



AMBER Performance on HPCx

Lorna Smith, Chris Johnson, Arthur Trew

EPCC, The University of Edinburgh, King's Buildings, Mayfield Road, Edinburgh, UK

In this document we examine the performance of AMBER on HPCx. We investigate the percentage of absolute efficiency obtained by the code, before looking at the speed-up obtained on multiple processors. We profile the code and consider the collective communications involved in the code. Finally we consider PMEMD, a separate package that has been developed from AMBER and offers substantial performance improvement.

The code

AMBER (Assisted Model Building Energy Refinement) is a molecular dynamics application designed specifically for biomolecular systems [1]. The code has been developed by a collaboration between the Scripps Research Institute, the University of Utah, NIEHS, Penn State and SUNY-Stony Brook. AMBER is actually a suite of codes, and in this study we focus on Sander, the main parallel molecular dynamics program of the suite. This code is written in Fortran and parallelised using a master / slave, replicated data model. Currently we have AMBER7 installed on HPCx.

The HPCx System

HPCx consists of 40 IBM pSeries 690 Regatta nodes, each containing 32 POWER4 processors (a total of 1280 processors). The peak computational power of the HPCx system is 6.6 TeraFlop/s, or up to 3.5 TeraFlop/s sustained.

Each Regatta node consists of 32 1.3 GHz POWER4 processors. In the POWER4 architecture, a chip contains two processors, together with the Level 1 (L1) and Level 2 (L2) cache. Each processor has its own L1 instruction cache of 64 kB and L1 data cache of 32 kB integrated onto the chip. Also on-board the chip is the L2 cache (instructions and data) of

1.44 MB, which is shared between the two processors. Four chips (8 processors) are integrated into a multi-chip module (MCM). Four MCMs (32 processors) comprise one frame. Each MCM is configured with 128 MB of Level 3 (L3) cache and 8 GB of main memory, shared between the eight processors of the MCM.

The nodes in the HPCx system are connected via IBM's SP Switch2 (Colony) network. In order to make optimal use of the connections from the frames to the switch, each frame is logically partitioned into four 8-way nodes or LPARs. Each LPAR runs its own copy of the AIX operating system. Even when two LPARs in the same frame communicate, they must use the Switch network, just like any other pair of LPARs. From the programmer's point of view the system appears as 160 nodes of 8 processors each.

The POWER4 is a 64-bit RISC processor implementing the PowerPC instruction set architecture. It has a 1.3 GHz clock rate, and has a 5-way superscalar architecture with a 20 cycle pipeline. There are two floating point multiply-add units each of which can deliver one result per clock cycle, giving a theoretical peak performance of 5.2 Gflop/s. There is one divide and one square root unit, which are not pipelined.

Methodology

In order to investigate the performance of AMBER we have made extensive use of three tools: hpmcount, xprofiler and mpitrace.

Hpmcount is an IBM tool that counts hardware events, such as instructions issued, while the program is being executed. By the selection of different event sets, it offers different views of the behaviour of the program. The reported number of floating point operations per second gives us a Gflop/s rate from which we can derive the percentage of peak performance. Other important metrics are the FMA (fused multiply-add) percentage and the computational intensity [5,6].

Xprofiler is a simple profiler that profiles both sequential and parallel code at the source level. With a GUI type interface xprofiler provides a profile of CPU utilisation and execution time. Xprofiler does not report I/O or communication time [7].

MPITrace is an IBM tool that provides details of the amount of time spent in communication routines, including information on the number of times a particular routine is called, and the average message size for that routine [8].

Profiling of Sander

For this investigation we have used a benchmark involving factor IX protein with Ca⁺⁺ ions. This system has 90906 atoms. Table 1 shows the most computationally intense routines of Sander 7 on a relatively low number of processors (8 processors). These results are obtained from the Sander 7 output file. These results indicate that the majority of the time is spent calculating the forces between atoms, which is as we would expect for a molecular dynamics code. In particular, the majority of the time is spent in the Ewald routines. Using xprofiler, we can determine the actual name of the subroutine carrying out the majority of the work. This is the subroutine short_ene, which accounts for around 80% of the work for this particular run. Xprofiler does not give you any indication of the time spent carrying out I/O or communication, and we will consider parallel speedup in the next section. As mentioned above, results presented here are for Sander 7. At this point it is interesting to note that this routine has changed substantially from Sander 6 to Sander 7. One of the key changes was the introduction of the IBM MASS (Mathematical Acceleration SubSystem) libraries. These IBM libraries consist of an accelerated set of frequently used intrinsic mathematical functions (the scalar library) and a set of vector functions (the vector library).

Routine Name	CPU time (s)
Total Execution time	261
Force time	248
Nonbond force	237
Ewald time	211
Direct Ewald time	114
Recip Ewald time	86

Table 1: CPU times for the most computationally intense routines of Sander, produced using Xprofiler. This is for factor IX protein with Ca⁺⁺ ions, with 90906 atoms, for 500 steps. Note that the Nonbond force execution time is part of the overall force time, Ewald time is part of the Nonbonded force execution time and Direct and Recip Ewald time are part of the overall Ewald time.

The performance increase achieved from using the IBM MASS libraries within Sander 6 was discussed in a presentation by Carlos P. Soza (IBM) at a recent IBM Biological workshop [9]. He observed a 15% increase in performance for a single processor run on an IBM POWER system when vector MASS libraries were added. Our studies broadly agree with this finding,

with Sander 7 (including the MASS libraries) showing similar increases in performance over Sander 6.

Efficiency

In this section we consider the efficiency of Sander 7, i.e. the fraction of the peak performance that is achieved on a processor. The POWER4 processor has a clock rate of 1.3 GHz and its arithmetic units, if continually supplied with data, are capable of generating one result per clock cycle, i.e. 1.3 Gflop/s. The addition and multiplication units can be overlapped and the processor has two of each unit, giving a peak performance of four times this, or 5.2 Gflop/s. This peak rate is a theoretical maximum and it is often important to look at what fraction of the peak performance can actually be achieved.

For this investigation we used a set of input files available from the Scripps Research Institute. The files describe a protein (dihydrofolate reductase, dhfr) in an explicit water bath with cubic periodic boundary conditions. This system has 23,558 atoms [2].

Figure 1 shows the percentage of peak for Sander for this benchmark. On a single processor we achieve a percentage peak performance of over 11%. While this percentage may be low when compared to those observed on other platforms, this is a reasonable percentage when compared to other applications on HPCx [3]. However, the percentage of peak drops as we leave an LPAR, and continues to drop as we increase the number of processors.

The percentage of peak efficiency obtainable is limited by the number of FMA (fused multiply-add) instructions that can be carried out (i.e. the number of additions (and subtractions) that can be overlapped with multiplications). For Sander, the FMA percentage is around 58%, partly explaining the 11% percentage efficiency.

Also of importance is the computational intensity (the ratio of floating point instructions to load/store instructions). For Sander this is only 2.0, a relatively low number. This low number is perhaps unsurprising, as the nature of the molecular dynamics algorithms often involves many read and writes to temporary variables, resulting in a large number of load and stores.

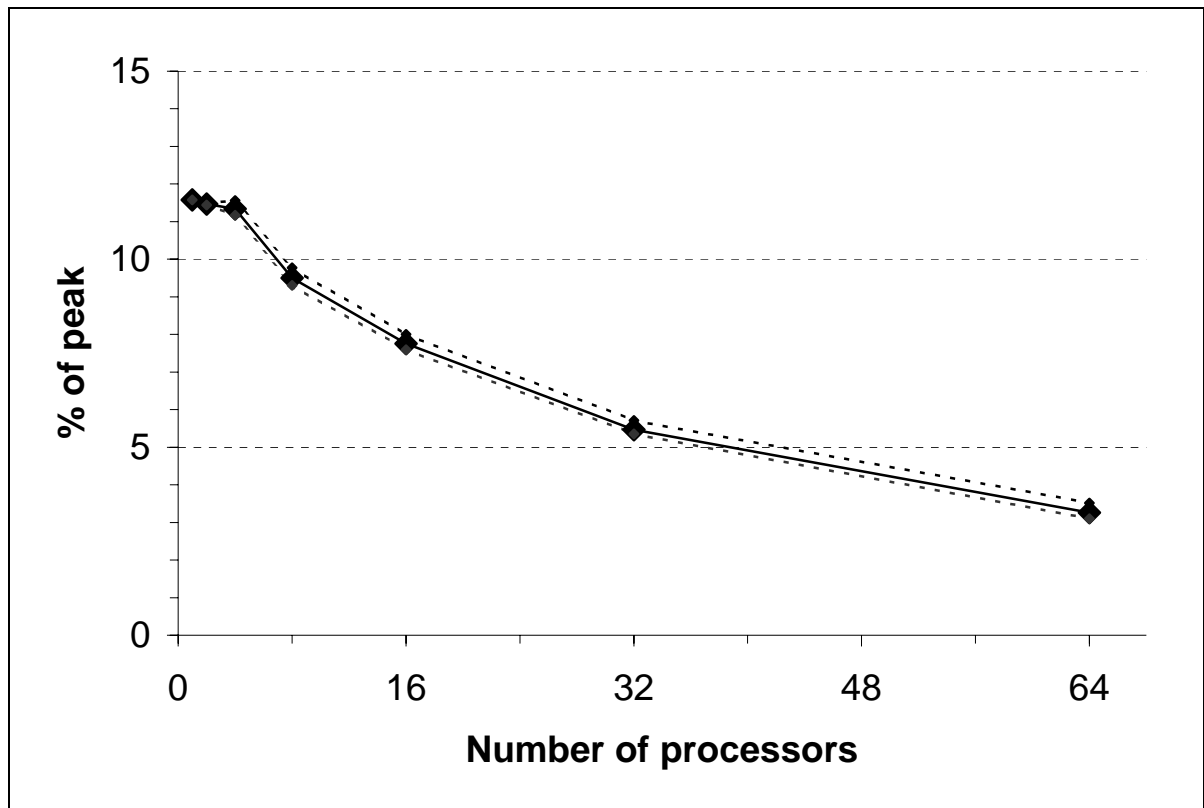


Figure 1: Performance of the Sander code showing percentage of peak (measured Gflop/s relative to 5.2 Gflop/s peak). The central solid line is the average across all processors and the two dashed lines show minimum and maximum. This is for a protein dhfr with 23,558 atoms, for 1000 steps. These results are based on the total execution time (including initialisation time).

Parallel Scaling

In this section we consider the parallel scaling of Sander. For this investigation we have used a benchmark involving factor IX protein with Ca^{++} ions. This system has 90906 atoms and was run for 500 steps.

Figure 2 shows the speed-up obtained for Sander on up to 128 processors. We achieve relatively good scaling within an LPAR, with a speed-up of around 6 on 8 processors. However, once we leave an LPAR the scaling reduces, achieving a speed-up of only around 11 on 32 processors. For more than 64 processors the execution time increases with increasing numbers of processors.

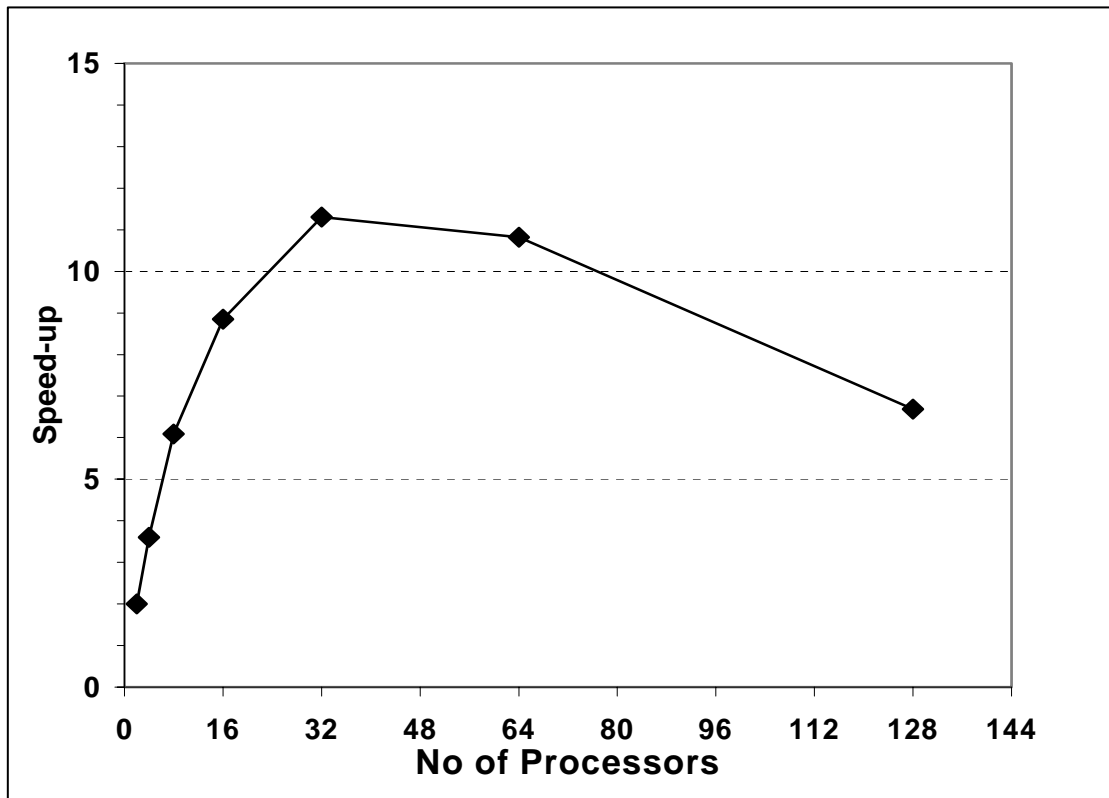


Figure 2: Speed-up of the Sander code on up to 128 processors for factor IX protein with Ca⁺⁺ ions, with 90906 atoms, for 500 steps. These results are based on the total execution time (including initialisation time).

To understand the reason for the relatively poor scaling of Sander, we have looked more closely at the communications involved in this code.

Parallel Profiling of Sander

As mentioned previously, xprofiler does not provide details of the time spent in communication routines. This has been measured using mpitrace, the results for this are shown in Table 2. As the number of processors increases, the amount of time spent in communication increases substantially. For example the amount of time spent in communication is 12% on 8 processors and 44% on 32 processors. It is clear from these results that the relatively poor scaling of Sander is partly due to this substantial increase in communication time.

It is interesting to note the variation in timings. Most of this variation in timing is due to time spent in BARRIER operations, and simply highlights load imbalance issues.

MPITrace provides more detailed information about the type of MPI routines being utilized and the amount of time spent within these. The results of these are shown in Table 3 for 128 processors.

Number of Processors	Percentage of total execution time (s) spent in communications		
	max	min	average
1	0.1	0.1	0.1
2	11	4	8
4	18	5	13
8	14	7	12
16	29	26	29
32	48	40	44
64	54	47	52
128	87	79	84

Table 2: Percentage of total execution time spent in communication for varying number of processors for Sander. This is for factor IX protein with Ca++ ions, with 90906 atoms, for 500 steps.

Hence the majority of this time is spent in an MPI_REDUCE_SCATTER operation. Sander has the option to use an MPI_REDUCE_SCATTER operation or to use a Sander specific routine, which uses a tree-like algorithm and MPI_SendRecv. As such a large quantity of time is spent carrying out this reduction / scatter operation, we investigated the most efficient of these options. The results are shown in Table 4.

MPI Routine	No of Calls	Av Bytes	Time
MPI_Comm_size	2691	0.0	0.0
MPI_Comm_rank	2691	0.0	0.0
MPI_Send	1	568	0.0
MPI_Isend	63000	2312	0.3
MPI_Recv	63000	2312	13.1
MPI_Wait	63000	0.0	2.2
MPI_Bcast	54	1322538	4.4
MPI_Barrier	6098	0.0	11.1
MPI_Allgatherv	502	34128	8.8
MPI_Allreduce	2001	312	4.3
MPI_Reduce_scatter	500	2181744	55.5

Table 3: Total time spent in communication for 64 processors for Sander. The total execution time for this run was 153 seconds. This is for factor IX protein with Ca++ ions, with 90906 atoms, for 500 steps.

Number of Processors	Execution Time
MPI_REDUCE_SCATTER	158 seconds
Sander specific routine	147 seconds

Table 4: Execution time spent in communication for various options of Sander on 64 processors. This is for factor IX protein with Ca++ ions, with 90906 atoms, for 500 steps.

It is clear from this table that the Sander specific routine is the most efficient option. One possible explanation for this is the performance of the IBM library on collective communications. Other studies have suggested that this is less than optimal for this clustered shared memory system. By introducing a two level system, using multiple

communicators, we obtain substantially better performance, for example on the CASTEP code.

Populated Logical Partitions

As mentioned previously, HPCx is divided into logical partitions, each logical partition containing 8 processors. Some codes exhibit better performance when run with fewer than the full 8 processors per logical partition [10]. For example, we may see a difference in execution time for a code run on 128 processors within 16 logical partitions (8 processors per partition) and the same code run on 128 processors within 32 logical partitions (4 processors per partition).

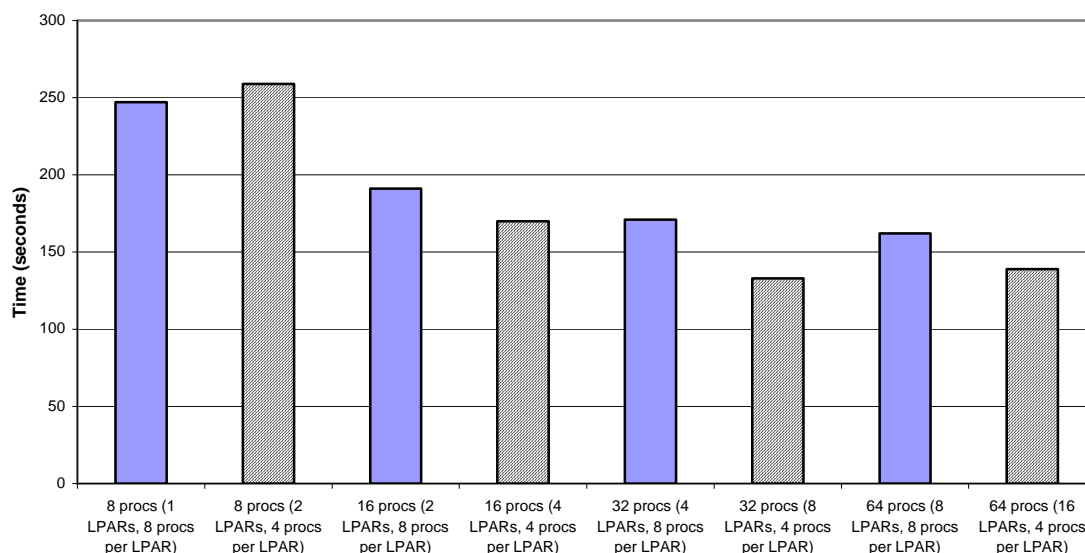


Figure 3: Execution time for Sander on various processor combinations and logical partitions (LPARs). This is for factor IX protein with Ca⁺⁺ ions, with 90906 atoms, for 500 steps. These results are based on the total execution time (including initialisation time).

Sander is one such code. Figure 3 shows the execution time for Sander on a range of processors and logical partitions. It is clear from the diagram that in most cases, we see a significant reduction in execution time when we do not use all 8 processors within a logical partition. Within a logical partition, level 3 cache and main memory are shared between the processors, hence the fewer processors utilized within a logical partition, the less competition there is for level 3 cache and memory. For memory bound codes this can make a substantial difference. However, on HPCx users are charged for the use of an entire logical partition, no matter how many processors are used. Hence it may be cheaper to run

the code for longer on fewer logical partitions.

The exception is on 8 processors, where running across 2 logical partitions (4 processors per partition) takes longer than running on 8 processors within 1 logical partition. This is because running across 2 logical partitions forces the code to use the switch, whereas running within a single logical partition keeps everything within a shared memory system.

PMEMD

PMEMD (Particle Mesh Ewald for Molecular Dynamics) is a piece of code based on Sander developed by Robert Duke [4], using HPCx. This code involves a substantial rewrite from Sander, such as incorporating Fortran 90 functionality to get better data structures, dynamic memory allocation, subroutine call interface checking and so on. It also involves a rewrite of the direct force evaluation code, creating a version that uses much less memory, and furthermore, that had a significantly better cache hit ratio due to better "image" locality. The performance of this code and Sander 7 on up to 256 processors is shown in Figure 3. As this figure shows, substantial performance improvement is achieved using PMEMD. For example, on 128 processors, PMEMD takes 40.8 seconds and Sander7 156.5 seconds, representing a 74% improvement in performance. PMEMD is available on HPCx, please see:

<http://www.hpcx.ac.uk/research/chemistry/amber.html>

for further details.

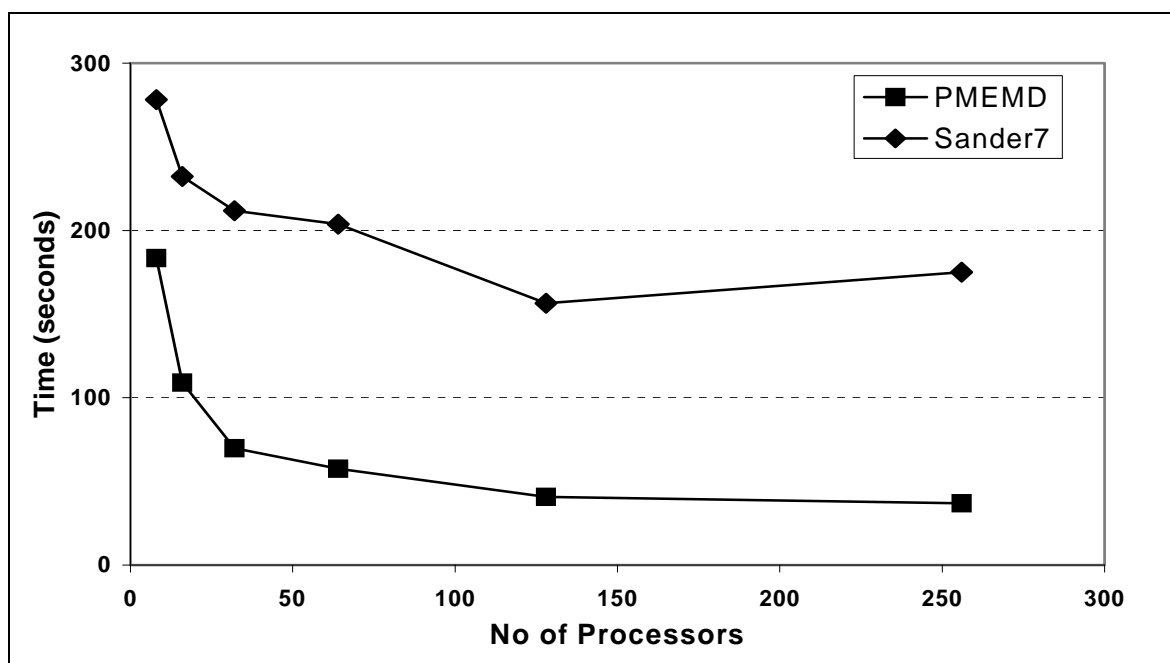


Figure 3: Execution time for Sander 7 and PMEMD on 256 processors. This is for factor IX protein with Ca⁺⁺ ions, with 90906 atoms, for 500 steps. These results are based on the total execution time (including initialisation time).

Conclusions

This document examines the performance of the molecular dynamics code AMBER, or more specifically Sander, on HPCx. This study only investigated two benchmark systems, and it is important to note that the performance observed on this code can change substantially when studying different systems.

Future work will involve further investigation of the performance of PMEMD and AMBER on the phase 2 system of HPCx. The Phase 2 system represents a substantial upgrade, with an increase in processor speed, a change in switch technology, and in the MPI software implementation, so it is important to repeat these studies under this new environment.

References

- [1] UCSF Amber home page. See: <http://www.scripps.edu/>
- [2] Joint Amber Charm (JAC) benchmark, See: www.scripps.edu/brooks/Benchmarks
- [3] M. Ashworth, I. Bush, M.F. Guest, M. Plummer, A.G. Sunderland, H.J.J. Van Dam, J. Hein, L. Smith, Application Performance on HPCx: The Impact of Serial Efficiency, Internal HPCx report, this will release as an external technical report in 2004.
- [4] PMEMD was developed by Dr. Robert Duke in Prof. Lee Pedersen's Lab at UNC-Chapel Hill, starting from the version of Sander in AMBER 6. See: <http://amber.scripps.edu/pmemd-get.html>
- [5] J. Hein, Using the Hardware Performance Monitor Toolkit on HPCx, HPCx Technical Report HPCxTR0307. See: http://www.hpcx.ac.uk/research/hpc/technical_reports/HPCxTR0307/index.html
- [6] L. DeRose, Hardware Performance Monitor (HPM) Toolkit. See: <http://www.hpcx.ac.uk/support/documentation/IBMdocuments/HPM.html>
- [7] IBM Parallel Environment for AIX. See: http://www-1.ibm.com/servers/eserver/pseries/library/sp_books/pe.html
- [8] MPITrace documentation. See: <http://www.hpcx.ac.uk/support/documentation/IBMdocuments/mpitrace>
- [9] IBM Workshop on High Performance Computing in Chemistry and Life Sciences, Max

Planck Institute of Plasma Physics in Garching, Germany, October 2002.

[10] J. Hein and J.M. Bull, Capability Computing, Achieving Scalability on over 1000 Processors, HPCx Technical Report HPCxTR0301, See:

http://www.hpcx.ac.uk/research/hpc/technical_reports/HPCxTR0301.pdf