990	4.97	0.310
	17	968	992	945	5.22	0.307
	18	959	974	945	5.27	0.292
	22	1260	1290	1240	3.98	0.181
	23	1360	1390	1330	3.70	0.161
	24	1370	1380	1370	3.68	0.153

Table 14: The times obtained from running the Game of Life with Wildfire (N=1920).

The $N = 1920$ data from the OpenMP and one dimensional MPI runs of the code are shown in Table 3.4, with the speedup plotted on Figure 23.

It can be seen that with the larger problem size the MPI results now scale better, but not enough that conclusions can be drawn. Again the OpenMP scales much better that the MPI. There is a performance dip at eighteen processors indicating the presence of the O/S overhead. There is no evidence of Wildfire latency jumping from eighteen to nineteen processors.

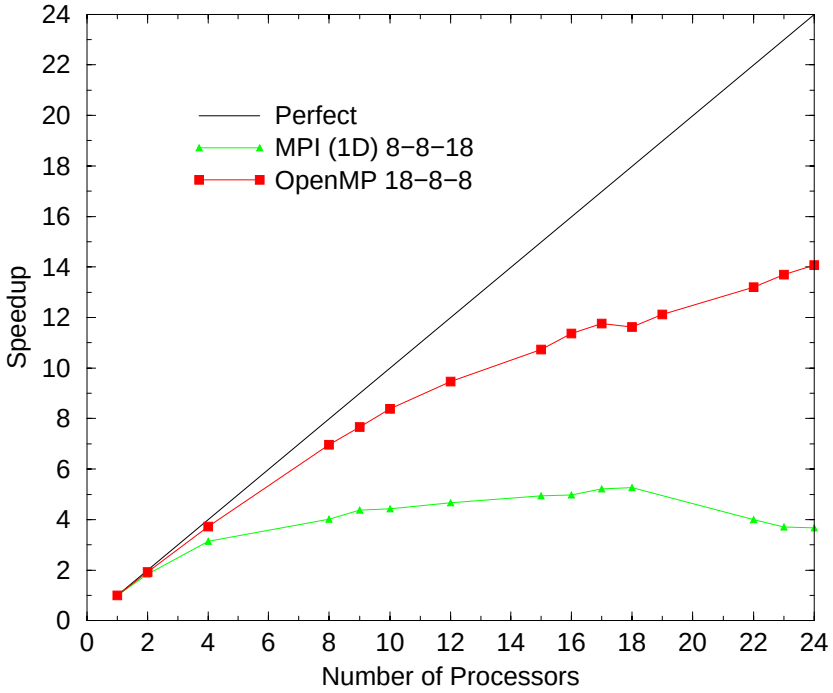


Figure 23: A graph illustrating the speedup the OpenMP and MPI versions of the Game of Life on the Wildfire (N=1920).

Type	Processors	Time	Max	Min	Speedup	Efficiency
OpenMP 8-8-18	1	3210	3210	3210	1	1
1000 iterations	7	533	534	533	6.02	0.860
	8	488	490	487	6.58	0.822
	9	440	440	439	7.30	0.811
	10	400	400	400	8.02	0.802
	12	338	338	338	9.50	0.792
	15	279	279	278	11.5	0.768
	16	265	265	265	12.1	0.757
	17	250	250	250	12.8	0.754
	18	239	239	238	13.4	0.747
	22	201	201	201	15.9	0.725
	23	194	194	194	16.5	0.719
	24	187	187	186	17.1	0.715
OpenMP 18-8-8	1	31600	31600.	31600.	1	1
10000 iterations	7	4920	4920	4920	6.42	0.917
	8	4340	4340	4340	7.27	0.909
	9	3890	3890	3890	8.11	0.902
	10	3550	3550	3550	8.90	0.890
	12	3030	3030	3030	10.4	0.869
	15	2580	2580	2580	12.2	0.816
	16	2490	2490	2490	12.7	0.794
	17	2440	2440	2430	12.9	0.762
	18	2490	2490	2490	12.6	0.703
	19	2260	2260	2260	13.9	0.735
	22	2120	2120	2120	14.9	0.677
	23	2040	2040	2040	15.4	0.672
	24	1970	1980	1970	15.9	0.666
MPI 1D	1	35400	35400	35400	1	1
10000 iterations	7	6410	6420	6400	5.53	0.790
	8	6140	6160	6110	5.77	0.721
	9	5700	5720	5680	6.21	0.690
	10	5340	5360	5310	6.64	0.664
	12	5000	5020	4980	7.08	0.590
	15	4680	4690	4670	7.56	0.504
	16	4600	4630	4570	7.70	0.481
	17	4700	4700	4690	7.54	0.443
	18	4670	4690	4650	7.58	0.421
	22	4790	4820	4760	7.39	0.336
	23	4810	4850	4780	7.36	0.320
	24	4900	4980	4810	7.23	0.301

Table 15: The times obtained from running the Game of Life with Wildfire (N=4896).

Type	Processors	Time	Max	Min	Speedup	Efficiency
Mixed Mode	1	3220	3220	3220	1	1
10000 iterations	3	1110	1110	1110	2.88	0.961
	6	597	597	596	5.39	0.899
	9	418	418	418	7.70	0.855
	12	327	327	327.6	9.83	0.819
	15	274	275	274	11.7	0.781
	18	239	239	239	13.4	0.747
	21	216	216	215	14.9	0.710
	24	199	199	199	16.1	0.674

Table 16: The times obtained from running the Game of Life with Wildfire (N=4896).

The $N = 4896$ data from the OpenMP and one dimensional MPI runs of the code are shown in Table 15 with the mixed mode runs shown in Table 16. The speedup plotted on Figure 24.

There is another improvement in the scaling of the MPI code. Now a small dip can be witnessed in jumping from seven to eight processors, giving evidence of an O/S overhead.

There is a distinctive difference in the speedup pattern between the OpenMP runs. The 18-8-8 run scales better until seventeen processors, which is the expected result since until this point only one box is being used. At eighteen processors the O/S overhead causes a sharp decrease in performance. This effect is witnessed to a lesser degree on the 8-8-18 results at eight processors.

The mixed mode results appear to be the most Amdahl in nature. The O/S overhead is never apparent in these results because at no point are the boxes solely filled. The scaling seems to be generally as good as the OpenMP runs and there is no evidence of Wildfire latency.

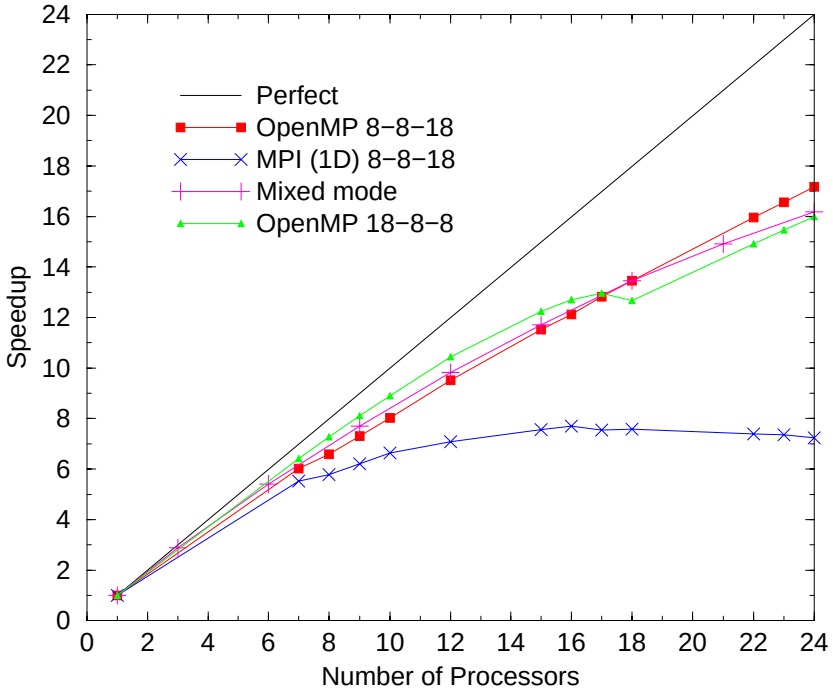


Figure 24: A graph illustrating the speedup the OpenMP and MPI versions of the Game of Life on the Wildfire (N=4896).

4 Conclusion

The O/S overhead is clearly apparent when comparing the performance of the HPC 3500 and HPC 6500. However, this appears to only be true when there is significant memory access involved. From quantitative analysis the value of the O/S overhead appears to be less than 6% of the available CPU time, and it is the opinion of the author that this does not warrant sole processor allocation.

Using OpenMP on large shared memory machines can give rise to large performance benefits due to caching issues which are replicated in MPI. Caching advantages and disadvantages should be taken into account when deciding which parallelisation strategy to use.

The Wildfire machine appears, performance wise, to be almost indistinguishable from an equivalent single SMP machine. There is no evidence to suggest that Wildfire gives rise to a significant latency, but the O/S overhead can be observed when all boxes are not in use.

References

[1] Dr Mark Bull. High resolution timing library. EPCC Internal.

[2] Dr Mark Bull. Openmp binding library. EPCC Internal.

[3] Erik Hagersaten and Michael Koster. Wildfire: A scalable path for smps. Technical report, Sun Microsystems, Inc.

[4] Dr Harvey Richardson. Mpi binding library. EPCC Internal.

[5] Dr Lorna Smith. Mixed mode mpi/ openmp programming. Technical report, EPCC.

A Perl Script

```
#!usr/bin/local/perl -w

$maxproc = 1;
while (defined($filename = glob("*.log")) ) {
    open (DATA, $filename ) ||
die "can't open any log files: $!";
    $flag = 0;
    until ( !defined ($string = <DATA>) ){
        chomp ($string);
        if ($string =~ /^ Number\b/ && $flag eq 0) {
            $nproc = substr($string,22,3);
            if ($nproc > $maxproc) {
                $maxproc = $nproc;
            }
            $flag = 1;
        }elsif($string=~/^  2  loop\b/&&$flag eq 1){
            $temp = substr($string,42,8);
            $flag = 2;
        }
        if ($flag eq 2) {
            if (!defined ($loop[$nproc]) ){
                $loop[$nproc] = $temp;
                $count[$nproc] = 1;
                $max[$nproc] = $temp;
                $min[$nproc] = $temp;
            } else {
                $loop[$nproc] = ($loop[$nproc]*
$count[$nproc] + $temp) / ($count[$nproc] + 1);
                $count[$nproc]++;
                if ($temp > $max[$nproc]) {
                    $max[$nproc] = $temp;
                }
                if ($temp < $min[$nproc]) {
                    $min[$nproc] = $temp;
                }
            }
            $flag = 0;
        }
    }
}
if ( !defined ($loop[1])) {
    die "No single processor data\n";
```

```
}
for ($nproc = 1; $nproc <= $maxproc; $nproc++) {
    if ( defined ($loop[$nproc]) ) {
        $snproc[$nproc] = $loop[1]/$loop[$nproc];
        $eff[$nproc] = $snproc[$nproc]/$nproc;
        write;
    }
}

format STDOUT_TOP =

Processors Time   Maximum Minimum Count Speedup Efficiency
========== =====  ======= ======= ===== ======= ==========
.

format STDOUT =
@<<<<<<<<< @<<<<< @<<<<<  @<<<<<  @<<<< @<<<<<  @<<<<<
$nproc,$loop[$nproc],$max[$nproc],$min[$nproc],
$count[$nproc],$snproc[$nproc],$eff[$nproc]
.
```

B Peak Performance Code

B.1 OpenMP

```
! Simple programme - calculates the sum squared
! of the numbers of up to n (OpenMP version).
! Michael Clark - 4 August 2000.

program max_performance

  implicit none
  integer i,n,size
  parameter (n=100000000)
  integer omp_get_num_threads
  real sum

! Initialise the timer
  call hrtname(1, 'whole code')
  call hrtname(2, 'loop')
! Begin the programme timer
  call hrton(1)

!$omp parallel
  size = omp_get_num_threads()
!$omp end parallel

  write(6,*) 'Number of processors', size

  call hrton(2)

!$omp parallel do private(i) reduction(+:sum) shared(n)
  do i=1,n
     sum = sum + i * i
  end do
!$omp end parallel do

! End the loop timer
  call hrtoff(2)

  write(6,*) 'Sum is', sum

! End the programme timer
  call hrtoff(1)
```

```
  call hrtprint()

end program max_performance
```

B.2 MPI

```
! Simple programme - calculates the sum squared
! of the numbers of up to n (MPI version).
! Michael Clark - 4 August 2000.

program max_performance

  implicit none
  include 'mpif.h'
  integer i,j,n,size,rank,ierror
  parameter (n=100000000)
  real temp,sum
  integer, pointer, dimension(:,:) :: interval
  integer, pointer, dimension(:) :: sub_n

! Initialise the timer
  call hrtname(1, 'whole code')
  call hrtname(2, 'loop')
! Begin the programme timer
  call hrton(1)

  call mpi_init(ierror)
  call mpi_comm_size(mpi_comm_world,size,ierror)
  call mpi_comm_rank(mpi_comm_world,rank,ierror)

  allocate(interval(2,0:size-1), sub_n(0:size-1))

! Split n calculations between the processors
  sub_n(:) = n / size

! Check for remainder
  j = n - size * sub_n(0)

! Divide remainder up between the processors
  if (j .ne. 0) then
     do i = 1, size-1
        sub_n(i) = sub_n(i) + 1
        j = j - 1
        if(j .eq. 0) exit
     end do
  end if

! Assign intervals for each processor
```

```
  j = 1
  do i = 0, size-1
     interval(1,i) = j
     interval(2,i) = j + (sub_n(i) - 1)
     j = j + sub_n(i)
  end do

  if (rank.eq.0) then
     write(*,*) 'Number of processors', size
  end if

! Begin the loop timer.
  call hrton(2)

  temp=0.0
! Each processor calculate the sum squared
  do i=interval(1,rank),interval(2,rank)
     temp = temp + (i * i)
  end do

! Reduce to a single value
  call mpi_reduce(temp,sum,1,mpi_real,mpi_sum,0,
     & mpi_comm_world,ierror)

  if (rank.eq.0) then
     write(*,*) 'Sum is', sum
  end if

! End the loop timer
  call hrtoff(2)

  call mpi_finalize(ierror)

! End the programme timer
  call hrtoff(1)
  call hrtprint()

end program max_performance
```

Michael is currently studying Computational Physics (MPhys.) at the University of Edinburgh.

Supervisors on this project were:
Dr Alan Simpson, Applications Group Manager
Dr Lorna Smith, Applications Consultant