

HPCx: TOWARDS CAPABILITY COMPUTING

Submitted to *Concurrency and Computation: Practice and Experience*

Mike Ashworth, Ian J. Bush, Martyn F. Guest and Andrew G. Sunderland
*Computational Science and Engineering Department, CLRC Daresbury Laboratory, Daresbury,
Warrington, Cheshire, WA4 4AD, UK*
Email: m.ashworth@dl.ac.uk, m.f.guest@dl.ac.uk

Stephen Booth, Joachim Hein, Lorna Smith and Kevin Stratford
*Edinburgh Parallel Computing Centre, University of Edinburgh, JCMB, The King's Buildings,
Mayfield Road, Edinburgh, EH9 3JZ, UK*

Alessandro Curioni
IBM Research, Zurich Research Laboratory, 8803 Rueschlikon, Switzerland

Abstract

We introduce HPCx - the UK's new National HPC Service - which aims to deliver a world-class service for capability computing to the UK scientific community. HPCx is targeting an environment that will both result in world-leading science and address the challenges involved in scaling existing codes to the capability levels required. Close working relationships with scientific consortia and user groups throughout the research process will be a central feature of the service. A significant number of key user applications have already been ported to the system. We present initial benchmark results from this process and discuss the optimisation of the codes and the performance levels achieved on HPCx in comparison with other systems. We find a range of performance with some algorithms scaling far better than others.

1. INTRODUCTION

HPCx is the name of the UK's new National High Performance Computing Service. It is a large IBM p690 cluster whose configuration is specifically designed for high-availability capability computing. We define *capability computing* as running jobs which use a significant fraction of the total resource in contrast with *capacity computing*, which is the running of a large number of smaller jobs. The Engineering and Physical Sciences Research Council (EPSRC) is overseeing the project, on behalf of the UK scientific community. HPCx is a joint venture between the Daresbury Laboratory of the Central Laboratory for the Research Councils (CLRC) and Edinburgh Parallel Computing Centre (EPCC) at the University of Edinburgh. IBM (UK) Ltd has been chosen as the hardware supplier for the six-year duration of the project.

Perhaps the key challenge for the HPCx service is to deliver on the capability aspirations of the UK community across a broad spectrum of scientific and engineering disciplines. In this article we describe the initial progress towards this goal. Following an overview of the HPCx Project in section 2, we provide an overview of the challenges involved in capability-based computing (section 3) and specifically the challenges involved in communications and in managing the memory hierarchy (section 4). Sections 5 to 10 outline initial progress towards this goal by illustrating the current levels of delivered performance from nine well-known codes from six different applications areas, chosen to be representative of the workload on the HPCx system and to illustrate the challenges of achieving Terascale performance from a range of algorithms.

2. THE HPCX PROJECT

The British Government, through EPSRC, has funded the project to approximately £53M (US\$ 85M). The scope of the project is to operate and support the principal academic and research high performance computing service for the UK. The principal objective is to provide a capability computing service to run key scientific applications that can only be run on the very highest performing computing platforms.

HPCx is a collaborative project between three partners: CLRC Daresbury Laboratory, Edinburgh Parallel Computing Centre (EPCC) and IBM. Within HPCx, CLRC and EPCC both provide "Added Value" services (primarily scientific support, code optimisation, and code development) and operations and system support services. The physical accommodation is provided at CLRC's Daresbury Laboratory. IBM is the technology partner, providing the hardware, system software and appropriate maintenance and support services. They offer world-leading and competitively priced HPC technology, an aggressive road map over the next 6 years and significant resources for science support.

HPCx is a six-year project and the technology will be provided in three phases, with defined performance levels at year 0, year 2 and year 4. The performance targets for the three phases are defined in terms of Linpack Rmax performance. The targets for the three phases are approximately 3 Tflop/s, 6 Tflop/s and 12 Tflop/s, roughly following Moore's Law. Online and offline storage will increase proportionately, with 50, 100 and 200 TB in the three phases. There is a flexible approach to the technology refreshes in phases 2 and 3, and future upgrades are likely to feature the "Federation" switch and POWER5-based architectures.

The issue of a well-defined road map proved decisive in the choice of technology, and in addressing the relative merits of commodity-based and proprietary-based solutions. While serious consideration was given to the former, the underlying requirements of the procurement - demands for the delivery of technology against a contractually-binding 6-year roadmap, together with an exceedingly high and challenging level of full-system RAS features (Reliability, Availability and Serviceability) - argued strongly for a proprietary solution. Furthermore the procurement demanded features that were not then available from the commodity arena - a robust parallel file system supporting 1000s of processors and commitments to check-point restarting - features that are only just achieving the desired level of robustness. Within the proprietary space, the offerings from HP/Compaq following the demise of the Alpha processor coupled with the on-going debate over the future of the joint company and its associated range of products led to problems over commitment to a credible roadmap. SGI and Cray did not bid for the contract as their new offerings, respectively the SGI Altix Itanium product and the Cray SV-2, would not have been available on the timescales of the project. This left IBM as the preferred technology provider given its road map over the next 6 years, and its ability to respond to the contractually bound features characterising the service.

The focus is on maximising the delivery of capability computing; the service is not intended to be used as a task farm or for multiple modest-sized throughput jobs. There is a comprehensive support service providing porting, optimisation, training and new applications outreach. The service has a 24 x 7 operating regime with high RAS requirements.

2.1. The HPCx phase 1 system

Delivery of the Phase 1 system commenced on 4th October 2002. The full user service opened officially on 9th December 2002, although many users had benefited from up to a month's early access prior to this date. The system comprises 40 "Regatta" pSeries 690 (p690) SMP compute nodes connected by the "Colony" SP Switch2.

Each shared memory node has 32 1.3GHz POWER4 processors and 32 GB memory giving the system an aggregate memory of 1.28 TB. The POWER4 processor has dual floating point units each of which, through the fused multiply-add instruction, is capable of delivering two results per clock cycle. This gives each processor a peak performance of 5.2 Gflop/s and the whole system a peak of some 6.6 Tflop/s.

In order to increase connectivity to the switch and improve communications throughput, each compute node is configured as four 8-way logical partitions (LPARs). The "Colony" SP Switch2 allows each LPAR to have two connections (PCI adapters) into the switch fabric (dual plane), providing approximately 20 μ sec latency and 350 MBytes/sec bandwidth. Two additional 16 processor nodes are provided as I/O systems. The system runs AIX version 5, with GPFS for file system support (18TB EXP500) and HSM for backup and archive to tape storage (35TB LTO tape library).

3. TOWARDS CAPABILITY COMPUTING

Perhaps the key challenge for the HPCx service is to deliver on the capability aspirations of the UK community across a broad spectrum of disciplines. Following a summary in sections 3 and 4 of the key challenges to be addressed, we outline in sections 5-10 initial progress towards our goal by illustrating the current levels of delivered performance from a number of codes across a variety of disciplines, including materials science, molecular simulation, atomic and molecular physics, molecular electronic structure, computational engineering and environmental science.

Considering a total of nine application codes, we compare performance on the IBM p690 cluster with that found on a number of platforms from other vendors. These include the Cray T3E/1200E ("Turing"), the SGI Origin 3800/R12k-400 system ("Green") and the SGI Altix 3700/1.3GHz system ("Newton"), all operated by CSAR¹ at the University of Manchester, UK, and the SGI O3800/R14k-500 system ("Teras") at SARA², Amsterdam. Also included is the Compaq AlphaServer ES45/1000, the TCS-1 system at Pittsburgh Supercomputing Centre (PSC)³. Readers should be aware that these systems cover a range of technologies with some being obviously more recent than others.

3.1. The Challenges of Capability Computing

During the next few decades, advances in computing technologies will increase the speed and capacity of computers, storage, and networks by several orders of magnitude. At the same time, advances in theoretical, mathematical, and computational science will result in computational models of ever increasing predictive capability and utility. The key goal for computational scientists and engineers is then to harness the power offered by present and future high-performance computers to solve the most critical problems in science and engineering. Such a goal demands a capability-driven approach, an approach in which the full power of a Terascale computer is brought to bear on a given scientific problem through effective utilisation of all available resources - CPUs, memory, interconnect, and in many cases high levels of I/O performance. Capability Computing contrasts strongly with the alternative capacity-based approach where many more modest problems are tackled, often simultaneously, on a machine, each with less demanding requirements. The primary mission of HPCx is that of Capability Computing, an approach reflected by our drive to ensure that the majority of jobs on the IBM p690 cluster are capable of utilising at least a significant fraction of the available resource.

Before considering our progress to date, we provide a brief overview of the challenges that must be addressed if we are fully to realise the benefits offered by high-performance computing [2]. The current generation of parallel supercomputers are, of course, built from thousands of processors using commodity memories (DRAM, the same memories used in personal computers) interconnected by a communications fabric that, although fast by networking standards, is still far slower than access to local memory. To make full use of such machines, computational scientists and engineers must address the three major challenges outlined below:

a. Management of the memory hierarchy: A key characteristic of today's parallel machines is the presence of a deep memory hierarchy. Each processor in a parallel supercomputer has a memory hierarchy - registers, on- and/or off-chip caches, main memory, and virtual memory (disk) - with each level requiring successively more time to access (latency) and having slower transfer rates (bandwidth). A parallel computer adds an additional level to this hierarchy—remote memory. In exploiting the capabilities of today's parallel machines, it is critical carefully to manage this memory hierarchy. A programming model based on the concept of non-uniform memory access (NUMA) explicitly recognises this hierarchy, providing numerous advantages to the scientific programmer. We note in passing that the Global Arrays (GA) library [3] implements a very efficient NUMA programming model.

b. Expression and management of concurrency: Most scientific algorithms are rich in parallelism, although understanding the performance of these algorithms on parallel supercomputers requires more information into

¹ Computer Services for Academic Research, University of Manchester, UK, <http://www.csar.cfs.ac.uk/>

² SARA Computing and Networking Services, Amsterdam, Netherlands, <http://www.sara.nl/>

³ Pittsburgh Supercomputing Center, Pittsburgh, PA, USA, <http://www.psc.edu/>

just how much parallelism is needed and at what level of granularity. The analysis of ref. 1 suggests the following:-

- Fine grain parallelism is essential to obtain efficient sequential execution given the mismatch in speed between that of the processor and that of the memory subsystem. Further, modern super-scalar processors achieve peak speed by executing multiple instructions per cycle, increasing the demand for fine grain parallelism.
- The granularity of coarse grain parallelism is determined by the ratio of the single processor speed to the average latency of remote memory references. This latency is controlled by the characteristics of both the communications hardware and the algorithm.

Current portable parallel environments require the user manually to express coarse grain parallelism (*e.g.*, with MPI or Global Arrays), while relying upon compilers or libraries *e.g.* the basic linear algebra subroutines (BLAS) to describe the fine grain parallelism. Significant progress has been made in addressing this area of concurrency, a key requirement in any capability-driven scenario *e.g.* the applications work at Pittsburgh Supercomputing Centre and the CRYSTAL and POLCOMS applications on HPCx (see below).

c. Efficient sequential execution: While efficiently using a large number of processors is recognised as a key requirement in any capability-driven approach, the more fundamental issue of ensuring efficient use of each of these processors is often overlooked. This is a non-trivial problem; to illustrate the magnitude of this problem, it is useful to compare the B/F ratio (the bandwidth to memory in Bytes/sec divided by the speed of the processor in operations/sec) for vector and parallel supercomputers. With a large B/F ratio, a processor can keep its functional units busy because the required data can be rapidly transferred from memory to the units. A small B/F ratio means, however, that the processor may often sit idle, as it will be waiting for data to be moved from memory to the functional units. The mathematical approaches and algorithms developed in the Cray era often made explicit use of the high memory bandwidths of Cray supercomputers - the last supercomputer built by Cray Research, Inc. (Cray T90) had a B/F ratio of 15. This was sufficient to run many vector operations at full speed out of main memory. The B/F ratio for the processor/memory subsystem in ASCI White, on the other hand, is just 0.67-1.33. This means that many of the traditional algorithms used in computational science and engineering will not perform well on parallel supercomputers without substantial tuning, revision, or even replacement.

There are numerous examples of this effect, where commendably high levels of concurrency are nevertheless accompanied by low overall delivery of peak performance. On the 3,000 processor TCS-1 system at PSC, the atmospheric general circulation model code, CCM/MP-2D, achieved 83% scaling efficiency on 2048 processors, but only 23% of peak. On the same machine, Klaus Schulten's NAMD code sustained a speedup factor of 1599 using 2250 processors on a molecular dynamics simulation of the 327,000 atom F1-ATPase system - exceptional scaling for a bio-molecular MD code - but only 18% of peak.

Designers of microprocessors and parallel supercomputers have attempted to mitigate the memory performance problem by a number of mechanisms *e.g.*, by interposing a high-speed cache between the processor and main memory (with an 8 MByte cache, the B/F increases on the processors of ACI White to 5.3). Clearly, if an algorithm can be structured so that the data are in cache when needed, the performance of the microprocessor will improve substantially. For many applications the key to successful utilization of the cache is data reuse, but this often requires a significant change in the algorithm. Other features designed to enhance memory performance include instruction and data pre-fetch, and the ability to schedule multiple outstanding memory requests. Although we expect to see substantial improvements in the B/F ratio of microprocessor-based parallel supercomputers in the next few years, there remains much to be gained by research on algorithms that promise to improve the sequential performance of parallel scientific and engineering codes.

4. CHALLENGES OF CAPABILITY COMPUTING

4.1. Management of Memory Hierarchy

The previous section discussed the challenges associated with fully exploiting the capabilities of today's parallel machines. In this section we examine these challenges in more detail, through a range of simple benchmark codes.

Where appropriate, these benchmark codes have been studied on a range of platforms. Not all of these benchmarks are relevant on multiple platforms however, as they study performance issues specific to the HPCx platform. In these situations results are only presented on HPCx.

In the following section we discuss the performance and scaling of a range of applications across a variety of platforms. Many of the issues highlighted in this section contribute to the observed performance of these applications. Hence this section also provides insight into the performance and scaling of these applications on modern HPC systems.

Before considering the benchmarks we provide details of the memory hierarchy of HPCx. Following this we look at communication issues such as collective communications and overlapping communication with computation. We then consider sequential performance and the effective management of the memory hierarchy.

4.2. HPCx memory hierarchy

HPCx consists of 40 IBM p690 nodes, each containing 32 POWER4 processors. Within a node there are 16 chips: each chip contains two processors with their own level 1 caches and a shared level 2 cache. The level 1 cache is split into a 32 KByte data cache and a 64 KByte instruction cache, has 128-byte lines, is 2-way set associative and write-through. The level 2 cache is a 1.44 MByte combined data and instruction cache, with 128 byte lines and is 8-way set associative and write-back.

The chips are packaged into a Multi-Chip Module (MCM) containing 4 chips (8 processors) and a 128 Mbyte level 3 cache, which is shared by all 8 processors in the MCM. The level 3 cache has 512 byte lines, and is 8-way set associative and write-back.

Each p690 node contains 4 MCMs and 32 GBytes of main memory. The MCMs are connected to each other and to main memory by a 4-way bus interconnect to form a 32-way symmetric multi-processor (SMP). Note that the level 3 caches lie on the memory side of the interconnect. Cache coherency is maintained on the level 2 caches via a snoopy bus protocol. This means that every processor in effect has access to all 512 MBytes of level 3 cache in the system.

In order to increase the communication bandwidth of the system each P690 node has been divided into 4 logical partitions (LPAR), coinciding with each MCM. Each LPAR runs its own copy of the AIX operating system and operates as an 8-way SMP.

4.3. Parallel Execution - Communications

In this section we look at two particular aspects of communications: collective communications and overlapping communication and computation. Both show interesting performance characteristics and both are relevant to the applications' discussed in the following sections.

a. Overlapping communication with computation

The latency and bandwidth of the Colony switch has been described in section 2. It is interesting to speculate whether overlapping computation and communication may be profitable on HPCx, and indeed on other systems with relatively long latency and low bandwidth interconnect. To investigate this, we ran a simple ping-pong code, where one process also does some computation on a work array. In one version of the code this computation was performed between the asynchronous sends and receives of the data and the wait for completion of all communications. For reference, another version had the calculation placed after all communications were complete. The time difference between these versions is shown in Figure 1 for a range of message sizes and work array sizes on the IBM p690 and the AlphaServer ES45/1000. On the left side of the figure we show the absolute time saved. The time saved increases with the size of the work array and with the size of the message. On HPCx there is also evidence of a limitation and reduction in the time saved for large work arrays, which may be due to level 2 cache conflicts between the calculation and the communication. On the right side of the figure we show the relative time saved with respect to the total execution time. For small messages (up to 16kB) there is the potential to save around 20-25% of the total execution time, but for larger messages the saving drops to 5% or less. Runs of this benchmark on the SGI Origin 3800 and SGI Altix 3700 systems showed less potential for overlapping, with maximum values for the absolute time saved of 261 μ s and 168 μ s respectively and maximum relative savings of 10% in both cases.

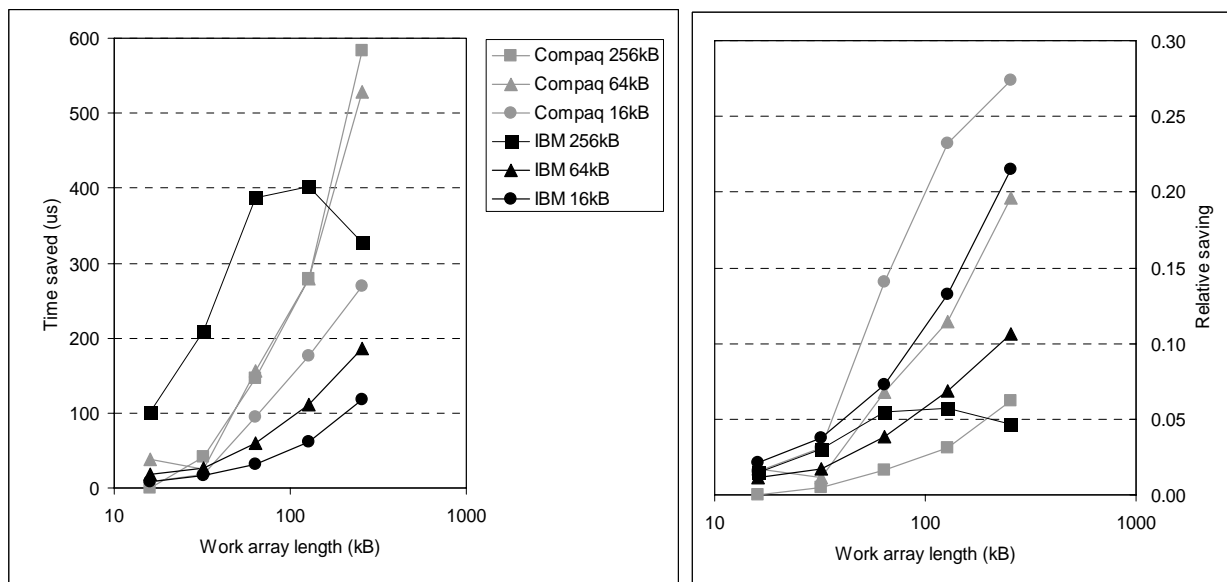


Figure 1. Results of the overlapping benchmark showing the absolute time saved in μs (left) and the time saved relative to the total execution time (right) for different message sizes as a function of the work array size. Results are shown for the IBM p690 (black) and the AlphaServer ES45/1000 (grey).

These results suggest that if an application has few large messages ($\sim 256\text{kB}$), overlapping may not be worthwhile as the saving is low at 5%. On the other hand if it has many small messages ($\sim 16\text{kB}$), and there are suitably sized units of work with which to overlap the communications, a worthwhile saving of up to 25% may be available.

b. Collective communications

Many scientific applications use collective communications and to achieve good scaling on clusters SMP systems these communications need to be implemented efficiently. Collective communications were included in the MPI standard to allow developers to implement optimised versions of essential communication patterns. A number of collective operations can be efficiently implemented using tree algorithms - including Broadcast, Gather, Scatter and Reduce. On systems where communication is faster within a node than between nodes, the tree algorithm can be constructed such that communications corresponding to branches of the tree at the same level should run at the same speed, otherwise the speed of each stage of the algorithm will be limited by the performance of the slowest communication.

Although this issue may apply generally to many clustered SMP platforms, it is particularly important on HPCx, where communications within an LPAR are significantly faster than between LPARs. Hence results are only presented for HPCx within this section.

To demonstrate this issue, a library has been developed that, by creating multiple communicators, performs collective operations in two stages. For example, an AllReduce operation involves a reduction operation carried out across the processors within an LPAR, followed by a reduction of these results across LPARs. A simple benchmark code, that executes an AllReduce operation across a range of processors, has been used to compare the performance of this reduction operation against the standard all reduce operation of the IBM MPI library. Figure 2 shows the performance (in MB/s) for each operation on a range of data sizes. It is clear from this diagram that the two-stage operation, using multiple communicators, is significantly faster than the standard MPI operation. Hence for applications that are dominated by collective communications, this library offers a simple and quick mechanism to obtain significant performance improvement. The library is simple to implement as it uses the MPI profiling interface and hence requires no code modification.

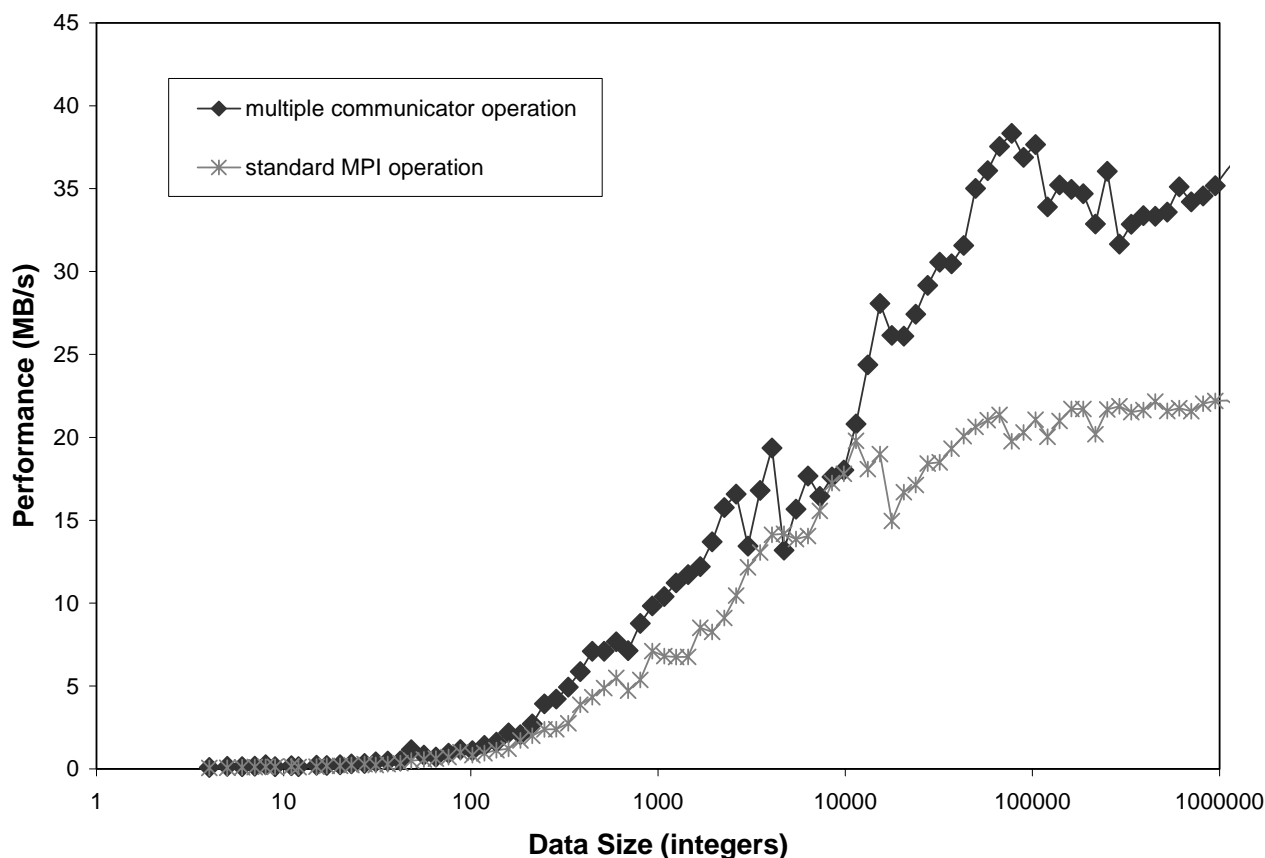


Figure 2. Performance in MB/s of the two-stage AllReduce operation involving multi-communicators and the standard all reduce operation of the IBM MPI library.

4.4. Management of Memory Hierarchy

Section 2 discussed the deep memory hierarchy of today's parallel machines, and how important it is to carefully manage this memory hierarchy, if we wish to exploit the full capabilities of these systems. In this section we justify the need for this careful management, through a range of simple benchmark codes.

a. Sequential Execution

The discussion on efficient sequential execution in section 3 concluded that the issue of ensuring efficient sequential processor execution is often overlooked, but an algorithm that is restructured so that data is in cache when needed, can show substantial performance improvement. The performance benefits achieved through this mechanism are very platform specific and may indeed offer little improvement on platforms with a relatively flat memory system. However, this issue is particularly important on HPCx, where the cache hierarchy is complex (see Section 4.2). Hence to highlight this importance, we have run a simple benchmark code on HPCx that performs a lattice Laplacian in 2 dimensions by using an iterative Jacobi Algorithm. This code is written in Fortran and has been parallelised using MPI.

The sequential performance of the code has been investigated on a range of different problem sizes, the results of which are given in Table 1. For each problem size we have specified its location in memory. The largest problem size is 1024 times larger than the smallest problem size and ideally, would be 1024 times slower. However the performance is significantly slower than this, at around 2150 times slower. This is because the smallest problem size resides in level 2 cache and the largest problem size in main memory. This effect is less pronounced between level 2 and level 3 cache, with problem size 2 being around 5 times slower than problem size 3, when it is only 4 times as large.

Array Size (integers)	Initial Code		Memory Efficient Code	
	Execution Time (s)	Memory Location	Execution Time (s)	Memory Location
210 * 252	0.60	level 2	0.52	level 2
420 * 504	2.8	level 3	2.2	level 3
840 * 1008	12.5	level 3	9.6	level 3
1680 * 2016	52	level 3	39	level 3
3360 * 4032	277	main memory	172	level 3
6720 * 8064	1299	main memory	800	main memory

Table 1. Time in wall clock seconds for the Jacobi benchmark executed on one processor for a range of problem sizes.

The computationally intense component of the code involves updating the elements of a 2D array, based on an element's current nearest neighbour values. The original algorithm writes the updated elements to a temporary array and copies the values to the final array once the updates are complete. By rewriting this routine to update elements of the final array and overwrite elements of the temporary array after each iteration, we can reduce the size of the temporary array and hence the amount of data being written to memory. Comparing the execution times of both codes (Table 1), it is clear that the more memory efficient code is faster for all problem sizes, with the biggest performance gain observed for the larger problem sizes.

While this is a simple example, this is a fairly standard mechanism for reducing the memory requirements of the code, and highlights the significant improvements in sequential performance that can be achieved through efficient sequential algorithm development.

b. Parallel execution

Parallellising an application code complicates the cache issue further. The discussion above highlights the need for efficient parallel algorithms that minimise the amount of memory accesses. For example, the scaling observed for CRYSTAL (see section 5.1) has only been achieved through rationalising the memory management in the code - dynamically allocating arrays and rewriting the algorithm to use a distributed, rather than replicated data strategy, which requires unacceptable levels of memory.

In order to study the effect of the complicated nature of the memory hierarchy on parallel applications, the performance of the simple Jacobi benchmark has been measured on varying numbers of processors and LPARs for a problem size of 1680 x 2016. The results are shown in Figure 3. Again this issue is particularly relevant to HPCx and has only been studied on this system.

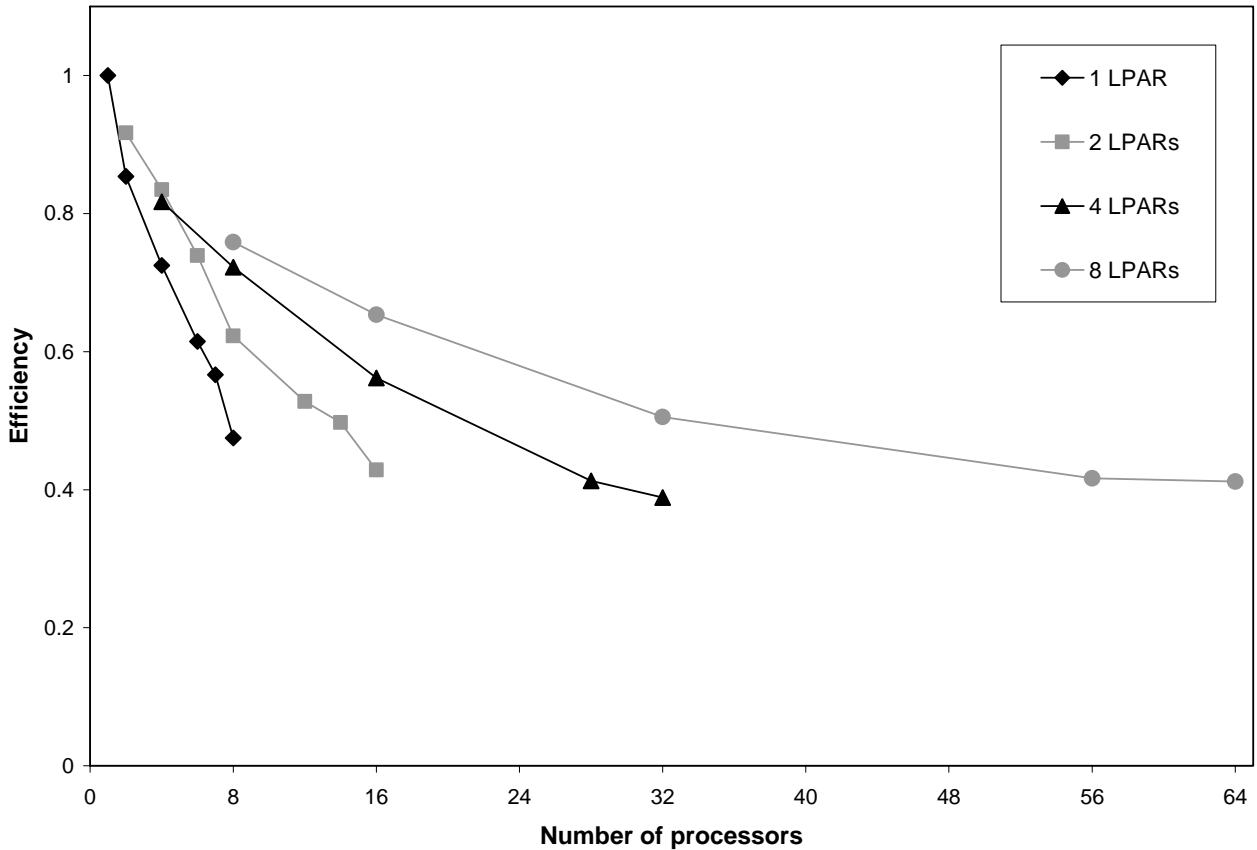


Figure 3. Parallel efficiency for the Jacobi benchmark (size 1680x2016) executed on varying numbers of processors and LPARs.

The p690 32-processor frame is designed in such a way that much of the hardware, especially the level 2 and level 3 caches and the memory hardware, is shared between more than one processor. A processor may at times, when other processors in the frame are idle or not using cache/memory, obtain a higher access rate to cache/memory than its share. This may be seen in Figure 3, where we show the parallel efficiency, defined as $t_1/(n.t_n)$ where t_n is the execution time on n processors. As the number of processors utilised on an LPAR increases, we observe a drop in efficiency to slightly less than 50 percent. The small B/F ratio of HPCx discussed in the previous section means that the processors may often sit idle, waiting for data to be moved from memory to the functional units. When a single processor has access to all the hardware it can sustain 11 GB/s to main memory, but with a larger number of processors per LPAR the memory bandwidth saturates at a lower level.

We observe a significant reduction in execution time for a fixed number of processors run using under-populated LPARs, compared to fully-populated LPARs. For example, running an 8 processor job across 8 LPARs (i.e. 1 processor per LPAR) provides a 35% reduction in execution time over an 8 processor job run within 1 LPAR. By using 8 separate LPARs, each processor is no longer sharing its memory bus, level 3 cache and level 2 cache, and can therefore access data at a higher rate. In order to utilise HPCx efficiently, a user is charged for the full use of an LPAR, even if they do not utilise all the processors. Hence utilising only 1 processor of an LPAR is likely to be costly. However, there may be a potential benefit to using 6 or 7 processors per LPAR, if the performance increase outweighs the cost of paying for a full LPAR.

The final complication associated with HPCx and memory hierarchy relates to the fact that cache coherency is carried out in hardware and cannot be disabled. As mentioned above, each p690 node has been divided into four LPARs, each with their own copy of the operating system. Cache coherency is however still active between LPARs. The effect of this is demonstrated by running four copies of our benchmark code simultaneously on a single p690 node, and by running four copies on four different nodes simultaneously. This test has been run on various data sizes and the results are given in Figure 4. The smallest problem size fits into level 2 cache and no real difference is observed in execution time between the two experiments.

However the larger problem sizes do not fit into level 2 cache, and all level 2 cache misses will result in all the LPARs answering requests for cache coherency. When running four copies of the benchmark code simultaneously on a single p690 node, the tightly synchronised nature of this benchmark results in all the LPARs throwing cache misses at a similar time, creating a large number of cache coherency requests and resulting in a performance degradation. In comparison, by running four copies of our benchmark code on four different nodes, the remaining LPARs will be executing other user codes that may not stress the memory system as much, or at the same time.

This is a common result for clusters of SMP nodes, due to contention within the node for the shared caches, memory bus and network interfaces when using multiple processors per node.

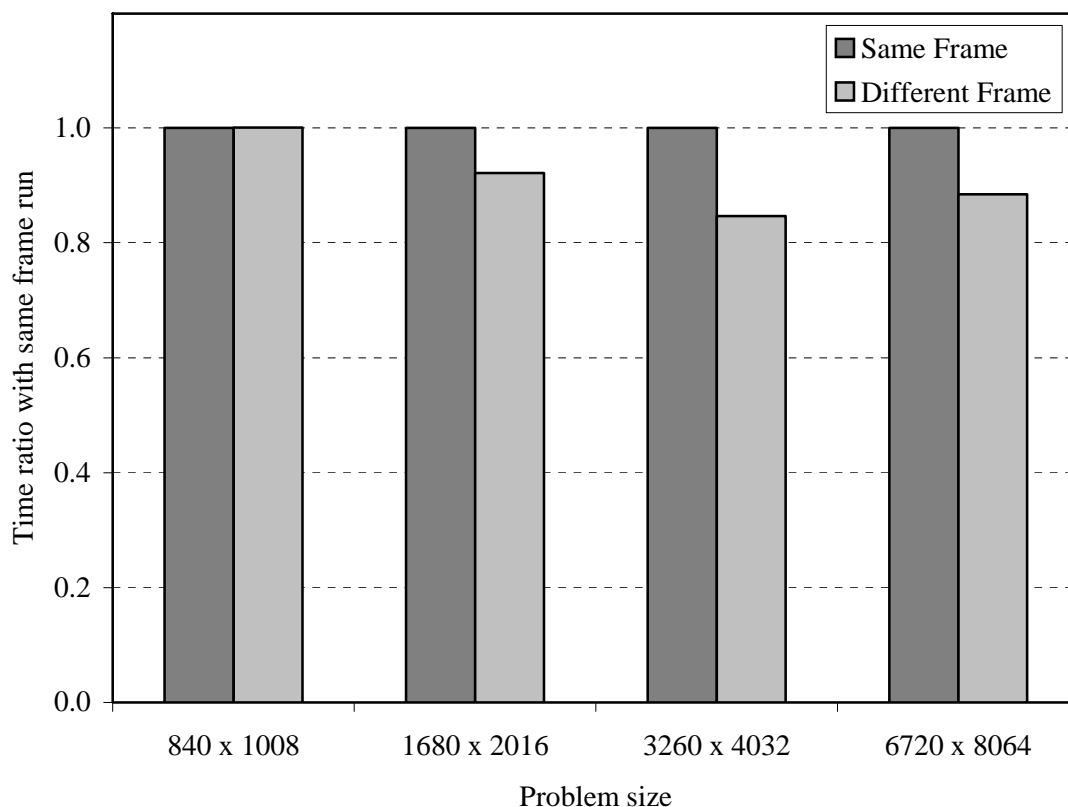


Figure 4. Wall clock time for the update component of the Jacobi benchmark on a range of problem sizes, when running on 4 LPARs on the same frame or 4 LPARs in different frames. All results are for 8 processors per LPAR.

In this section we have attempted to highlight the importance of making efficient use of the memory hierarchy and of understanding and optimising the communication patterns of our applications. In the following sections we examine the performance and scaling of a range of applications across a variety of disciplines, including materials science, molecular simulation, atomic and molecular physics, molecular electronic structure, computational engineering and environmental science.

5. MATERIALS SCIENCE

5.1. CPMD and CRYSTAL

CPMD: The CPMD code is based on the original code by Car and Parrinello. It is a production code with many unique features and currently has about 150,000 lines of code, written in Fortran 77 with the MPI communications library. Besides the standard Car-Parrinello method, the code is also capable of computing many different types of properties, including the inclusion of quantum effects on nuclei with the path integral method and interfaces for QM/MM calculations. Since January 2002 the source code has been freely available

for non-commercial use. Several thousand registered users from more than 50 countries have compiled and run the code on platforms ranging from notebooks to some of the largest high performance parallel computers.

Fast Fourier Transforms (FFT) are an essential part of all plane-wave calculations. In fact, it is the near linear scaling of FFTs that make large scale DFT calculations with plane wave basis sets possible. FFTs are used to transform the charge density and local potential between real space and reciprocal space. The number of these transforms is fixed and does not depend on the system size. On the other hand the transforms of wave functions from reciprocal space to real space and back (needed in the force calculation) has to be done for each state and dominates execution time for small and medium sized systems. Only for large systems (number of atoms larger than 1000) do the cubic scaling inner products and orbital rotations become dominant.

Different strategies are followed in parallel implementations of plane-wave / pseudo-potential codes. Parallelization of the CPMD code was done on different levels. The central parallelization is based on a distributed-memory coarse-grain algorithm that is a compromise between load balancing, memory distribution and parallel efficiency. This scheme achieves good performance on computers with up to about 200 CPUs, depending on system size and communication speed. In addition to the basic scheme, a fine-grain shared-memory parallelization was implemented. The two parallelization methods are independent and can be mixed. This allows us to achieve good performance on distributed computers with shared memory nodes and several thousands of CPUs, and also to extend the size of the systems that can be studied completely *ab initio*, to several thousand atoms.

Some methods implemented in CPMD allow a further level of parallelization. These methods, such as path-integral molecular dynamics or linear response theory, are embarrassingly parallel on the level of the energy calculation. Typically 2 to 16 copies of the energy and force calculation can be run in parallel. For these methods, an efficient use of computers with tens of thousands of CPUs can be envisaged. However, in this work only the main Car-Parrinello molecular dynamics part of the code has been used.

Our coarse-grain distributed-memory parallelization is driven by the distribution of wave-function coefficients for all states to all CPUs. Real-space grids are also distributed, whereas all matrices that do not include a plane-wave index are replicated (especially overlap matrices). All other arrays are only distributed if this does not cause additional communications. For a general data distribution in both spaces, each transform making up the 3D FFT would include communication between all processors. The data distribution in CPMD tries to minimize the number of communication steps while still having optimum load balancing in both spaces. This scheme requires only a single data communication step after the first transform. In addition, we can make use of the sparsity of the wave-function representation still present after the first transform and only communicate non zero elements.

The various load-balancing requirements are interrelated, and we use a heuristic algorithm to achieve near-optimum results. Our experience is that for all cases good load balancing is achieved for the reciprocal space. The restriction to full-plane distributions in real space, however, introduces severe problems in the case of a large number of processors. The number of planes available is typically about 50 for small systems and 200 to 300 for large systems. This restricts the maximum number of processors that can be used efficiently. The coarse granularity of this approach is also responsible for the appearance of magic numbers of processors where especially good performance can be achieved. This is no major problem because the appearance of these numbers is fully transparent. The efficiency of the scheme described above has a number of limitations which are discussed elsewhere [8].

Shared-memory parallelization on the loop level is achieved by using OpenMP compiler directives and multi-threaded libraries (BLAS and FFT) if available. In our tests, IBM's multi-threaded ESSL library has been used. Compiler directives have been used to ensure parallelization of all longer loops (those that depend on the number of plane waves or the number of grid points in real space), and to avoid parallelization of the shorter ones. This type of parallelization is independent of the MPI parallelization and can be used alone or in combination with the distributed-memory approach. Tests on various shared-memory computers have shown that an efficient parallelization up to 16 processors can be achieved.

The performance of CPMD has been tested on an IBM p690 using up to 1280 processors (Figure 5). The maximum performance achieved was above 1 Teraflop/s for a system consisting of 1000 atoms. This represents ~20% of the peak performance (33% of the Linpack Benchmark performance) and an overall parallel efficiency (calculated with an assumed single processor performance estimated from smaller systems) of 45%. This has to be compared with the maximum performance that can be achieved using BLAS3 library

calls of about 66% of peak performance on a POWER4 processor. Tests with smaller systems reveal that the main performance bottlenecks are the memory bandwidth on the shared-memory nodes and the latency of the node interconnect. The novel mixed parallelization scheme presented here is opening new frontiers for the *ab initio* study of molecular systems with several thousand of atoms on modern supercomputers.

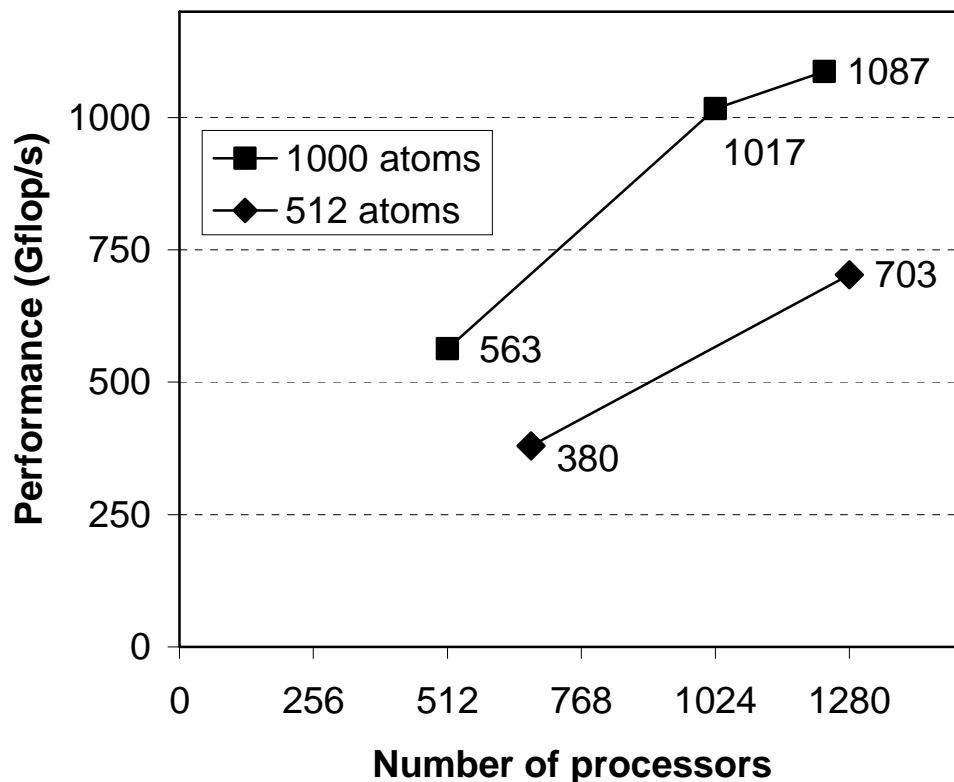


Figure 5. Performance of the CPMD code on the IBM p690

CRYSTAL: The CRYSTAL [4] code uses a periodic Hartree-Fock or density functional Kohn-Sham Hamiltonian and various hybrid approximations in the calculation of wave-functions and properties of crystalline systems. The wavefunctions are expanded in atom centred Gaussian type orbitals (GTOs) providing a highly efficient and numerically precise solution with no shape approximation to the density or potential. The code is developed within a long-standing collaboration between the Theoretical Chemistry Group at the University of Torino, Italy, and the Computational Materials Science Group at Daresbury Laboratory, and is widely distributed internationally. Details of the code and its applications can be found on the CRYSTAL web pages [4, 5].

Recent enhancements to the parallel distributed data version of the code, MPP CRYSTAL 2003, include the incorporation of a somewhat faster, and more numerically stable version of the parallel Jacobi diagonalizer [6]. Further, since it can diagonalize not only real symmetric but also Hermitian matrices, the ScaLAPACK library routines are no longer required. This is advantageous because the latter routines both scale poorly and also, dependent upon the eigenvalue spectrum, may require unacceptably large amounts of replicated memory.

The rationalization of the memory management within the code has continued. All large arrays are now dynamically allocated, and further general purpose Fortran 90 modules are available for more complex data structures, such as linked lists. On MPP systems disk access is often an expensive process, and so as far as is possible this is now avoided. Data is either recalculated or, for distributed objects, stored in memory, the latter being possible because of the memory capacity typically found on these machines.

Table 2. Time in Wall Clock Seconds for CRYSTAL calculations of Crystalline Crambin on the IBM p690 and SGI Origin 3800/R12k-400 for Three Different Basis Sets.

No. of CPUs	STO-3G Basis (3,948 GTOs)		6-31G Basis (7,194 GTOs)		6-31G** Basis (12,354 GTOs)	
	SGI O3800 / R12k-400	IBM p690	SGI O3800 / R12k-400	IBM p690	SGI O3800 / R12k-400	IBM p690
32	6559	3238	23457	11970		
64	3400	1762	12130	6333		
96	2327	1183	8240	4248		
128	1810	921	6251	3305		
192	1289	682	4437	2317		
256	1038	531	3496	1801		1924
384	772	416	2449	1321		
512		343		1043		1099
768		278		817		
1024		245		668		716

Timings on the IBM p690 and Origin 3800/R12k-400 of CRYSTAL 2003 for a benchmark calculation on crystalline crambin [7] with 1284 atoms per cell are reported in Table 2. The structure of Crambin is derived from XRD data at 0.52 Å. These calculations were performed in basis sets of increasing quality, with reported times for the STO-3G (3,948 GTOs) and 6-31G (7,194 GTOs) basis sets referring to 3 cycles of the iterative SCF process, and the times for the 6-31G** basis (12,354 GTOs) being for a single SCF iteration. The performance (an arbitrary inverse time metric fixed for each basis set) shown in Figure 6 reveals excellent scalability that is enhanced with improvements in the basis set.

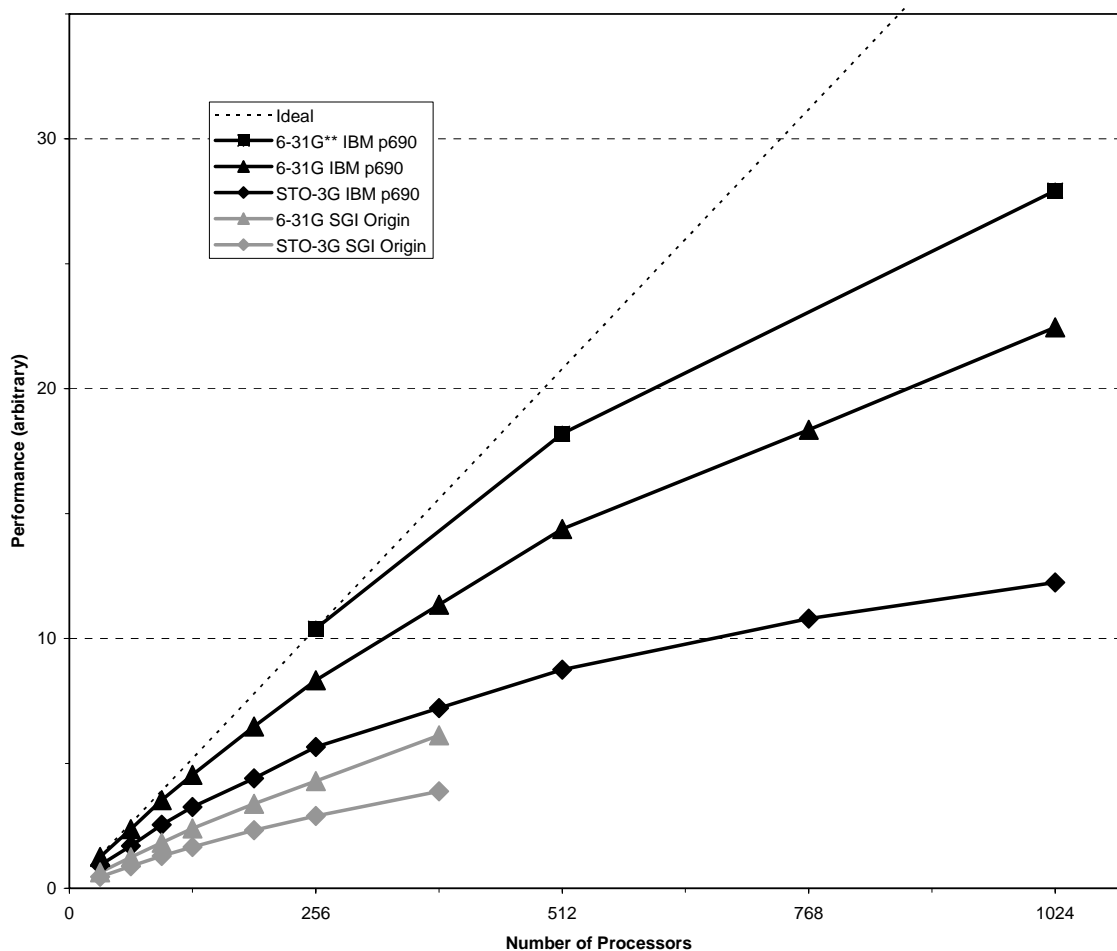


Figure 6. Scalability of CRYSTAL-2003 in Calculations on Crystalline Crambin.

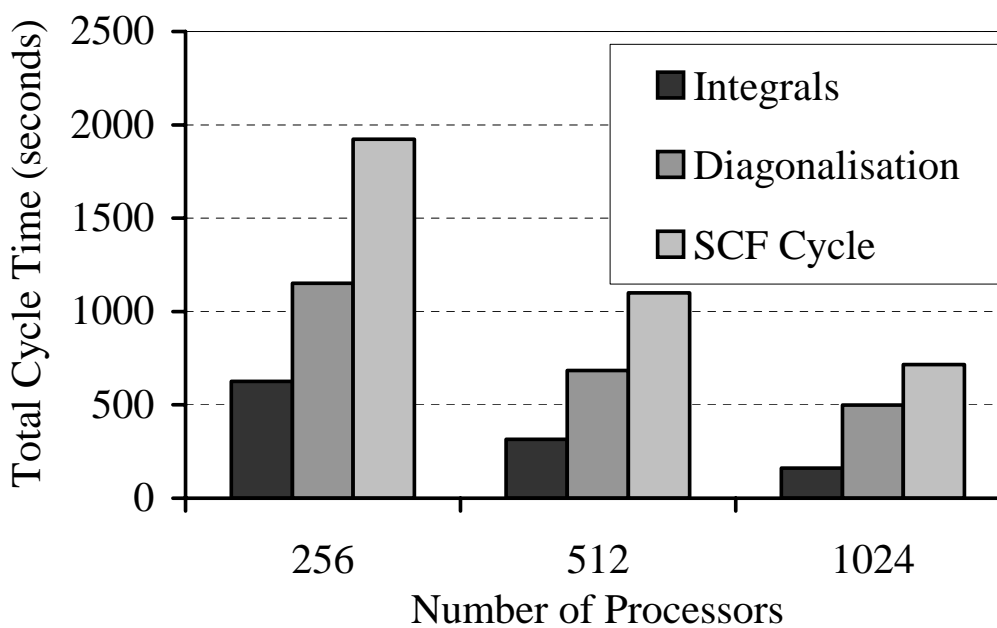


Figure 7. Breakdown of times for the first SCF cycle of CRYSTAL on the IBM p690 for 6-31G** calculations on Crystalline Crambin.

For the 12,354 GTO calculation, the integrals scale almost perfectly, with the diagonalisation around 3.1 times quicker on 1024 compared to 256 CPUs. A fit of measured data to Amdahl's law provides an estimated speed up of about 700 on 1024 processors; Figure 7 provides a breakdown of the cycle time as a function of processor count.

6. MOLECULAR SIMULATION

6.1. DL_POLY and NAMD

DL-POLY: DL_POLY [9] is a general-purpose molecular dynamics simulation package designed to cater for a wide range of possible scientific applications and computer platforms, especially parallel hardware. Two graphical user interfaces are available, one based on the CERIU² Visualiser from Accelrys and the other on the Java programming language.

DL_POLY supports a wide range of application areas, including [10] ionic solids, solutions, metals, zeolites, surfaces and interfaces, complex systems (e.g. liquid crystals), minerals, bio-systems, and those in spectroscopy. Comprehensive benchmarking of the replicated data (RD) version (Version 2.11) of DL_POLY [11,12] clearly reveals the limitations inherent in the RD strategy, with restrictions in the size of system amenable to study, and limited scalability on current high-end platforms. These limitations apply not only to systems possessing complex molecular topologies and constraint bonds, but also to systems requiring simple atomic descriptions, systems that historically exhibited excellent scaling on the Cray T3E/1200E. Significant enhancements to the code's capabilities have arisen from the recent release of the distributed data (or domain decomposition) version (DL_POLY 3) [16], developments that have been accelerated in light of the arrival of the HPCx system.

Evaluation of the Coulomb potential and forces in DL_POLY is performed using the smooth particle mesh Ewald (SPME) algorithm [13]. As in all Ewald [14] methods, this splits the calculation into two parts, one performed in real space and one in Fourier space. The former only requires evaluation of short ranged functions, which fits in well with the domain decomposition used by DL_POLY 3, and so scales well with increasing processor count. However the Fourier component requires 3 dimensional FFTs to be performed. These are global operations and so a different strategy is required if good scaling is to be achieved.

The original implementation involved replicating the whole FFT grid on all processors and performing the FFTs in serial after which each processor could evaluate the appropriate terms for the atoms that it held. This method clearly has a number of well known drawbacks.

While both open source 3D parallel FFTs (such as FFTW [15]) and proprietary routines (such as Cray's PCCFFT) are available, neither adequately address all the issues. The problem is that they impose a data distribution, typically planes of points, that is incompatible with DL_POLY's spatial domain decomposition, so while a complete replication of the data is not required, it is still necessary to perform extensive data redistribution which will limit the scaling of the method.

To address these limitations, a parallel 3D FFT has been written [16] which maps directly onto DL_POLY's data distribution; this involved parallelizing the individual 1D FFTs in an efficient manner. While the method will be slower than the proprietary routines for small processor counts, at large numbers it is attractive, since (a) while moving more data in total, the method requires much fewer messages, so that in the latency dominated regime it should perform better, and (b) global operations, such as the *all to all* operations used in both FFTW and PCCFFT, are totally avoided. More generally the method is extremely flexible, allowing a much more general data distribution than those of other FFTs, and as such should be useful in other codes which do not map directly onto a "by planes" distribution.

In the present section we present recent results obtained on the IBM p690, Compaq AlphaServer ES45/1000 and SGI Origin 3800/R14k-500, results which highlight the drastic improvements in both system size and performance made possible through these developments. The four benchmarks reported in Table 3 include two Coulombic-based simulations of NaCl, one with 27,000 ions, the second with 216,000 ions. Both simulations involve use of the Particle Mesh Ewald Scheme, with the associated FFT treated by the algorithm

outlined above [16] in which the traditional all-to-all communications are replaced by the scheme that relies on column-wise communications only. The reported timings are for 500 time steps in the smaller calculation, and 200 time steps in the larger simulation. The other two benchmarks are macromolecular simulations based on Gramicidin-A; the first includes a total of 99,120 atoms and 100 time steps. The second, much larger simulation, is for a system of eight Gramicidin-A species (792,960 atoms), with the timings reported for just 50 time steps. In terms of time to solution, we see that the AlphaServer SC outperforms the Origin 3800 at all processor counts in all four benchmarks; both 256 CPU runs for the larger NaCl and Gramicidin-A simulations suggest factors of 1.3-1.4.

These results show a marked improvement in performance compared to the replicated data version of the code [11], with the gratifying characteristic of enhanced scalability with increasing size of simulation, both in the ionic and macromolecular simulations. Considering the NaCl simulations, we find speedups of 139 and 122 respectively on 256 processors of the Origin 3800 and AlphaServer SC in the 27,000-ion simulation. These figures increase to 172 and 171 respectively in the larger simulation featuring 216,000 ions. While the total times to solution on the IBM p690 are broadly in line with the AlphaServer SC, the scalability remains inferior.

Table 3. Time in Wall Clock Seconds for four DL_POLY 3 benchmark Calculations on the Compaq AlphaServer SC ES45/1000, IBM p690 and SGI Origin 3800.

CPU's	SGI Origin 3800 / R14k-500	Compaq Alpha ES45 / 1000	IBM p690
NaCl ; 27,000 ions, 500 time steps			
16	313	183	160
32	168	103	97
64	92	57	61
128	53	37	42
256	36	24	
NaCl; 216,000 ions, 200 time steps			
16	764	576	455
32	387	326	234
64	201	168	128
128	116	91	78
256	71	54	48
512			41
Gramicidin A; 99,120 atoms, 100 time steps			
16	282	173	166
32	167	109	107
64	100	75	74
Gramicidin A; 792,960 atoms, 50 time steps			
32	661	370	349
64	273	186	189
128	140	109	114
256	97	68	77
512			56

For these systems the FFT routine remains the poorest scaling; the time required to exchange scales poorly on the IBM p690 with increased processor count. Doubling the number of processors roughly halves the amount of data a processor has to send, so one would expect the time to roughly halve. In practice this is not the case, and at certain processor counts the time dramatically increases. This can be traced to the message passing occurring via the switch rather than through shared memory. A more detailed timing breakdown than

presented here reveals that for the x direction, which never has to use the switch in the present case, the scaling is reasonable, while for the z direction, which has to use the switch for the first time at 16 processors, there is a major spike in the timings.

A more compelling improvement with system size is found in the macromolecular Gramicidin-A simulations. Again the SPME algorithm is used for evaluation of the Coulomb field, but in these simulations there is the extra complication of constraints on the atoms' motions, which reflects chemical bonds in the system. The shake algorithm is used to evaluate the constraints, and this is again potentially a global operation and so, as for the FFT, good scaling is difficult to achieve. In the distributed data implementation, both SHAKE and short-range forces require only nearest neighbour communications, suggesting that communications should scale linearly with the number of nodes, in marked contrast to the replicated data implementation. This is borne out in practice. In the larger simulation (with 792,960 atoms) we find speedups of 218 and 175 on 256 processors of the Origin 3800 and AlphaServer SC respectively. This level of scalability represents a significant advance over that exhibited by both DL_POLY 2 and CHARMM [11].

NAMD is the parallel, object-oriented molecular dynamics program designed for high performance simulations of large biomolecular systems [17]. NAMD employs the prioritized message-driven execution capabilities of the Charm++/Converse parallel runtime system, allowing excellent parallel scaling on both massively parallel supercomputers and commodity-based workstation clusters. An initial implementation of the code on HPCx was carried out by Rick Kufirin (NCSA), and Table 4 shows the elapsed time per time step for the standard NAMD ApoA-I benchmark [18], a system comprising 92,442 atoms, with 12Å cutoff and PME every 4 time steps, periodic. Also shown are the timings from the optimised version of the code running on PSC's AlphaServer SC ES45/1000. These show that the AlphaServer marginally outperforms the IBM

Table 4: Elapsed Time per time step (seconds) for the NAMD ApoA-I Benchmark on the IBM p690 and Compaq AlphaServer SC ES45/1000.

Number of processors	Compaq Alphaserver SC ES45/1000	IBM p690
1	7.86	7.97
2		4.47
4		2.19
8		1.11
16		0.580
32		0.305
64		0.163
128	0.0715	0.0985
256	0.0403	0.0664
512	0.0239	0.0424
1024	0.0176	0.0252

p690 on a single CPU, and with a speed-up of 447 on 1024 processors, exhibits superior scalability compared to the IBM p690 (a corresponding speed-up of 316). This is to be expected in light of the superior interconnect associated with the AlphaServer. We would expect these scalability figures to improve with larger simulations in light of performance attributes demonstrated on the TCS-1 system at Pittsburgh (e.g. a speedup of 778 on 1024 CPUs in a 327K particle simulation of F_1 -ATPase).

7. ATOMIC & MOLECULAR PHYSICS

7.1. H2MOL and PFARM

H2MOL [19] is a code from the Multiphoton and Electron Collision Consortium (CCP2) written by Ken Taylor & Daniel Dundas, Queens University Belfast. It solves the time-dependent Schrödinger equation to

produce estimates of energy distributions for laser-driven dissociative ionization of the H_2 molecule. A cylindrical computational grid is defined with ϕ , ρ and Z co-ordinates, with the Z domain distributed amongst an array of processors arranged logically in a triangular grid (to take advantage of symmetry). A feature of the way this code has been written is that it specifies as constant the number of grid points **per processor** in the z -direction. Thus with increasing numbers of processors it is working with an increasingly refined mesh i.e., for this benchmark, perfect scaling would be represented by a flat timing profile across the different processor counts.

The results of Table 5 refer to a problem definition with ϕ points = 11, ρ points = 30 and Z points = 11 (per processor). This is the maximum problem size that the restricted memory on the Cray-T3E/1200E (256 MByte) can accommodate.

The IBM p690 / Cray T3E performance ratio starts at around 3 on 6 processors and reduces to just above 2.5 on 231 processors. For calculations involving low numbers of processors the code spends most of its time in highly optimized ESSL and LIBSCI library routines (ZGEMM, ZAXPY and ZDOT). The performance achieves around one third of peak for both IBM and Cray machines. Therefore the best relative performance to be expected on low processor counts reflects the peak ratios of the two machines i.e. 5.2 Gflops (IBM p690) / 1.2 Gflops (Cray T3E) = 4.3. By 120 processors (8 tasks on 15 nodes) there is no more than a factor of two performance gain from the IBM p690. At 231 processors, with partially occupied nodes (7 tasks on each of 33 nodes), the performance ratio has improved to around 2.5. Table 5 demonstrates that the timings on large processor counts of HPCx are fairly flat, representing reasonable scaling. Runs undertaken with fully occupied L-PAR'd nodes (*) are slower than those with at least one free task per node.

Table 5: Total Elapsed Times (seconds) for the fixed-point H2MOL Benchmark (ϕ points = 11, ρ points = 30 and Z points = 11 (per processor)).

Number of processors	Cray T3E/1200E	IBM p690	(Tasks,T) x (Nodes,N)
6	5650	2098	6T x 1N
15	5935	2343	5T x 3N
28	6155		
45	6396		
66	6622	2660	6T x 11N
91	7123		
120	7198	3353	8T x 15N*
		2550	6T x 20N
231	7742	3004	7T x 33N
325		3071	5T x 65N
435	†	3269	5T x 87N
496	†	3952*	8T x 62N*

† Memory Exceeded

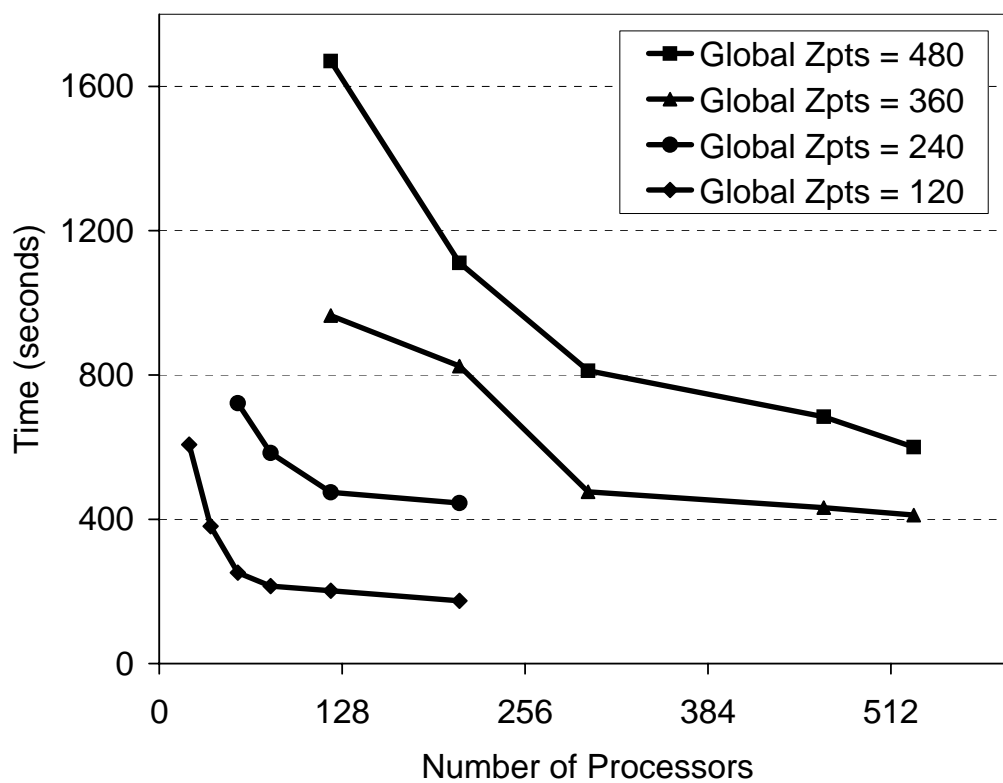


Figure 8. The H2MOL benchmark on the IBM p690 with fixed problem size

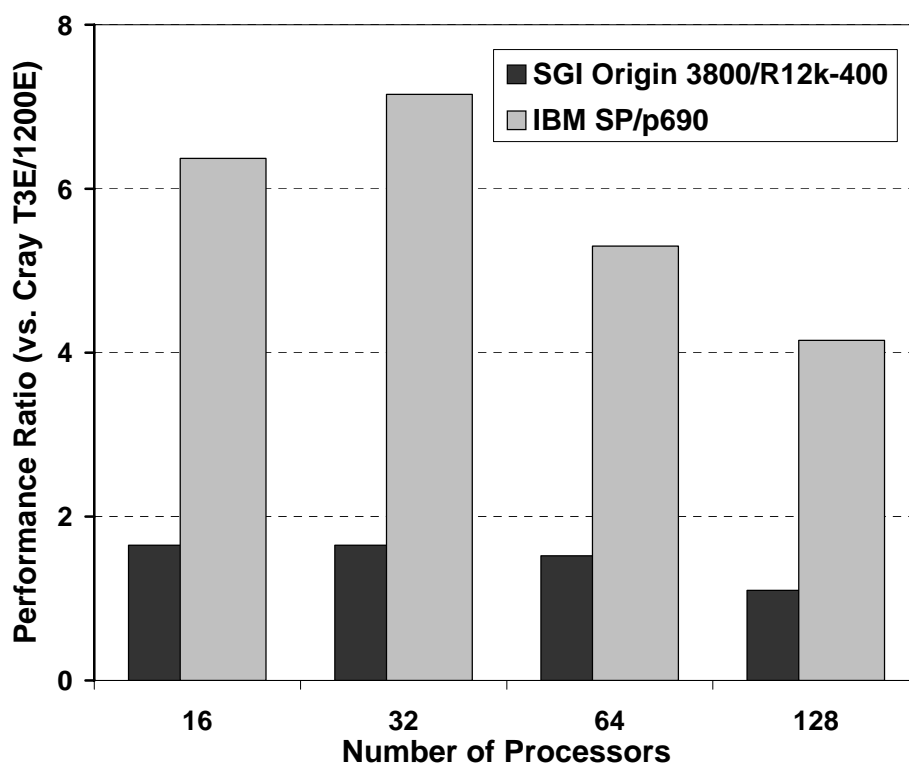


Figure 9. The PFARM benchmark showing performance on the IBM p690 and the SGI Origin 3800/R12k-400 relative to the Cray T3E/1200E performance.

Fixing a global number of Z points, rather than a local number of Z points, allows the scaling of the code to be examined more clearly. The results of Figure 8 (from the IBM p690) involve the ϕ and ρ parameters being set as above, with the total (global) number of Z points remaining fixed for different sized processor arrays. Hence, each run is now computing an identical problem. In the graph above, the number of timesteps has been reduced to increase throughput (this has no impact on scaling).

Evidently, the scalability improves as the problem size increases. Currently, users of the code are investigating problems with over 400 Z points on the SGI Origin 3800 at CSAR.

PFARM: In recent years new high performance codes and techniques have been developed to extend the successful R-matrix formalism to treat applications such as the description of the edge region in Tokamak plasmas (fusion power research) and for the interpretation of astrophysical spectra. The parallel R-matrix program PFARM has been optimised to treat open d-shell atoms and ions as well as intermediate energy scattering problems [20]. Several new facilities have been incorporated into PFARM to improve its efficiency for these calculations. In electron-ion scattering calculations it is necessary to perform separate computations at a very large number of scattering energies in order to compute the thermally averaged collision strengths required in applications. The program splits this energy mesh into a fine region (below all target thresholds) and coarse region (above all target thresholds). By treating each region separately, optimisations specific to each can be made. The parallel calculation has been designed to take advantage of the near-optimal performance of serial BLAS 3 routines; this operation is the computational rate-determining step. Therefore the effect of factors such as matrix dimension, matrix transposes and symmetry have been studied closely, and data is now arranged to maximise the BLAS 3 performance [20]. The major outstanding efficiency issues now relate to the required matrix diagonalisations (using the PeIGS parallel eigensolver [21,22]) and to load balancing. To address the latter problem, automated load balancing that adapts to each specific application has been developed. A computational model of the parallel calculation has been constructed that describes the relative speed of each component of the functional decomposition. Factors such as physical properties, hardware, relative BLAS/LAPACK performance and channel-splitting are all included in the model. The model is incorporated into a controlling Perl script that can predict dynamically an optimum configuration of processors for a particular physical problem. For each computation, processor configurations and the associated timing data is saved in order to refine the model for future runs.

The code has been ported to the IBM p690 and SGI Origin. Figure 9 shows the performance of the IBM p690 and Origin 3800/R12k-400 relative to the Cray T3E/1200E for a typical Ni^{3+} scattering problem. This figure suggests that the IBM p690 outperforms the SGI Origin 3800 and Cray T3E/1200E by factors of 3.8 and 4.2 on 128 processors.

8. MOLECULAR ELECTRONIC STRUCTURE

8.1. GAMESS-UK

GAMESS-UK [23] represents a typical established electronic structure code, comprising some 800K lines of Fortran that permits a wide range of computational methodology to be applied to molecular systems. Improving the scalability of the parallel code has been addressed in part by adopting a number of the tools developed by the High Performance Computational Chemistry (HPCC) group from the Environmental Molecular Sciences Laboratory at PNNL, in Richland, Washington. These include the Global Array (GA) toolkit [3] that provides an efficient and portable "shared-memory" programming interface for distributed-memory computers, and the scalable, fully parallel eigensolver, PeIGS whose numerical properties satisfy the needs of the chemistry applications [21,22].

The main source of parallelism in the DFT module is the computation of the one- and two-electron integrals together with the exchange correlation contributions, and their summation into the Fock matrix. The computation of these quantities is allocated dynamically using a shared global counter. With the capabilities afforded by the GA tools [3], some distribution of the linear algebra becomes trivial. As an example, the SCF convergence acceleration algorithm (DIIS - direct inversion in the iterative subspace) is distributed using GA storage for all matrices, and parallel matrix multiply and dot-product functions. This not only reduces the time

Table 6. Time in Wall Clock Seconds for Four GAMESS-UK Benchmark Calculations on the Compaq AlphaServer SC ES45/1000, IBM p690 and SGI Origin 3800/R14k-500.

CPU's	SGI Origin 3800 / R14k-500	Compaq Alpha ES45 / 1000	IBM p690
Cyclosporin (1000 GTOs) DFT/B3LYP 6-31G			
32	1191	713	666
64	704	424	424
128	481	310	322
Cyclosporin (1855 GTOs) DFT/B3LYP 6-31G**			
32	4731	2504	2614
64	2838	1584	1681
128	1867	1100	1281
Valinomycin (882 GTOs) DFT/HCTH			
32	2306	1301	1329
64	1228	705	749
128	734	415	493
Valinomycin (1620 GTOs) DFT/HCTH			
32	11053	5711	5557
64	5846	3081	3109
128	3388	1825	1940
256	2274	1162	1394
(C ₆ H ₄ (CF ₃) ₂) ₂ SCF 2nd Derivatives			
16	1490		860
32	803	501	621
64	494	360	371
128		246	213

to perform the step, but the use of distributed memory storage (instead of disk) reduces the need for I/O during the SCF process. Diagonalisation of the resulting Fock matrix is now based on the PeIGS module from NWChem [24].

Substantial modifications were required to enable the SCF 2nd derivatives [25] to be computed in parallel. The conventional integral transformation step has been omitted, with the SCF step performed in direct fashion and the MO integrals, generated by re-computation of the AO integrals, and stored in the global memory of the parallel machine. The GA tools manage this storage and subsequent access. The basic principle by which the subsequent steps are parallelised involves each node computing a contribution to the current term from MO integrals resident on that node. For some steps, however, more substantial changes to the algorithms are

required. The coupled Hartree-Fock (CPHF) step and construction of perturbed Fock matrices are again parallelised according to the distribution of the MO integrals. The most costly step in the serial 2nd derivative algorithm is the computation of the 2nd derivative two-electron integrals. This step is trivially parallelised through a similar approach to that adopted in the direct SCF scheme - using dynamic load balancing based on a shared global counter. In contrast to the serial code, the construction of the perturbed Fock matrices dominates the parallel computation. It seems almost certain that these matrices would be more efficiently computed in the AO basis, rather than from the MO integrals as in the current implementation, thus enabling more effective use of sparsity when dealing with systems comprising more than 25 atoms.

The performance of the DFT and SCF 2nd Derivative modules on the SGI O3800/R14k-500, Compaq AlphaServer SC ES45/1000 and IBM p690 are shown in Table 6. Note that the DFT calculations did not exploit CD fitting, but evaluated the coulomb matrix explicitly. Considering the DFT results, modest speedups of 81, 73 and 65 are obtained on 128 processors of the Origin 3800, AlphaServer SC and IBM p690 respectively for the larger cyclosporin calculation. Somewhat better scalability is found in both Valinomycin DFT calculation where a greater proportion of time is spent in integral evaluation arising from the more extended basis sets [26]; speedups of 104, 100 and 92 respectively are obtained on 128 processors of the Origin, AlphaServer and IBM p690 in the 1620 GTO calculation. The enhanced performance of the SCF 2nd Derivative module on the IBM p690 arises from the decreased dependency on latency exhibited by the current implementation compared to the DFT module. Thus the timings of Table 6 suggest that, at least for the 2nd derivatives module, the IBM p690 is outperforming the AlphaServer SC on 128 processors.

The less than impressive scalability of both GAMESS-UK (and also NWChem) on the IBM p690 arises to some extent from the dependency of both codes on a Global Array implementation that is based on IBM's LAPI communication library [27]. The current implementation of LAPI on POWER4-based architectures is far from optimal, with the measured latencies and bandwidths significantly inferior to those measured on corresponding POWER3-based systems. We are currently working with PNNL and IBM's LAPI team to further understand and address these shortcomings.

8.2. Parallel Eigensolver Performance

Many scientific MPP applications under development involve the computation of the standard real symmetric eigensystem problem. In particular, this diagonalization stage in parallel quantum chemistry codes often consumes a significant percentage of overall run time. Several public domain parallel eigensolvers are available for general use including ScaLAPACK library routines [28] and Parallel Eigensolver System software (PeIGS) library routines (from PNNL, [21,22]). The latest release of ScaLAPACK (v1.7) includes new parallel algorithms based on divide-and-conquer techniques [29]. Also, algorithms developed recently by Dhillon et al [30] are included in a new release of PeIGS (v3.0) [22]. In addition, Ian Bush at Daresbury Laboratory has developed BFG [31], a parallel one-sided Block Factored Jacobi implementation [21] based upon the techniques described by Littlefield et al [22].

The fastest serial diagonalisation algorithm is that based upon Householder reduction to tridiagonal form followed by diagonalization [32]. A number of parallel implementations of this algorithm exist, e.g. ScaLAPACK and the PeIGS package, but they all suffer from a number of drawbacks. The most obvious is that even for quite large matrices the scaling is less than impressive at large processor counts (see below).

However on closer investigation further problems can be seen. Both ScaLAPACK and PeIGS require a rather indeterminate amount of workspace to operate, and this workspace is replicated across all the processors; in ScaLAPACK in the worst case scenario the size of this workspace can be the size of the whole matrix. Obviously for the very large matrices that one wishes to diagonalize on high-end systems this is highly undesirable, and may even be impossible.

Further many algorithms can easily generate a good guess at the eigenvectors, and it would be very useful if this guess could be used to speed up the diagonalization. Unfortunately this is difficult within the Householder methodology; an alternative here is Jacobi's algorithm. Although slower in serial, typically by a factor of three to five, it has a number of attractive properties for practical parallel codes; the scaling is good (see below), the memory requirement scales strictly with the inverse of the number of processors and is known a priori, and guesses at the eigenvectors, provided they are mutually orthogonal, can be used very effectively to speed up the diagonalization. The DL-based BFG code implements a version of this algorithm as described by Littlefield and Maschoff [21], with the current version offering extended functionality, better numerical stability, increased single processor performance and better scaling than its predecessors. It can diagonalize

both real symmetric and Hermitian matrices, and has interfaces for matrices distributed either in a block-cyclic (like ScaLAPACK) fashion, or by columns (PeIGS). It is written in standard conforming Fortran and uses MPI for message passing, and so it is portable to a wide range of parallel machines. Further extensive use of MPI communicators make it flexible enough that matrices may be diagonalized across just a subset of the processors, thus allowing a number of diagonalizations to be carried out at once.

Using an example fully symmetric Fock matrix of dimension 3888 from the CRYSTAL quantum chemistry code, the performance of the following methods on the SGI Origin 3800 and IBM p690 platforms are compared in Figure 10

- PeIGS 3.0 (PDSPEV) [22] implements an optimised inverse iteration algorithm with orthogonalization performed in parallel [30].

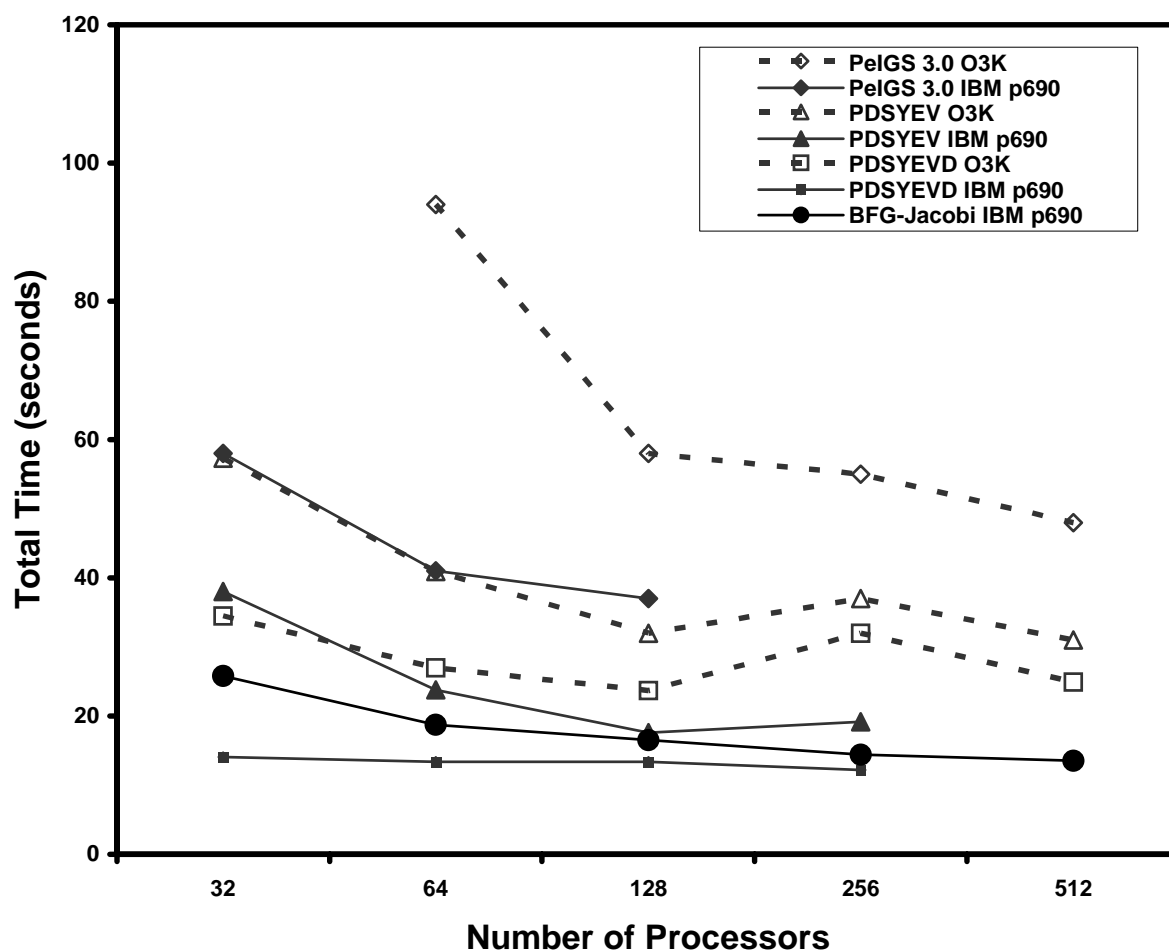


Figure 10. Timings (seconds) for the PeIGS, ScaLAPACK and BFG routines on the SGI Origin 3800 and IBM p690.

- PDSYEV (Scalpack 1.5 & 1.7) uses the QR algorithm [36].
- PDSYEVD (ScaLAPACK 1.7) uses the Parallel Divide and Conquer Algorithm [29]. This consists of an initial partition of the problem into sub-problems and then after appropriate computations, results are joined together using the rank-one update of Cuppen.
- The Block Factored Jacobi method (BFG) [21] is an iterative procedure that uses repeated applications of orthogonal Jacobi rotations, which, depending on the order of them, bring the matrix

more or less quickly to diagonal form. No initial reduction of the matrix to tridiagonal form is required.

Figure 10 shows the time taken to calculate the full set of eigenvalues and eigenvectors of a Fock matrix (dimension=3888) from CRYSTAL representing lithium fluoride with an F centre. The set of eigenvalues contains several tightly packed clusters and therefore re-orthogonalisation of the eigenvectors will be required in the PeIGS routines. All runs were undertaken on the SGI Origin 3800 and IBM p690 unless specified. The results suggest that the new ScaLAPACK v1.7 divide-and-conquer routine provides the best overall performance. On this test case, it is consistently 20-30% faster than the ScaLAPACK v1.5 routine. None of the methods based on Householder Reduction methods scale above 128 processors on the Origin 3800 as communication overheads begin to dominate run-times. Both ScaLAPACK routines outperform the PeIGS routines, mainly due to data blocking strategies that allow use of highly-optimised BLAS 3 operations. However, both ScaLAPACK routines may require prohibitive allocations of workspace to guarantee orthogonality of the eigenvectors, particularly in cases with clustered eigenvalues. Both PeIGS routines also require substantial amounts of workspace, though this is generally less than ScaLAPACK. For this test case, PeIGS v3.0 showed little performance improvement over PeIGS v2.1, however it should be stressed that PeIGS v3.0 is still under development, and the version tested does not yet implement fully the optimised inverse iteration algorithm described in [30]. The performance of the parallel BFG code developed by Bush performs slowly on lower processor counts, but it does scale well up to large processor counts, and is competitive with the ScaLAPACK divide-and-conquer routine on 512 Origin 3800 processors.

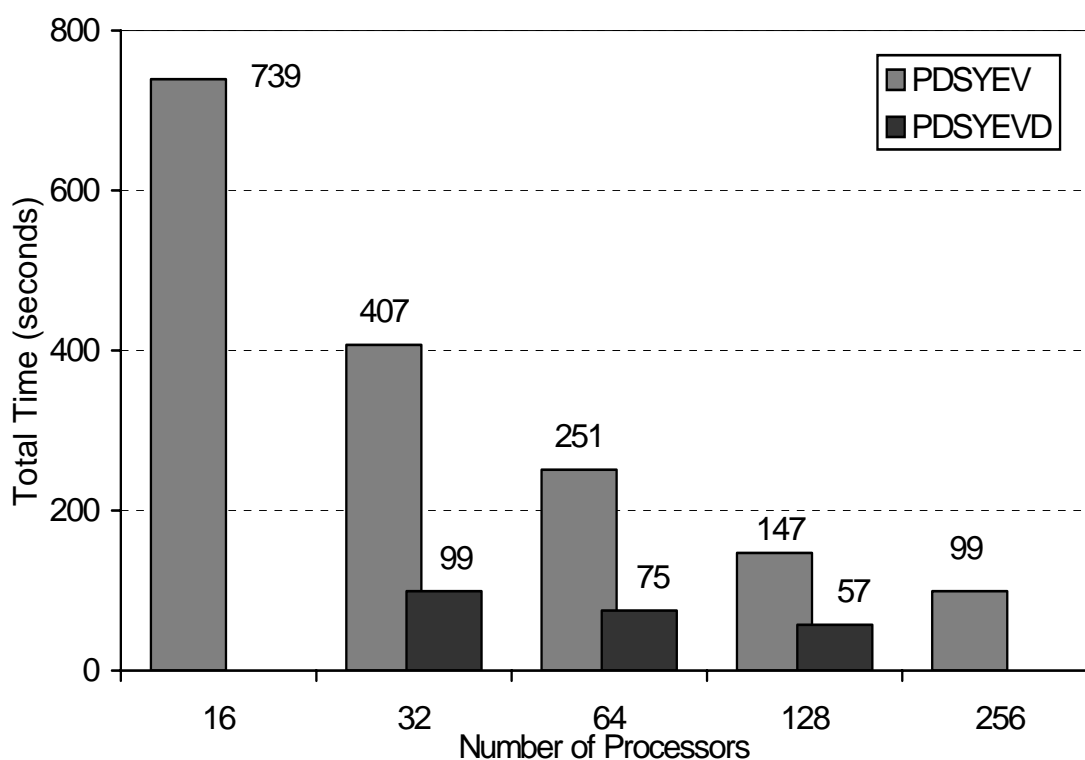


Figure 11. Timings (seconds) for the PDSYEV and PDSYEVD ScaLAPACK routines on the IBM p690 when diagonalising a matrix of dimension 9,000.

Finally we show in Figure 11 the improved scalability on the IBM p690 of both ScaLAPACK routines, PDSYEV and PDSYEVD, when increasing the matrix dimension size from 3888 to 9000. Of particular note are the impressive times to solution obtained from the divide-and-conquer routines.

9. COMPUTATIONAL ENGINEERING

9.1. PCHAN

Fluid flows encountered in real applications are invariably turbulent. There is, therefore, an ever-increasing need to understand turbulence and, more importantly, to be able to model turbulent flows with improved predictive capabilities. As computing technology continues to improve, it is becoming more feasible to solve the governing equations of motion – the Navier-Stokes equations – from first principles. The direct solution of the equations of motion for a fluid, however, remains a formidable task and simulations are only possible for flows with small to modest Reynolds numbers. Within the UK, the Turbulence Consortium (UKTC) has been at the forefront of simulating turbulent flows by direct numerical simulation (DNS). UKTC has developed a parallel version of a code to solve problems associated with shock/boundary-layer interaction. The code (SBLI) was originally developed for the Cray T3E and is a sophisticated DNS code that incorporates a number of advanced features: namely high-order central differencing; a shock-preserving advection scheme from the total variation diminishing (TVD) family; entropy splitting of the Euler terms and the stable boundary scheme [27]. The code has been written using standard Fortran 90 code together with MPI in order to be efficient, scalable and portable across a wide range of high-performance platforms.

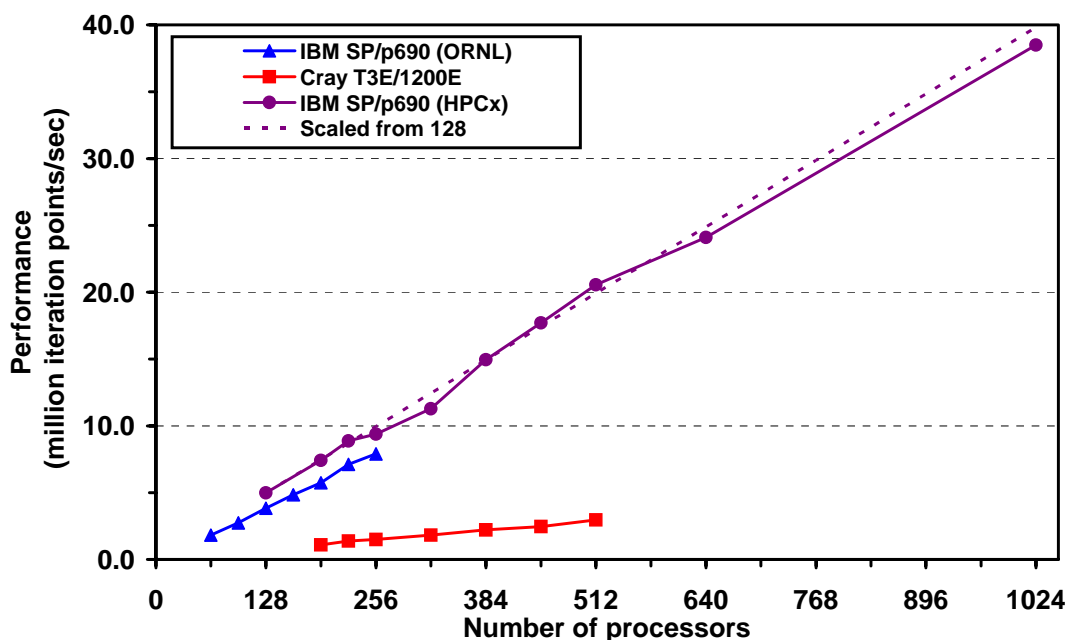


Figure 12. The PCHAN T3 (360x360x360) benchmark on the IBM p690 and Cray T3E/1200E systems. The dashed line shows the ideal performance scaled from the HPCx 128 processor figure.

The PCHAN benchmark is a simple turbulent channel flow benchmark using the SBLI code. Performance with the T3 Grid Benchmark data case (360x360x360) shows close to ideal scaling on both the Cray T3E/1200E and on the IBM p690 systems, for which we have data from both the Cheetah system (at ORNL) and from the HPCx system. Where benchmark runs allow direct processor-for-processor comparison between the IBM and Cray systems (192-512 processors), the performance ratio is fairly constant at around 6.2 to 7.2. e.g., the 512 CPU T3 Benchmark required 227 elapsed seconds, with the HPCx system outperforming the Cray T3E/1200E (1,575 seconds) by a factor of 6.94. Figure 12 shows performance results from the T3E and the two IBM systems. The HPCx runs incorporate code optimisations, which account for a 20%-30% improvement; these were not present in the Cheetah implementation.

The most important communications structure within PCHAN is a halo-exchange between adjacent computational sub-domains. Providing the problem size is large enough to give a small surface area to volume ratio for each sub-domain, the communications costs are small relative to computation and do not constitute a

bottleneck. Code optimisation for the POWER4 cache architecture has shown to be highly beneficial in increasing the performance.

10. ENVIRONMENTAL SCIENCE

10.1. POLCOMS

The Proudman Oceanographic Laboratory Coastal Ocean Modeling System (POLCOMS) has been developed to tackle multi-disciplinary studies in coastal/shelf environments [34]. The central core is a sophisticated 3-dimensional hydrodynamic model that provides realistic physical forcing to interact with, and transport, environmental parameters.

The hydrodynamic model is a 4-dimensional finite difference model based on a latitude-longitude Arakawa B-grid in the horizontal and S-coordinates in the vertical. Conservative monotonic PPM advection routines are used to ensure strong frontal gradients. Vertical mixing is through turbulence closure (Mellor-Yamada level 2.5).

In order to study the coastal marine ecosystem, the POLCOMS model has been coupled with the European Seas Regional Ecosystem Model (ERSEM) [35]. Studies have been carried out, with and without the ecosystem sub-model, using a shelf-wide grid at 12km resolution. This results in a grid size of approx. 200 x 200 x 34. In order to improve simulation of marine processes, we need accurate representation of eddies, fronts and other regions of steep gradients. The next generation of models will need to cover the shelf region at approximately 1km resolution.

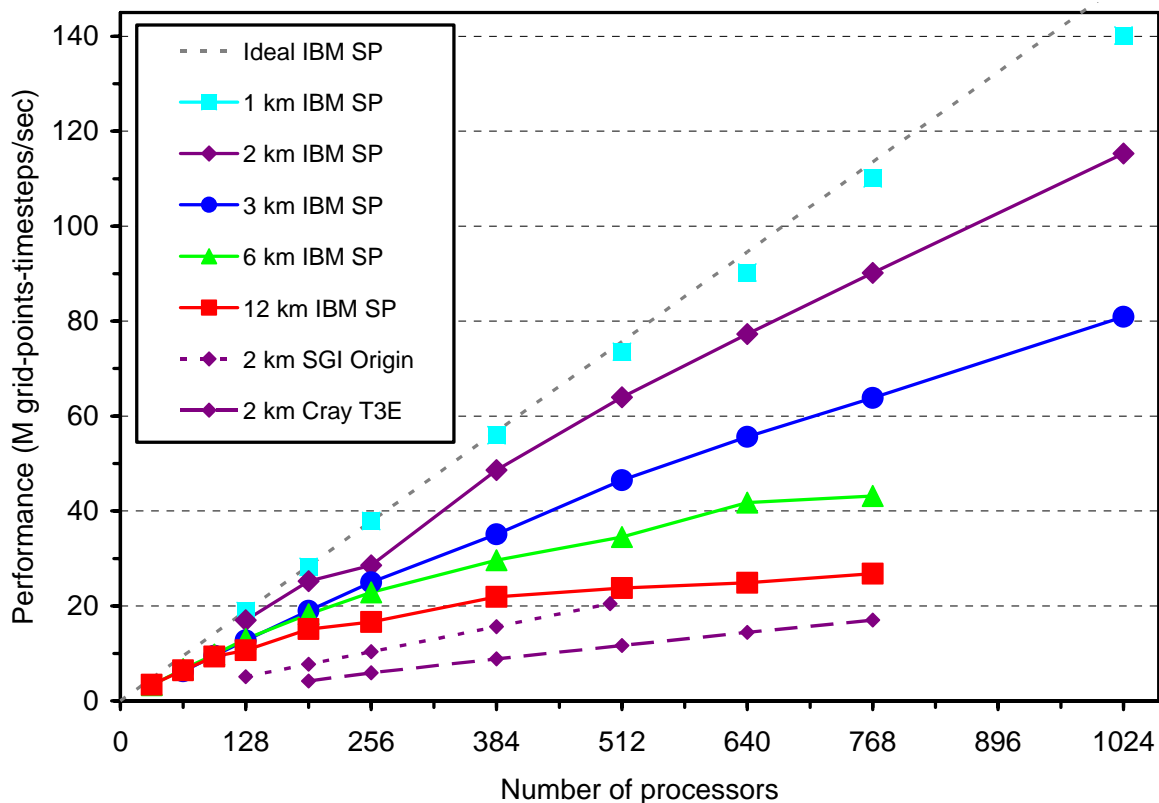


Figure 13. Performance of the POLCOMS hydrodynamics code as a function of grid resolution.

In order to assess the suitability of the POLCOMS hydrodynamic code for scaling to these ultra-high resolutions we have designed a set of benchmarks which runs (without the ecosystem model) at grid sizes representative of resolutions from the current 12km down to 1km. The resolutions (and horizontal grid

dimension) chosen are 12km (200), 6km (400), 3km (800), 2km (1200) and 1km (2400). The number of vertical levels was fixed at 34. In order to keep benchmark run times manageable, the runs were kept short (100 timesteps) and the initialisation and finishing times were subtracted from the total run time. So as to compare properly runs with different grid sizes, performance is reported in Figure 13 as the amount of work (gridpoints \times timesteps) divided by the time. Runs using up to 1024 processors of the HPCx system are shown compared to the Cray T3E-1200E and Origin 3800/R12k-400 systems operated by CSAR at the University of Manchester, UK.

We find, as expected, that, as the grid size increases, the ratio of communication to computation in the code improves and so does the scalability. At 2km resolution the code is scaling almost linearly on all three systems with the HPCx system delivering approx. 5.5 times the performance of the Cray T3E and 3.1 times the Origin 3800.

11. SUMMARY

We have introduced HPCx, the UK's new National HPC Service which aims to deliver a world-class service for capability computing to the UK scientific community. HPCx is targeting an environment that will both result in world-leading science and address the challenges involved in scaling existing codes to the capability levels required.

The key goal is to enable complex and increasingly coupled and multi-disciplinary scientific applications to harness the potential performance of current and future high-performance parallel systems in order to meet critical scientific objectives. We have identified three key issues: management of the memory hierarchy, expression and management of concurrency and efficient sequential execution. These are key issues on all current systems.

We find that applications which scale best across all systems (CPMD, CRYSTAL, PCHAN and POLCOMS) are those where the developers have most successfully optimised the memory access patterns for multi-level cache architectures and maintained a high compute to communications ratio. There is no substitute for careful design of data structures, data access (loop indexing) and data distribution. For legacy codes this may require a complete re-design.

When comparing single processor performance we find that the IBM POWER4 1.3 GHz and the ES45/1000 of the AlphaServer are generally on a par and substantially outperform the older R12k 400 MHz and R14k 500MHz processors of the Origin. For the POWER4, a ratio of between 5 and 7 times the T3E is typical. On larger numbers of processors the network of the AlphaServer is seen to deliver scaling which is superior to that from IBM's Colony switch, though we note that the latter will soon be superseded by IBM's follow-on product.

It is the applications with unavoidable large-scale global communications (DL-POLY, GAMESS, H2MOL, NAMM and PFARM) which provide the most severe test for the communications sub-systems of these machines. With processor speeds continuing to increase faster than interconnect speeds, scalability is not likely to improve for algorithms which depend on collective, global operations.

We have made significant progress in optimizing a range of key user applications for the HPCx system. The initial benchmark results from this process and the performance levels achieved have highlighted a wide range of performance, with some algorithms scaling far better than others. Our focus on algorithm development and the drive to remove dependencies on collective, global operations have been successful in several cases in improving the scalability of these codes. Where all these issues have been addressed we find excellent levels of scalability and performance.

12. REFERENCES

- [1] *Computational Chemistry Applications: Performance on High-End and Commodity-class Computers* M. F. Guest and P. Sherwood, Proceedings of HPCS 2002, Moncton, Canada, 2002.
- [2] *Promise and Challenge of High-Performance Computing, with Examples from Molecular Modelling*, T.H. Dunning Jr., R.J. Harrison, D. Feller and S.S. Zanteas, Phil. Trans. Soc. A, **360-1795** (2002) 1079-1105.

- [3] *Global Arrays: a nonuniform memory access programming model for high-performance computers*, J. Nieplocha, R.J. Harrison and R.J. Littlefield, *J. Supercomput.* **10** (1996) 197-220.
- [4] *CRYSTAL 98 User's Manual*, V.R. Saunders, R. Dovesi, C. Roetti, M. Causa, N.M. Harrison, C.M. Zicovich-Wilson, University of Torino, Torino, 1998. <http://www.chimifm.unito.it/teorica/crystal>
- [5] *Ab Initio Modelling in Solid State Chemistry*, European Summer School, MSSC2002, 8-13 Sept. 2002, Torino, Italy, <http://www.chimifm.unito.it/teorica/mssc2002>.
- [6] *A New Implementation of a Parallel Jacobi Diagonalizer*, I.J. Bush <http://www.cse.clrc.ac.uk/arc/bfg.shtml>
- [7] *Accurate protein crystallography at ultra-high resolution: Valence-electron distribution in crambin*, C. Jelsch, M.M. Teeter, V. Lamzin, V. Pichon-Lesme, B. Blessing, C. Lecomte, *PROC.NAT.ACAD.SCI.USA* V. 97 3171 2000.
- [8] *Dual-level Parallelism for ab initio Molecular Dynamics: Reaching Teraflop Performance with the CPMD code*, J. Hutter and A. Curioni, IBM Research Report, RZ 3503 (2003).
- [9] *DL_POLY: A general purpose parallel molecular dynamics simulation package*, W Smith and T R Forester, *J. Molec. Graphics* **14** (1996) 136.
- [10] *DL_POLY: Applications to Molecular Simulation*, W. Smith, C. Yong and M. Rodger, *Molecular Simulation* **28** (2002) 385.
- [11] *Application Performance on High-end and Commodity-class Computers*, M.F. Guest, <http://www.ukhec.ac.uk/publications/reports/benchmarking.pdf>
- [12] *The DL-POLY Molecular Simulation Package*, W. Smith, http://www.cse.clrc.ac.uk/msi/software/DL_POLY/
- [13] *A smooth particle mesh Ewald method*, U. Essmann, L. Perera, M.L. Berkowitz, T. Darden, H. Lee and L.G. Pedersen. *J. Chem. Phys.* **103** (1995) 8577.
- [14] *Computer Simulation of Liquids*, M.P. Allen and D.J. Tildesley, Clarendon Oxford 1987.
- [15] *The Fastest Fourier Transform in the West*, M. Frigo, S.G. Johnson, MIT Technical Report, MIT-LCS-TR-728, Sept. 11, 1997. <http://www.fftw.org/>
- [16] *A Parallel Implementation of SPME for DL_POLY 3*, I.J. Bush and W. Smith, <http://www.cse.clrc.ac.uk/arc/fft.shtml>
- [17] *NAMD2: Greater scalability for parallel molecular dynamics*, L. Kalé, R. Skeel, M. Bhandarkar, R. Brunner, A. Gursoy, N. Krawetz, J. Phillips, A. Shinozaki, K. Varadarajan, and K. Schulten. *J. Comp. Phys.*, **151** (1999) 283-312.
- [18] *NAMD Performance*, Theoretical and Computational Biophysics Group, NIH Resource for Macromolecular Modeling and Bioinformatics, <http://www.ks.uiuc.edu/Research/namd/performance.html>
- [19] Dundas D, Meharg K, McCann J F and Taylor K T, 2003, *Euro. Phys. J. D*, in press.
- [20] *A parallel R-matrix program PRMAT for electron-atom and electron-ion scattering calculations*, A.G. Sunderland, C.J. Noble, V.M. Burke and P.G. Burke, *Comput. Phys. Commun.*, **145** (2002) 311.
- [21] *Parallel inverse iteration with reorthogonalization*, G. Fann and R.J. Littlefield in: *Sixth SIAM Conference on Parallel Processing for Scientific Computing (SIAM)*, (1993) pp.409-413; R.J.Littlefield, K.J. Maschhoff, *Investigating the Performance of Parallel Eigensolvers for Large Processor Counts*, *Theoretica Chimica Acta* **84** (1993) 457-473.
- [22] *PeIGS - Parallel Eigensystem Solver*, G. Fann, <http://www.emsl.pnl.gov/docs/nwchem/doc/peigs/docs/peigs.html>
- [23] *GAMESS-UK* is a package of ab initio programs written by M.F. Guest, J.H. van Lenthe, J. Kendrick, K. Schoeffel and P. Sherwood, with contributions from R.D. Amos, R.J. Buenker, M. Dupuis, N.C. Handy, I.H. Hillier, P.J. Knowles, V. Bonacic-Koutecky, W. von Niessen, R.J. Harrison, A.P. Rendell, V.R. Saunders, and A.J. Stone. The package is derived from the original GAMESS code due to M. Dupuis, D. Spangler and J. Wendoloski, *NRCC Software Catalog*, Vol. 1, Program No. QG01 (GAMESS), 1980.

- [24] M.F. Guest, E. Apra, D.E. Bernholdt, H.A. Fruechtl, R.J. Harrison, R.A. Kendall, R.A. Kutteh, X. Long, J.B. Nicholas, J.A. Nichols, H.L. Taylor, A.T. Wong, G.I. Fann, R.J. Littlefield and J. Nieplocha, *Future Generation Computer Systems* 12 (1996) pp. 273-289.
- [25] *A Parallel Second-Order Møller Plesset Gradient*, G. D. Fletcher, A. P. Rendell and P. Sherwood. *Molecular Physics*. **91** (1997) 431.
- [26] N. Godbout, D. R. Salahub, J. Andzelm and E. Wimmer, *Can. J. Chem.* **70**, (1992) 560.
- [27] *Performance and Experience with LAPI – a New High-Performance Communication Library for the IBM RS/6000 SP*, G. Shah , J. Nieplocha , J. Mirza, C. Kim, R.J. Harrison, R.K. Govindaraju, K. Gildea, P. DiNicola, C. Bender, Supercomputing 2002.
- [28] *ScaLAPACK Home Page*, http://www.netlib.org/scalapack/scalapack_home.html
- [29] *A Parallel Divide and Conquer Algorithm for the Symmetric Eigenvalue Problem on Distributed Memory Architectures*, F. Tisseur and J. Dongarra, , *SIAM J. Sci. Comput.* Vol. 20, No. 6, pp. 2223-2236.
- [30] *Application of a New Algorithm for the Symmetric Eigenproblem to Computational Quantum Chemistry*, I. Dillon, G. Fann, B. Parlett, Proceedings of the Eighth SIAM Conference on Parallel Processing for Scientific Computing, Minneapolis, March 1997.
http://www.cs.utexas.edu/users/inderjit/public_papers/fannchem1.ps.gz
- [31] *A New Implementation of a Parallel Jacobi Diagonalizer*, I.J. Bush,
<http://www.cse.clrc.ac.uk/arc/bfg.shtml>
- [32] *Linear Algebra*, Wilkinson and Reinsch, Vol. 2 of Handbook for Automatic Computation (1971)
- [33] *Direct Numerical Simulation of Shock/Boundary Layer Interaction*, N.D. Sandham, M. Ashworth and D.R. Emerson, <http://www.cse.clrc.ac.uk/ceg/sbli.shtml>
- [34] *Coupled Marine Ecosystem Modelling on High-Performance Computers*, M. Ashworth, R. Proctor, J.T. Holt, J.I. Allen, and J.C. Blackford in *Developments in Teracomputing*, eds. W. Zwiefelhofer and N. Kreitz, 2001, 150-163, (World Scientific).
- [35] *A highly spatially resolved ecosystem model for the North West European Continental Shelf*, J.I. Allen, J.C. Blackford, J.T. Holt, R. Proctor, M. Ashworth and J. Siddorn, *SARSIA* **86** (2001) 423-440.
- [36] *Generalized QR factorization and its applications*, E. Anderson, Z. Bai and J. Dongarra, *Linear Algebra and its Applications*, 162-164 (1992), pp. 243-273.

Acknowledgements

The success of the early stages of the HPCx project and of the implementation of the application benchmarks owes considerably to the efforts of many people. Among those without whom this work would not have been possible are Richard Blake, Paul Durham, Paul Sherwood, Bill Smith and Nic Harrison from CLRC, Mike Brown, Alan Simpson and Arthur Trew from EPCC, and Luigi Brochard and colleagues from IBM.

The HPCx service is funded by the Engineering and Physical Sciences Research Council on behalf of the UK Government Office of Science and Technology.

We are grateful to CSAR, ORNL, PSC and SARA for access to machines, and to referees, whose comments, we believe, have been responsible for significant improvements to the paper.