# Parallel Application Software on High Performance Computers

# I. The IBM SP2 and Cray T3D

**The CCLRC HPCI Centre at Daresbury Laboratory**

**Software Engineering Group**

**R.J. Allan and M.F. Guest (Eds.)**

**Edition 2: 8/8/1996**

Abstract

This report documents experience of porting widely used codes from a range of science and engineering applications to a parallel IBM SP2 computer installed at the CCLRC Daresbury Laboratory. Comparisons with the EPCC Cray T3D are made and there are many comments relevant to that system. Most of the codes were previously used on other systems, such as the Cray T3D, Intel iPSC/860 and Meiko CS-1 and are supported by staff at Daresbury. Performance of the SP2 nodes and parallel speedup are discussed along with ease of use and software quality. Some comparison is also made with other systems, both sequential and parallel.

**This report is available from http://www.dl.ac.uk/TCSC/HPCI/**

# EXECUTIVE SUMMARY

This report describes the development, optimisation and performance of application software supported at the CCLRC Daresbury Laboratory through the UK's High Performance Computing Initiative (HPCI) and the Collaborative Computational Projects (CCPs). These codes are the mainstay of a number of UK academic and industrial research groups. Part of the work undertaken by the CCLRC HPCI Centre involves benchmarking these applications to gain a deeper understanding of all aspects of performance for several parallel computer systems, with a particular focus on the 16-node IBM SP2 installed at the Laboratory, and on the 384-node Cray T3D at the Edinburgh Parallel Computer Centre (EPCC).

The IBM SP2 at the Daresbury Laboratory was installed in April 1995, so that the range of work described herein is the product of little more than six months of effort. This is a significant achievement in terms of the spread of scientific and engineering disciplines covered, the number of HPCI consortia codes running, and the performance attained, which is in many cases markedly superior to that achieved on equivalent configurations of the Cray T3D.

Early attention focused on a variety of topics including message passing, programming languages, run time performance, linear algebra and mathematical libraries, activities that led to a growing awareness of the capabilities of the machine. Particular emphasis was given to the development and performance of a number of parallel matrix diagonalisation routines and FFT algorithms. We have summarised in this report a wealth of data that has been used to aid our understanding of a variety of performance issues around both the IBM SP2 and Cray T3D.

Our experiences in the development and implementation across many computer architectures of a large number of application codes from the disciplines of Atomic and Molecular Physics, Computational Engineering, Computational Chemistry, Molecular Modelling and Molecular Simulation, Materials Science, and the Earth's Environment have been well chronicled throughout the years. This report presents the latest chapter in this work; we have described a variety of algorithmic developments and associated performance figures in each of these areas, with many comparisons drawn between the IBM SP2 and Cray T3D. Specifically:

1. In *Atomic and Molecular Physics*, we describe the Fortran-90 implementation and performance of the (e,2e) collision code, and an outline of the parallelisation strategy adopted in the SP2 and T3D versions of the MOLSCAT general purpose non-reactive scattering package. Also presented are performance figures for the Oxford Photoionisation code, developed within the Atomic Multi-photon Processes HPCI Consortium, and the Time-dependent Reactive Scattering program (TDRS) of the Chemical Reaction and Energy Exchange Processes Consortium.

2. In *Computational Engineering*, we consider computations that range from modelling of flames, involving multi-phase flows and huge ranges of temperature, to flow around complex bodies, such as in chemical reactors or over aircraft wings, to structural design. The applications described in this report include the ANGUS and WING3D codes, the former developed with the Computational Combustion HPCI Consortium, the latter by Dr. Badcock of the External Aerodynamics consortium. Development and performance of the Turbulence Transition Modelling code (from the Turbulence Transition consortium), TMCODE, is also considered, together with initial work in implementing the FLOW3D program, used to simulate high-speed air flow over a cavity, on the IBM SP2 and Cray T3D.

3. In *Computational Chemistry, Molecular Modelling and Molecular Simulation*, we discuss parallel developments and performance of five codes. Computational chemistry covers, of course, a wide spectrum of activities ranging from quantum mechanical calculations of the electronic structure of molecules, to classical mechanical simulations of the dynamical properties of many-atom systems, to the mapping of both structure-activity relationships and reaction synthesis steps. The emphasis in this report is on both quantum chemistry and molecular dynamics. Performance data in the former category includes that from the Density Functional and Coupled Cluster modules of the NWChem package (from Pacific Northwest National Laboratory, USA), and from the SCF, MP2-gradient and SCF-second derivatives modules of the Daresbury-developed GAMESS-UK code. Work in progress using the parallel HONDO program is outlined, and consideration given to initial parallel developments around the MOLPRO code. Performance benchmarks for the molecular simulation program, DL_POLY, are presented on both the IBM SP2 and Cray T3D. The GAMESS-UK, DL_POLY and the MOLPRO codes are of course used by a number of HPCI consortia.

4. In *Materials Science*, we describe work in progress using the LMTO and Self-interaction-corrected Local Spin Density (SIC-LSD) parallel codes. An outline is given of the parallelisation and performance of both the CRYSTAL periodic Hartree Fock program and the CETEP code, the latter used by the UK Car-Parrinello consortium. CETEP performance data is given on both the Cray T3D and IBM SP2. The capabilities and performance of the AIMPRO code, from the Covalently-bonded Materials HPCI Consortium, are described, and investigations of gold-hydrogen complexes in silicon, currently in progress on the IBM SP2, are outlined. Recent developments in the area of micromagnetism are then considered, with descriptions given of the parallel implementation and performance of codes from the Micromagnetism HPCI Consortium (the Sheffield and Oxford programs), together with the development of the MagFEM library and the Erlangen Benchmark.

5. In *Earth Sciences*, we highlight work with the Sea-Shelf Consortium (NERC) on the Proudman Oceanographic Laboratory's Water Quality Model (WQM) code, together with some preliminary performance data on the IBM SP2.

We are confident of building on the experiences of the last six months and anticipate significant advances in all the areas covered by this report throughout the coming year. There can be little doubt that the IBM SP2 at Daresbury Laboratory is positioned to provide a significant and growing contribution to the national provision of high performance computing resources. This assessment is based both on the relative ease of developing parallel applications, and on the levels of performance subsequently realised in production runs. Both attributes, crucial to the successful exploitation of any high performance computing resource, are amply demonstrated in the chapters of this report.

# Contents

# List of Tables

# Chapter 1

# Introduction and Summary

## 1.1 How to read this document

This report describes the development, optimisation and performance of a range of application software supported at the CCLRC Daresbury Laboratory through the UK's High Performance Computing Initiative (HPCI) and the Collaborative Computational Projects (CCPs). These codes are the mainstay of a number of UK academic and industrial research groups. As part of the work undertaken by the CCLRC HPCI Centre, the Software Engineering Group is benchmarking these applications to gain a deeper understanding of all aspects of performance for several parallel computer systems, with a particular focus on the 16-node IBM SP2 installed at the Laboratory, and on the 384-node Cray T3D at the Edinburgh Parallel Computer Centre (EPCC).

The body of this report is organised into seven chapters, together with a set of appendices grouped together in chapter 8. An outline of the role of the CCLRC HPCI Centre, together with summary information on the Daresbury IBM SP2, is presented below, following a summary of the chapters.

Chapter 2 provides an overview of the system attributes of the IBM SP2, with consideration given to a variety of topics including message passing, programming languages, run time performance, linear algebra and mathematical libraries. This chapter summarises a wealth of data that has been used to aid our understanding of a variety of performance issues around both the IBM SP2 and Cray T3D, and should ideally be read with reference to the appendices of the report that present this data in more detail.

An earlied edition of this report contained sections on matrix diagonalisation and FFTs. These are now available separately as fuller reports:

R.J.Allan and I.J.Bush *Parallel Application Software on High Performance Computers: Parallel Diagonalisation Routines* (Daresbury Laboratory, 1996)

R.J.Allan, I.J.Bush, D.Henty and T.Bush *Parallel Application Software on High Performance Computers: Serial and Parallel FFT Routines* (Daresbury Laboratory, 1996)

They are also visible on the WorldWide Web at URL *http://www.dl.ac.uk/TCSC/HPCI/reports/* .

The remaining chapters document our experiences in the development and implementation of a large number of application codes from the disciplines of Atomic and Molecular Physics, Computational Engineering, Computational Chemistry, Molecular Modelling and Molecular Simulation, Solid State and Materials Science and finally, the Earth's Environment. A variety of algorithmic developments and associated performance figures are presented in each of these areas, with many comparisons drawn between the IBM SP2 and Cray T3D.

Chapter 3 considers the area of Atomic and Molecular Physics, with a description of the Fortran-90 implementation and performance of the (e,2e) collision code, CDWBX, and an outline of the parallelisation strategy adopted in the SP2 and T3D versions of the MOLSCAT general purpose non-reactive scattering package. Also presented are performance figures for the Oxford Photoionisation program, developed within the Atomic Multi-photon Processes HPCI Consortium, and the Time-dependent Reactive Scattering code (TDRS) of the Chemical Reaction and Energy Exchange Processes Consortium.

Chapter 4 considers a variety of codes associated with the discipline of Computational Engineering, a major and growing activity at the Laboratory. The computations range from modelling of flames, involving multi-phase flows and huge ranges of temperature, to flow around complex bodies, such as in chemical reactors or over aircraft wings, to structural design. The applications described in this chapter include the ANGUS and WING3D codes, the former developed with the Computational Combustion HPCI Consortium, the latter by Dr. Badcock of the External Aerodynamics consortium. Development and performance of the Turbulence Transition Modelling program, TMCODE, is also considered, together with initial work in implementing the FLOW3D code, used to simulate high-speed air flow over a cavity, on the IBM SP2 and Cray T3D.

Computational Chemistry, Molecular Modelling and Molecular Simulation are considered in Chapter 5, with parallel developments and performance presented for five codes. Computational chemistry covers, of course, a wide spectrum of activities ranging from quantum mechanical calculations of the electronic structure of molecules, to classical mechanical simulations of the dynamical properties of many-atom systems, to the mapping of both structure-activity relationships and reaction synthesis steps. The emphasis in this chapter is on both quantum chemistry and molecular dynamics. Performance data in the former category includes that from the Density Functional and Coupled Cluster modules of the NWChem package (from Pacific Northwest National Laboratory, USA), and from the second derivatives module of the Daresbury-developed GAMESS-UK code. Work in progress using the parallel HONDO program is outlined, and consideration given to initial parallel developments around the MOLPRO code. Performance benchmarks for the molecular simulation program, DL_POLY, are presented on both the IBM SP2 and Cray T3D.

Chapter 6 turns to a consideration of the disciplines of Materials Science Following a description of work in progress using the LMTO and Self-interaction-corrected Local Spin Density (SIC-LSD) parallel codes, an outline is given of the parallelisation and performance of both the CRYSTAL periodic Hartree Fock program and the CETEP code, the latter used by the UK Car-Parrinello consortium. CETEP performance data is given on both the Cray T3D and IBM SP2. The capabilities and performance of the AIMPRO code, from the Covalently-bonded Materials HPCI Consortium, are described, and investigations of gold-hydrogen complexes in silicon, currently in progress on the IBM SP2, are outlined. Recent developments in the area of micromagnetism are then considered, with descriptions given of the parallel implementation and performance of codes from the Micromagnetism HPCI Consortium (the Sheffield and Oxford programs), together with the development of the MagFEM library and the Erlangen Benchmark.

Finally, in Chapter 7, an overview is presented of the Proudman Oceanographic Laboratory's Water Quality Model (WQM) code, together with some preliminary performance data on the IBM SP2.

The codes described in the ensuing chapters have, in many cases been run by their authors or staff of the Theory and Computational Science (TCS) Groups to whom we remain indebted.

## 1.2 The CCLRC HPCI Centre

EPSRC has established three HPCI Centres, at Edinburgh (EPCC), Southampton and Daresbury Laboratory (DL), to offer wide-ranging support to consortia in the areas of parallel implementation, optimisation and applications development. The remit of the HPCI Centres is to provide support for

- Scientific consortia
- New applications
- New and advanced programming environments
- Matching of systems/methods to applications
- Access to software
- Access to novel HPC hardware
- Contact with industry
- Publishing information
- Education and training.

Some 75% of the DL Centre's effort continues to be devoted to the work of the HPCI consortia. There are twelve consortia formally associated with the Centre, of which five are *directly-supported* and seven *indirectly-supported*. The *directly-supported* consortia/projects are

- Computational Combustion for Engineering Applications
- Chemical Reactions and Energy Transfer Processess
- Large Eddy Simulation of Complex Engineering Flows at high Reynolds Numbers
- U.K. Car-Parrinello
- Macromolecular Modelling.

The *indirectly-supported* consortia/projects include

- External Aerodynamics
- Sea Shelf Hydrodynamic Modelling
- Transition Modelling
- Micromagnetism
- Atomic Multi-photon Processes
- Fundamental Electron-collision Processes
- *Ab initio* Simulation of Covalent Materials.

The Centre has three fixed-term RA personnel working full time with the consortia, Ian Bush, Peter Lockey and Kevin Maguire. In addition, a number of scientific advisors within TCS have been identified.

Our objective is to assist these consortia to develop state-of-the-art computer codes in leading edge science and engineering. We also aim to encourage new computational areas, which could initially make use of diverse HPC facilities such as distributed workstations, vector supercomputers or medium-sized novel architecture systems such as the IBM SP2. Some of the codes reported in this document are of this kind, or are numerical kernels which may be used for such developments.

Consortia supported at other centres are, at Edinburgh

- The Virgo project
- Lattice QCD consortium
- Human system modelling
- Colloid hydrodynamics
- Quantum Monte Carlo calculations for real solids
- Modelling vegetation-climate interactions

and at Southampton

- Materials Chemistry
- Global Ocean circulation
- UK Global Atmospheric modelling programme (UGAMP)
- Simulation and statistical mechanics of complex flows
- Direct numerical simulation of fluid flow
- UK particle cosmology consortium

Of these the Materials Chemistry and Direct Numerical Simulation consortia also rely on computer codes supported at Daresbury.

## 1.3 The Daresbury IBM SP2

The IBM SP2 at Daresbury consists of 14 thin node 2 (TNT) processors, each having 64 Mbytes of memory and 2 wide nodes, each with 128 Mbytes of memory. All nodes have 128 Kbytes of level 1 data cache and 32 Kbytes of instruction cache. The SP2 nodes are standard Power2 Architecture RS/6000 processors having a 66.7 MHz clock and a peak performance of 267 million floating point operations per seconds (MFLOPS).

The interconnect between the nodes is a high performance switching network with low latency and capable of sustaining a high bandwidth. In addition each node has an Ethernet connection and the two wide nodes an FDDI connection. All nodes have fast wide SCSI II discs attached for scratch space and access to permanent disc store is via NFS, using the wide nodes and FDDI connections. The wide nodes are used for login, editing and compilation as well as computation.

The nodes are normally partitioned into two pools:

- *Pool 1* consists of 8 TN2 nodes for dedicated single-user access;

- *Pool 2* consists of the 2 wide nodes and 6 TN2 nodes and provides shared multi-user access.

Dedicated access to all 16 nodes is provided once a week, by means of a locally developed booking system.

# Chapter 2

# Message Passing, Languages and Mathematical Libraries

## 2.1 Summary of Message-passing Performance

*R.J. Allan*

The performance of the message-passing system can be characterised using the parameters of Hockney and Carmona [21], namely

- $r_\infty$, the asymptotic transfer rate in MB/s and
- $n_{\frac{1}{2}}$, the message length at which half this rate is achieved.

A variety of message passing harnesses have been investigated, including MPL, MPI, PVM3, PVMe, TCGMSG, GA Tools, HPF and BSP. MPICH (Argonne National Laboratory, USA) has also been tested but since a bug was discovered no results can be reported. A full description of these tests are presented in Appendix II; at this point we merely summarise these results through the tables below, which collate the Hockney and Carmona parameters for the different software and networks investigated.

These parameters will be used throughout this document in our discussion of the performance of each application and, together with the sequential node performance of the machine, are critical in an overall characterisation and parallel scaling model.

The GA-Tools developed by Robert Harrison and Rick Littlefield at Battelle Pacific Northwest National Laboratory, USA [39], are available on both the T3D, SP2 and other systems at Daresbury. They provide an easy interface to virtual shared memory for the storage and asynchronous access of lare matrices. Some global operations are provided too. At this point we refer the reader to a rather thorough report by Howard Pritchard [93] that benchmarks most functionality of the GA-Tools on a Cray T3D, Intel Paragon and IBM SP1 and SP2. Some uses of GA-Tools are described in Chapter 5.

Other software and detailed performance tests are described in Appendix II of the current report.

Table 2.1: The IBM SP2 †

| software | protocol | $r_\infty$ | $n_{\frac{1}{2}}$ | latency [microsecs] |
|---|---|---|---|---|
| MPL | ip | 8.38 | 3037 | 270 |
| MPL | us | 35.2 | 3032.3 | 86 |
| MPI | ip | 6.26 | 4727 | 656 |
| MPI | us | 34.8 | 3966 | 114 |
| PVMe | us | 34.14 | 18297 | 535 |
| tcgmsg v4.01 | us | 34.8 | 3062 | 88 |

† We do not differentiate between wide and thin nodes as they are essentially the same.

Table 2.2: The Cray T3D

| software | protocol | $r_\infty$ | $n_{\frac{1}{2}}$ | latency [microsecs] |
|---|---|---|---|---|
| shmem_put | shmem | 120.2 | 780 | 6 |
| shmem_get | shmem | 58 | 362 | 6 |
| PVM | shmem | 25.7 | 3357 | 138 |
| TCGMSG | shmem | 47.5 | 1729 | 36.4 |
| MPI (EPCC) | shmem | 50.4 | 2662 | 50 |

Many of these tests were carried out using an 'unsupported' version of the MPI (Message Passing Interface) environment. Its point-to-point bandwidth is satisfactory, as reported in the chapter on message-passing, since it is built on top of MPL. However we await the supported IBM release of the standard MPI software with the release of Parallel Environment V2.1.

In practise, a number of machines are still lacking efficient MPI routines, including the SP2. This is particularly important for some of the applications requiring global reduction routines, e.g. implicit solvers. Only naive implementations of global operations are available in most cases in order to achieve the functionality of the standard as early as possible. To overcome this problem, some applications are still using hand-written communication routines, for this purpose. The full (and very rich) MPI functionality for global operations will be used when it is available in an efficient form.

## 2.2 Programming Languages

A number of HPCI consortia are developing, or will be developing, new codes to take advantage of the Cray T3D machine. The vast majority of codes running at present on the T3D are written in Fortran. It is therefore natural to use the current international standard (ISO/IEC 1539:1991), so called Fortran-90. Fortran-90 has a number of important features which make it far superior to any previous Fortran and is, in fact, better than C for many scientific applications.

For this development work we are currently using the 16-processor IBM SP2 machine at Daresbury, which permits shared access to nodes for test purposes and has a full optimising Fortran-90 compiler in common with all IBM RS/6000 workstations including the PowerPC-based systems. In addition to this the xldb visual debugger makes software development a particularly quick operation.

As an example we cite in Chapter 3 the CDWBX code, which is used by CCP2, members of the

"Fundamental Atomic Physics" consortium and a European HC&M consortium for "Coincidence studies of electron and photon impact" [80].

The compiler at present available on the T3D does not fully support the standard. Whilst a number of the new features are included they are either only partially implemented (e.g. array syntax), or differ from the standard in detail making codes non-portable (the MINLOC intrinsic function is an example). Much documentation searching is required to discover what is implemented, and to what degree, and whether the Cray definition is standard compliant. We have been told that another release of the compiler is due, which has more features implemented. We, and the consortia we work for, will continue to develop our codes on the IBM SP2 in the hope that we will soon be able to run them on the Cray T3D.

## 2.3 Run Time Performance

Some difficulty was experienced in obtaining repeatable execution times for a number of codes, especially in the area of computational chemistry and materials structure where large memory and temporary disc requirements coexist. It appeared at first that running on a dedicated system gave different times on different days. A lengthy investigation was undertaken using CRYSTAL as the typical application. Different message-passing interfaces (IP or US) were used and the application was monitored closely during a number of runs on the dedicated system.

Several reasons for this became apparent as the benchmarks were studied. Discussion with Dr. Ron Bell (IBM, Bedfont Lakes) was useful to eliminate some initial ideas. In particular it was understood that no nfs-mounted discs should be used as the response of nfs is unpredicatable. Therefore all temporary files and executables were moved onto the local scratch disks (a copy of the executable being made on every one). This could be done with a typical LoadLeveler script as follows:

**LoadLeveler script**

```
#@ executable = /usr/bin/poe
#@ arguments = myscript -euilib us
#@ min_processors = 4
#@ max_processors = 4
#@ output = output.txt
#@ error = error.txt
#@ job_type = parallel
#@ requirements = (Adapter == "hps_user" && Pool == 3)
#@ class = book
#@ queue
```

**myscript**

```
cd /scratch/rja
cp /h/rja/peigs2.1/example/geneig .
./geneig
```

After doing this we observed two things. Firstly `stderr` and `stdout` will still use the NFS mounted `/h` partition for the files `error.txt` and `output.txt`. This is assumed to be a negligible effect on the timing. Second, and far more important, the executable was paging.

We therefore statically linked the US library using the `-us` flag on `mpxlf` to see if this was contributing to the paging. By doing this the size of the `bss` section increased by 16Mbytes. A similar effect was also seen with the IP library but this time the `bss` increased by 32Mbytes. IBM UK informed us that this was due to 8K send and receive system buffers,(16K in te case of IP), being allocated by a executable process for every other node it could possibly communicate with i.e. the maximum number of nodes in an SP2 is 1024, making the total system buffer space 1024*16K. Although, in theory, this implies a user's application increases by 16Mbytes when statically linking the US, in practise the executable will only use 16K times the number of nodes requested by the user e.g. an 8 node job would use an additional 128K of memory for system buffer space using the US protocol. Discussion on this issue is continuing with IBM.

We therefore suspect the unpredictable timings were caused by the system paging virtual memory onto the same physical disc as that used by the executables for temporary file storage. A single disc cannot keep up the required throughput for any part of the code doing heavy i/o while paging is taking place - exactly as first seen.

We suggest paging space and scratch space should therefore be on physically separate discs, allowing the SCSI-2 controller to provide the necessary throughput. We are confident that many of the benchmark results reported in this document can be significantly improved (perhaps by a factor of 2) if this action is carried out.

## 2.4 Linear Algebra and Mathematical Libraries

*R.J. Allan*

Many scientific and engineering applications perform numerically intensive calculations relying heavily on vector and/or matrix operations. As a result the Basic Linear Algebra Subprograms library (BLAS), the Engineering Scientific Subroutine Library (ESSL) and the FORTRAN-77 equivalent were benchmarked on a single node on the IBM SP2 as well as on the Cray T3D and Intel iPSC/860.

One of the fundamental requirements in realising the potential of past and present generations of high performance computers is the availability of efficient implementations of the Basic Linear Algebra Subprograms, whether these machines be based on vector or scalar pipelined (usually RISC) processors. The BLAS represent a fundamental measure of the 'computational accessibility' of the system. One imagines that the aim of compiler writers must be to approach the BLAS performance as closely as possible by loop unrolling and re-scheduling operations. Indeed the matrix notation and intrinsics which are a fundamental part of the Fortran-90 standard enable a one-to-one mapping of source code onto the BLAS library.

This is of course not easy. Details of the computer architecture including cache and memory system, registers and floating point pipelines need to be considered.

It is of course impossible to present a complete analysis of BLAS performance, and much depends on the capability of the library's author. This is well exemplified on the IBM SP2, where two such libraries are available. The first, `libblas.a`, was simply compiled from optimised Fortran-77 code and while now obsolete, is unfortunately still shipped with the systems. The second, `libessl.a`, is highly optimised using assembly code and full cache management in sequencing of instructions. A similar set of routines, part of `libsci.a` exists on Cray systems, including the T3D.

Attempts to rationalise and explain performance aspects of the IBM POWER2 architecture exist on the Internet at URL *http:/www.austin.ibm.com/tech/p2ppc_tech.html* for instance. Performance tuning manuals may also be consulted for all machines.

Three classes of BLAS exist and encapsulate different amounts of computation to memory access.

- **Level-1**: scalar and vector operations (Lawson, Hanson, Kincaid and Krogh ACM Trans. Math. Software 5 (1979) 308-23).
- **Level-2**: matrix-vector operations (Dongarra, Du Croz, Hammarling and Hanson ACM Trans. Math. Software 14 (1988) 1-32).
- **Level-3**: matrix-matrix operations (Dongarra, Du Croz, Duff and Hammarling ACM Trans. Math. Software 16 (1990) 1-28).

A useful list of all the BLAS routines is given in appendix A of the book by Freeman and Phillips [99]. In general level-3 BLAS are able to make best use of the processor for a given amount of memory access as explained below. Since the latter is usually the limiting factor, level-3 BLAS are expected to offer the best performance of any code on a given computer.

We shall see that memory access plays a crucial rôle in cache-based systems and, in particular, the lack of a set associative cache on the T3D, leads to bottlenecks, even though the DEC Alpha EV4 processor can support a full cache system. The on-chip cache of the EV4 is direct mapped to memory by physical address in the Cray T3D so that extreme care has to be taken in placing vectors into storage so that their addresses do not map to the same cache line, as this would lead to continual cache thrashing. The T3D data cache is 8 kB made up of 256 lines of 32 B each (i.e. four DP numbers per line). One word can be loaded into registers per clock cycle through a load/store pipe 3 cycles deep. A cache miss results in a 24 cycle delay (39 cycles if a page miss) to get the next 4 words from main memory into cache. Memory is divided into two banks of pages, each with a select register. The 64-bit CPU is clocked at 151 MHz and has a six-stage add/multiply pipe which can deliver one FP op per cycle when fully loaded, so a peak of 151 MFlops could be achieved.

A 32 byte 'read-ahead' buffer is provided on the T3D as an elementary second-level cacheing mechanism so speed up loads in such situations. In practice this is what limits the performance. It is however capable of delivering into the cache in 12 cycles. 9 cycles are required to read 4 words into the buffer from main memory (24 for a page miss). Four similar write buffers exist. In the Cray T3E EV5 architecture this concept has been formalised into the idea of 'streams' and six such buffers are provided together with a set-associative cache. Workstations using the EV4 processor normally have a full 256 kB second level cache. We will not further consider these buffers.

The IBM SP2 with RS/6000 POWER2 superscalar CPU has a four-way set-associative dual-ported cache with load and store pipes controlled by the two fixed-point FXU processors. It is possible to have cache with 32 Byte data paths, so in total 4 DP words can be loaded and 4 stored in one clock cycle. The 4-way associativity means four banks of memory can be pointed to from a relative base address and locations incremented without the need to re-compute addresses. In the Daresbury SP2 however only two memory banks are currently installed per node. It has 1024 cache lines each of size 128 bytes holding 16 double precision numbers each. The memory bus width and load and store pipe width is 128 bits or four 32-bit words. For this reason the load and store operations are referred to as Quads. For our purposes however two DP numbers can be loaded simultaneously in each pipe. There are also two FP processors (FPU) each having a pipeline able to do a multiply and add in two cycles called an FMA. Effectively two multiplies and two adds can be performed by the CPU each cycle providing operations and loads and stores can be scheduled by the compiler. The clock rate of the CPU is 66.7 MHz giving a potential peak of 267 MFlops. 15-20 cycles are required to recover from a cache miss.

The Intel iPSC/860 has i860 long instruction word CPUs with a simple direct-mapped 8 kB on-chip data cache. 128 bits, or two DP numbers can be loaded into registers per cycle. An adder pipeline is able to do one DP add per cycle and an additional multiply pipeline is able to do one DP multiply per 2 cycles (or one single precision multiply per cycle). Since the CPU is clocked at 40 MHz a potential peak performance of 60 MFlops is possible. It however takes four cycles to start a multiply and three cycles to start an add. Access to main memory is somewhat complicated. Roughly 14 cycles are required to recover from a cache miss but then 32 Bytes become available. This however depends on the CPU state. Additional pipelined load instructions (which bypass the cache) can also be used taking 2 cycles. We will not discuss issues with the long instruction words (so-called dual mode).

The node architecture of these computers is parameterised by three numbers

1. CPU maximum performance if all FP pipes operational with on-chip cache
2. CPU to cache memory access rate (on chip and/or off chip)
3. CPU to main memory access rate with no page misses
4. CPU to main memory access rate with paging if there are pages (although no code of course pages continuously)

We summarise performance of the 3 classes of BLAS below, where we will analyse performance compared to items 1-3 above, quoting the percentage of (1) achievable as a means of normalisation. The raw performance of items 1-4 is shown in Table 2.3; 64-bit operations are assumed and this is classed as one word for the purpose of this section.

Table 2.3: CPU performance characteristics, all ops per clock cycle

| | CPU Operations | Cache Words | Main Memory Words | Page Miss Words |
|---|---|---|---|---|
| Cray T3D EV4 | 1+ or 1× | 1 | 4/24 | 4/39 |
| IBM SP2 TNT | 2+ and 2× | 4 | approx. 4/15 | approx. 4/35 |
| Intel iPSC/860 | 1+ and 1/2× | 2 | approx. 4/14 | no pages |

It can be seen that in the worst-case alignment abysmal performance can be obtained on any machine.

### 2.4.1 Level-1 BLAS

Level-1 BLAS are typified by the DDOT and DAXPY operations (Table 2.4). The first is just the familiar scalar product which requires one multiplication and one addition for every two operands per iteration until the length of the vector is reached. The dot product is however conceptually sequential since for each iteration the product must be added to the result of the previous iteration. Only one FPU of the POWER2 CPU can therefore be active. At each iteration one load Quad, and one FMA are generated plus a branch. However to accumulate the result the FMA has a latency of two cycles, so only one result will be produced every two cycles resulting in a peak speed of 66.7 MFlops. This is not changed by loop unrolling, however a sophisticated implementation would do unrolled partial summations to a depth of four to achieve up to 4 FMAs in 3 cycles (because the next branch will then be outside the instructions gathered). This leads to 8/3*66.7 = 177.9 MFlops. Actually due to other effects (register renaming) only 106.7 MFlops is possible.

Reading two vectors into cache on the T3D (assuming no conflicts) would prevent the register-load pipeline from working, since corresponding elements of the vectors are not in consecutive cache locations. Six cycles would then be needed for two FP ops yielding only 50 MFlops. On the T3D it would be necessary to interleave the two vectors in cache to effectively pipeline the register loads. If this is done 2 loads will take two cycles during which time a multiply and an add can be performed. This is not done in practice, although loop unrolling (usually to a maximum depth of 8 to prevent overflowing the 32 available FP registers) would allow 8 entries from each vector to be loaded in 18 cycles. Floating point multiplies cannot start however until cycle 12 and the FP pipeline has a 5-cycle startup delay. The same applies for the final multiplications. A total of 16 FP operations could therefore be performed in 37 cycles giving a performance of 16/37*151=65.3 MFlops.

On the Intel i860 similar techniques can be used with two vectors stored in two halves of the cache. Care must be taken with physical addresses to avoid thrashing as on the Cray. A FP multiply can then be performed every 2 cycles with an intervening add of the previous result and 40 MFlops should result.

The DAXPY operation benefits from simple loop unrolling and on the POWER2 can be reduced to four load Quads, four FMAs, two store Quads and a branch per iteration. These take 3 cycles because of the branch and again result in 177.9 MFlops theoretical peak.

On the Intel the situation is the same as for a dot product except that the result must be stored (in cache) so two FP ops will now take 4 cycles leading to a peak of 20 MFlops.

Unrolling to a depth of 8 on the Cray T3D enables multiplication by the constant (assumed in a register) to start on cycle 4 and addition of the second vector and storing of results on cycle 17. Writing back results would finish 2 cycles late, so 31 cycles would be required for 16 FP operations yielding a performance of 16/31*151=77.9 MFlops. However in practice it is limited by access from the main memory and, unrolling to a depth of 4, generates 8 loads, 4 stores and 8 flops for each 4 iterations. This results in 8 FP ops in 39 cycles resulting in 31 MFlops.

None of the theoretical peak speeds are observed, most probably because of difficulties in keeping the cache correctly filled since no data can be re-used. Memory to cache latency then begins to dominate.

It is interesting to note that Fortran code outperforms than ESSL on this.

Table 2.4: Level-1 BLAS: DDOT and DAXPY Performance

| | SP2 | T3D | iPSC/860 |
|---|---|---|---|
| DDOT | 12.6 | | 17.6 |
| DDOT initialised | 46.4 | 33.5 | 17.6 |
| % of theoretical peak achieved | 43.5 | 51.3 | 44.0 |
| DAXPY | 12.4 | | 10.2 |
| DAXPY initialised | 34.7 | 29.3 | 10.2 |
| % of theoretical peak achieved | 19.5 | 94.5 | 51.0 |

### 2.4.2 Level-2 BLAS

The level-2 BLAS, e.g. DGEMV, typically require one multiplication and one addition for each two data words fetched per iteration. The speed is simply proportional to the speed of data access. This behaviour can be kept up for the length of the vector only before a result must be stored and the pipelining will probably stall. However the pipelining could be kept going and the vector (or part of it) held in cache while the next part of the matrix is loaded. A new portion of the left hand matrix must be gathered by a strided access to main memory. Results are shown in Table 2.5.

If loop unrolling is used each vector element can be multiplied simultaneously by a number of matrix row elements and partial sums collected.

On the RS/6000 POWER2 systems, unrolling to a maximum depth (actually 24 because there are only 32 FP registers) could enable about 96% of the peak performance, and 80% when the matrix does not all fit in cache. Cache prefetching using the spare capacity of one FXU must be done to achieve this.

On the Cray and Intel systems loop unrolling can achieve roughly one (pipelined) load per FP operation and will run at 151 MFlops on the T3D and 40 MFlops on the Intel. A detailed analysis would be needed to find the correct theoretical peak. On the T3D a theoretical figure has been quoted including access to external memory of 50 MFlops, which is however exceeded in the example given below.

Table 2.5: Level-2 BLAS: DGEMV Performance

| | SP2 | T3D | iPSC/860 |
|---|---|---|---|
| DGEMV | 24.0 | | 26.5 |
| DGEMV initialised | 95.9 | 54.7 | 26.6 |
| % of theoretical peak achieved | 44.9 | 36.2 | 66.5 |

### 2.4.3 Level-3 BLAS

Level-3 BLAS are expected to perform better than level-2, see Table 2.6, again because of the greater re-use of data in registers. Depending on the size of the problem, FP operations can be kept up for a long time before a result needs to be stored at which point the pipelining will probably stall. A whole row of the right-hand matrix (or part of it) can be kept in cache, allowing update of a portion of the right hand matrix with sequential main memory access.

The level-3 BLAS, for instance DGEMM, typically require one multiplication and one addition for each two data words. On the RS/6000 systems, with 2 FMAs and 2 Quad loads every clock cycle to achieve peak performance. Loop unrolling allows the branch time to be absorbed and also helps with data loading into cache making the case simpler than the level-2. The speed is simply proportional to the speed of data access.

Similar arguments to the above apply to the Intel and Cray. A quoted peak of 120 MFlops applies to the Cray.

Table 2.6: Level-3 BLAS: DGEMM Performance

| | SP2 | T3D | iPSC/860 |
|---|---|---|---|
| DGEMM | 233.0 | | 36.7 |
| DGEMM initialised | 242.0 | 107.9 | 36.7 |
| % of theoretical peak achieved | 90.6 | 89.9 | 91.8 |

We can see that the systems are now performing at a high fraction of their theoretical peak for this ideal case. Clearly any application codes should aim to use matrix-matrix operations as much as possible. Finally, we are not certain why the level-1 and level-2 BLAS do not perform better on our SP2, since higher speeds have been reported on other systems.

# Chapter 3

# Atomic and Molecular Physics

## 3.1 The CDWBX code.

*R.J. Allan*

### 3.1.1 Parallel Programming with Fortran-90

The CDWBX program [80] is a development of the original DWE2E program for (e,2e) collisions to incorporate any general single-centre spherical-harmonic operator expansion. It uses BLAS to compute the 2-dimensional gaussian quadratures with reduction of the multiplications by pre-computing the inner integral in the external angular-momentum loops. A $960 \times 960$ grid is used.

Much of the code consists of a large set of nested loops over the angular momentum decomposition with special functions being called in the interior to evaluate Wigner recoupling coefficients, Bessel functions etc. IF tests are used to decide on contributions to singlet/triplet and direct/exchange terms inside calls to the quadrature routines.

Initial wavefunction generation by the SUBROUTINE DWFN is performed using tabulated SCF bases duplicated on all processors, but is a relatively small overhead. A 4000 point grid is used with a Numerov ODE solver. The angular momentum loops are computed in SUBROUTINE CDWBX on different processors with each processor storing intermediate integrals in tables to avoid redundant computation. A global reduction summation is required to obtain the geometry-dependent transition amplitudes. Angular terms are computed in the inner loop if the operator is not angle-dependent, or parallelised over if it is.

The code is written in structured Fortran-90 with some limited use of array intrinsics and direct calls to BLAS. MPI is used for parallel environment calls and reduction operations. Input of small data files are performed on all nodes, but output of results is restricted to node 0.

Compilation is performed using the command line:

```
mpxlf90 -O3 -qhot -qarch=com -qnosave -I../dwe2e -c -ip < ... >
```

and the linker uses library definitions

```
-lblas $(MOD)mpi.o -L/usr/local/mpi/lib/RS6K -lmpi
```

Comparing the older version of the DWE2E code before and after converting to Fortran-90 yields the following execution times (seconds) for dest deck (2) on an RISC/6000 model 370.

| DWE2E F77 | DWE2E F90 | CDWBX F90 |
|---|---|---|
| 410.9 | 491.5 | 1040.2 |

It is difficult to know the true cause of the increased execution time for DWE2E. Some re-coding has obviously occurred as well as use of dynamic memory. Some data copies have also had to be made in interfacing routines where pointers to array sections were not accepted (e.g. in interfacing to the Fortran-77 MPI library and BLAS).

Finally, although the new algorithm is slower it permits considerably more flexibility in the form of correlation in the interaction operator and wavefunctions used (see Franz and Altick [17]). It is also formulated to use the BLAS.

### 3.1.2 Test Jobs and Results

Three test examples have been used to illustrate performance. Test job (1) is a small case, asymmetric H(1s) scattering with explicit exchange, involving a total of 1680 integrals calculated, 17128 duplicated and 1016 cut off (see Table 3.1).

The second case is somewhat larger, symmetric coplanar Ar(3p) scattering with implicit exchange, and involves a total of 30795 + 30795 integrals calculated, with 1,025760 + 1,025167 duplicates and 279212 + 279806 cut off (see Table 3.2).

The final example, Test job (3) is for a large(ish) case, asymmetric coplanar Xe(5p) scattering with exchange. A total of 449535+449236 integrals are calculated, with 28,525,547 + 28,373,203 duplicated and 2,264326 + 2,265965 cut off (see Table 3.3).

Table 3.1: CDWBX: Test Job 1

| Platform | dwfn Time | cdwbx Time | Total Time | Speedup vs. PowerPC/250T | Efficiency |
|---|---|---|---|---|---|
| PowerPC/250T | 6.59 | 54.37 | 60.96 | 1 | |
| IBM RS/6000-370 | 1.35 | 21.86 | 3.21 | 2.6 | |
| IBM SP2/1 node | 1.39 | 12.57 | 14.05 | 4.3 | 1.0 |
| IBM SP2/2 nodes | 1.30 | 6.80 | 8.10 | 7.5 | 1.7 |
| IBM SP2/4 nodes | 1.50 | 4.41 | 5.91 | 10.3 | 2.4 |

The constancy of column 2 (dwfn time) provides a measure of activity on pool 2.

Clearly for the large case (Table 3.3) the parallel efficiency is almost 100% and the apparent super-linear speedup may be attributed to memory access.

Table 3.2: CDWBX: Test Job 2

| Platform | dwfn Time | cdwbx Time | Total Time | Speedup vs. PowerPC/250T | Efficiency |
|---|---|---|---|---|---|
| PowerPC/250T | 6.35 | 1644.0 | 1650.4 | 1.0 | |
| IBM RS/6000-370 | 4.36 | 1035.9 | 1040.2 | 1.6 | |
| IBM SP2/1 (ip) | 3.37 | 623.38 | 626.75 | 2.6 | 1.0 |
| IBM SP2/2 (ip) | 3.33 | 324.19 | 327.51 | 5.0 | 1.9 |
| IBM SP2/4 (ip) | 3.34 | 163.98 | 167.33 | 9.9 | 3.8 |
| IBM SP2/8 (ip) | 3.43 | 90.12 | 93.55 | 17.6 | 6.7 |
| IBM SP2/1 (us) | | 610.50 | 613.40 | 2.7 | 1.0 |
| IBM SP2/2 (us) | 2.71 | 298.62 | 301.33 | 5.5 | 2.0 |
| IBM SP2/4 (us) | 2.93 | 143.56 | 146.49 | 11.3 | 4.2 |
| IBM SP2/8 (us) | 2.88 | 79.51 | 82.39 | 20.0 | 7.4 |

Table 3.3: CDWBX: Test Job 3

| Platform | dwfn Time | cdwbx Time | Total Time | Speedup vs. PowerPC/250T | Efficiency |
|---|---|---|---|---|---|
| PowerPC/250T | 19.92 | 17613. | 17633. | 1 | |
| IBM RS/6000-370 | 10.00 | 9417.8 | 9427.8 | 1.9 | |
| IBM SP2-1 | 7.69 | 6581.9 | 6589.6 | 2.7 | 1 |
| IBM SP2-2 | 7.80 | 3028.2 | 3036.0 | 5.8 | 2.2 |
| IBM SP2-4 | 7.76 | 1558.4 | 1566.2 | 11.3 | 4.2 |
| IBM SP2-8 | 8.16 | 813.7 | 821.9 | 21.4 | 8.0 |

## 3.2 MOLSCAT

*I.J. Bush and J. Hutson*

MOLSCAT is a general purpose non-reactive scattering package, developed by S. Green and J. Hutson [22]. It may be used to calculate a number of properties of molecular collisions, such as the full S matrix and state-to-state cross sections, for a wide variety of collision partners, and within a number of different approximations. Part of the full package has been parallelised by I.J. Bush [6], and it is this parallel version that has been run on both the SP2 at DL and the T3D at EPCC.

The parallelisation strategy is driven by the need to store a large coupling constant hypermatrix, which for large cases is distributed. However rather than distribute the matrix across all nodes, for each angular momentum and parity the minimum number of nodes required to store the matrix is calculated, that number of idle nodes is found by a master process and the hypermatrix is calculated and distributed across those nodes. These nodes then propagate the wavefunction for the different energies. The resulting code scales well with the number of nodes for jobs consisting of large numbers of angular momenta and energies, which is typical of production work. However there are load balancing problems for small jobs.

The test problem was the collision of two hydrogen fluoride molecules, HF-HF. Four energies, one total angular momentum and four parities were covered. The code was compiled with the -O3 flag on the SP2, both with and without the KAP preprocessor, and with default optimisation on the Cray T3D. The timings are

Table 3.4: MOLSCAT on the IBM SP2 - No Pre-processing

| Nodes | Time | Speed Up |
|---|---|---|
| 1 | 388.0 | — |
| 3 | 150.5 | 2.6 |
| 4 | 104.8 | 3.7 |
| 7 | 77.9 | 5.0 |
| 8 | 64.4 | 6.0 |

Table 3.5: MOLSCAT on the IBM SP2 - With Pre-processing

| Nodes | Time | Speed Up |
|---|---|---|
| 1 | 370.3 | — |
| 2 | 190.4 | 2.0 |
| 4 | 98.5 | 3.8 |
| 8 | 62.6 | 5.9 |

Table 3.6: MOLSCAT on the Cray T3D

| Nodes | Time | Speed Up |
|---|---|---|
| 1 | 1350.6 | — |
| 3 | 521.8 | 2.6 |
| 7 | 271.7 | 5.0 |

In the above the node column does not include the master process, i.e. it is the total number of calculating nodes.

The deviation from linearity in the speed up figures is due to the job being quite small. For the larger numbers of nodes each process is typically propagating for one angular momentum, and so differences in times can have a large impact on the scaling of the job.

The figures show that an SP2 node is around three and a half times quicker than the T3D.

## 3.3 The Oxford Photoionisation Code

*I.J. Bush and A. Sanpera*

This code was developed by I.J. Bush in collaboration with A. Sanpera of the Atomic Multi-Photon Processes consortium. It calculates the time evolution of the wavefunction of a laser irradiated atom, within the single active electron approximation.

The kernel of the code involves the solution of a large (order 120,000), but sparse, set of linear simultaneous equations. The algorithm used for this is the conjugate gradient (CG) method.

The code has been tested on both the T3D and the SP2. Timings are as follows:

| Machine | Nodes | Time (seconds) |
|---|---|---|
| IBM SP2 | 8 | 5572 |
| IBM SP2 | 16 | 2854 |
| Cray T3D | 32 | 1653 |

As a general guide, we find that the T3D and SP2 timings for a given number of nodes are roughly comparable. This is because of the fine-grained nature of the CG algorithm and frequent use of global sums; latency ($n_{\frac{1}{2}}$) is an important factor (see the section on message passing for a discussion of this point).

Work is currently in progress to generalise this code for the 2-active electron approximation.

## 3.4 Time-dependent Reactive Scattering

*R.J. Allan and G.G. Balint-Kurti*

The TDRS code is used by the Consortium for Chemical Reaction and Energy Exchange Processes. It was originally written by G. Balint-Kurti and A. Offer (University of Bristol) and the parallel version has been the subject of a recent Ph.D. thesis by Fahrettin Gögtas (1995). This parallel version was run on the Meiko CS-1 at Bristol and also the Intel iPSC/860 at Daresbury. R.J. Allan has more recently modified the code to use the BLAS and MPL on the Daresbury IBM SP2. Work is in progress to develop a multigrid version of this code.

The program propagates the time-dependent Schrödinger equation using a number of time steps. The duration (in pseudo time) is specified in the input file, as are also the total energy and molecular parameters.

Use of the multi-sequence 1D FFTs from VFFTPK is described in the section on FFTs. The propagation algorithm uses a Chebyshev polynomial expansion and 3-term recurrence relation as described by Tal-Ezer and Kosloff [73] with modifications to the DVR grid techniques by Dixon and Balint-Kurti (see Gögtas 1995 and references therein). In brief an initial (Gaussian) wavepacket is computed in the reactant channel on the potential energy surface of an atom-diatom system. This is decomposed into parts represented by a Chebyshev polynomial series. Each part must be propagated using the FFT method in two dimensions, the third dimension being the atom-diatom hyper angle which is used for parallelisation within an IOS type of approximation. An alternative parallelisation scheme would be to propagate each Chebyshev term on a different processor (or both for large MPPs).

Typical numbers of Chebyshev terms are between 10 and 100 depending on the initial wavepacket's total energy and the time step. Typical numbers of angles are O(30).

Different Jacobi coordinates are used in the reactant and product channels. A coordinate transformation takes place (also using FFT interpolation) once the wavepacket is localised near the reaction barrier or well. Identical propagation then takes place in the product channel and the wavepacket is projected onto product vibrational states. Variation of the amplitude of these states as a function of time is written to a file and again an FFT is used to convert it to an energy-dependent reaction cross section for each final state. The program thus yields state-to-state cross sections from a well-defined initial vibrational state with total angular momentum J=0 into a number of final vibrational states. This is done in a final analysis stage.

Execution time depends on the size of the 2D DVR grid in a complex way which reflects the scaling of the FFT routines and BLAS used. Time depends linearly however on the number of hyper angles and number of time steps used and also the number of terms in the Chebyshev decomposition.

Two test cases have been run for the reaction Li + HF $\rightarrow$ LiF + H at an initial energy of 0.25eV. The execution time is reported for just one time step.

**Small job, times in seconds**

dt=100.0, vmax=0.2, nx=256, ny=54, nth=16

Table 3.7: TDRS on the IBM SP2

| nnode | protocol | nterms | setup cpu | setup wall | cpu per term | wall per term | total |
|---|---|---|---|---|---|---|---|
| 1 | us | 70 | | 26.6 | 2.2 | 2.2 | 180.6 |
| 2 | us | 70 | 9.5 | 25.0 | 1.3 | 1.3 | 116.0 |
| 4 | us | 70 | | 20.4 | 1.6 | 1.6 | 132.4 |
| 8 | us | 70 | | 24.3 | 0.9 | 0.9 | 87.3 |

Note larger grid in this run size dt=100.0, vmax=0.2, nx=256, ny=54, nth=64 and consequently larger number of Chebychev terms.. Only the total time used is shown for one time step, excluding startup time.

Notes:

1. tinit is done once only per superstep and includes i/o for restart purposes
2. tprop=nter*(t1+t2+t3+trecur)

Table 3.8: TDRS on the Cray T3D

| nproc | nterm | tinit | t1 | t2 | t3 | trecur | tprop |
|---|---|---|---|---|---|---|---|
| 1 | | | | | | | *4011 |
| 2 | 211 | 98.7 | 4.44 | 4.35 | 0.0364 | 0.155 | *2009 |
| 4 | 212 | 72.0 | 2.24 | 2.17 | 0.0187 | 0.0777 | 1008 |
| 8 | 213 | 63.0 | 1.26 | 1.08 | 0.00913 | 0.0388 | 509.8 |
| 16 | 215 | 54.7 | 0.773 | 0.535 | 0.00453 | 0.0195 | 299.7 |
| 32 | 219 | 52.0 | 0.549 | 0.269 | 0.00227 | 0.0097 | 178.3 |
| 64 | 228 | 52.5 | 0.443 | 0.135 | 0.00115 | 0.0048 | 137.8 |

3. all times are prefectly parallel except t1 which includes global comms (with shmem)

4. * estimated using Amdahl's law

In this code the initialisation step involves computing and broadcasting information required by all nodes, so that the elapsed and cpu time differ. The propagation step is executed independently on each processor, the non-linear speedup reflecting some duplicated computational work (a global summation step between time steps has not been measured). The extra elapsed time noted on the ip network reflects additional system activity.

## 3.5 CFG

*R.J.Allan and M.L.Brewer*

CFG

This code implements a new semiclassical initial value method for the calculation of polyatomic bound-bound Franck-Condon spectra [104]. The method combines the Frozen Gaussian Approximation (FGA) of Herman and Kluk and the cellular dynamics algorithm of Heller. This overcomes problems of an oscillatory integrand in previous methods and allows an accurate treatment of absorption spectra of polyatomic bound-state potentials.

The test results below show a 3-atom case with 512 trajectories. Work is currently underway with 6 atoms which uses 8192 trajectories. The first run of this completed in 1 1/2 hours on 256 processors. The code was developed by M.Brewer and D.Manolopoulos.

Table 3.9: Parallel performance on the T3D [seconds]

| Code | Number of Nodes | | | | | | |
|---|---|---|---|---|---|---|---|
| | 1 | 2 | 4 | 8 | 16 | 32 | 64 |
| CFG | | | 570.1 | 293.2 | 149.5 | 77.9 | 41.9 |

## 3.6 DVR3D

*H. Mussa, C.J. Noble and R.J. Allan*

A parallel version of DVR3D is partly completed by EPSRC CASE Ph.D. student Hamse Mussa directed by the author Jonathan Tennyson with support from Cliff Noble and Robert Allan (Daresbury staff). The code uses MPI.

Both 2D vibrational Hamiltonian and 3D ro-vibrational Hamiltonian generation of order 10,000 is complete solved using the Parallel General Symmetric Eigensolver system PeiGSv2.1 by David Elwood, George Fann and Rick Littlefield (PNNL). Solution of the final block-structured sparse matrix of order 100,000 is likely to use the PARPACK implementation of the implicitly-restarted Arnoldi method of Richard Lehoucq, Danny Sorensen and Kristyn Maschhoff (Rice University).

## 3.7 R-Matrix Propagators

*J.W.Heggarty, A.G.Sunderland, C.J.Noble, N.S.Scott, V.M.Burke, R.J.Allan, M.S.T.Watts*

R-matrix methods are often used to investigate electron-atom and electron-molecule collisions.

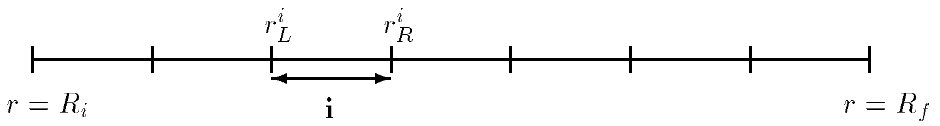
R-Matrix propagators provide an efficient method of calculating the R-matrix at radial distances $r = R_f$ (where matching to an asymptotic solution is feasible) from R-matrices obtained on an inner boundary $r = R_i$ from R-matrix calculations.

This external region problem involves solving:

$$[\frac{d^2}{dr^2} - \frac{\mathbf{l}(\mathbf{l}+1)}{r^2} + \mathbf{k}^2]\mathbf{F} = \mathbf{V}\mathbf{F}$$

where

$$V_{ij} = \sum_{\lambda=1}^{m} a_{ij}^{\lambda} r^{-\lambda-1}$$




The R-matrix given by:

$$\mathbf{F}^i(r_R^i) = \mathbf{R}^i \frac{d\mathbf{F}^i}{dr} \mid_{r=r_R^i}$$

is propagated using the equations

$$\mathbf{R}^i = -[\mathbf{G}^i(r_R^i \mid r_R^i) + \mathbf{G}^i(r_R^i \mid r_L^i) \frac{1}{\mathbf{R}^{i-1} - \mathbf{G}^i(r_L^i \mid r_L^i)} \mathbf{G}^i(r_L^i \mid r_R^i)]$$

which involve the Green's function $\mathbf{G}$ calculated on the sector boundaries.

Large-scale calculations (500-2000 channels) are required to treat scattering of atoms in the iron peak region of the periodic table.

These may be performed using massively parallel distributed memory computers such as the Cray T3D and IBM SP2.

Simple decompositions such as distributing scattering symmetries or energies are insufficient in these cases. We demonstrate parallel decompositions of the two forms of R-matrix propagator used in the serial package FARM which are successful for problems involving more than 1000 channels.

### 3.7.1 Baluja-Burke-Morgan Propagation

#### Outline of Method

The basis function approach which was proposed by Baluja, Burka and Morgan [2] is referred to here as the BBM propagator. It divides the range into a number of sectors, the sector size being determined by the number of nodes of the wavefunction within the sector. Green's function within this subrange can then be expanded in terms of an orthonormal set of shifted Legendre polynomials.

Rather than undertaking a potentially costly parallel diagonalization of the resulting sector Hamiltonian matrix, this approach invokes ScaLAPACK parallel linear equation solving routines (LU factorizations and triangular solves).

This parallel 'solution following' approach ensures good load balancing and a guaranteed high level of accuracy regardless of large variations in interaction potentials. It is therefore particularly suitable for propagations in the proximity of the inner boundary layer.

#### Parallelisation Strategy

The parallel propagation code is written in Cray Fortran 90. In effect this is a subset of standard Fortran 90 (for instance, modules and contains functions are excluded), therefore the progression of the sequential F90 code to a parallel Cray f90 version has involved much redesign.

A distributed data approach is used, reduced-width amplitude matrices, potential matrices, and Hamiltonian matrices are distributed across the network of processors.

With 64 MBytes of memory available on each node of the T3D, and up to 256 processing elements (PEs) available, this means that problem sizes in excess of 1000 channels can be solved. The first stage of the propagation requires calls to explicit MPI message passing routines in order to globally broadcast the problem characteristics. From this information the memory required can be dynamically partitioned on each node. However, one important note is that the SHMEM message passing primitives at the heart of the Cray MPP ScaLAPACK implementation require identical memory addressing on each processor for distributed objects. This condition is in no way guaranteed using Fortran 90 allocate. The Cray specific shpalloc routines must be used to dynamically allocate memory for ScaLAPACK data objects.

The reduced-width amplitude matrix from the inner boundary layer is read and distributed across the processor network. This data can then be used to construct the distributed initial R-matrix 'in place'.

The program loops over scattering energies and sectors within each energy.

### ScaLAPACK on the Cray T3D

In order for ScaLAPACK to operate efficiently some information regarding the distributed objects must be provided. This is achieved via a descriptor vector. The entries of this descriptor uniquely determine the mapping of the matrix entries onto the local processes' memories. Each distributed object is initially associated with a descriptor variable. Thereafter a handle and the descriptor vector is all that is required to reference the distributed object.

ScaLAPACK routines generally consist of blocked algorithms. To this end a block size must be specified for each distributed object. Determining optimal, or near optimal block sizes for the data distribution is complex, as it depends on many problem and implementation dependent factors. For the purposes of these results, the block size nb for the distribution and computation is set at nbrow = nrows/nprows, nbcol = ncols/npcols for an nprows(npcols processor grid. Preliminary tests have shown this block size selection to be fairly efficient, though more experimentation in this field is required before concrete conclusions can be drawn regarding performance.

### Advantages of this method

- Accuracy generally unaffected by large variations in interaction potential
- Efficient use of processors and guaranteed load balancing
- Better scaling than Light-Walker method over large numbers of processors. Very large problems (approx. 1000 channels) can be solved by simply adding more processors.
- The higher level structure of the program should appear very similar to the sequential code, with calls to ScaLAPACK routines, rather than LAPACK routines. Unfortunately, as mentioned previously, the Cray Fortran 90 implementation does not provide all the features of standard Fortran 90. This has resulted in quite extensive modifications to the original sequential code. However, these discrepencies will be addressed when standard Fortran 90 is available on the T3E.

### Disadvantages of the method

- Requires more computation than the Light-Walker method for the same propagation distance.
- Communication overheads affect speed-up on large processor arrays.
- The method is unsuitable for small arrays of processors.

### 3.7.2 Light-Walker Propagator

#### Outline of Method

The Light-Walker (LW) propagation approach employed [3] is a potential following method which assumes that the interaction potential is constant within each propagation sector. The size of each sector is determined by estimating the error resulting from variations in the potential over the sector width. The interaction potential at the centre of each sector is diagonalised to obtain an orthogonal transformation matrix used to decouple the Green's functions within each sector. An initial R-matrix may then be propagated across each of the sectors until the final propagation distance is reached.

#### Parallelisation Strategy

Data from an internal region R-matrix program is read in from a file and sent to appropriate PEs as necessary.

Initial R-matrices at the required energies are computed from the reduced-width amplitudes by a dedicated group of PEs.

Remaining PEs form a data-dependant pipe. The number of sectors required for a given propagation distance is calculated and sectors distributed as evenly as possible over the PEs in the pipe. For each sector in the pipe the sector potential matrix is then diagonalised to obtain the transformation matrix.

Once setup is complete the first R-matrix to be propagated is requested by the front-end of the pipe. The R-matrix is propagated over the appropriate number of sectors local to that PE and then sent to the neighbouring PE in the pipe. The front end of the pipe is now free to request another initial R-matrix while its neighbour begins to propagate the frist R-matrix over the sectors local to it. Each time a PE element finishes propagating the R-matrix over its sectors it sends it to the next PE and receives a new R-matrix from the previous PE. This process continues down the length of the pipe for each of the required energies.

#### Advantages of this method

- Less computation per sector than BBM method
- Matrices are size N(N compared to (Nb(N)((Nb(N), where N = no. channels, Nb = no. basis functions in BBM method. This greatly reduces demands on available memory.
- Sector sizes may be varied.

#### Disadvantages of this method

- Maximum efficiency requires one sector calculation per processor.
- Potentially perfect load balancing degraded by wind-up and wind-down times. This cost penalty becomes negligible when propagating over a large number of scattering energies
- Error control determined phenomenologically.

**Propagation times on the Cray T3D**

Table 3.10: R-Matrix propagators on the Cray T3D [seconds]

| 240 channel problem, 5 energies | | |
|---|---|---|
| Processors | Light-Walker Pipeline 53 sectors | BBM Basis Function 2 sectors |
| 4 | 288.4 * | 2393 |
| 8 | 182.3 | 1336 * |
| 16 | 119.6 | 992.1 |
| 32 | 103.3 | 520.2 * |
| 64 | 85.4 * | 370.4 |
| 128 | 78.6 * | 316.3 * |
| 256 | 75.3 * | 196.4 |

| 480 channel problem, 5 energies | | |
|---|---|---|
| Processors | Light-Walker Pipeline 53 sectors | BBM Basis Function 2 sectors |
| 4 | 1942 * | 22690 * |
| 8 | 1199 * | 11788 * |
| 16 | 828.0 | 6304 * |
| 32 | 642.3 | 3611 * |
| 64 | 549.5 * | 2411 |
| 128 | 503.0 * | 1567 * |
| 256 | 479.8 * | 1096 |

* denotes extrapolation of timings using least squares fit to dataPropagation interval = 9.4 Bohr (inner region boundary layer) to 18.8 Bohr. Results agree to within a 1

### 3.7.3 Conclusions

To date the research has successfully produced two contrasting parallel R-matrix propagation implementations. The results presented suggest that both the Light-Walker pipeline approach and the Baluja-Burke-Morgan basis function approach to the propagation can be parallelised efficiently.

Vast improvements have been made over the sequential implementations, both in the speed of the propagation, and in the size of the problem that can be undertaken.

The above graph shows reasonably good speed-ups for the BBM method, particularly on larger problems. LU factorizations and, more especially triangular solves are inherently sequential operations, so near 100despite the fact that the ScaLAPACK implementation is highly optimized for the T3D, at present only general routines are available, so the linear equation solvers take no advantage of symmetry. It is expected that a full range of ScaLAPACK routines will be available for the next generation T3E machines.

The Light-Walker pipeline performs best over large numbers of energies where start-up and wind-down times become negligible.

As expected, the above results show that if the interactive potential field varies relatively smoothly (as in the problem described here) , the parallel Light-Walker propagator performs best. However, close to the inner region boundary, this is often not the case, and to obtain acceptable results the BBM approach will be used.

The BBM method becomes competitive on larger processor arrays and larger problem sizes, where the LW pipe length is in excess of the number of sectors. In the problem outlined above, there is no advantage to be gained by running the LW propagator on more than 64 processors.

**Further Work**

Study how block size affects ScaLAPACK performance in the BBM propagator Overlap communication and computation to a greater extent in the L-W propagator

**References**

[1] V.M.Burke and C.J.Noble FARM - A flexible asymptotic R-matrix package Computer Physics Communications 85 (1994) 471-500

[2] K.L.Baluja, P.G.Burke and L.A.Morgan Computer Physics Communications 27 (1982) 299

[3] J.C.Light and R.B.Walker Journal of Chemical Physics 65 (1976) 4272

# Chapter 4

# Computational Engineering

Computational engineering is a major and growing activity with a substantial support effort provided for the community by the Daresbury Laboratory. On the national scale CCP12 has many active members and brings together a number of projects involving academics and industrialists. On a European scale ERCOFTAC and a number of other bodies serve a similar purpose.

The computations range from modelling of flames involving, multi-phase flows and huge ranges of temperature to flow around complex bodies, such as in chemical reactors or over aircraft wings, to structural design. The applications detailed below are a sample of these areas.

## 4.1 Direct numerical simulation for computational combustion

*K. Maguire, D.R. Emerson and R.S. Cant*

The ANGUS code was developed by Stewart Cant and David Emerson of the Computational Combustion Consortium. The method employed, together with an initial performance comparison of the parallel code on the Cray T3D and Intel iPSC/860, are presented in [4]. The notes below are based on that paper.

### 4.1.1 Description of the ANGUS Program

The purpose of the ANGUS code is to carry out DNS of turbulent premixed combustion in order to generate statistical data in support of modelling. The equations to be solved are the Navier-Stokes equations for fluid flow, augmented by two additional equations each describing the transport of a single scalar variable and together specifying the thermochemical state of the system in the presence of differential diffusion effects. Thus, in total there are six partial differential equations to be solved: a continuity equation; three momentum equations; a scalar equation representing temperature; and a scalar equation representing a reaction progress variable. These equations are described in detail in [4].

In solving the equations a low Mach number approximation is made in order to decouple the density from the pressure and thus avoid the need to resolve acoustic length and time scales in the simulation. Note that no turbulence-modelling assumptions are made anywhere in this set of equations, which therefore remain exact in both laminar and turbulent flow at low Mach number.

In order to simulate non-reacting flow it is sufficient to solve only the first four differential equations, with the density assumed constant. In order to simulate combustion it is necessary to solve at least the first five differential equations with a specified chemical reaction rate. In order to take account of unequal diffusion of heat and mass all six differential equations must be solved with the Lewis number (Le) specified different from unity. A great deal of useful data has been obtained from combustion simulations carried out in this manner at constant density, but for full realism the density (and the molecular transport properties) must be allowed to vary with the temperature.

The purpose of the present investigation is to examine the fluid-mechanical implications of combustion in a turbulent environment, and the inner workings of the chemistry are of secondary importance. Thus we assume that the chemical reaction of interest can be modelled in the most simple way by a one-step irreversible reaction governed by Arrhenius kinetics.

Discretisation of the equations is carried out using standard second-order central differences on a three-dimensional grid of unit cells. The velocity nodes are located at the face-centres of each cell, giving a staggered-grid arrangement that conserves kinetic energy as well as mass and momentum [81]. This is a low-order discretisation by comparison with much previous DNS work where pseudo-spectral or high-order finite-difference methods have been used [82, 83]. However the use of low-order discretisation in the present work is justified physically by the high level of natural diffusion in the combustion problem. Time advancement is achieved by a modified explicit second-order Adams-Bashforth procedure incorporating the solution of a Poisson equation for the pressure.

The usefulness of any direct numerical simulation depends *inter alia* on the turbulence Reynolds number that can be attained. In the present case the limit of resolution is set by the need to maintain an adequate representation of a laminar flame. A minimum of about ten computational cells is required, and in order to remain in a regime of combustion of technological interest it is necessary to set the size of the smallest scale of turbulence to be comparable with this figure. At a small-scale resolution of ten cells the Reynolds number is about 30 for a grid size of 1283. This represents the current state-of-the-art in combustion DNS. Increased grid size increases the Reynolds number to about 70 for a grid size of 2563 and about 120 for the maximum grid size of 3683 which appears to be achievable on the Cray T3D. The last figure is becoming comparable with the turbulence Reynolds numbers attained in simple laboratory burner experiments.

The class of problem to be addressed using ANGUS concerns turbulent premixed flame propagation into an oncoming stream of reactants. A planar laminar flame is positioned near the centre of the computational domain perpendicular to the streamwise direction. It is supplied with fresh reactants by means of a turbulent inflow condition and burned products are exhausted by means of an outflow boundary condition. The spanwise boundary conditions are periodic. This problem class offers the possibility of achieving statistical stationarity and of collecting genuinely time-averaged statistics. For development purposes a simpler problem will be tackled involving two laminar flames initialised back-to-back. This problem is time-dependent, but avoids the need to specify a turbulent inflow condition and yields double the statistical sample size for flame-related quantities. The pressure solver utilises a conjugate gradient method with a Modified ILU (MILU) preconditioner [84]. As with many CFD algorithms, the resulting matrix is both sparse and symmetric. In this case, it is heptadiagonal and the periodic boundary conditions also mean that the matrix is singular. In this case, the MILU preconditioner is used and the diagonal is evaluated only once at the initialisation phase of the program. The parallel preconditioning is done locally (there is no fill in). Pseudo code for the main execution steps is given in the paper [4].

### 4.1.2 Parallel Implementation of ANGUS

A grid partitioning strategy is employed for ANGUS which is quite typical of many domain decomposition techniques used in parallel CFD. Due to the finite difference stencil it is necessary to introduce "halo" or "ghost" cells at the interface boundaries. These are used as a message cache and allow derivatives to be determined in these regions with only local variables. The halo cells are then updated as required. The difference stencil employed in ANGUS uses a halo only one cell thick. In other applications, it may be necessary to accommodate two halo cells. The data transfer routines are however written in a general way and this can be accomplished by simply setting a parameter NHALO=1 or 2.

A staggered grid approach is used with the pressure located at the cell centre. Owing to this staggering it is necessary to transfer data from diagonally adjacent neighbours as well as from vertical and horizontally adjacent neighbours. This can be done in two ways: (i) by transferring a single element to the point where it is required; or (ii) by utilising the fact that the data has already been transferred to the domain's nearest neighbour in the horizontal or vertical move. In the first approach, the logic can get considerably more involved whereas in the latter approach it necessitates only the inclusion of the extra two halo cells in the data transfer. This adds only a minor additional overhead and is far easier to implement and maintain. Cells are mapped logically to processors in a rectangular grid pattern.

Message passing was initially done using standard PVM. However, whilst PVM is good for general purpose computing and portability on workstation clusters, other options on the T3D give enhanced performance and reduced latency $n_{\frac{1}{2}}$ (see section on message passing). Finally SHMEM was used on the Cray T3D and MPI on the IBM SP2.

### 4.1.3 Optimisations

During parallelisation a thorough performance analysis was undertaken. It was already known that the pressure solver was the most compute-intensive part of the calculation. Indeed, the pressure solver takes in excess of 95% of the computing time at each time step. This was confirmed with Apprentice on the Cray and it also indicated the relatively low floating point performance of the routine, which is coded in standard Fortran 77. As previously discussed, the pressure solver uses a preconditioned conjugate gradient method which, apart from the preconditioner, consists of essentially dot products and saxpy-type operations. It is straightforward to code and has a unit stride, which provides good access to data in cache. However, the relatively small cache, and the lack of a secondary cache on the T3D, appear to play a significant part in the relatively poor floating point performance of standard Fortran 77. This feature appears to be typical in RISC based microprocessors [85] and is, in part, related to the small cache but is also compounded by the fact that access to main memory takes many additional clock cycles; typically factors of 3 to 5 are involved. In this particular case, the absence of the secondary cache on the T3D may also be important.

Owing to the domination of the pressure routine early efforts were concentrated on trying to optimise that section of the code. The conjugate gradient method is an ideal candidate for using the BLAS, however only level-1 BLAS can be used because the matrix is sparse and advantages may be limited (see section on BLAS).

Results were obtained by performing a fixed number of iterations in the conjugate gradient solver. This was done because the number of iterations necessary to obtain a converged solution with the parallel preconditioner varied slightly at each timestep. Therefore, to give a clearer indication of where the time was being spent, the number of iterations was fixed at 300 for the solver without preconditioning and at 150 when the preconditioner was being applied. This is in broad agreement with what is observed in practice where the number of iterations are reduced by approximately a factor of two.

Detailed timing analysis of each individual loop section was performed and is discussed in the paper [4].

The preconditioner and matrix-vector product, q = Ap, are both executed in standard Fortran-77. Typical times for these sections (see Table 4.1) therefore give an indication of how real sections of code perform on parallel computers. When the results obtained were analysed (these sections of code are identical on both machines whereas the matrix-vector loop for the BLAS routines was unrolled on the CRAY T3D) it was found that the typical improvement in performance is about 2.5 in moving from the Intel to the Cray. When the BLAS/Fortran sections were compared (these sections involve the saxpy and dot product operations) it was clear that the Cray routines are significantly better. However the Fortran performance on the Intel is better than its level-1 BLAS and the real improvement is again about 2.5.

Table 4.1: Typical Performance of ANGUS Code Sections on the Cray T3D, 64 processors and problem of size $128^3$

| | code type* [time in seconds] | | | |
|---|---|---|---|---|
| routine | 1 | 2 | 3 | 4 |
| global sum | 0.47 | 1.01 | 0.46 | 0.87 |
| message passing | 10.48 | 22.81 | 10.47 | 22.79 |
| preconditioning | 36.39 | 4.35 | 36.32 | 8.53 |
| q=Ap | 11.89 | 24.32 | 15.98 | 34.32 |
| BLAS/Fortran | 18.45 | 40.21 | 33.44 | 71.73 |
| Total time | 77.68 | 92.70 | 96.67 | 138.34 |

* Note code types: 1 BLAS + preconditioning; 2 BLAS without preconditioning; 3 F77 + preconditioning; 4 F77 without preconditioning.

### 4.1.4 Summary and Conclusions

A description of the ANGUS code [4] and its implementation on the IBM SP2 has been provided, and performance comparisons made with the Cray T3D. Whilst there is room for improvement with the floating point performance of standard Fortran-77, it is possible to improve the performance by the utilisation of the level-1 BLAS in solvers such as conjugate gradient algorithms. However, preconditioning does present a problem for efficient cache utilisation and the lack of a secondary cache clearly plays a role. In comparison to the Intel iPSC/860 and Cray T3D, the IBM SP2 shows good performance particularly for small numbers of processors. This may be attributed to the more powerful nodes of the SP2, although the communications network on the T3D, providing a low latency ($n_{\frac{1}{2}}$) and a high bandwidth, acts to narrow the performance differential on larger numbers of processors. These features are clearly important for iterative schemes, such as the conjugate gradient algorithm, where global sums are performed frequently and significant amounts of interface data need to be transferred.

Furthermore the level-1 BLAS on the Intel and IBM are little better than raw Fortran-77 (see section on BLAS) so no improvement is seen. Higher level BLAS cannot be used in this code because of the very sparse (heptadiagonal) matrix.

The Table 4.2 below shows a comparison of the total times for the code running on the Cray T3D and IBM SP2 for a number of problem sizes. The Cray T3D was using UNICOS-MAX 1.1.0.2 and the cf77_6.1 compiler. Compilation on the IBM used mpxlf -O3 and the ESSL BLAS.

Table 4.2: Performance of ANGUS on the Cray T3D and IBM SP2 for Various Problem Sizes

| Procs | Cray T3D | IBM SP2 |
|---|---|---|
| Problem size 32x32x32 | | |
| 1 | 5.28 | 2.40 |
| 2 | 3.15 | 1.77 |
| 4 | 1.56 | 1.00 |
| 8 | 0.78 | 0.57 |
| Problem size 96x96x96) | | |
| 8 | 40.86 | 20.97 |
| 16 | 22.52 | 17.14 |
| Problem size 128x128x128) | | |
| 16 | 65.43 | 44.13 |

Note: in these tests the code used preconditioning and the level-1 BLAS.

## 4.2 WING3D - 3D wing code

*K. Maguire*

This code was developed by Ken Badcock (University of Glasgow) of the External Aerodynamics Consortium for modelling airflow over a 3D aircraft wing.

### 4.2.1 Code Description

This code was originally developed to run on a cluster of SGI workstations, using PVM for message passing. Versions run on the IBM workstation cluster and the Intel iPSC/860 at Daresbury, the Cray T3D and now the IBM SP2.

The code is used to solve the thin-layer Reynolds' averaged Navier Stokes equations for aerofoil and wing flows, in 3 dimensions using an implicit method. Spatial terms are discretised using Osher's flux approximation with MUSCL interpolation and Von Albada limiter, and central differencing for the viscous fluxes. The Baldwin-Lomax turbulence model is used; although not linearised in time in the current version, this has been found not to degrade the stability properties of the method. At each time step, a linear system is required to be solved, the inverse of the approximate ADI factorisation is used as the preconditioner in a conjugate gradient algorithm.

This code is dominated by computation, particularly on systems with high bandwidth like the Cray T3D, and communication costs are typically very low. Details of the parallelisation of the code for other parallel systems are contained in a paper by Badcock [3]. The SP2 port did not present many difficulties, although some of the communication patterns can cause deadlock when using blocking sends and receives.

The supplied test data set, an OneraM6 aerofoil, is very large and required at least 4 processors to run on the IBM SP2 (and for comparison 8 processors on the Cray T3D), so it is difficult to be definite about overall performance, but an SP2 node typically runs at 3 times the speed of a T3D node, roughly in line with other similar codes. Little optimisation has yet been possible for the SP2, and this figure can be expected to improve once this has been done.

The code makes few calls to libraries, no BLAS or LAPACK, but spends a lot of time ( 10%-20%) in Fortran INTRINSIC routines SQRT, LOG, EXP. In some places the coding is rather crude, it is expected some code restructuring would have a beneficial effect on performance. In particular some routines have been produced by a symbolic package, making them unreadable and unmaintainable, it as expected that these will be replaced in future versions of the code

This code uses an explicit scheme to get a starting point, then uses an implicit scheme for the remainder (and majority) of the solution process.

### 4.2.2 Parallel Port

Cray T3D Port/Code Type: Dr. Badcock, Cray Fortran, T3D PVM message passing [3].

IBM SP2 Port/Code Type: K.Maguire, Fortran 77 plus extensions, MPL, message passing

The following points should be noted:

1. This code is too big to run on less than 8 processors on the Cray T3D and 4 processors (using some virtual memory) on the IBM SP2.
2. Efficiency is good on both machines, with communication costs low for this code.

Table 4.3: WING3D Performance: Average Time (seconds) per Implicit Iteration

| Processors | Cray T3D | IBM SP2 |
|---|---|---|
| 4 | | 28.96 |
| 8 | 43.30 | 14.85 |
| 16 | 23.54 | 8.19 |
| 32 | 11.46 | |

3. No BLAS calls are present in WING3D.

## 4.3 Turbulence Transition Modelling

*K. Maguire*

The code TMCODE is used to calculate the solution of the three-dimensional unsteady Navier Stokes equations for a flow in a lid-driven cubical cavity. It was written by Dr Gajjar and Mr Mustafa Turkyilmazoglu (University of Manchester). The equations are solved by using a finite difference approximation to the velocity-vorticity form of the differential equations, and solving the resulting linear systems using Red-Black Successive Over-relaxation (SOR) methods. Solutions can be obtained for Reynolds numbers up to 1000.

A more detailed description can be found by Turkyilmazoglu [78].

The code was originally written for the MASPAR MP-1104 parallel machine, and required significant re-engineering to produce acceptable performance on the Cray T3D. The code was initially developed using Fortran-90, compilers. Unfortunately one is not available as yet on the Cray T3D, and thus a Cray Fortran-77 version was produced by Dr Gajjar. This was achieved by adopting a straightforward domain decomposition scheme in the X and Y directions, and using a message-passing (shmem based) coding style rather than the data-parallel style originally used on the MASPAR.

The IBM SP2 version of the code is also message passing, given the current unavailability of a data-parallel compilation system. It is based upon IBM's MPL, and has been fully tested up to 16 processors. Note that as this code is comparatively new, little optimisation effort has been possible on the SP2. Additionally all message passing is currently implemented using blocking MPL calls, so that some additional improvement would be expected when computation and communication are overlapped. There are few calls to library routines, and performance analysis has been limited. There is a low i/o requirement, while masked operations are crucial due to the red-black method used, as are collective operations such as global summation

## 4.4 3D Supersonic Flow Over a Cavity

*J. Tracey and K. Maguire*

FLOW3D is a 2nd order finite-difference grid-based code to simulate high-speed air flow over a cavity. The flow is characterised by a boundary layer where the velocities of the flow change rapidly from

Table 4.4: Execution times (seconds) for TMCODE on the SP2 and T3D

| Processors | Cray T3D | IBM SP2 |
|---|---|---|
| 1 | 12.64 | 4.43 |
| 2 | 6.45 | 2.12 |
| 4 | 2.97 | 1.07 |
| 8 | 1.46 | 0.63 |
| 16 | 0.77 | 0.32 |

zero at the surface to the free stream values. The presence of the cavity causes the boundary layer to separate and re-attach at some point downstream. Where the flow re-attaches is determined by the cavity geometry and external conditions. So called "open" cavity flow is characterised by the stream re-attaching on a cavity face. Open cavity flow produces serious aerodynamic effects, such as fluctuating pressures, noise, heat transfer and increased drag. The study of flow in and around cavities is therefore of fundamental interest to aeronautical engineers. The high speed cavity airflow problem demands that both the compressibility and the viscosity of the fluid be accounted for accurately and provides a computational task characteristic of a range of commercial CFD applications.

The characteristic equations and method of solution adopted are described in documentation of the 2D FLOW benchmark code [95]. The FLOW3D code is still under development on both the T3D and SP2 and timings will be reported at a later date.

# Chapter 5

# Computational Chemistry

Computational chemistry covers a wide spectrum of activities ranging from quantum mechanical calculations of the electronic structure of molecules, to classical mechanical simulations of the dynamical properties of many-atom systems, to the mapping of both structure-activity relationships and reaction synthesis steps. Although chemical theory and insight play important roles in the work of computational chemists, the prediction of physical observables is almost invariably bounded by the available computer capacity. Massively parallel computers, with hundreds to thousands of processors (MPPs), promise to significantly outpace conventional supercomputers in both capacity and price-performance. The development of molecular modeling software applications that realise this promise, providing 10-100 times more computing capability than has been available with more traditional hardware, will enable substantial scientific and commercial gains by increasing the number and complexity of chemical systems that can be studied. While increases in raw computing power alone will greatly expand the range of problems that can be treated by theoretical chemistry methods, a significant investment in new algorithms is needed to fully exploit the potential of present and future generations of MPPs. Merely porting presently available software to these parallel computers does not provide the efficiency required to exploit their full potential. Most existing parallel applications show a significant deterioration in performance as greater numbers of processors are used. In some cases, the efficiency is so poor that the use of additional processors decreases, rather than increases, the performance. Thus, new algorithms must be developed that exhibit parallel scalability (i.e., show a near linear increase in performance with the number of processors).

Current MPP technology typically delivers Livermore FORTRAN Kernel (LFK) maximum megaflops per second (MFLOPS) ratings of 100-200, interprocessor communication bandwidths of 30-150 megabytes (MB) per second, and point-to-point latencies of 20-60 microseconds ($\mu$s). Figure 5.1 illustrates the relative computing power required for molecular computations at four different levels of theory (and accuracy): configuration interaction, Hartree-Fock, density functional theory (DFT), and molecular dynamics (MD). Each of these levels of theory formally scale from $N^2$ to $N^6$ with respect to computational requirements. Standard screening techniques which in essence ignore interactions for atoms far apart can however bring these scalings down significantly. Each level of theory also exhibits differing requirements for interprocessor communication (bandwidth and latency) and we will provide a brief description of this requirement in the next section. It is our impression that the next generation MPP will have LFK maximum rates of 300-400 MFLOPS, interprocessor communication bandwidths of 100-300 MB per second, and point-to-point latencies of 10-30 $\mu$s.

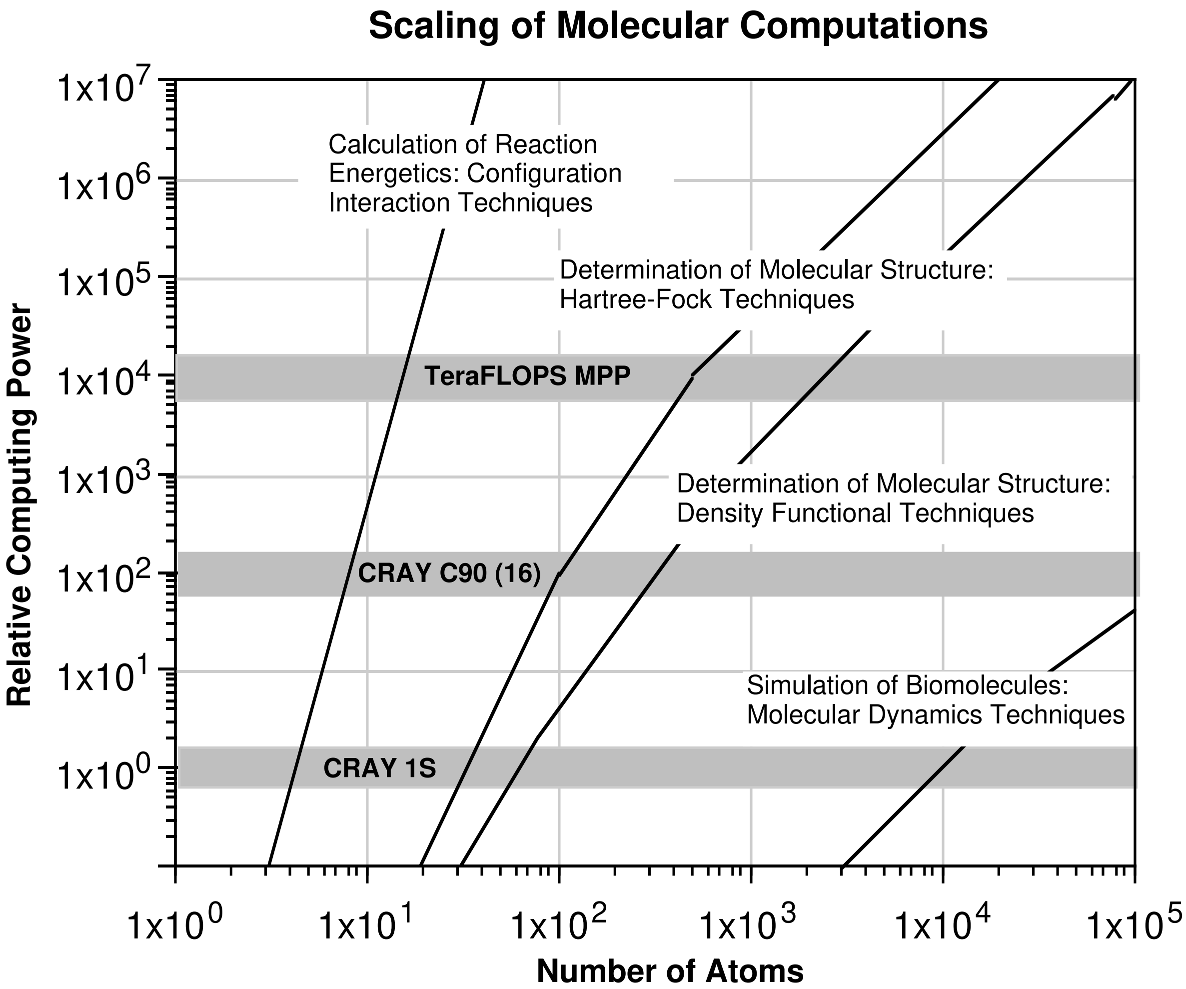



Figure 5.1: The relative computing power required for molecular computations at four levels of theory. The formal scaling for configuration interaction, Hartree-Fock, density functional, and molecular dynamics is: $N^6$, $N^4$, $N^3$, and $N^2$, respectively. Standard screening techniques which in essence ignore interactions for atoms far apart can bring these scalings down significantly.

## 5.1 NWChem and Fully Distributed Parallel Applications

*M.F. Guest*

NWChem is a computational chemistry package currently under development by the High Performance Computational Chemistry Group (HPCC), part of the Environmental Molecular Sciences Laboratory (EMSL), at the Battelle Pacific Northwest Laboratory (PNL), Richland Washington, USA. Targeting both present and future generations of massively parallel processors (MPP), this software features a variety of efficient parallel algorithms for a broad range of methods commonly used in computational chemistry. The emphasis throughout is on scalability and the distribution, as opposed to the replication, of key data structures so that memory is used efficiently. To facilitate such capabilities, a shared non-uniform access memory model has been developed which simplifies parallel programming while at the same time providing for portability across both distributed- and shared-memory machines. This is the GA-Tools discussed in the section on message passing.

In addition to the creation of new algorithms for computational chemistry on parallel processors, the HPCC group are also creating the high-level data and control structures needed to make parallel programs easier to write, maintain, and extend. These efforts have culminated in the NWChem package (for Northwest Chemistry). The package includes a broad range of functionality, and will continue to grow for some time yet. Algorithms for Hartree-Fock self-consistent field calculations [29, 30], a four-index transformation [31], several forms of second-order perturbation theory [3-5], density functional theory, and molecular dynamics with classical, quantum mechanical, or mixed force-fields are already in the code, and others such as multiconfiguration SCF (MCSCF) and higher-level correlated methods are under development. A wide range of properties are also available or under development: gradients, Hessians, electrical multipoles, NMR shielding tensors, etc.

### 5.1.1 Density Functional Theory - Self Consistent Field Module

A complete description of all of the algorithms employed in NWChem is not possible in the space available here. In order to demonstrate how the structure and tools are used by a high-level application, we will sketch the operation of the density functional theory module of NWChem. We will then mention briefly some of the other high-level functionality which is already in place or under development for the package.

An essential core functionality in a suite of electronic structure programs is the direct self-consistent field (SCF) module. It is increasingly evident that the application of direct SCF to large molecules is best performed on MPPs, owing to the enormous computational requirements. However, targeting systems with over 1,000 atoms and 10,000 basis functions requires a re-examination of the conventional algorithm and memory capacities assumed. The majority of previous parallel implementations use a replicated-data approach [35] which is limited in scope since the size of these arrays will eventually exhaust the available single-processor memory.

Our implementation of the parallel direct DFT (as in the Hartree-Fock module) distributes these arrays across the aggregate memory using the GA-Tools. This ensures that the size of systems treated scales with the size of the MPP and is not constrained by single processor memory capacities.

The software we have developed is a MPP implementation of the Hohenberg-Kohn-Sham formalism [50, 51] of DFT. This method yields results similar to those from correlated *ab initio* methods, at substantially reduced cost. It assumes a charge density, and approximations are made for the Hamiltonian (the exchange correlation functional); in contrast with the traditional *ab initio* molecular orbital method that assumes an exact Hamiltonian and chooses approximations to the wavefunction [52]. The Gaussian-basis DFT method in NWChem breaks the Hamiltonian down into the same basic one-electron and two-electron components as do traditional Hartree-Fock methods, with the two-electron component further reduced to a Coulomb term and an exchange-correlation term. The treatment of the former can be accomplished in a fashion identical to that used in traditional SCF methods, e.g. from a fitted expression similar to that found in RI-SCF [49], or from the commonly used Dunlap fit [53, 54]. DFT is really distinguished from traditional methods, however, by the treatment of the exchange-correlation term. This term is typically integrated numerically on a grid, or fitted to a Gaussian basis and integrated analytically.

It is instructive to elaborate on the computation/communication requirements of the time-consuming components. The computationally dominant step, the construction of the Fock matrix, is readily parallelised as the integrals can be computed concurrently. A 'strip-mined' approach is used where the integral contributions to small blocks of the Fock matrix are computed locally and accumulated asynchronously into the distributed matrix. By choosing blocking over atoms, the falloff in interaction between distant atoms can be exploited, while satisfying local memory constraints simultaneously. The three time-consuming steps contributing to the construction of the Fock matrix are: fit of the charge density (FitCD); calculation of the Coulomb potential (VC); and evaluation of the exchange-correlation potential (VXC). The fit of the charge density and the calculation of the Coulomb potential both typically consume similar amounts of floating point cycles, primarily evaluating three-center two-electron integrals at a rate of about 300-400 flops each. Very few communications are required beyond a shared counter and global array accumulate, these components are easily parallelised and display high efficiencies. The computation of the exchange-correlation potential requires far fewer flops (evaluating Gaussian functions numerically on a grid, as opposed to analytical integrations) but approximately the same communication effort. This results in a higher communication/computation ratio than the preceding two components.

Figure 5.2 shows the speedup on a variety of MPPs, using up to 128 processors to build the Fock matrix for the zeolite fragment $Si_{28}O_{67}H_{30}$ (1673 basis functions) on a variety of representative MPP systems, the IBM SP2, Kendall Square Research KSR-2, the Intel Paragon and Cray T3D. Near-linear speedups are obtained with all of these platforms. The total time to solution also approximately the same for three of the four MPPs, but for different reasons.

Data in Table 5.1 further contrasts the architectural differences in todays MPPs. For example, the high efficiencies of the VXC construction on the KSR-2 and T3D are a direct reflection of the hardware/software shared memory. The degradation of efficiency in the construction of VXC on the SP2 is a direct reflection of the higher processor power and the corresponding need to increase the granularity of work. This is a simple modification and has now been implemented.

### 5.1.2 Other Application Modules

NWChem is designed to be a full-featured computational chemistry package serving a broad user community. As such, we have developed, or will be developing, a wide range of application modules for the package. The Hartree-Fock (HF) module differs slightly from the DFT module described above

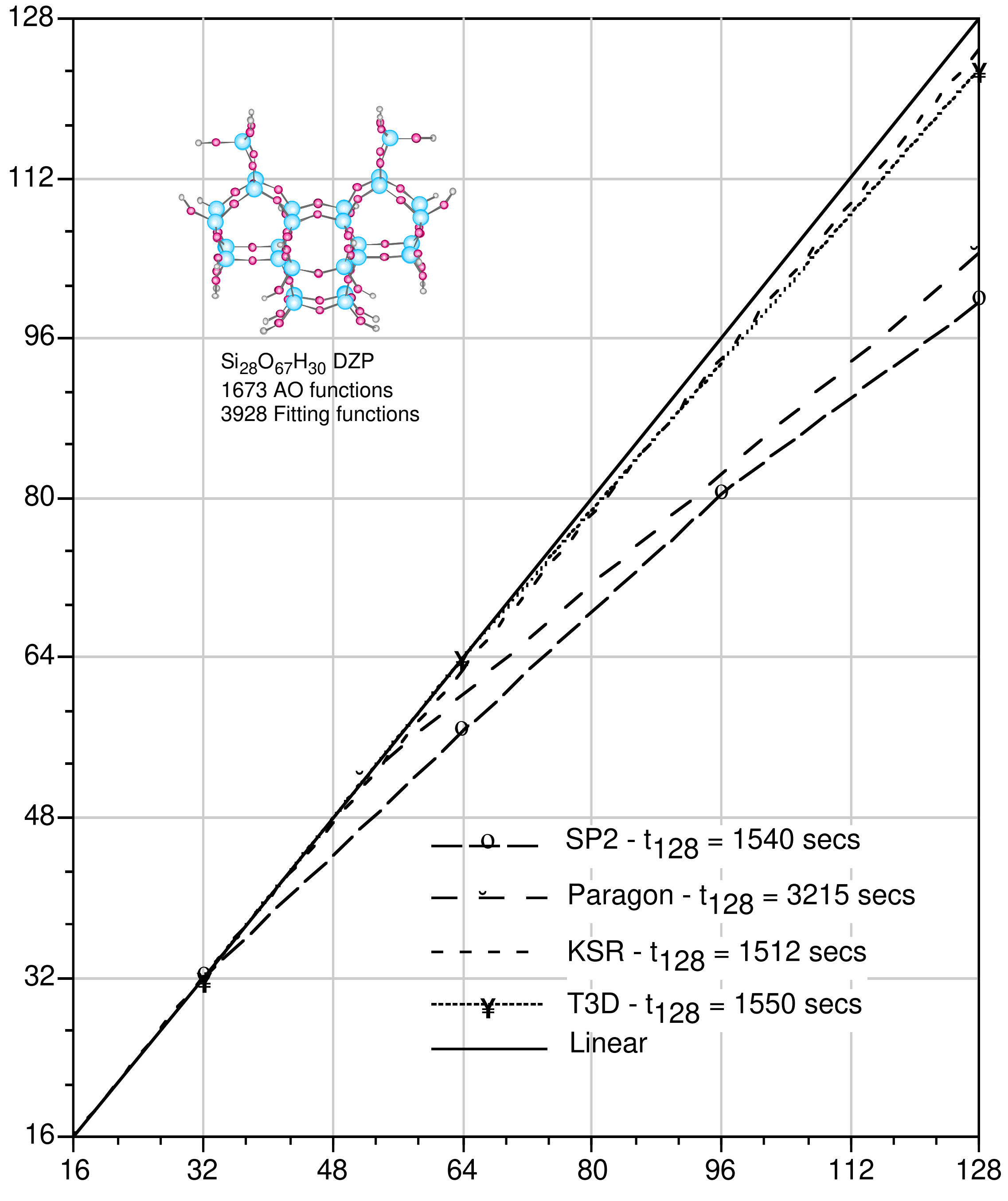



Figure 5.2: Parallel scaling of the NWChem Fock matrix construction for the zeolite fragment $Si_{28}O_{67}H_{30}$ using density functional theory (in the local density approximation) on a variety of massively parallel systems.

Table 5.1: Time in wall-clock seconds for the construction of the primary components of the density functional theory Fock matrix: Dunlap fit of the charge density, FitCD; construction of the Coulomb Potential, VCoul; construction of the exchange-correlation potential, VXC; and the total time

| Machine | Nodes | FitCD | VCoul | VXC | Total |
|---|---|---|---|---|---|
| IBM SP2 | 32 | 1667 | 801 | 2330 | 4798 |
| | 64 | 708 | 571 | 1430 | 2709 |
| | 96 | 445 | 425 | 1040 | 1910 |
| | 128 | 370 | 338 | 832 | 1540 |
| Intel Paragon | 52 | 1675 | 1889 | 2895 | 6459 |
| | 128 | 774 | 830 | 1611 | 3215 |
| KSR-2 | 16 | 3785 | 4252 | 3775 | 11812 |
| | 32 | 1875 | 2138 | 1870 | 5883 |
| | 48 | 1247 | 1439 | 1298 | 3984 |
| | 64 | 929 | 1066 | 1007 | 3002 |
| | 128* | 461 | 538 | 513 | 1512 |
| Cray T3D | 32 | 2144 | 2447 | 1391 | 5982 |
| | 64 | 1039 | 1212 | 736 | 2987 |
| | 128 | 519 | 619 | 417 | 1555 |
| | 256 | 260 | 305 | 242 | 807 |

in that it is based on a quadratically convergent SCF (QCSCF) approach [55]. The SCF equations are recast as a non-linear minimisation which bypasses the diagonalisation step. This scheme consists of only data-parallel operations and matrix multiplications which guarantees high efficiency on parallel machines. Perhaps more significantly, QCSCF is amenable to several performance enhancements that are not possible in conventional approaches. For instance, orbital-Hessian vector products may be computed approximately which significantly reduces the computational expense with no effect on the final accuracy. Gradients are available for both HF and DFT codes, with analytic Hessians and other properties under development. A parallel multiconfiguration self-consistent field (MCSCF) module is under development as well. In addition to efficient parallel scaling of algorithms as a route to large-scale simulations, we are also interested in alternative formulations of methods which may allow more efficient calculations for large systems. Therefore, besides the traditional direct second-order many-body perturbation theory (MP2) code, we are exploring methods which will allow us to take advantage of the locality of interactions in large systems [33, 56, 57, 58], as well as the use of integral approximations such as the so-called "resolution of the identity" approximation [48, 49], which we have implemented at both the SCF [59] and MP2 [32] levels. NWChem is also capable of molecular dynamics calculations using classical, quantum mechanical, or mixed Hamiltonians. Quantum mechanical forces may be obtained from any method in NWChem for which analytical gradients are implemented as well as from a number of third-party packages which can be easily interfaced with the NWChem structure.

A performance analysis of calculations on the zeolite fragment $Si_{28}O_{67}H_{30}$ on present generation MPP systems suggests that our goal of scalability, and hence the ability to treat more complex species in a routine fashion, is well in hand.

## 5.2 Coupled Cluster Code

*R. Kobayashi, University of Cambridge*

### 5.2.1 Theory and Implementation

The coupled cluster method, in the form in which it is used today, was introduced to chemistry by Čížek and Paldus [8], and first implemented by Purvis and Bartlett [60]. It is based on writing the coupled cluster wavefunction as an exponential of excitations operating on a Hartree-Fock (HF) reference.

$$|CC\rangle = exp(T)|HF\rangle \tag{5.1}$$

which ensures size-consistency. The cluster operator $T$

$$T = T_1 + T_2 + ... \tag{5.2}$$

is a sum of singles, doubles, etc. excitations commonly truncated at double excitations to give Coupled Cluster Singles and Doubles (CCSD). Substituting this wavefunction into the Schrödinger equation and projecting onto the ground and excited subspaces gives the set of coupled cluster equations for the energy and amplitudes.

$$\langle\Psi_0|(\mathcal{H} - E_0)(1 + T_1 + T_2 + \frac{1}{2}T_1^2)|\Psi_0\rangle = E_{CCSD} \tag{5.3}$$

$$\langle\Psi_0|(\mathcal{H} - E_0)(1 + T_1 + T_2 + \frac{1}{2}T_1^2 + T_1T_2 + \frac{1}{6}T_1^3)|\Psi_0\rangle = 0 \tag{5.4}$$

$$\begin{aligned}\langle\Psi_0|(\mathcal{H} - E_0)(1 + T_1 + T_2 + \frac{1}{2}T_1^2 + T_1T_2 + \frac{1}{6}T_1^3 \\ + \frac{1}{2}T_1^2T_2 + \frac{1}{24}T_1^4)|\Psi_0\rangle = 0\end{aligned} \tag{5.5}$$

These equations were efficiently formulated by Scuseria et al. [64], as below, in terms of intermediates to minimise the operation count and it is these equations on which we base our implementation. In these equations and throughout we use the convention that $i, j, k...$ label occupied orbitals, $a, b, c...$ label virtual orbitals, $n$ labels the whole range of molecular orbitals and $\alpha$,$\beta$ label atomic orbitals. The notation $o$ and $v$ are also used in general to describe occupied and virtual indices.

$$\begin{aligned}\sigma_i^a &= h_c^a t_i^c - h_i^k t_k^a + f_i^a + h_c^k\left(2t_{ki}^{ca} - t_{ik}^{ac} + t_i^c t_k^a\right) \\ &\quad + \left[2(ck|ai) - (ik|ac)\right]t_k^c + \left[2(ck|ad) - (dk|ac)\right]\tau_{ki}^{cd} \\ &\quad - \left[2(ck|il) - (cl|ik)\right]\tau_{kl}^{ca}\end{aligned} \tag{5.6}$$

$$\begin{aligned}\sigma_{ij}^{ab} &= \frac{1}{2}(ia|jb) + \frac{1}{2}a_{ij}^{kl}\tau_{kl}^{ab} + b_{ab}^{cd}\tau_{ij}^{cd} + g_c^a t_{ij}^{cb} - g_i^k t_{kj}^{ab} - f_k^c(t_{ij}^{cb}t_k^a + t_{ik}^{ab}t_j^c) \\ &\quad + \left[(ia|bc) - (ik|bc)t_k^a\right]t_j^c - \left[(ai|jk) + (ai|ck)t_j^c\right]t_k^b \\ &\quad + (j_{ic}^{ak} - \frac{1}{2}k_{ic}^{ka})(2t_{kj}^{cb} - t_{kj}^{bc}) - \frac{1}{2}k_{ic}^{ka}t_{kj}^{bc} - k_{ic}^{kb}t_{kj}^{ac}\end{aligned} \tag{5.7}$$

The intermediate quantities are:

$$h_i^k = (1-\delta_{ki})f_i^k + [2(kc|ld) - (kd|lc)]\,\tau_{il}^{cd} + f_c^k t_i^c \tag{5.8}$$
$$h_c^a = (1-\delta_{ca})f_c^a - [2(kc|ld) - (kd|lc)]\,\tau_{kl}^{ad} - f_c^k t_k^a \tag{5.9}$$
$$h_c^k = f_c^k + [2(kc|ld) - (kd|lc)]\,t_l^d \tag{5.10}$$
$$g_i^k = h_i^k + [2(ik|cl) - (il|ck)]\,t_l^c \tag{5.11}$$
$$g_c^a = h_c^a + [2(ac|dk) - (ad|ck)]\,t_k^d \tag{5.12}$$
$$a_{ij}^{kl} = (ik|jl) + (ik|cl)t_j^c + (ck|jl)t_i^c + (kc|ld)\tau_{ij}^{cd} \tag{5.13}$$
$$b_{cd}^{ab} = \frac{1}{2}(ac|bd) - (ac|dk)t_k^b \tag{5.14}$$
$$j_{ic}^{ak} = (ai|ck) - (il|ck)t_l^a + (ad|ck)t_i^d - \frac{1}{2}(ck|dl)(t_{il}^{da} + 2t_i^d t_l^a) + \frac{1}{2}[2(ck|dl) - (dk|cl)]\,t_{il}^{ad} \tag{5.15}$$
$$k_{ic}^{ak} = (ik|ac) - (ik|cl)t_l^a + (dk|ac)t_i^d - \frac{1}{2}(dk|cl)(t_{il}^{da} + 2t_i^d t_l^a) \tag{5.16}$$
$$\tau_{ij}^{ab} = t_{ij}^{ab} + t_i^a t_j^b \tag{5.17}$$

A crucial part of any efficient implementation is the consideration of memory, disk space, operation count, and when dealing with massively parallel computers, interprocessor communication and disk i/o. Not all these factors are independent of each other and sometimes it is necessary to compromise to find the best balance. With this in mind we have specially designed an algorithm to run optimally on massively parallel machines. For the coupled-cluster equations the costly quantities are the $b$, $j$ and $k$ terms which scale as $n^6$ in operation count, and in in terms size, the integrals containing three or four virtual indices. The chief limitation of the earlier Molecular Orbital (MO)-driven coupled cluster implementation of Rendell et al. [62] was in handling these integrals. As they could not, in general, be held in global memory they were written to disk, leading to potential i/o bottlenecks. To overcome this problem it was decided to replace the contribution from these integrals using Atomic Orbital (AO) integrals driven in a 'direct' fashion. To best balance the number of operation counts, amount of communication, degree of parallelisation and use of local memory among other factors the following strategy was settled on.

The structure driving the solution of the sigma equations (above) in the original code (Scheme 1)

do a = 1,nvirt<br>
do b = 1,a

| | |
|---|---|
| get $t_{oo}^{av}$, $t_{oo}^{bv}$ | amplitudes from global memory |
| get [ao\|oo], [av\|oo], [ao\|vo] | |
| [bo\|oo], [bv\|oo], [bo\|vo] | integrals from global memory |
| read [ab\|vo], [av\|bo], [ab\|vv] | integrals from disk |
| form $\sigma_o^v$, $\sigma_{oo}^{av}$, $\sigma_{oo}^{bv}$ | |

enddo

enddo

*Scheme 1: Old scheme for evaluating the CCSD sigma equations*

was replaced by that in Scheme 2.

back transform $t^{vv}_{oo}$ to $t^{\alpha v}_{oo}$

do ish=1,nsh
do jsh=1,ish

generate AO integrals $(\alpha\beta|\gamma\delta)$, $(\alpha\gamma|\beta\delta)$
transform to partial mo integrals $(\alpha\beta|nn)$, $(\alpha n|\beta n)$

do i=$\alpha_{lo}$,$\alpha_{hi}$ — range of basis functions
do j=$\beta_{lo}$,min(i,$\beta_{hi}$) — corresponding to shell

get $t^{av}_{oo}$, $t^{bv}_{oo}$, $\tilde{t}^{av}_{oo}$, $\tilde{t}^{bv}_{oo}$ — amplitudes from global memory
get [$\alpha$o|oo], [$\alpha$o|vo], [$\beta$o|oo], [$\beta$o|vo] — integrals from global memory

form $\sigma^{v}_{o}$, $\sigma^{\alpha v}_{oo}$,$\sigma^{\beta v}_{oo}$

enddo
enddo

transform $\sigma^{\alpha v}_{oo}$ to $\sigma^{vv}_{oo}$

*Scheme 2: Scheme for evaluating ccsd sigma equations in a 'direct' manner.*

Thus the sigma equations are distributed over pairs of shells as $nsh * (nsh + 1)/2$. Computational load is balanced in the shell ordering since shells with higher angular momentum basis functions are dealt with first.

The $\sigma$ equations now have the form

$$\sigma^a_i = h^c_a t^c_i - h^k_i t^a_k \tag{5.18}$$

$$\begin{aligned}\sigma^\alpha_i &= h^k_c(2\tilde{t}^{c\alpha}_{ki} - \tilde{t}^{c\alpha}_{ik} + t^c_i \tilde{t}^\alpha_k) \\ &+ [2(\beta k|\alpha i) - (ik|\alpha\beta)]\, t^\beta_k + [2(\beta k|\alpha d) - (dk|\alpha\beta)]\, \tau^{\beta d}_{ki}\end{aligned} \tag{5.19}$$

$$\sigma^a_i = [2(\alpha k|il) - (\alpha l|ik)]\, \tau^{\alpha a}_{kl} \tag{5.20}$$

$$\begin{aligned}\sigma^{\alpha b}_{ij} &= \frac{1}{2}(i\alpha|jb) + \frac{1}{2} a^{kl}_{ij}\tau^{\alpha b}_{kl} + b^{\beta d}_{\alpha b}\tau^{\beta d}_{ij} + h^c_b \tilde{t}^{\alpha c}_{ij} \\ &+ [2(\alpha\beta|dk) - (\alpha d|\beta k)]\, t^d_k t^{\beta b}_{ij} - g^k_i t^{\alpha b}_{kj} \\ &+ [(i\alpha|b\beta) - (ik|b\beta)t^\alpha_k]\, t^\beta_j - \left[(\alpha i|jk) + (\alpha i|\beta k)t^\beta_j\right] t^b_k \\ &+ (j^{\alpha k}_{i\beta} - \frac{1}{2}k^{k\alpha}_{i\beta})(2t^{\beta b}_{kj} - t^{b\beta}_{kj}) - \frac{1}{2}k^{k\alpha}_{i\beta} t^{b\beta}_{kj} - k^{k\alpha}_{i\beta} t^{a\beta}_{kj}\end{aligned} \tag{5.21}$$

with the corresponding intermediates analogous to those in Eqns. (5.8 – 5.17).

The $h$, $g$ and $a$ intermediates, Eqns. (5.8 – 5.17) and the energy only depend on integrals of at most two occupied and two virtual indices and are evaluated in the MO basis for convenience. These integrals were obtained by slightly modifying an MP2 program, called prior to the coupled cluster code, which was also used to generate the integrals with one AO index. These remaining parts of the code were distributed as much as possible over the number of virtual orbitals.

The above structure of the equations loses some of the efficiency of the MO formalism in that it loses the $a \leftrightarrow b$, $i \leftrightarrow j$ interchange symmetry and it introduces an extra set of amplitudes transformed with respect to the inverse MO coefficient matrix. However, at this point our primary concern was to minimise the amount of communication and local memory usage. The amount of available local memory was the main limiting factor of this implementation. After allocating memory for the program and other fixed quantities we were left with 45 MB to divide between local and global memory. The main quantities stored in global memory were species of size $o^2v^2$. The main quantities in local memory were of size $o^2v$ and the partial MO integrals $(\alpha\beta|nn)$,$(\alpha n|\beta n)$. The memory required for the integrals could be reduced further by evaluating the integrals in batches as was done for the perturbative triples correction described next.

### 5.2.2 The Perturbative Triples Correction

To take into account fully the triple excitation contributions in the coupled cluster framework the cluster operator in Eq. (5.2) should be truncated after T3. However, this results in an iterative $n^8$ cost which is still considered to be largely impractical. Therefore, Raghavachari et al. [61] came up with the idea of a perturbative triples correction. This correction, denoted (T), is an estimate from perturbation theory of the contribution of the triple excitations and is evaluated using the optimised cluster amplitudes at the end of a CCSD calculation. The formula has been given in compact form by Lee et al. [25] using sums over occupied and virtual orbitals:

$$E_{(T)} = -\frac{1}{3}\sum_{ijk}^{occ}\sum_{abc}^{vir}\left(4W_{ijk}^{abc} + W_{ijk}^{bca} + W_{ijk}^{cab}\right)\left(V_{ijk}^{abc} - V_{ijk}^{cba}\right)/D_{ijk}^{abc} \tag{5.22}$$

where

$$W_{ijk}^{abc} = X_{ijk}^{abc} + X_{jik}^{bac} + X_{kji}^{cba} + X_{ikj}^{acb} + X_{jki}^{bca} + X_{kij}^{cab} \tag{5.23}$$

$$X_{ijk}^{abc} = \sum_d (bd|ai)t_{kj}^{cd} - \sum_l (ck|jl)t_{il}^{ab} \tag{5.24}$$

$$V_{ijk}^{abc} = W_{ijk}^{abc} + (bj|ck)t_i^a + (ai|ck)t_j^b + (ai|bj)t_k^c \tag{5.25}$$

$$D_{ijk}^{abc} = \epsilon_a + \epsilon_b + \epsilon_c - \epsilon_i - \epsilon_j - \epsilon_k \tag{5.26}$$

This arrangement of the triples correction is not however easy to parallelise and a form similar to that for MP4 by Rendell *et al.* [63] was used instead. In this version, the triples energy is calculated as a different grouping of the various permutations of $X$ stored in local memory as $f(b,c)$. From Eq. (5.24) it can be seen that the triples correction requires integrals with three virtual indices and these are calculated in batches depending on how much global memory is available. Thus in our implementation we evaluate these integrals for a range $a$ of the virtual indices, compute the contribution to the triples energy and then go onto the next batch of integrals. The parallel task is within the $a, i$ loops and all other integrals are read from global memory. This gives a loop structure as in Scheme 3.

| | |
|---|---|
| Work out how many integrals will fit in memory. | |
| Compute integrals of the form (ao\|vv),(av\|ov) | where a is the range of virtual indices corresponding to the integral block. |
| do a=a$_{lo}$,a$_{hi}$ | |
| get $t_{oo}^{av}$ | from global memory |
| get integrals (av\|vo), (ao\|vv) | from global memory |
| do i=1,nocc | |
| get $t_{io}^{vv}$ | from global memory |
| get integrals (iv\|oo), (io\|vo) | from global memory |
| do j,k=1,nocc | |
| do b,c=1,nvir | |
| form $f_{1n}(b,c) = \sum_d (bd\|ai) t_{kj}^{cd} - \sum_l (ck\|jl) t_{il}^{ab}$ | |
| similarly form $f_{1t}$, $f_{2n}$, $f_{2t}$,$f_{3n}$, $f_{3t}$, $f_{4n}$, $f_{4t}$ | |
| evaluate contribution to the triples energy | |
| end all loops | |

*Scheme 3: Scheme for evaluating the perturbative triples correction to the CCSD energy in a 'direct' manner.*

### 5.2.3 Sample Calculations

To demonstrate the performance of the code we have carried out two sets of calculations on the Cray T3D using the NWChem package. The first, on 2-hydroxypyridine, was the largest case that we could do under the current implementation due to local and global memory requirements, and required all 256 processors available. The second, on glycine, was to examine the scalability of the code. For 2-hydroxypyridine, the geometry used was that optimised at the SCF level using Dunning's cc-pVTZ basis (Dunning [14]). For the actual energy calculation, this basis was used without the f functions on the heavy atoms to make the calculation feasible. This still left 220 basis functions with 18 active occupied orbitals and 195 active virtuals. The perturbative triples correction required 195 integral passes which was deemed to take too long to be practical. The timings for the various parts of the code for one CCSD iteration are given in Table 5.2. Also given is the wall-clock time for the total calculation which took 9 iterations to converge to $10^{-6}$ in energy units.

In order to examine the scalability of the implementation the smaller case of glycine was chosen at the 3-21G SCF-optimised geometry using the Dunning cc-pVDZ basis set. CCSD(T) calculations were carried out on 32,64,128 and 256 processors and the results are given in Table 5.3.

From the results it can be seen that the code scales well even up to 256 processors. The apparent superscalability of some of the terms arises from our use of memory where available. Advantage was taken of the storing of integrals in both the CCSD sigma equations and the perturbative triples correction. In the CCSD sigma equations the first CCSD iteration evaluates all the integrals and stores as many as possible in the available memory. All subsequent CCSD iterations read in these integrals (the percentage of integrals stored is given in parentheses). Similarly, for the perturbative triples correction the amount of available memory dictates the number of integral evaluation passes required (also given in parentheses). The lack of change in the second CCSD iteration in going from 128 to 256 processors is probably an indication that communications are starting to dominate at this stage. An encouraging feature of this parallel implementation is that the code seems to be efficient and can produce large CCSD results quite rapidly in real time.

Table 5.2: Breakdown of time taken for one CCSD Iteration for 2-hydroxypyridine on 256 Processors.

| Time taken on one processor (in cpu s) | |
|---|---|
| Forming intermediates | 2.4 |
| Transforming amplitudes | 1.5 |
| Forming inverse coefficients | 5.6 |
| Generating and transforming integrals | 356.8 |
| Forming sigma in AO basis | 158.5 |
| Forming sigma in MO basis | 1.7 |
| Various overheads | 46.7 |
| **Total** | **573.2** |
| **Total wall-clock time** | **5196** |

SCF energy -321.672172 a.u.
MP2 energy -322.788800 a.u.
CCSD energy -322.805919 a.u.

Table 5.3: CCSD(T) Timings for Glycine on 32,64,128 and 256 Processors (7 Iterations).

| Time taken on one processor (in cpu s) | Number of processors | | | |
|---|---|---|---|---|
| | 32 | 64 | 128 | 256 |
| 1st CCSD iteration | 267 | 142 | 78 | 58 |
| 2nd CCSD iteration (%integrals stored) | 236 (33%) | 66 (98%) | 42 (100%) | 42 (100%) |
| One triples integral pass | 356 | 201 | 102 | 53 |
| Triples correction (no. passes) | 2176 () | 1238 () | 600 () | 310() |
| Generating/transforming integrals | 513 | 78 | 38 | 23 |
| Forming sigma vectors | 422 | 225 | 107 | 54 |
| **Total** | **3606** | **1813** | **916** | **559** |
| **Total wall-clock time** | **4671** | **2051** | **1118** | **750** |

SCF energy -282.850714 a.u.
MP2 energy -283.684162 a.u.
CCSD energy -283.713845 a.u.
CCSD(T) energy -283.739775 a.u.

**Acknowledgment**

The NWChem program package was developed at Pacific Northwest Laboratory, PO Box 999, Mail Stop K1-96, Richland, WA 99352.

## 5.3 GAMESS-UK

*G.D. Fletcher, P. Sherwood, M.F. Guest*

In this section we provide a description of the results of the parallelisation of the GAMESS-UK program, conducted largely within the Esprit-funded IMMP (Interactive Modelling through Parallelism) consortium of EUROPORT 2. Partners in the IMMP project include CCLRC Daresbury Laboratory, Vreije Universiteit Amsterdam, FAU Erlangen-Nürnberg, ICI (Wilton), Zeneca, and Oxford Molecular. We describe below the principles behind the parallel implementation, and report the results of related benchmarking exercises.

During the past 15 years development work on GAMESS-UK has included both extensions to the program's functionality and efforts to optimise the performance of the code on the current state-of-the-art hardware. This has included particular attention to vector supercomputers, such as the Cray XMP, YMP, C90, Cyber 205 and pipeline processors such as the FPS x64 series. Present functionality of the code lies at varying levels of approximation, including SCF (RHF, UHF, CASSCF, MCSCF, GVB), MP2, MP3, MP4, CCSD(T), CI (MRDCI, Direct-CI, Full-CI) and analytic SCF and MP2 second derivatives

As the first parallel machines were introduced, the potential for greatly enhanced price-performance became apparent and work to move GAMESS-UK to parallel hardware was started. Initial implementations included those for the Intel iPSC-2 and iPSC/860 hypercubes installed at Daresbury Laboratory, and Alliant FX4 and i860-based shared-memory hardware. The early Intel parallel code, including parallel direct SCF capability, was based on the Intel NX operating system and message-passing library, and has formed the starting point for work within EUROPORT. The package is currently available on Cray Y-MP, C-90 and T3D, Intel iPSC/860, Kendall Square KSR-2 and IBM SP2 platforms, plus a wide variety of workstations. There are over 60 academic sites and a number of industrial users, including ICI, Shell, Unilever, Zeneca and Norsk Hydro.

### 5.3.1 Overview of Applications Area

The potential of *ab initio* molecular electronic structure calculations in industry is widely recognised, with the ability to solve many challenging problems in, for example, the design of drugs, chemicals and polymers and materials, and to understand the molecular problems relevant to combustion, atmospheric chemistry, chemical engineering of processes and environmental restoration, treatment of waste etc. GAMESS-UK represents a typical electronic structure code, comprising some 300,000 lines of Fortran-77 that permits a wide range of computational methodology to be applied to molecular systems; other such codes include the GAUSSIAN series, CADPAC, MOLPRO, Turbomole, HONDO etc. Although not all of the methods contained within these codes are strictly relevant to industrial chemistry, the scaling properties of many of the methods involved has meant a limitation historically to rather small systems, and only semi-quantitative accuracy. The development of MPP architectures, with the promise of significantly enhanced performance, promises to extend the size of system amenable to treatment, and to increase the predictive capability of the software. The capability targeted with EUROPORT includes further development of the parallel Hartree Fock SCF and gradient module (perhaps the most widely used *ab initio* technique in industry), and development of two features (MP2 and SCF second derivatives) that historically have been little used in industry due to computational expense.

### 5.3.2 Parallel Developments

The work on GAMESS-UK within IMMP has addressed a number of aspects of the program's functionality, outlined below.

**Parallel Direct SCF and Gradient**

The SCF program is parallelised in a replicated data fashion, each node maintaining a copy of all data structures present in the serial version. The main source of parallelism is the computation of the one- and two-electron integrals and their summation into the Fock matrix. The one-electron integrals and derivative integrals are distributed using a simple round-robin allocation of tasks followed by global summation, whereas the more costly two-electron quantities are allocated dynamically using a shared global counter. The result of parallelism implemented at this level is a code scalable to a modest number of processors (around 32), at which point the cost of other components of the SCF procedure starts to become relatively significant. The first of these to be addressed is the diagonalisation (Jacobi), which is replaced in the parallel code by a distributed Householder algorithm. This is the stage of parallelisation reached at the point the first IMMP deliverable was presented, all code being based on a message-passing model. Once the capability for Global Array storage [39] had been added to support the MP2 and SCF second-derivative modules, some distribution of the linear algebra was trivial. As an example, the SCF convergence acceleration algorithm Direct Inversion in the Iterative Subspace (DIIS), module was distributed using Global Array storage for all matrices, and parallel matrix multiply and dot-product functions. This not only reduced the time to perform the step, but the use of distributed memory storage (instead of disk) reduced the need for i/o during the SCF process.

**Parallel MP2 Energy and Gradient**

Substantial modifications were required to enable the MP2 gradient to be computed in parallel. Specifically, the conventional integral transformation step, (in which two-electron integrals in the AO basis are read from disk, partially transformed, sorted, transformed to the MO basis, and written back out to disk) has been omitted. The SCF step is performed in a direct fashion (i.e. without integral storage), and the MO integrals generated by recomputation of the AO integrals, and stored in the global memory of the parallel machine. The storage and subsequent access is managed by the Global Array tools, which implement a portable virtual shared memory model. The basic principle by which the subsequent steps are parallelised involves each node computing a contribution to the current term from the MO integrals resident on that node. For some steps, however, more substantial changes to the algorithms are required. In the construction of the Lagrangian (the right hand side of the CPHF equations) for example, MO integrals with three virtual orbital indices are required. Due to the size of this class of integrals they are not stored, but the required terms of the Lagrangian constructed directly from AO integrals. A second major departure from the serial algorithm concerns the MP2 two-particle density matrix. This quantity, which is required in the AO basis, is of a similar size to the two-electron integrals and is stored on disk in the conventional algorithm, but generated as required during the two-electron derivative integral generation from intermediates stored in the Global Arrays.

**Parallel SCF Second Derivatives**

Like the MP2 gradient, the computation of the SCF second derivative module includes a number of terms constructed from two-electron integrals in the MO basis, and the algorithm is adapted in a similar way. The conventional transformation is replaced by a direct transformation, with integrals written to Global Array storage. These integrals are used in the CPHF step and in the construction of perturbed fock matrices, with these steps therefore parallelised according to the distribution of the MO integrals. Thus the main steps involved in the parallel implementation are:

1. perform SCF run using dynamic load balancing with a shared counter scheme.
2. SCF gradient calculation also using load balancing giving $S^x$ and $F^{(x)}$.
3. 1- and 2-electron integral transformations and one-electron operators is done serially.
4. MO integral classes VOVO, VVOO, VOOO, and OOO are generated and a parallel MP2 gradient transformation is performed.
5. CPHF for each perturbation to obtain $P^x_{(OV)}$ for each perturbation $x$

   $\sum G_{iajb} P^x_{jb} + (\epsilon_a - \epsilon_i) P^x_{ia} = L^x_{ia} = \sum S^x_{jk} G_{aijk} + \epsilon_i S^x_{ia} - F^{(x)}_{ia}$
   Formation of $L^x$ is integral driven. The Hessian $G_{iajb}$ is distributed, the product of Hessian and trial vectors is parallel. Trial vectors are globally summed and a final small set of linear equations are solved serially.
6. 1-electron integrals (dipole, quadrupole and first and second derivatives) are computed serially.
7. 2-electron second derivative integrals are computed using the load balancing method.

The most costly step in the serial second derivative algorithm is the computation of the second derivative two-electron integrals. This step is trivially parallelised using the approach adopted in the direct SCF scheme – using dynamic load-balancing based on a shared global counter.

### 5.3.3 Test Cases and Benchmarks

In the examples below we identify the nature of the calculations undertaken, the industrial interest in the benchmark, and the performance obtained across a variety of parallel platforms. In addition to the 16-node IBM SP2 and HP/9000-735 workstation cluster, a variety of benchmark platforms were available to the IMMP Consortium, including a 512-node IBM SP2 at the Cornell Theory Centre, a 64-node Parsytec GC machine at Paderborn, and an SGI PowerChallenge Array at Cortailloid in Switzerland (with R8000 processor nodes running at 75 MHz and a main-memory configuration of 2-2.5 Gb). Full details of these machines are given in the Appendix I.

**Titanium Chloride on Magnesium Chloride Support**

Titanium chloride supported on magnesium chloride is a well know catalyst for the production of polypropylene as well as other polymers. The benchmark illustrates the ability of quantum codes to incorporate solid state effects in order to treat problems in heterogeneous catalysis.

In this example, a cluster is used to represent the surface of $MgCl_2$. The system consists of 2 Mg ions, 4 Cl ions and a Ti atom treated quantum mechanically. Another 33 ions are treated as point charges. The functionality tested includes the SCF Energy and first derivatives, the calculation being conducted in a TZVP basis (see table 5.4).

Table 5.4: GAMESS-UK Benchmark - Titanium Chloride (times in seconds)

| | IBM SP2 | | SGI Power Challenge | | Parsytec GC | | Workstation Cluster | |
|---|---|---|---|---|---|---|---|---|
| Nodes | Time | Speed-up | Time | Speed-up | Time | Speed-up | Time | Speed-up |
| 1 | 12874 | [1] | 20987 | [1] | | | 17250 | [1] |
| 2 | 6471 | 2.0 | 10576 | 2.0 | | | 8836 | 2.0 |
| 4 | 3279 | 3.9 | 5311 | 3.9 | 11099 | [4] | 4626 | 3.7 |
| 8 | 1735 | 7.4 | 2271 | 9.2 | 6538 | 6.8 | | |
| 16 | 1196 | 10.8 | 1471 | 14.4 | 4293 | 10.3 | | |
| 32 | 540 | 24.0 | | | 3302 | 13.0 | | |
| 64 | 374 | 34.0 | | | | | | |
| 128 | 283 | 45.0 | | | | | | |

The following points should be noted:

- Comparison of the 1-node times suggests that some improvement in the single-processor SGI performance may be possible, as this processor typically performs comparably to the RS/6000 RIOS-2 architecture.
- The FDDI network was not dedicated – a 5-node workstation cluster time of 4742 seconds probably reflected traffic on the network.

**Cyclosporin ($C_{63}H_{113}N_{11}O_{12}$)**

Cyclosporin-A is a powerful immune-system suppressant valuable in organ transplant post-operative treatment. It was used as a "proof-of-principle" study for the vectorised Direct-SCF module of GAMESS-UK on the FPS 164 in 1989 (the SCF calculation took nearly 3 weeks to perform). The functionality tested is the SCF energy, with the calculation performed in a 6-31G split-valence basis.

Table 5.5: GAMESS-UK Benchmark - Cyclosporin (times in seconds)

| | IBM SP2 | | Workstation Cluster | |
|---|---|---|---|---|
| Nodes | Time | Speed-up | Time | Speed-up |
| 1 | | | 286579 | [1] |
| 2 | | | 149412 | 1.9 |
| 5 | | | 70347 | 4.2 |
| 8 | 34709 | [8] | | |
| 16 | 18586 | 15.0 | | |
| 32 | 10203 | 27.0 | | |
| 64 | 6755 | 41.0 | | |

Note that the timings on the SP2 (Table 5.5) include parallel DIIS and diagonalisation, which was not employed on the workstation cluster

**The Chlorotriazine/$OH^-$ Transition State**

An important section of the Colours business in Zeneca Specialities is that involved in reactive dyeing. This includes the dyeing of fibres such as cotton, in which dyestuffs are attached directly to the cellulose fibres by a covalent bond from a linking unit known as a reactive group. In Zeneca, the most commonly used such group is chlorotriazine, which binds to cotton via a nucleophilic replacement of the chlorine with alkoxy groups from the cellulose chain. In competition with this is hydrolysis of the chlorotriazine by hydroxide ions. The product of this reaction is a coloured molecule with a good affinity for the fibre, but which cannot form a permanent bond. Thus it is a costly effort for the dye house to wash out the hydrolysed dye, and it means that the dyeing process is not 100% efficient.

It is therefore of interest to understand the underlying reactivity of the chlorotriazine group. To do this and to gain understanding of the different reaction rates of the incoming nucleophiles, it is necessary to find the transition state of each of the reactions. Even at semi empirical levels of theory, this can be a difficult and time-consuming process, and it often requires *ab initio* levels of theory to give reliable transition states. Transition state location requires the calculation of second derivatives (or Hessian) in two ways :

- The initial transition state search can be greatly improved by the computation of the Hessian
- The final geometry has to be characterised to make sure it is a true transition state . This involves a Hessian calculation, and should yield exactly one negative (or imaginary) vibrational frequency.

In addition, if a high enough level of theory is used, it is also possible to use second derivative calculations to predict and interpret infrared and Raman spectra of molecules.

The benchmark run is on a calculated transition state geometry obtained using MOPAC and the COSMO solvation method. Obviously, the geometry would have to be further refined at the *ab initio* level before a true transition state can be found, but the calculation serves to illustrate the requirements of such a calculation. The calculation involves a single point Hessian calculation using either (a) a DZ basis set (for 1-16 nodes, see table 5.6), or (b) a TZVP basis set (185 basis functions, see table 5.7) on 8 and more nodes.

Table 5.6: GAMESS-UK SCF 2nd Derivatives Benchmark - Chlorotriazine/$OH^-$ (DZ, seconds)

| | IBM SP2 | | SGI Power Challenge | |
|---|---|---|---|---|
| Nodes | Time | Speed-up | Time | Speed-up |
| 1 | | | 2807 | [1] |
| 2 | 1491 | [2] | 1432 | 2.0 |
| 4 | 819 | 3.6 | 765 | 3.7 |
| 8 | 445 | 6.7 | | |
| 16 | 277 | 10.8 | | |

The following points should be noted:

- A time of 3829 seconds (elapsed) was required for the serial version of the code (HP/9000-735, conventional algorithm, DZ basis).
- The parallel requirements were 6 MW of RAM (13 MW if one-node case, basis a) is run) and 200 MB/node of disk space.

Table 5.7: GAMESS-UK SCF 2nd Derivatives Benchmark - Chlorotriazine/$OH^-$ (TZVP, seconds)

| | IBM SP2 | |
|---|---|---|
| Nodes | Time | Speed-up |
| 8 | 3400 | [8] |
| 16 | 1829 | 15.0 |
| 32 | 1216 | 22.0 |
| 64 | 806 | 34.0 |
| 128 | 705 | 39.0 |

- A 4-processor HP/9000-735 workstation cluster time of 3462 seconds clearly indicates the unsuitability of the platform for calculations of this type. The processor utilisation is low as a results of the high-latency of the communications and the fact that the GA-Tools implementation for a workstation cluster requires an extra process on each node to manage the global memory
- Memory requirement for this type of calculation are extremely heavy, making single processor calculations difficult.

### MP2 Calculations on Cytosine and Trinitrotoluene

Cytosine (an amino-acid) and Trinitrotoluene (TNT, an explosive) are medium-sized molecules, representative of systems studied in the pharmaceutical and materials industries respectively. Calculations on cytosine (6-31G basis, 82 basis functions, see table 5.8) and tri-nitrotoluene, (6-31G** basis, 265 basis functions, see table table 5.9) are included to test the MP2 gradient functionality of the program.

Table 5.8: GAMESS-UK MP2 Gradient Benchmark - Cytosine (times in seconds)

| | IBM SP2 | | SGI Power Challenge | |
|---|---|---|---|---|
| Nodes | Time | Speed-up | Time | Speed-up |
| 1 | | | 1403 | [1] |
| 2 | | | 705 | 2.0 |
| 4 | 529 | [4] | 355 | 3.9 |
| 8 | 228 | 9.2 | 178 | 7.9 |
| 16 | 129 | 16.4 | 100 | 14.0 |
| 32 | 77 | 27.5 | | |
| 64 | 53 | 39.9 | | |

Table 5.9: GAMESS-UK MP2 Gradient Benchmark - Trinitrotoluene (times in seconds)

| | IBM SP2 | |
|---|---|---|
| Nodes | Time | Speed-up |
| 16 | 5473 | [16] |
| 32 | 3308 | 26.5 |
| 64 | 2403 | 36.4 |
| 128 | 1589 | 55.1 |

The following points should be noted:

- A serial time of 1627 seconds was obtained using the conventional algorithm on the HP/9000-755.

- The observed superlinear scaling of the IBM numbers on the DZ test case are probably the results of an artificially slow 4-node time, rather than unexpected efficiency of the parallel code. The 4-node test was performed on the interactive pool at Cornell and it is possible the node was used by another task.

### 5.3.4 Lessons Learned

#### Ease of Use of Parallel Platforms

1. *IBM SP2*: Our experiences on the SP2 were positive, both interactive and batch runs were performed. Some knowledge of IBM LoadLeveler was essential, as was an understanding of the LoadLeveler pools implemented at the site (a local feature). Once this is established (and Cornell have invested greatly in making information available on the WWW) the machine is very reliable. We have experience with the compilers, which now appear to be quite reliable. Some unexpected failures did occur for example nodes running out of paging space or jobs failing to start. The system is not 100% predictable, but is clearly a genuine production machine.

2. *SGI Power Challenge*: This platform only offers modest parallelism unless multiple nodes are linked together (which we didn't get chance to test) but within this constraint it is an impressive machine. Not only is it possible for the machine to carry a mixed workload (of serial and parallel jobs) but the symmetric multiprocessing O/S can redistribute the tasks to use the nodes most effectively, a feature not yet available on distributed-memory machines. This allows interactively parallel work to be performed much more readily, as there is no need to ensure free nodes are available, if they become available during a run they will be effectively exploited. The platform also has the obvious advantage that the core memory available is independent of the number of nodes employed - large memory jobs can be run on small numbers of processors if this is required, either for timing or if this is appropriate given the nature of the existing load. The main disadvantage of the configuration is that dedicated nodes are difficult to guarantee, so a failing of the set-up at Cortailloid was the lack of a batch system for long jobs that needed dedicated processors for timing purposes.

3. *Parsytec GC*: Running in the Parsytec environment was complicated by PC2's use of the CCS front-end. We still don't understand it well enough to consider whether it is a good environment for production work - but we are not aware of any batch capability. The compilers were well below the quality expected for commercial production machines, especially during the early stages of the project, when even test programs proved to large to link without taking extra steps.

4. *Workstation Cluster*: The workstation cluster is quite simple to use, in most respects comparable to a single workstation. The extra effort required for parallel processing falls to the system manager, rather than the user, as consistent user accounts and file systems (permanent and scratch) need to be set up across the machine. In addition, most workstation clusters need additional software installing to provide batch capabilities, we use DQS (Distributed Queuing System) on our HP cluster, but it was not trivial to set up.

#### Performance of the Parallel Platforms

Both the IBM SP2 and SGI PowerChallenge array may be considered to have performed well in our benchmarking studies. Both these machines are based on competitive processors, and both have high-performance interconnects. Performance of the coarse-grained parallel direct-SCF module is generally similar on these two platforms. The parallel modules based on the Global Array model generate the most significant communication, and performance is expected to be highly sensitive to the message-passing latency in particular. The SGI seems to perform particularly well on such codes, which is to be expected as it is shared-memory machine. The SP2, while still showing speed-up as further processors are added, clearly does not show such clear scalability. However, such an analysis may be rather misleading, as the SGI timings were all collected on small machines (1-16 nodes), while the SP2 results (because of the memory requirements) could typically only be run on 8 processors or above. As noted elsewhere, it is one of the benefits of the SGI shared-memory architecture that the full memory of the machine ,may be utilised without the need to parallelise over all processors. A fairer analysis of the SP2 and the PowerChallenge array will require an SP2 configuration with some large-memory nodes, and larger SGI configurations, spanning more than one shared-memory node. We hope to be able to carry out such benchmarks, but it is not clear it will be possible to do so within the timescale of EUROPORT.

In contrast, the Parsytec GC was not felt to be a high-performance machine. The processor is considerably slower than the RIOS-2 or R8000 series, and, by comparison. the communication system shows large latencies and low bandwidths. Even for the coarse-grained direct-SCF module performance was poor, probably because the machine could not support the I/O activity that occurs in this module (even though the I/O is performed by node 0 only with results broadcast to the other nodes).

The workstation cluster is a viable platforms for cost-effective conventional and direct SCF calculations, but does not have the interconnect performance (or, in general, the memory) to support Global-Array based virtual-shared-memory codes such as the parallel MP2 and second-derivative modules of GAMESS-UK.

Finally, we should mention the crucial role of I/O capabilities in the development of parallel electronic structure codes, One of the principle differences between the current generation of parallel platforms is the availability of high-performance I/O from the nodes. The lack of such a capability has always been one of the weaknesses of parallel computers, especially in the field of computational chemistry, where codes have traditionally relied heavily on disk storage. Of the machines benchmarked, each provides a different implementation and the exploitation (or not) of the capabilities of the a given machines will generally affect the scalability observed.

## 5.4 MOLPRO

*A.J. Dobbyn, P.J. Knowles and R.J. Allan*

There are several important factors that need to be considered when calculating potential energy surfaces (PESs) that will be used in dynamical studies of chemical reactions:

- The surfaces must be global, i.e. cover not only the interaction region but extend well into both the entrance and exit channels. In order to achieve this a very large number of points have to be calculated, e.g. $\approx O(10^3)$ for a triatomic system.

- The surfaces must have uniform accuracy. This necessitates the use of multi-reference configuration interaction (MRCI) methods, which in contrast to other methods that utilise only a single reference configuration, are capable of producing correct results in the dissociation channels, as well as in the interaction region. These methods are inevitably very costly in time, as well as requiring quite large amounts of memory.

These two factors imply that to obtain a high quality surface for the study of a chemical reaction will be very expensive. To make such calculations possible, we thus require highly efficient computer programs and the use of excellent computer facilities, e.g. the supercomputing facilities available in Edinburgh and at Daresbury.

**Example Application**

The exothermic reaction:

$$\mathrm{O}(^1D) + \mathrm{H}_2(^1\Sigma_g^+) \rightarrow [\mathrm{H}_2\mathrm{O}]^* \rightarrow \mathrm{OH}(X\,^2\Pi, A^2\Sigma^+) + \mathrm{H}(^2S),$$

is of fundamental importance to many branches of chemistry. It is one of the essential reactions in combustion and in the atmosphere. Surprisingly, there is still no clear consensus as to the mechanism by which this reaction occurs, despite many theoretical and experimental studies.

Three potential energy surfaces (PESs) are required to describe this reaction: the $\tilde{X}(A_1)$, $\tilde{A}(B_1)$ and $\tilde{B}(A_1)$ states. The $\tilde{A}$ and $\tilde{B}$ states have Rydberg character at geometries not too far from the equilibrium on the ground state. For linear HOH the $\tilde{A}$ and $\tilde{B}$ states are degenerate, and the surfaces are coupled via the Renner-Teller interaction for small angular displacements from linearity. There are conical intersections between the $\tilde{X}$ and $\tilde{B}$ states for linear HOH and HHO, giving rise to avoided crossings away from linearity.

It is clear then that this reaction, with its excited Rydberg states and many nonadiabatic interactions, provides a huge challenge to any quantum chemistry program, which perhaps explains why so little work has been done to characterise the global PESs of this system. Thus the use of supercomputing facilities is essential to ensure that these calculations are carried out without a significant delay.

**MOLPRO**

MOLPRO is a complete system of *ab initio* programs for molecular electronic structure calculations. The particular emphasis in this quantum chemistry program is on highly accurate computations for small molecules. The methods utilised in the MRCI program, which forms a part of MOLPRO, are almost uniquely capable of dealing with the complexity of the problems under consideration here, as they are able to efficiently include a large proportion of the correlation energy, as well as being able to yield a balanced description of PESs.

**Approaches to Parallelisation**

Given the nature of the problem, i.e. the calculations at very many different geometries, it might seem the best strategy would be a data farming one, whereby different geometries were calculated on each node. Such a version of MOLPRO has been developed for the IBM SP2 computer at Daresbury. In this version of the program individual points on the surfaces are calculated on separate nodes of the computer. The SP2 is superbly equipped for these calculations: the single node performance is excellent, and the 1 GB scratch discs attached to each node make the large amount of i/o involved not only possible but also fast. This, however, is not a sensible approach for the T3D which not only has relatively poor i/o rates, but also has fairly limited amounts of disc space. It was thus necessary to develop a distributed data version of the code.

The first stage of this parallel implementation was to produce a version which could do replicated work on 'N' nodes. This mostly involved a reorganisation of the i/o subsystem. This was done in two ways. The first simply ensured that all the i/o went through one node: this single node writes the data to disc, reads the data and then broadcasts it to the others. The second method involved the complete mapping of the i/o into memory.

Once the replicated work version was written the next stage was to divide the work, and were possible, the arrays i.e. memory requirements, across the nodes. This has involved changes to over 30 subroutines, as the work done is not localised in only a few places. Unfortunately, this makes it difficult to achieve very good speed up rates, though those obtained do justify the use of 64 or 128 nodes.

The implementation of this parallel work has been carried out using **Global Array tools** (GA). This is a **portable** 'shared memory' programming model for distributed memory computers, developed at Pacific Northwest Laboratory by Jaroslaw Neiplocha, Robert J. Harrison and Richard J. Littlefield, specifically for use in parallel quantum chemistry codes. The major advantage of this programming model over a message-passing model is the ease with which it can be applied to even highly complex codes. The key feature of GA is that it enables nodes to asynchronously access data held on other nodes, without 'interrupting' the node on which the data is stored.

Some sample timings are shown below for two different test cases, both of which are calculations for $H_2O$ utilising an active space for the reference configurations of the valence space plus three $p$ like orbitals on the oxygen. Test 1 uses an augmented valence triple zeta basis set, and test 2 uses a basis set similar, though slightly smaller, than the augmented valence quadrupole zeta set. The results suggest that the code is slightly more than 97 % parallel.

Table 5.10: Timings for the MRCI code of MOLPRO'96 on the T3D

| number of nodes | 16 | 32 | 64 | 128 | 256 |
|---|---|---|---|---|---|
| test 1 | 480.78 | 310.11 | 229.24 | — | — |
| test 2 | — | — | 441.39 | 353.82 | |

**Future Work**

There has still to be some fine tuning of the MRCI program to increase the amount of parallel work done. It is doubtful that it will be possible to improve significantly on the speed ups obtained to date, but some improvement may be achieved if a greater emphasis is placed on nodes using only local data. The major work that this needs to be done is on the other programs within MOLPRO. In any calculation involving the MRCI program it is necessary first to compute the one and two electron integrals, and then to obtain molecular orbitals, using the self-consistent field (SCF) program followed by the multi-configuration self-consistent field (MCSCF) program.

So far the evaluation of the two electron integrals has been parallelised, but the sort of the integrals has not. The SCF program is a relatively fast program, but the MCSCF field program can be slow if convergence is not reached in a few iterations. Unfortunately the work within this latter program is spread fairly evenly throughout several subroutines, and it may be an even harder task to parallelise this program than the MRCI program.

## 5.5 Nonlinear Optical Properties using HONDO

*J. Waite, The National Hellenic Research Foundation, Athens*

The main thrust of this project is the calculation of the Non-Linear Optical (NLO) properties of atoms and molecules and since such properties are at least fourth order, their estimation is computationally intensive. The work on the IBM SP2 uses two versions of the *ab-initio* code HONDO, namely version 8.5 (as modified by J. Waite) and HONDO95, part of the Chem-Station package from IBM. HONDO was written by Dr. Michel Dupuis *et al.*, IBM, Department MLM/078, Neighborhood Road, Kingston, and was chosen since it both specialises in the calculation of NLO properties and is designed to run on IBM machines, including the SP2. In fact Michel Dupuis set up HONDO95 on the Daresbury system. Within this program NLO properties can be computed at the SCF, MP2, MP4, CI or MCSCF levels using derivative Hartree-Fock, sum-over-states or finite-field methods.

Computation of the Nth order NLO properties is currently under way, for N up to 40, of the Hydride ion, Helium, the Lithium cation, Lithium Hydride and Beryllium, using large bases out to g-orbitals at the RHF level, in collaboration with Dr. Jerry Silverman, Texas, USA. In addition, test calculations of $\alpha$, $\beta$ and $\gamma$ of ammonia, at MP4 were satisfactory, for the purpose of running a ro-vibrational study of all components of these tensors. This work is being done in collaboration with Dr. Brian Sutcliffe, Chemistry, York.

In the future, hopefully running simultaneously with the ammonia project, it is proposed to compute the NLO properties of nickel carbonyls in collaboration with Prof. Doug Smith, Toledo, USA; the frequency dependent NLO properties of Nickel and Palladium Dithiolenes and - phenylenes with Prof. Allan Underhill, Chemistry, Bangor and some transition metal, mixed valence complexes of Dr. Bob Denning, Inorganic Chemistry, Oxford; together with other complexes with Dr. Paul Raithby, Chemistry, Cambridge.

The SP2 is found to be a friendly and reasonably easy computer to use, with an excellent turn-round. The users guide, manuals, initial help from Michel Dupuis and on-going assistance from Paul Walmsley have been extremely helpful. It is hoped that more disk space will be available shortly, for this will be essential if larger molecules are to be treated. Future plans include installing and testing Prof. Mike Zerner's (University of Florida) ZINDO code for the semi-empirical determination of the NLO properties of large systems, at the SCF and MP2 levels. The manual for HONDO95 may be found in:

*/h/jwt/hondo/master/hondo95.listps* or *hondo95.script* .

## 5.6 Molecular Dynamics and DL_POLY

*T.R. Forester, W. Smith*

DL_POLY is a general purpose parallel molecular dynamics (MD) package. It has the capability to model a wide range of systems including ionic materials, macromolecules, solutions and simple metals, by numerically solving Newton's classical equations of motion for many body systems.

It can handle a wide range of two body, three body (angle) and four body (torsional) potentials, with direct summation, reaction field of Ewald summation available for electrostatic terms. It incorporates all the standard periodic boundary conditions (including non-periodic) along with either atomistic or group based cutoff methods.

The two body terms are usually the most computationally demanding and a Verlet neighbourhood table is (re)constructed periodically with the parallelised Brode-Ahlrichs algorithm. In systems with cells large in comparison with the cutoff, a link-cell version of this is used. A link-cell scheme is also used in the calculation of nonspecific three body forces (e.g. in glasses).

The method of integration of the equations of motion is based on the Verlet leapfrog algorithm. Integration in the NVE, NVT and NPT (both isotropic and anisotropic) ensembles is available as is a steepest-descent structure optimisation routine. A multiple-timestep algorithm is available for large systems. Rigid bodies or bonds are handled by either bond constraints (the SHAKE algorithm) or quaternion algorithms or a combination of both.

### 5.6.1 Timings for DL_POLY_2.0 Benchmarks

Timings are reported below for three different benchmarks on the Cray T3D and IBM SP2. In each case the times are reported for three distinct phases of the computation, namely (a) the initialisation phase, reading in the data etc, (b) the MD cycle time, reflecting the numerically intensive phase of the computation, and (c) the wind up time - writing out restart files and calculating the final properties etc. Note that the performance of the code is determined solely by phase (b) of the computation, given the many thousands of time steps involved in a typical simulation run; the speed-ups quoted in the results below are derived from just this MD cycle time.

**Benchmark 1**

Wall Times (seconds) on both the IBM SP2 and Cray T3D are shown in table 5.11 for a simulation of a system of 19652 Lennard Jones (Argon) atoms. The neighbourhood table was calculated using the link-cell scheme. The following CONTROL deck was used:

```
temperature       50. K
steps             100
timestep          0.005
cutoff            8.60
delr              0.58
```

**Benchmark 2**

Wall Times (seconds) on the IBM SP2 and Cray T3D are shown in tables 5.12 and 5.13, for simulations of a system of 3456 SPC/e and 6912 SPC/e water molecules (rigid bodies + neutral group), respectively. The reaction field method was used to calculate the long range electrostatic forces. The CONTROL deck was as follows:

Table 5.11: DL_POLY Benchmark 1 on the T3D and SP2 (all times are in seconds)

| T3D (Phase) | 2 | 4 | 8 | 16 | 32 | 64 |
|---|---|---|---|---|---|---|
| a | 20.3 | 21.8 | 24.3 | 26.4 | 29.8 | 31.6 |
| b | 385.6 | 207.2 | 111.7 | 64.2 | 44.1 | 34.2 |
| c | 28.5 | 28.2 | 28.3 | 28.6 | 28.5 | 28.5 |
| Total | 434.4 | 257.2 | 164.3 | 119.2 | 102.4 | 94.3 |
| Speed-up | [2] | [3.7] | [6.9] | [12.0] | [17.5] | [22.5] |
| SP2 (Phase) | 2 | 4 | 8 | 16 | | |
| a | 34.3 | 50.5 | 74.7 | | | |
| b | 163.0 | 99.8 | 64.2 | | | |
| c | 27.6 | 70.8 | 14.3 | | | |
| Total | 225.0 | 221.1 | 153.1 | | | |
| Speed-up | [2] | [3.3] | [5.1] | | | |

```
temperature           300. K
steps                   10
timestep             0.002
cutoff                15.0
rvdw                  10.0
delr                   1.0
reaction field   eps 78.0
ensemble nvt berendsen
neutral groups
```

Table 5.12: DL_POLY Benchmark 2/3456 on the T3D and SP2 (all times are in seconds)

| T3D (Phase) | 4 | 8 | 16 | 32 | 64 | 128 |
|---|---|---|---|---|---|---|
| a | 18.5 | 14.3 | 14.4 | 12.2 | 11.0 | 12.6 |
| b | 330.0 | 171.4 | 90.4 | 48.2 | 25.6 | 15.8 |
| c | 15.0 | 14.9 | 15.0 | 15.0 | 15.1 | 15.3 |
| Total | 363.5 | 200.6 | 119.8 | 75.4 | 51.7 | 43.7 |
| Speed-up | [4] | [7.7] | [14.6] | [27.4] | [51.6] | [83.5] |
| SP2 (Phase) | 4 | 8 | 16 | | | |
| a | 15.9 | 27.0 | | | | |
| b | 196.8 | 105.5 | | | | |
| c | 2.7 | 2.7 | | | | |
| Total | 215.4 | 135.2 | | | | |
| Speed-up | [4] | [7.5] | | | | |

**Benchmark 3**

Wall Times (seconds) on the Cray T3D are shown in table 5.14 for the third benchmark, a sodium chloride system of 27,000 ions. The electrostatics were handled with an Ewald sum with convergence parameter 0.16373 reciprocal Angstroms. The strategy adopted involves parallelising reciprocal space over k vectors (formally 4913 k vectors are included in the sum). The control deck used was as follows:

Table 5.13: DL_POLY Benchmark 2/6912 on the T3D (all times are in seconds)

| T3D (Phase) | 8 | 16 | 32 | 64 | 128 |
|---|---|---|---|---|---|
| a | 38.0 | 28.9 | 24.3 | 22.5 | 23.0 |
| b | 517.9 | 272.6 | 147.0 | 81.2 | 45.2 |
| c | 29.8 | 30.0 | 29.9 | 29.8 | 30.5 |
| Speed-up | [8] | [15.2] | [28.2] | [51.0] | [91.7] |

```
temperature 1000 K
ewald precision  1d-6
timestep          0.005
multiple          5
steps            10
cutoff           19.0
primary          10.76
rvdw             10.0
delr              0.75
ewald precision 1d-6
```

Table 5.14: DL_POLY Benchmark 3 on the T3D (all times are in seconds)

| T3D (Phase) | 16 | 32 | 64 | 128 |
|---|---|---|---|---|
| a | 21.1 | 23.3 | 22.3 | 23.0 |
| b | 314.0 | 185.6 | 120.8 | 89.2 |
| c | 39.6 | 39.5 | 39.6 | 41.7 |
| Total | 374.7 | 248.8 | 182.7 | 153.9 |
| Speed-up | [16] | [27.1] | [41.6] | [56.3] |

### DL_POLY_2.0 Benchmark 9: amber case (1) – 15mer in 1247 waters

CONTROL deck amber test case : 15mer in 1247 waters

```
temperature           10
equilibration steps   10
steps                 50
timestep              0.001
cutoff               8.0
rvdw                 8.0
delr                  1.0
collect
zero
reaction field
eps       78
ensemble nvt berendsen 1.0
rdf every 10 steps
print   rdf
```

```
print every 100
stats every 100
cap         100000
scale every     1
```

This test case has 3993 atoms, 1264 charge groups, periodic boundaries pair list based on closed contact between any atoms in charge groups. The water is integrated as rigid bodies (quaternion algorithm)

Table 5.15: DL_POLY Benchmark 9 on the T3D and SP2 (all times are in seconds)

| T3D (Phase) | 1 | 2 | 4 | 8 | 16 |
|---|---|---|---|---|---|
| a | 16.1 | 14.4 | 14.5 | 13.6 | 13.3 |
| b | 487.3 | 273.3 | 149.3 | 83.2 | 46.7 |
| c | 6.9 | 6.1 | 6.0 | 6.8 | 6.1 |
| Total | 510.4 | 293.8 | 169.8 | 103.6 | 66.1 |
| Speed-up | 1 | 1.74 | 3.0 | 4.9 | 7.7 |
| SP2 (Phase) | 1 | 2 | 4 | 8 | 16 |
| a | 3.7 | 8.0 | 10.3 | 39.1 | 23.9 |
| b | 276.7 | 159.7 | 96.8 | 55.6 | 36.6 |
| c | 1.0 | 1.1 | 1.1 | 1.1 | 1.2 |
| Total | 281.4 | 168.8 | 108.2 | 95.8 | 61.7 |
| Speed-up | 1 | 1.66 | 2.6 | 2.9 | 4.6 |

Same system with r= 16.0 rvdw = 12.0

Table 5.16: DL_POLY Benchmark 9 on the T3D and SP2 (all times are in seconds)

| T3D (Phase) | 1 | 2 | 4 | 8 | 16 |
|---|---|---|---|---|---|
| a | | | | 12.5 | 12.5 |
| b | | | | 347.0 | 183.3 |
| c | | | | 6.0 | 6.0 |
| Total | | | | 356.6 | 201.8 |
| Speed-up | | | | [8] | [14.1] |
| SP2 (Phase) | 1 | 2 | 4 | 8 | 16 |
| a | 3.6 | 8.1 | 7.3 | 28.8 | 41.1 |
| b | 1197.4 | 644.0 | 344.0 | 187.4 | 111.2 |
| c | 1.0 | 1.1 | 1.1 | 1.2 | 1.2 |
| Total | 1202.0 | 653.2 | 352.4 | 217.4 | 153.5 |
| Speed-up | 1 | 1.84 | 3.41 | 5.53 | 7.83 |

### DL_POLY_2.0 Benchmark 10

Control deck Transferrin 27539 atoms system

```
temperature 300.0
presure  0.001
```

```
stats        250
print every 250
timestep     .0001
steps           20
equilibration   50
cap 100000
collect
zero
scale 1
cutoff 8.00
delr   1.00
coulombic potential
shake     5d-4
```

The system has 27539 atoms, including 8102 water molecules, 8 Angstrom cutoff, 27539 atoms, 8441 charge groups, periodic boundaries. The pair list is based on closed contact between any atoms in charge groups. water integrated as rigid bodies (quaternion algorithm)

Table 5.17: DL_POLY Benchmark 10 on the T3D and SP2 (all times are in seconds)

| SP2 (Phase) | 4 | 8 | 16 |
|---|---|---|---|
| a | 130.7 | 119.6 | 160.3 |
| b | 423.8 | 235.2 | 157.7 |
| c | 7.6 | 7.6 | 7.5 |
| Total | 562.3 | 362.4 | 325.5 |
| Speed-up | 1 | 1.55 | 1.73 |

# Chapter 6

# Materials Science

## 6.1 LMTO code

### On the Gap Anisotropy in the Layered High Temperature Superconductors

*W.M. Temmerman and Z. Szotek*

Recently, high-resolution photoemission experiments have been providing fairly convincing evidence that the gap in the excitation spectrum of the layered $high-T_c$ superconductors is strongly anisotropic, see for instance [10, 66]. Whilst this may be decisive information, the interpretation of the data is allusive due to the lack of a credible model which deals with the normal and superconducting states on equal footing [1, 16].

A new strategy for constructing a model which deals with the normal and superconducting states on an equal footing has now been developed by Temmerman et al. [74]. This combines the eight-band model advocated by Andersen et al. [2] for describing the electronic structure of these materials, near the Fermi energy, in the normal state, and the semi-phenomenological density functional approach to superconductivity of Suvasini et al. [68]. Only singlet pairing is considered and the pairing potential has a complicated matrix form in both site and orbital indices. This allows us to attribute the rich variety of consequencies, predictions for various experiments, of the theory to interactions between electrons at specific sites and in specific orbitals. Guided by the data from such searching probes as high-resolution photoemission experiments this approach is in a position to identify the sites and the orbitals relevant to superconductivity, and hence provide new bases for speculations about the nature of pairing as described in Temmerman et al. [75].

The code which incorporates the above eight-band model to describe the electronic structure together with the density functional formulation of superconductivity, was parallelised using IBM's MPL message passing harness. This code also runs on the Intel iPSC/860. On the IBM SP2 it scales perfectly; performance on 64 nodes is exactly twice as fast as on 32 nodes. It performs 5.5 faster on an IBM SP2 node than on an i860 node.

## 6.2 SIC-LSD code

**Role of $5p$ electrons in description of $\gamma \to \alpha$ transition in cerium**

*W.M. Temmerman and Z. Szotek*

*Ab initio* self-interaction-corrected local spin density (SIC-LSD) calculations have, for the first time, described the $\gamma$ phase of Ce (see Szotek et al. (1994), Svane (1994)). This high-volume phase of elemental Ce is characterised by the localisation of the f electron. Moreover, the *ab initio* calculations seem also to be able to describe the phase diagram of the $\gamma \to \alpha$ transition (see Johansson et al. (1995), Svane (1995)). This indicates that the calculated total energy differences between the $\alpha$ and $\gamma$ phase are correctly described. Whilst this can be construed as evidence for the Mott transition model for the $\gamma \to \alpha$ transition in Ce, the serious underestimation of the theoretical volume of $\alpha$ Ce by 13% could indicate that the Kondo volume collapse model is more appropriate.

Recently, we have explored the effects of the self-interaction-correction on the semi-core *5p* electrons in Ce, i.e. the degree of localisation of these semi-core p electrons in Temmerman et al. (1995). We find that the self-interaction-correction of the *5p* electrons lowers the total energy of both $\alpha$ and $\gamma$ Ce and that the theoretical volume of $\alpha$ Ce is now underestimated by only 4%, which is of the same magnitude as the SIC-LSD volume of the $\gamma$ phase. The total energy difference between the $\alpha$- and $\gamma$-phase is of the order of 0.1 mRy and correctly the $\alpha$ phase is the ground state.

These results were generated with a new SIC code which incorporates greater flexibility in the nature of the localised states. However, this code is much more memory demanding, which makes it impossible to be run on the Intel iPSC/860 nodes which have only 16 MB memory. It however fits perfectly on the larger-memory nodes of the IBM SP2. This parallel code uses IBM's MPL message passing harness.

## 6.3 A Replicated Data Parallel Implementation of CRYSTAL95

*N.M. Harrison and V.R. Saunders*

The CRYSTAL program calculates the wave-functions and properties of crystalline systems, using an Hartree-Fock (HF) Linear-Combination-of-Atomic-Orbitals (HF-LCAO) approximation. The code has been jointly developed by the University of Torino, Daresbury Laboratory and their collaborators.

It is used to investigate the electronic structure of systems periodic in 0 (molecules), 1 (polymers), 2 (slabs), and 3 dimensions (crystals). It is able to compute, at a common level of accuracy, HF energies and first-order density matrices, and a number of properties of these systems. Useful information on the distribution of one-electron levels is also obtained.

A discussion of the present scheme in relation to other approaches, a thorough presentation of the algorithms employed and of the possible applications of CRYSTAL, and a selection of relevant literature may be found in [94]. In this section we present the replicated data algorithm for the parallelisation of CRYSTAL and report its performance on current hardware.

### 6.3.1 Theoretical Framework

The main approximation made in CRYSTAL is to expand the single-particle orbitals in a finite basis set of atom-centred Gaussian-type orbitals (GTOs). Thus, each Crystalline Orbital, (**CO**) $\psi_i(\mathbf{r};\mathbf{k})$, is a linear combination of Bloch functions (**BF**) $\phi_\mu(\mathbf{r};\mathbf{k})$, defined in terms of local functions, $\varphi_\mu(\mathbf{r})$ (hereafter indicated as 'Atomic Orbitals', **AOs**).

$$\psi_i(\mathbf{r};\mathbf{k}) = \sum_\mu a_{\mu,i}(\mathbf{k})\phi_\mu(\mathbf{r};\mathbf{k}) \tag{6.1}$$

$$\phi_\mu(\mathbf{r};\mathbf{k}) = \sum_{\mathbf{T}} \varphi_\mu(\mathbf{r} - \mathbf{A}_\mu - \mathbf{T})\ e^{i\mathbf{k}\cdot\mathbf{T}} \tag{6.2}$$

$\mathbf{A}_\mu$ denotes the coordinate of the nucleus in the zero reference cell on which $\varphi_\mu$ is centred, and the $\sum_{\mathbf{T}}$ is extended to the set of all lattice vectors $\mathbf{T}$.

The local functions are expressed as linear combinations of a certain number $n_G$ of individually normalised GTOs characterised by the same centre, with fixed coefficients $d_j$ and exponents $\alpha_j$, defined in input:

$$\varphi_\mu(\mathbf{r} - \mathbf{A}_\mu - \mathbf{T}) = \sum_j^{n_G} d_j\ G(\alpha_j; \mathbf{r} - \mathbf{A}_\mu - \mathbf{T}) \tag{6.3}$$

The AOs belonging to a given atom are grouped into 'shells' $\lambda$. A shell can contain all AOs with the same quantum numbers $n$ and $\ell$ (for instance 3s, 2p, 3d, shells), or all the AOs with the same principal quantum number $n$, if the number of GTOs and the corresponding exponents are the same for all of them (mainly sp shells, 'sp constraint'). These groupings permit a reduction in the number of auxiliary functions that need to be calculated in the evaluation of electron integrals, and relevant speed-up.

A single, normalised, s-type GTO $G_\lambda$ is associated with each shell ('adjoined Gaussian' of shell $\lambda$). The $\alpha$ exponent is the smallest of the $\alpha_j$ exponents of the Gaussians in its contraction. The adjoined Gaussian is used to estimate the AO overlap and select the level of approximation to be adopted for the evaluation of the integrals.

The expansion coefficients of the Bloch functions, $a_{\mu,i}(\mathbf{k})$, are calculated by solving for each reciprocal lattice vector $\mathbf{k}$ the matrix equation:

$$\mathbf{F}(\mathbf{k})\mathbf{A}(\mathbf{k}) = \mathbf{S}(\mathbf{k})\mathbf{A}(\mathbf{k})\mathbf{E}(\mathbf{k}) \tag{6.4}$$

in which $\mathbf{S}(\mathbf{k})$ is the overlap matrix over the Bloch functions, $\mathbf{E}(\mathbf{k})$ is the diagonal energy matrix and $\mathbf{F}(\mathbf{k})$ is the Fock matrix in reciprocal space:

$$\mathbf{F}(\mathbf{k}) = \sum_{\mathbf{T}} \mathbf{F}^{\mathbf{T}}\ e^{i\mathbf{k}\cdot\mathbf{T}} \tag{6.5}$$

The matrix elements of $\mathbf{F}^{\mathbf{T}}$, the Fock matrix in direct space, can be written as a sum of one-electron and two-electron contributions in the basis of the AO:

$$F_{12}^{\mathbf{T}} = H_{12}^{\mathbf{T}} + B_{12}^{\mathbf{T}} \tag{6.6}$$

The one-electron contribution is the sum of the kinetic and nuclear attraction terms:

$$H_{12}^{\mathbf{T}} = T_{12}^{\mathbf{T}} + Z_{12}^{\mathbf{T}} = \langle \varphi_1^{\mathbf{0}} \mid \widehat{T} \mid \varphi_2^{\mathbf{T}} \rangle + \langle \varphi_1^{\mathbf{0}} \mid \widehat{Z} \mid \varphi_2^{\mathbf{T}} \rangle \tag{6.7}$$

In core pseudopotential calculations $\hat{Z}$ includes the sum of the atomic pseudopotentials.

The two-electron term is the sum of the Coulomb and exchange contributions:

$$B_{12}^{\mathbf{T}} = C_{12}^{\mathbf{T}} + X_{12}^{\mathbf{T}} = \tag{6.8}$$

$$\sum_{3,4} \sum_{\mathbf{Q}} P_{3,4}^{\mathbf{Q}} \sum_{\mathbf{S}} [(\varphi_1^0 \varphi_2^{\mathbf{T}} \mid \varphi_3^{\mathbf{S}} \varphi_4^{\mathbf{S+Q}}) - \frac{1}{2}(\varphi_1^0 \varphi_3^{\mathbf{S}} \mid \varphi_2^{\mathbf{T}} \varphi_4^{\mathbf{S+Q}})] \tag{6.9}$$

The Coulomb interactions, electron-nucleus, electron-electron and nucleus-nucleus are individually divergent, due to the infinite size of the system. Grouping of corresponding terms is necessary in order to eliminate this divergence (Ewald summation).

The $\mathrm{P}^{\mathbf{Q}}$ density matrix elements in the AOs basis are computed by integration over the Brillouin zone (BZ) volume,

$$P_{3,4}^{\mathbf{Q}} = 2 \int_{BZ} d\mathbf{k} e^{i\mathbf{k}\cdot\mathbf{Q}} \sum_n a_{3n}^*(\mathbf{k}) a_{4n}(\mathbf{k}) \theta(\epsilon_F - \epsilon_n(\mathbf{k})) \tag{6.10}$$

where $a_{in}$ denotes the ith component of the nth eigenvector, $\theta$ the step function, $\epsilon_F$ the Fermi energy, and $\epsilon_n$ the nth eigenvalue.

The total electronic energy per unit cell is given by:

$$E^{elec} = \frac{1}{2} \sum_{1,2} \sum_{\mathbf{T}} P_{12}^{\mathbf{T}} (H_{12}^{\mathbf{T}} + B_{12}^{\mathbf{T}}) \tag{6.11}$$

Further details of the algorithms used and the computational parameters have been published elsewhere [94].

### 6.3.2 The Parallel Implementation

The computational task can be divided into two stages. Firstly, **integral evaluation** in which the one-electron and two-electron integrals defined in equations 6.7 and 6.8 are evaluated. Secondly, the

**SCF cycles**, in each of which the Fock matrix is built according to equations 6.5 and 6.6 and equation 6.4 solved at each **k**-point by diagonalisation. The output density matrix is then evaluated according to equation 6.10. CRYSTAL95 may operate in an orthodox mode in which the integrals are stored to disk and re-read during each SCF cycle or a direct-SCF mode in which the integrals are regenerated when required.

The direct space matrices ($P^{\mathbf{Q}}$ and $\mathrm{F}^{\mathbf{T}}$) are efficiently screened; the number of finite elements scaling linearly with system size, O(N). These matrices are also subject to the symmetry of the system and thus the storage associated with the irreducible representation is 1/O(g) of that of the reducible matrix, where O(g) is the order of the symmetry group.

In the replicated-data parallel implementation of CRYSTAL95 the main data structures are present on each processor. The problem is considered to be made up of a number of *tasks* each of which is allocated to a processor using a *synchronous global index*. During integral evaluation each task is the evaluation of the integrals which contribute to a Fock matrix element involving a particular pair of shells. The number of contributions is $C * N_{sh}$ where $C$ depends on the efficiency of direct space screening in the system studied but is usually in the range 10-100. Thus, the total number of such tasks is typically of order 10,000. Each task may be computed independently without inter-processor communication. The global index ensures dynamic load balancing while the number of tasks is significantly greater than the number of processors.

During an SCF cycle each processor merges its local copy of $P^{\mathbf{Q}}$ with the subset of integrals it has generated to produce a partial Fock matrix, $F^{\mathbf{T}}$. A *global summation* (GSUM) is then performed which accumulates a full copy of $\mathrm{F}^{\mathbf{T}}$ on each processor. As the diagonalisation of $\mathbf{F}(\mathbf{k})$ at each **k**-point is an independent task these are assigned to processors using a global index. Each processor then contains a subset of the eigenvalues, $\mathbf{E}(\mathbf{k})$, and eigenvectors, $\mathbf{A}(\mathbf{k})$. A GSUM is performed on $\mathbf{E}(\mathbf{k})$ and each node computes the Fermi energy of the system. Using Equation 6.10 a partial copy of the output density matrix $P^{\mathbf{Q}}$ can then be generated by each processor and a GSUM performed. The total number of tasks is equal to the number of irreducible **k**-points ($N_k$) which is typically of order 10 to 30. The CPU time required per processor for the diagonalisation is proportional to $N_{AO}^3/N_{proc}$ whereas the total amount of data passed between processors it proportional to $N_{AO} \times N_{proc}$. The algorithm will therefore achieve high parallel efficiency while $N_{proc} < N_k$. For any given system size there will alway be a number of processors on which the communication overhead becomes dominant.

This algorithm has been implemented in PVM and TCGMSG [20]. It is thus portable to a wide variety of computer platforms including clusters of heterogeneous workstations connected by standard networks and purpose-built shared or distributed memory parallel computers.

### 6.3.3 Performance

The performance of CRYSTAL on a 16-node IBM SP2 is presented in Table 6.1.

The case studied is of moderate size which illustrates the points made above on a relatively small (16-node) computer. The electronic structure of the anti-ferromagnetic phase of $CaCuO_2$ is computed, which requires 140 AO, 40 shells and 75 irreducible **k**-points. The screening tolerances are high (8 7 8 8 16 – see [94] for further details) and thus some 8 million one-electron and 111 million two-electron integrals are calculated. The direct-SCF algorithm was employed.

Table 6.1: CRYSTAL95: Performance and Scaling in Calculations of $CaCuO_2$

| $N_{procs}$ | Wall Time (seconds) | Speedup |
|---|---|---|
| 1 | 4785 | - |
| 2 | 2394.1 | 2.0 |
| 4 | 1335 | 3.6 |
| 8 | 716 | 6.7 |
| 16 | 471.3 | 10.2 |

As expected on small numbers of nodes the parallel efficiency is high. On 16 nodes the communications overhead becomes more significant: dynamical load balancing of a small number of tasks becomes less efficient, the observed speedup is only a factor of 10.

### 6.3.4 Conclusions

A simple algorithm for parallelisation of periodic Hartree Fock calculations has been developed and implemented. With an appropriate matching of system size with parallel hardware, high efficiency results are obtained. The algorithm does not depend on very low latency or high bandwidth communications and is thus also suitable for use on a heterogeneous network of workstations linked using standard network technology (ethernet or FDDI). The implementation is based on PVM and TCGMSG and may thus be ported with minimum effort to a wide range of hardware

## 6.4 CETEP

*P. Lockey*

The CETEP code used by the UK Car-Parrinello consortium has been ported from Cray T3D PVM to MPI to improve portability. The MPI version has so far been run on the Cray T3D and the IBM SP2.

### 6.4.1 Communications Routines

Since the MPI standard is relatively recent, not all machines have efficient MPI routines for global operations such as reductions, broadcasts etc. To overcome this problem, two versions of the CETEP communication routines have been provided. The first version makes use of the full (and very rich) MPI functionality for global operations. The second version carries out the global operations using hand coded 'reasonably efficient' routines based on the simpler point-to-point MPI functionality. The particular version is chosen at compile-time using a C-preprocessor flag `MPI_GLOBALS`. It is anticipated that the second version of the routines will gradually become redundant as manufacturers improve the efficiency of their MPI libraries.

### 6.4.2 Performance Results

The timings shown are for the full code execution, and include all computation, communication, and file i/o. The following compile and i/o options were used:

**Cray T3D:** Compiled with -O1 -Oscalar3. File i/o done on LDCACHED disk.
**IBM SP2:** Compiled with -O3 -qhot. File i/o done on local node scratch disk.

Table 6.2: CETEP Test Problem Solution (all times in seconds)

| | IBM SP2 | | Cray T3D | |
|---|---|---|---|---|
| Nodes | Globals | No-globals | Globals | No-globals |
| 4 | 536.0 | 513.8 | 656.8 | 700.9 |
| 8 | 255.8 | 256.9 | 341.6 | 389.6 |
| 16 | 180.3 | 192.5 | 189.3 | 257.6 |

Timings are also shown for the routine `ZMEXCH`, which carries out an **all-to-all** exchange of complex data between the nodes. The majority of the communication time is spent in this routine.

Table 6.3: CETEP Communication Costs - ZMEXCH (all times in seconds)

| | IBM SP2 | | Cray T3D | |
|---|---|---|---|---|
| Nodes | Globals | No-globals | Globals | No-globals |
| 4 | 125.9 | 96.3 | 25.0 | 64.0 |
| 8 | 72.6 | 64.7 | 19.4 | 57.1 |
| 16 | 69.2 | 66.4 | 19.8 | 73.4 |

Clearly the code on the SP2 runs little quicker than on the T3D. There is however a good reason for this. A design decision was made, somewhat over 1 year ago, to optimise the performance on the T3D in line with CETEP's use in the grand-challenge consortium. This resulted in a data layout of distributed 'fat' columns in the 3D arrays being optimal, given the low latency and high bandwidth of SH_MEM comms relative to node cpu performance. In the older version of CETEP however, still running on the Intel iPSC/860, a 'fat' layer distribution was preferable, and we believe this to be the case for the SP2. It is necessary to incorporate improvements to the T3D code into the Intel version (now code 3 years old) and port to MPI to realise the potential of this. Such developments are in progress.

## 6.5 O(N) Density Functional Methods

*I.J. Bush, P. Madden, S. Watson, M. Pearson, E. Smargassi, M. Foley*

Paul Madden's O(N) molecular dynamics code is, in many ways, similar to the CETEP code. In both cases a density functional description of the quantum system is used to calculate forces on the classical ions. The major difference is that the O(N) code represents the kinetic energy of the quantum system as a functional of the electronic density, thus avoiding the need to introduce a basis set for the orbitals. The major advantage of this is that no orthogonality constraints need to be imposed, a $O(N^3)$ calculation, and instead the speed of the method depends on the FFTs (O(NlogN) approaching

O(N)). The disadvantage is that the kinetic energy operator is inherently non local, and at present the functionals can only describe sp metals, e.g. cesium and aluminium.

## 6.6 AIMPRO and the *Ab initio* Studies of Covalent Materials

*S. Öberg, (Department of Mathematics, University of Luleå, SWEDEN), J. Goss and R. Jones, (Department of Physics, University of Exeter), and P.R. Briddon, (Department of Physics, University of Newcastle upon Tyne)*

There has been remarkable advances in computational modelling defects in solids over the last ten years, so much so that sophisticated quantum mechanical methods can be brought to bear on *contemporary* problems and produce results that influence the direction of progress of experimental groups actively engaged on a study of these defects. To illustrate the capabilities of present theory we describe the results obtained from such an approach to gold-hydrogen defects in silicon.

It has recently been found [100] that hydrogen, introduced by an acid etc, can react with substitutional gold in silicon creating a number of defects. DLTS studies show that one such defect has a donor level at $E_v$+0.21 eV, an acceptor level at $E_c$-0.19 eV and a deeper mid-gap level. Another defect has no gap levels at all and is then a passive complex of hydrogen with gold. It seems very surprising that Au with deep gap $t_2$-levels arising from combinations of the four $sp^3$ hybrids on Si dangling bonds surrounding a substitutional Au impurity, with the *d*-orbitals on Au itself, can be filled (or emptied) and displaced out of the gap by H atoms. If this is correct, it suggests that troublesome impurities like transition metals (TM) can be passivated by H and their harmful electronic activity eliminated. To investigate this problem, it is necessary to solve quantum mechanically the electronic properties of Au and Au-H complexes embedded in a silicon crystal.

To achieve this we have used the modelling program (AIMPRO), employing techniques in use by the Covalent Materials Consortium working on the *ab initio* studies of covalently bonded materials. The first principles pseudopotential method uses a real space basis set and either H-terminated clusters or supercells. Typically clusters/unit-cells of size 70-800 are used. Details of the method are found in [103].

The functionality of AIMPRO includes the following:

1. self-consistent local density functional theory;
2. norm-conserving pseudopotentials;
3. *s*-, *p*-, *d*- and *f* Gaussian basis functions;
4. analytic evaluation of atomic forces;
5. conjugate gradient optimisation;
6. optimisation with constraints for the search of diffusion trajectories;
7. numerical evaluation of second order energy derivatives for the construction of the dynamical matrix;

8. the extraction of valence force potentials, from the *ab initio* derivatives, to determine bulk phonon spectra;

9. the merging of dynamical matrices calculated using *ab initio* technique s with those calculated with valence-force potentials and hence the determination of both local and resonant defect modes;

10. evaluation of effective charges for determination of the strength of local mode absorption;

11. spin-polarisation;

12. a scissor operator which allows the more accurate estimate of the positions of donor/acceptor levels;

13. PVM, BLACS, PBLAS and SCALAPACK routines which extend the code to parallel processors, such as an array of workstations, or massively parallel machines such as the SP2 and T3D.

The code has been tuned for the IBM SP2, and uses MPL rather than PVM. A representative speed-up, over 1 processor, is a factor of 7 on 8 processors and 13 on 16 processors, this factor depending on the job. A more demanding job clearly has a higher efficiency for 16 processors. Representative timings for the evaluation of the initial energy for a cluster of 114 atoms, with Hamiltonian dimension 1588, are 7872 seconds on an IBM RS/6000-590 workstation, 1243 seconds on 8 processors of the IBM SP2, and 334 seconds on 128 processors of the Cray T3D. The code has proved to be highly successful at analysing complex defects and defect processes and we describe below the application to the work on Au-H complexes.

### 6.6.1 Gold hydrogen complexes in silicon

We have investigated five Au-H complexes. These are $Au_s$-$H_n$ with $0 \leq n \leq 4$.

The starting point for the investigation was a structure of $AuH_n$ defects where the $n$ H atoms lay outside the vacancy. This is the preferred configuration found by previous calculations for $NiH_2$ [101] and is the configuration deduced from EPR investigations into $PtH_2$ [102].

The calculations were carried out in two stages: first, the structure of the complexes were found by inserting $AuH_n$ into a 71+$n$ atom cluster, $AuSi_{34}H_{36+n}$. This was then relaxed to deduce the Au-Si, Si-H lengths and angles. The defect was then inserted into a much larger 297+$n$ atom cluster $AuSi_{180}H_{116+n}$, with the same Au-Si, Si-H bond lengths and angles as found from the relaxed smaller cluster. The larger cluster possesses a gap closer to the experimental gap for Si, but the difference was still sufficiently great to prevent the elucidation of the donor/acceptor levels. Consequently, a scissors operator was used which forced the gap of the 297 cluster, without any defects, to be 1.2 eV. This is the normal procedure for determining deep donor/acceptor levels. These were then determined by calculating the total energies of the charged clusters and comparing them with the energies of charged clusters without defects. Thus the donor level relative to the valence band top, and the acceptor level relative to the conduction band bottom are found from:

$$E^0(X) - E^+(Si) - (E^+(X) - E^0(Si))$$

$$E^0(X) + E^-(Si) - (E^-(X) - E^0(Si))$$

respectively. Here $E^0(X), E^0(Si)$ are the energies of the neutral clusters with and without the defect. This formulation ignores any energy changes caused by additional relaxation of the charged defects.

Our findings may be summarised as follows:

1. Each defect has a $t_2$ gap level, which has 3 electrons for $Au_s$, and contains an additional electron for each H atom added until $AuH_3$ is reached. The $t_2$ level splits due to loss of $T_d$ symmetry for $n < 4$.
2. The $t_2$ level is high up in the gap in agreement with previous calculations.
3. Substitutional Au has a acceptor level at $E_c$-.2 eV and a donor level at $E_v$+.52 eV. These are to be compared with DLTS measurements of $E_c$-.53 and $E_v$+.34 eV respectively. Their separation, 0.48 eV is the Hubbard U-parameter which is in fair agreement with the experimental value of 0.34 eV. The actual position of the levels is about 0.25 eV too high in in the gap. These errors may be due to relaxation effects and errors in the technique.
4. $Au_s$ does not possess a second acceptor or donor level in agreement with experiment.
5. The donor-acceptor levels of Au-$H_n$ are given in Table 6.4.
6. $AuH_3$ is a passive defect except possibly for a shallow acceptor level. It is remarkable that the H atoms have succeeded in filling the $t_2$ level, displacing it to the band edges, and yet not introducing any additional deep acceptor state. This result confirms that H can indeed passivate Au impurities.
7. $AuH_4$ has a deep mid-gap $a_1$ level. We tentatively associate that with the mid-gap level G3.
8. All the other complexes possess deep states and could explain the G1, G2 donor/acceptor levels seen experimentally at $E_v$+0.21 eV, and $E_c$-0.19 eV.
9. We do not think the observed levels are related to a single defect, but rather to defects $AuH_n$ with $n$ not equal to 3.

Table 6.4: The donor-acceptor levels of Au-$H_n$

| Defect | Acceptor level $E_c$- | Donor level $E_v$+ |
|---|---|---|
| Au | 0.21 | 0.52 |
| $AuH_1$ | 0.28 | 0.44 |
| $AuH_2$ | 0.27 | 0.33 |
| $AuH_3$ | 0.03 | 0.03 |
| $AuH_4$ | 0.84 | 0.06 |

Considerable progress has been made in understanding these complexes but further experiments now need to be carried out to find the other, as yet undetected, levels.

## 6.7 Sheffield Micromagnetism Code

*P. Lockey*

Micromagnetism is essentially a theoretical formalism which enables the prediction of magnetisation structures such as domain walls and the investigation of magnetisation reversal mechanisms in bulk magnetic materials. As such it forms an important link between atomic scale magnetism and meso- and macro-scopic phenomena.

Micromagnetism is an important and challenging field for the computational physicist and is currently in a very high state of activity world-wide. The reasons for this are twofold. Firstly the arrival of large-scale computing facilities in the early 1980's made possible numerical solutions to the micromagnetic problem. This encouraged a number of groups to take on the problem of developing advanced theoretical and computational models for comparison with experimental data. Secondly, and contemporaneously, the technological evolution of a number of important magnetic materials reached a stage at which an increasingly detailed understanding of their fundamental magnetic behaviour emerged as being of vital importance.

Perhaps the best example of this is in the area of recording media. Over the 50 or so years of the existence of particulate recording media advances relied on the simple prescription of decreasing the particle size and increasing the coercivity. However, the large increase in information storage density required of magnetic recording media during the current decade and into the 21st century necessitates a new approach. Currently research is being undertaken into a number of materials including thin metal films with longitudinal and perpendicular anisotropy and Metal Evaporated (ME) tapes in addition to advanced particulate media. Common to all these disparate materials is the existence of complex magnetisation structures and reversal processes which are highly sensitive to the physical microstructure of the material and to the existence of short (exchange) and long range (magnetostatic) interactions. Micromagnetism is at the forefront of understanding these problems and has already found many practical applications.

The code used by the group at Sheffield University was written by John Tucker and J.Bishop. It is used to study the formation of domain walls in magnetic materials. The code was previously run on a transputer array using CSTools and Paramid i860 array. It has now been converted to MPI for increased portability.

The code has been run on Intel iPSC/860, Cray T3D and an IBM SP2. Compiler options used were **-O3 -qhot** on the IBM SP2, but no optimisation so far on the Cray T3D.

Table 6.5: Micromagnetism Test Problem Solution (Times in seconds)

| Nodes | Intel iPSC/860 | IBM SP2 | Cray T3D |
|---|---|---|---|
| 1 | 707.0 | 520.2 | 1161.4 |
| 2 | 385.6 | | 492.3 |
| 4 | 205.0 | 113.1 | 252.6 |
| 8 | 98.6 | 96.8 | 123.2 |
| 16 | 52.4 | | 55.9 |

The code is achieving about 11 Mflop per node on the Cray T3D.

## 6.8 The MagFEM Library and the Erlangen Benchmark

*K. Ramstöck and P. Lockey*

The Finite Element Method (FEM) is used for several engineering problems, the oldest perhaps being the computation of the deformation of a rigid body under a load. Other, more non-linear problems include the computation of fluid flows or magnetic fields. These applications are today well established and commercial program packages are available. Micromagnetism, however, being of more theoretical and less commercial interest has no such programs. Furthermore, there is no agreement in the scientific community about the best procedure to tackle the non-linearity and non-locality of the system equations which govern the problem. The aim of this work is to improve this situation as far as possible by

- Defining a benchmark problem. This has been a successful technique in other areas, the idea being to define a simple yet meaningful problem, which can be solved using a variety of methods. The result is thus well known and can be used to check new algorithms for bugs.
- Measuring the performance of the algorithms. Apart from verification purposes, once a benchmark problem and algorithm are defined, one can measure the performance of a new method with respect to this standard.

The MagFEM Library is a set of routines which can be used to compute micromagnetic configurations as Bloch Walls on a regular spaced grid, using advanced discretisation, advanced stray field evaluation (FFT) and advanced solving strategies. The routines can handle both free and 'embedded' surfaces. These can be used to compute structures like Bloch lines which only exist embedded in the environment of another structure (a Bloch wall, in this case).

This code is still under development and full timings will be presented at a later date.

## 6.9 Oxford Micromagnetism Code

*P. Lockey*

This code, used for studies of domain walls by the micromagnetism group at Oxford University as part of the micromagnetism consortium, was written by John Jakubovics. It has been ported to the T3D by Pete Lockey using a distributed data scheme. The scientific aim is to increase the thickness of the simulated film towards experimental values. For a realistic simulation the area of film must also increase in proportion to the thickness. This implies a large cpu and memory requirement.

The largest arrays of 3D coefficients are distributed, allowing a factor of 30 increase of problem size over previous grids. The memory requirement for this factorisation scales as

$1.8 \times n^2 + \frac{8 \times n^3}{p}$

For comparison this main loop takes 90-120 seconds on an HP workstation at Oxford. An aggregate performance of some 1.9 Gflop is achieved on the T3D from 128 nodes. Performance is shown in table

Table 6.6: T3D Timings: Main Loop of Oxford Micromagnetism Code

| nnode | time | speedup |
|---|---|---|
| 1 | 253.0 | |
| 4 | 64.7 | |
| 8 | 30.5 | 8.3 |
| 16 | 16.4 | 15.5 |
| 32 | 9.0 | 28.1 |
| 128 | 3.7 | 68.8 |

6.6.

This code is still under development, and full timings will be presented at a later date.

# Chapter 7

# The Earth's Environment

## 7.1 Water Quality Modelling

*P. Lockey*

The Proudman Oceanographic Laboratory (POL) Water Quality Model (WQM) code is a 3D baroclinic finite-difference hydrodynamic model of the North Sea, which has successfully been used to model a number of observed phenomena. A portable parallel version of the code has been produced, based on the MPI (Message Passing Interface) standard, and run on a number of parallel machines using a simplified test problem. A brief overview of the code is given here together with some preliminary performance results on the IBM SP2.

### 7.1.1 Overview of the WQM Code

As a full description of the hydrodynamic model formulation is given in [97], only brief details are provided here. The model solves the non-linear 3-dimensional baroclinic shallow water equations in spherical polar coordinates, in finite difference form on an Arakawa B grid. The B grid is preferred to the more commonly used C grid because it introduces less numerical smoothing into the representation of the Coriolis term, an important factor for accurate frontal representation. The finite difference equations are solved using explicit forward time-centred space methods in the horizontal, and an implicit method for vertical diffusion. The equations are time-split, with the timestep for the barotropic component fixed by the Courant-Friedrichs-Levy condition, and a longer timestep (typically 10 times the barotropic timestep) used for the baroclinic motion. A depth-following sigma coordinate provides the same number of vertical levels at each horizontal grid point, and allows bottom boundary conditions to be easily implemented. Vertical mixing of momentum, temperature and salinity is formulated in a simple Richardson number dependent algorithm having similar performance characteristics to a Mellor-Yamada level 2 scheme. To best preserve frontal boundaries the model has a sophisticated 3-D advection scheme, the Piecewise Parabolic Method (PPM), which has been shown [98] to have excellent long-time integration conservation properties.

### 7.1.2 Evolution of the WQM code

The model has previously been run on a 4-processor Cray EL94 machine. This has limited the size of grid that can be used; currently the grid covers the Southern North Sea between 51N (Dover Straits) and 56N, with 10 vertical levels, and a gridsize of $\sim$2.4km. There are three main objectives of moving the code to more powerful MPP machines:

1. To increase the coverage of the grid, allowing the modelling of the Northwest European continental shelf, with vertical resolution of 100 levels.
2. To enhance resolution in order to resolve the impact of fronts (either tidal or freshwater in origin) on the circulation, requiring a gridsize of 1km or less.
3. To include additional processes to describe sediment transport and resuspension, nutrient cycling and microbiological production.

The parallel code has been designed to be portable over different platforms with minimal source change. All code is written in Fortran-77 (plus MIL-STD extensions), with message passing carried out using the MPI standard. It should be noted that there is very little scope for use of BLAS in this code.

The parallelisation scheme is typical of many codes of this class. The grid is partitioned into rectangular subdomains in two of its dimensions, with full vertical columns of grids points held on each processor. Each subdomain has a 'halo' region which acts as a local communication cache for values on neighbouring subdomains. Updating the halo region requires only local communication and is carried out as soon as updated values are available. The halo update communications make heavy use of MPI derived datatypes. This considerably simplifies the programming and reduces the scope for errors, allowing each 3D halo edge to be transferred with a single message passing call. The current finite difference stencils used in the WQM code require a halo of only one grid point in width. However to allow for future developments of the model, the data setup and halo update routines have been written in a general way so that a larger halo region can be used by changing a single parameter.

The parallel code also makes heavy use of dynamic array allocation at run-time, allowing runs on different grid sizes and different numbers of processors, without recompilation. Since standard Fortran-77 does not explicitly include facilities for dynamic array management, the dynamic array allocation has been carried out using integer pointers into a single large statically allocated array. This scheme has been successfully used in the POLMP (POL Multiprocessing Program) code [96], which has been run on parallel machines from all major vendors. This scheme increases the complexity of the code at the top level, but once the 'dynamic arrays' have been passed down into subroutines they can be used as normal.

### 7.1.3 Performance Results and Conclusions

Initial tests of the parallel WQM code have been confined to a simplified problem involving a rectangular area of open sea, and no land areas. Runs have been carried out on the IBM SP2 and on the Cray T3D. No optimisation has yet been carried out, so these results should be treated as 'preliminary'.

Table 7.1: WQM: Initial Timings (secs) per timestep on the SP2 and T3D

| Parallel Timings | | | |
|---|---|---|---|
| Problem | No of nodes | | |
| Size | 1 | 4 | 16 |
| | IBM SP2 | | |
| $120\times60\times15$ | 11.2 | 3.5 | |
| $120\times120\times15$ | 23.0 | 6.4 | |
| $240\times120\times15$ | — | 12.4 | |
| $240\times240\times15$ | — | 24.5 | |
| | Cray T3D | | |
| $120\times60\times15$ | 14.3 | 4.1 | 1.3 |
| $120\times120\times15$ | 28.7 | 7.7 | 3.0 |
| $240\times120\times15$ | — | 15.3 | 4.6 |
| $240\times240\times15$ | — | 29.2 | 8.8 |

All SP2 figures in this section were obtained using the optimisation flags `-O3 -qhot` where possible. In practice, the program module containing the main timestepping loop had to be compiled without the optimisation in order to give correct results. However, the 6 routines called during the main timestepping loop (and accounting for all the execution time in practice) were compiled with the optimisation flags. The message passing library used was the Argonne/Mississippi State public domain MPI implementation, version 1.08.

Floating point operation rates were calculated by keeping an explicit running total of the number of FP operations carried out. This is updated at the end of each routine call, using a simple calculation. At the end of the final timestep the FP operation count for the main time-step loop is divided by the total time for the loop to give the Flop/s rate. Overall floating point operation rates are thus affected by not just per-node performance, but also by time spent in communications.

Table 7.1 shows times for the main time-step loop, on a number of different grid sizes, for varying node counts on the IBM SP2. For comparison, figures are also given for the Cray T3D. Table 7.2 shows total achieved Mflop/sec rates for varying node counts.

In practice, the halo updates can take a significant amount of time. Since this affects both overall timings and overall Mflop/s rates, timings for the halo updates alone are given in table 7.3. It is possible that the high cost of these operations on the SP2 is partly attributable to the use of the non-native MPI implementation; these tests need to be repeated with IBM's native MPI library.

Table 7.2: WQM: Initial Performance (Mflop) on the SP2 and T3D

| Total Mflop/sec | | | |
|---|---|---|---|
| Problem Size | No of Nodes | | |
| | 1 | 4 | 16 |
| | IBM SP2 | | |
| 120×60×15 | 16.0 | 51.4 | |
| 120×120×15 | 15.8 | 56.9 | |
| 240×120×15 | — | 59.6 | |
| 240×240×15 | — | 60.1 | |
| | Cray T3D | | |
| 120×60×15 | 12.6 | 44.0 | 141.4 |
| 120×120×15 | 12.7 | 47.3 | 123.2 |
| 240×120×15 | — | 48.2 | 159.2 |
| 240×240×15 | — | 50.8 | 167.9 |

Table 7.3: WQM: Halo Update Timings (secs) per Timestep on the SP2 and T3D

| Halo Timings | | |
|---|---|---|
| Problem Size | No of Nodes | |
| | 4 | 16 |
| | IBM SP2 | |
| 120×60×15 | 0.79 | |
| 120×120×15 | 0.87 | |
| 240×120×15 | 1.09 | |
| 240×240×15 | 2.09 | |
| | Cray T3D | |
| 120×60×15 | 0.18 | 0.18 |
| 120×120×15 | 0.43 | 0.25 |
| 240×120×15 | 0.49 | 0.30 |
| 240×240×15 | 0.80 | 0.65 |

# Bibliography


[1] A.A. Abrikosov, Physica C 244 (1995) 243.

[2] O.K. Andersen et al., Phys. Rev. B 49 (1994) 4145.

[3] K.J. Badcock, *The Implementation of Two and Three Dimensional Viscous Codes on Various Parallel Platforms*, Glasgow University, Aerospace Engineering Report 9415 (1995).

[4] D.R. Emerson and R.S. Cant, *Direct simulation of turbulent combustion on the Cray T3D - inital thoughts and impressions from and engineering perspective*, Parallel Computing (1995) submitted.

[5] G G Balint-Kurti, F Gögtas, S P Mort, A R Offer, A Laganà and O Gervasi *Comparison of time-dependent and time-independent quantum reactive scattering Li+HF - LiF+H* J. Chem. Phys. 99 (1993) 9567-84.

[6] I.J. Bush *Documentation for parallel MOLSCAT* (Daresbury Laboratory, 1994).

[7] J. Choi and J. Dongarra and D.W. Walker *PB-BLAS Reference Manual (Version 1.0Beta)*, Oak Ridge National Laboratory, Technical report, ORNL/TM-12469 (March 1994), Available by anonymous ftp from Netlib in directory scalapack.

[8] J. Čížek and J. Paldus, Int. J. Quantum Chem. Symp. 5 (1971) 359.

[9] J. Demmel and K. Stanley, *The Performance of finding eigenvalues and eigenvectors of dense symmetric matrices on distributed-memory computers*, preprint, University of California, Berkeley (1994).

[10] H. Ding et al., Phys. Rev. Lett. 74 (1995) 2784.

[11] J.J Dongarra and R.A. van de Geijn and R. Clint Whaley, *A Users' Guide to the BLACS*, (February 1993), University of Tennessee, Technical Report, Available by anonymous ftp from Netlib in directory pvm3.

[12] J.J. Dongarra and D.C. Sorensen, *A fully parallel algorithm for the symmetric eigenproblem*, SIAM J. Sci. Statist. Comput. 8, 139-154, (1987).

[13] J.J. Dongarra et al. *ScaLAPACK*.

[14] T.H. Dunning Jr, J. Chem. Phys. 90 (1989) 1007.

[15] D. Elwood, G. Fann and R. Littlefield *Parallel Eigensystem Solver*, PeIGS v0.0 reference manual, Batelle, Pacific Northwest Laboratory (2/8/93).

[16] R. Fehrenbacher et al., Phys. Rev. Lett. 74 (1995) 3884.

[17] A. Franz and P.L. Altick, *Electron impact ionisation of helium: coplanar, triply differential cross sections at high and intermediate energies*', J. Phys. B 25 (1992) 1577-90.

[18] GAMESS-UK is written by M.F. Guest, J.H. van Lenthe, J. Kendrick, K. Schoffel 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 parallel MP2 code is under development as part of the ESPRIT Europort2 project, IMMP.

[19] S. Hammarling *Parallel NAG* presentation at NAG annual colloquium (1994)

[20] R.J. Harrison and J. Nieplocha *TCGMSG* "readme" file in the source directory (PNNL 1994).

[21] R.W. Hockney and E.A. Carmona, *Comparison of communications on the Intel iPSC/860 and Touchstone Delta*, Parallel Computing 18 (1992) 1067-72.

[22] J.M. Hutson and S. Green, *MOLSCAT version 12* (1993). distributed in the U.K. by J.M. Hutson via CCP6.

[23] B. Johansson, I.A. Abrikosov, M. Alden, A.V. Ruban and H.L. Skriver, Phys. Rev. Lett. 74 (1995) 2335.

[24] M.M. Law, J.M. Hutson and A.Ernesti, *Fitting Molecular Potential Energy Surfaces*, (Daresbury Laboratory and CCP6 1994), ISBN 0-9522736-0-8.

[25] T.J. Lee, A.P. Rendell and P.R. Taylor, J. Chem. Phys. 94 (1990) 5463.

[26] R.J. Littlefield and K.J. Maschoff, *Investigating the performance of parallel eigensolvers for large processor counts*, PNL preprint (1992)

[27] R.J. Littlefield and K.J. Maschoff, *Investigating the performance of parallel eigensolvers for large processor counts*, preprint, Batelle Pacific Northwest Laboratory (1992).

[28] MPI Forum, *Document for a Standard Message-Passing Interface*, (April 1994), University of Tennessee, USA, available by anonymous ftp.

[29] Foster, I.T., Tilson, J.L., Wagner, A.F., Shepard, R., Bernholdt, D.E., Harrison, R.J., Kendall, R.A., Littlefield, R.J., Wong, A.T.: High Performance Computational Chemistry: (I) Scalable Fock Matrix Construction Algorithms. J. Computat. Chem. (1995) in press.

[30] Harrison, R.J., Guest, M.F., Kendall, R.A., Bernholdt, D.E., Wong, A.T., Stave, M., Anchell, J.L., Hess, A.C., Littlefield, R.J., Fann, G.I., Nieplocha, J., Thomas, G.S., Elwood, D., Tilson, J., Shepard, R.L., Wagner, A.F., Foster, I.T., Lusk, E., Stevens, R.: High Performance Computational Chemistry:(II) A Scalable SCF Program. J. Computat. Chem. (1995) in press

[31] A.T. Wong, R.J. Harrison and A.P. Rendell, Parallel Direct Four Index Transformation, manuscript in preparation

[32] D.E. Bernholdt and R.J. Harrison, Large-Scale Correlated Electronic Structure Calculations: the RI-MP2 Method on Parallel Computers, manuscript in preparation

[33] D.E. Bernholdt and R.J. Harrison, Orbital Invariant Second-Order Many-Body Perturbation Theory on Parallel Computers. An Approach for Large Molecules, J. Chem. Phys. (1995) in press

[34] MPI, University of Tennessee, MPI: A Message-Passing Interface Standard (1994).

[35] R.A. Kendall, R.J. Harrison, R.J. Littlefield, and M.F. Guest, *High Performance Computing in Computational Chemistry: Methods and Machines*, Reviews in Computational Chemistry, 6 (1995) pp 209-316, Ed. K.B. Lipkowitz and D.B. Boyd, and published by VCH Publishers, Inc., New York.

[36] The MPI Forum: A Message Passing Interface, Supercomputing '93 (IEEE computer Society Press, Los Alamitos, California, Portland, OR, 1993), pp. 878-883.

[37] M.E. Colvin, C.L. Janssen, R.A. Whiteside, and C.H. Tong, Theor. Chim. Acta 84 (1993) 301–314.

[38] T.R. Furlani, H.F. King, J. Comp. Chem. (1995) in press.

[39] J. Nieplocha, R.J. Harrison, and R.J. Littlefield, *Global Arrays; A Portable Shared Memory Programming Model for Distributed Memory Computers*, Supercomputing '94 (IEEE Computer Society Press, Washington, D.C., 1994.

[40] A.P. Rendell, M.F. Guest and R.A. Kendall, J. Comp. Chem. 14 (1993) 1429–1439.

[41] J. Choi, J.J. Dongarra, L.S. Ostrouchov, A.P. Petitet, D.W. Walker, R.C. Whaley, *The Design and Implementation of the SCALAPACK LU, QR, and Cholesky Factorization Routines*, Oak Ridge National Laboratory, Oak Ridge, TN, report ORNL/TM-12470 (LAPACK Working Note 80), September 1994. Available by anonymous FTP from ftp.netlib.org.

[42] J. Choi, J.J. Dongarra, D.W. Walker, *The Design of A Parallel Dense Linear Algebra Software Library: Reduction to Hessenberg, Tridiagonal, and Bidiagonal Form*, Oak Ridge National Laboratory, Oak Ridge, TN, report ORNL/TM-12472 (LAPACK Working Note 92), January 1995. Available by anonymous FTP from ftp.netlib.org.

[43] J. Choi, J.J. Dongarra, L.S. Ostrouchov, A.P. Petitet, R.C. Whaley, J. Demmel, I. Dhillon, K. Stanley, LAPACK Working Note: Installation Guide for ScaLAPACK, Department of Computer Science, University of Tennessee, Knoxville, TN, February 28, 1995. Available by anonymous FTP from ftp.netlib.org.

[44] E.R. Jessup, Ph. D. Thesis (1989) Yale University.

[45] S.S. Lo, B. Phillipps, A. Sameh, IAM J. Sci. Stat. Comput. 8(2) (1987).

[46] G. Fann, R.J. Littlefield, *Parallel Inverse Iteration with Reorthogonalization*, Sixth SIAM Conference on Parallel Processing for Scientific Computing (SIAM, 1993), 409–413.

[47] Anderson, E., Bai, Z., Bischof, C., Demmel, J., Dongarra, J.J., Du Croz, J., Greenbaum, A., Hammerling, S., McKenney, A., Ostrouchov, S., Sorensen, D.: LAPACK User's Guide (SIAM, 1992).

[48] M. Feyereisen, G. Fitzgerald and A. Komornicki, Chem. Phys. Lett. 208 (1993) 359–363 .

[49] O. Vahtras, J. Almlof and M. Feyereisen, Chem. Phys. Lett. 213 (1993) 514.

[50] P. Hohenberg and W. Kohn, Phys. Rev. B. 136 (1964) 864–871.

[51] W. Kohn and L.J. Sham, Phys. Rev. A. 140 (1965) 1133–1138.

[52] E. Wimmer, in Density Functional Methods in Chemistry, J.K. Labanowski, J.W. Andzelm, Eds. (Springer-Verlag, 1991) 7–31.

[53] B.I. Dunlap, J.W.D. Connolly and J.R. Sabin, J.R.: J. Chem. Phys. 71 (1979) 3396.

[54] B.I. Dunlap, J.W.D. Connolly and J.R. Sabin, J.R., J. Chem. Phys. 71 (1979) 4993.

[55] G.B. Bacskay, Chem. Phys. 61 (182) 385.

[56] P. Pulay, Chem. Phys. Lett. 100 (1983) 151–154.

[57] P. Pulay and S. Saebo, Theor. Chim. Acta 69 (1986) 357–368.

[58] S. Saebo and P. Pulay, Annu. Rev. Phys. Chem. 44 (1993) 213–236.

[59] H.A. Fruechtl and R.A. Kendall, *A Scalable Implementation of the RI-SCF algorithm*, manuscript in preparation.

[60] G.D. Purvis and R.J. Bartlett, J. Chem. Phys. 76 (1982) 1910.

[61] K. Raghavachari, G.W. Trucks, J.A. Pople and M. Head-Gordon, Chem. Phys. Lett. 157 (1989) 479.

[62] A.P. Rendell, T.J. Lee, A. Komornicki and S. Wilson, Theor. Chim. Acta 84 (1993) 271.

[63] A.P. Rendell, M.F. Guest and R.A Kendall, J. Comput. Chem. 14 (1993) 1429.

[64] G.E. Scuseria, C.L. Janssen and H.F. Schaefer, J. Chem. Phys. 89 (1988) 7382.

[65] G.E. Scuseria, Chem. Phys. Lett. 243 (1995) 193.

[66] Z.-X. Shen et al., Phys. Rev. Lett. 70 (1993) 1553.

[67] G.L.G. Sleijpen and H.A. van der Vorst, *A Jacobi-Davidson iteration method for linear eigenvalue problems*, Department of Mathematics, University of Utrecht, Preprint nr. 856. (1995)

[68] M.B. Suvasini et al., Phys. Rev. B 48 (1993) 1202.

[69] P.N. Swarztrauber, *Vectorizing the FFTs, in Parallel Computations*, (G. Rodrigue, ed.), Academic Press, 1982, pp. 51-83.

[70] Z. Szotek, W.M. Temmerman, H. Winter, Phys. Rev. Lett. 72 (1994) 1244.

[71] A. Svane, Phys. Rev. Lett. 72 (1994) 1248.

[72] A. Svane Phys. Rev. B. (1995) submitted.

[73] H. Tal-Ezer and R. Kosloff J. Chem. Phys. 81 (1984) 3967.

[74] W.M. Temmerman, Z. Szotek, B.L. Gyorffy, O.K. Andersen, O. Jepsen, Phys. Rev. Lett. (1995) (submitted).

[75] W.M. Temmerman, Z. Szotek, A. Svane and H. Winter (1995) to be submitted.

[76] C. Temperton, *Self sorting mixed-radix fast Fourier transforms*, J. Comp. Phys. 52 (1983) 1-23.

[77] J. Tennyson, J.R. Henderson and N.G Fulton, *DVR3D for the fully pointwise calculation of the ro-vibrational spectra of triatomic molecules* Computer Phys. Comm. (1994) submitted.

[78] M. Turkyilmazoglu (1994), *A Numerical Study of Three Dimensional Unsteady Viscous Flows In A Lid Driven Cavity.* Dept. of Mathematics, University of Manchester.

[79] H.-J. Werner and P.J. Knowles, *MOLPRO*, (1993) University of Sussex, distributed in the U.K. by P.J. Knowles via CCP1.

[80] C.T. Whelan, H.R.J. Walters, R.J. Allan and X. Zhang, *e-2e, effective charges, distorted waves and all that!*, in '(e,2e) and related Processes' Proc. NATO ARW Cambridge, 27/9-1/10/92, ed. C.T.Whelan, H.R.J.Walters, A.Lahmam-Bennani and H.Ehrhardt (Kluwer Academic Publishers: 1993) p1-32. ISBN 0-7923-2458-7.

[81] F.H. Harlow and J.E. Welch, Phys. Fluids 8, 2182-2189, 1965.

[82] S.A. Orszag, Phys. Fluids supplement II, 250-257, 1969.

[83] S. Lele, J. Comput. Phys. 103, 16-43, 1992.

[84] T.F. Chan and C-C.J. Kuo, *Parallel Elliptic Preconditioners: Fourier Analysis and Performance on the Connection Machine*, Computer Physics Communications, Vol. 53, 1989, pp 237–252.

[85] D. Bailey, *RISC Processors and Scientific Computing*, NAS Technical Report RNR-93–004 (1993).

[86] T. Brandes, *ADAPTOR: Language Reference Manual, Version 2.0 (March 1994)*, German National Research Center for Computer Science (GMD), Report Adaptor 3 (March 28, 1994).

[87] V. Getov, T. Brandes, B. Chapman, A. Dunlop, A.J. Hey and D. Pritchard, *Comparison of HPF-like Systems*, University of Southampton, GMD, University of Vienna, PPPE Deliverable D4.3a, (November 22, 1993)

[88] T. Brandes, *Evaluation of High Performance Fortran on some Real Applications*, German National Research Center for Computer Science (GMD), (1993).

[89] L. Valiant, *A Bridging Model for Parallel Computation*, Communications of the ACM 33 (1990).

[90] R. Miller and J. Reed, *The Oxford BSP Library: Users Guide*, Version 1.0, Oxford Parallel (1993).

[91] C.Temperton, *A Generalised Prime Factor FFT Algorithm for any N = (2**P)(3**Q)(5**R)*, SIAM J. Sci. Stat. Comp. (May 1992).

[92] R.J. Littlefield and K.J. Maschoff, *Investigating the performance of parallel eigensolvers for large processor counts*, Theor. Chim. Acta 84 (1993) 457-73.

[93] H. Pritchard, *The Global Array Tool*, CRI (13/4/95).

[94] C. Pisani, R. Dovesi and C. Roetti, *Hartree-Fock ab-initio treatment of crystalline systems*, Lecture Notes in Chemistry, Vol. 48, Springer Verlag, (Heidelberg, 1988).

C. Pisani and R. Dovesi, Int. J. Quantum Chem. 17 (1980) 501.

R. Dovesi, C. Pisani, C. Roetti, M. Causà and V.R. Saunders, *CRYSTAL 88*, QCPE program n. 577, (Bloomington, Indiana, 1989).

R. Dovesi, V.R. Saunders, C. Roetti, *CRYSTAL 95 User Manual*, University of Torino, (Torino 1992).

V.R. Saunders, Faraday Symp. Chem. Soc. 19 (1984) 79.

[95] R.J. Blake, D.R. Emerson and R.J. Allan, *FLOW: a parallel benchmark code for high speed air flow*, version 1, Daresbury Laboratory (1992).

[96] M. Ashworth, *Proudman Oceanographic Laboratory Multiprocessing Program (POLMP): Benchmarking Guide*, NERC Computer Services, August 1994.

[97] R. Proctor and I.D.James, *A Fine-Resolution 3D Model of the Southern North Sea*, Journal of Marine Systems (in press).

[98] I.D. James, *Advection schemes for shelf sea models*, Journal of Marine Systems (in press).

[99] T.L. Freeman and C. Phillips, *Parallel Numerical Algorithms*, Prentics Hall (1992), ISBN 0-13-651597-5.

[100] E. Ö. Sveinbjörnsson, and O. Engström, Phys. Rev. B 52 (1995) 4884.

[101] R. Jones, S. Öberg, J. Goss, P.R. Briddon, A. Resende, Phys. Rev. Lett., 75 (1995) 2734-37.

[102] S.J. Uftring, M. Stavola, R.M. Williams, and G.D. Watkins, Phys. Rev. B 51 (1995) 9612.

[103] R. Jones et al., J. Phys. C 20 (1987) L271-3.

[104] A.R.Walton and D.E.Manolopoulos *A new semiclassical initial value method for Franck-Condon spectra* Mol. Phys. 87 (1996) 961-78

# Chapter 8

# APPENDICES

## 8.1 Appendix I: Computer Systems Used and Compilation Options

The following benchmark platforms were available for test purposes

### 8.1.1 IBM SP2

Timings are from the 512-node machine at the Cornell Theory Center and the 16-node machine at Daresbury Laboratory (DL). Nodes were 66.7 MHz RS/6000 RIOS-2 processors, of the "thin" variety at Cornell and the TN2-nodes at Daresbury.

The IBM SP2 at Daresbury consists of 14 thin node 2 (TNT) processors, each having 64 Mbytes of memory and 2 wide nodes, each with 128 Mbytes of memory. All nodes have 128 Kbytes of level 1 data cache and 32 Kbytes of instruction cache. The processors standard Power2 Architecture RS/6000 processors capable of four floating-point arithmetic instructions per clock cycle providing the quad-load pipes can keep the cache filled. The SP2 nodes therefore yield a peak performance of 267 Mflops.

The interconnect between the nodes is a high performance switching network with low latency and capable of sustaining a high bandwidth. In addition each node has an Ethernet connection and the two wide nodes an FDDI connection. All nodes have fast wide SCSI II discs attached for scratch space and access to permanent disc store is via NFS, using the wide nodes and FDDI connections. The wide nodes are used for login, editing and compilation as well as computation.

The high-performance switch configuration at Cornell is worthy of note, as it is somewhat nonstandard due to the unusually large configuration installed there. The additional switching hardware required may be expected to degrade message-passing performance when compared to a smaller configuration.

Compilation options
`xlf -O3 -qhot -qnosave -qarch=pwr2 -lesslp2`

### 8.1.2 Cray T3D

Timings are from the 512-processor T3D at EPCC. The Cray T3D system uses DEC alpha EV4 nodes with 151 MHz clock therefore rated at 151 MFlops peak floating points arithmetic performance. The use of a direct-mapped cache with only a single read-ahead buffer makes some code re-structuring essential for optimisation.

Compilation options
`cf77 -c -Ccray-t3d -O1 -Oscalar3` with BLAS library and level-2 and a level-3 Fortran code
`cf77 -c -Ccray-t3d -Wf''-o unroll''` with Fortran BLAS level-1

### 8.1.3 iPSC/860

Timings are from the 64-processor Intel iPSC/860 at Daresbury.

Compilation options
`if77 -O4 -Mvect -lclasspack`

### 8.1.4 Parsytec GC

The Parsytec GC at The University of Paderborn, Germany consists of 64 nodes, each comprising two PowerPC 601 processors and T9000 transputer based switching hardware. Message-passing was performed using Parsytec's PowerPVM implementation, GA-Tools was ported to the Parsytec by dedicating one of the two processors per node to global memory management, with communication using PowerPVM. 64Mb of memory are available on each node.

### 8.1.5 SGI PowerChallenge Array

Timings were collected from the European Power Challenge Array at Cortailloid in Switzerland. All nodes used for benchmarking were R8000 processors running at 75 MHz with a main-memory configuration of 2-2.5 Gb. Message passing is performed using TCGMSG and GA-Tools, which within a node is implemented using shared-memory, and between nodes uses TCP-IP to communicate over the HIPPI switch (the latter capability has not yet been benchmarked).

### 8.1.6 HP Workstation cluster

The HP workstation cluster at Daresbury consists of one HP/9000-755 workstation with 128 Mb memory, and 4 HP/9000-735s with 80 Mb memory. They are connected by ethernet and FDDI networks, the latter being used for the tests reported here.

### 8.1.7 IBM Workstation cluster

The IBM workstation cluster at Daresbury consists of six IBM RS/6000. Tci11-tci13 are three model 530H systems (approx. 50M Flops) and tci14-tci16 are three model 370 systems (approx. 100 MFlops). They are connected by ethernet, FDDI and ATM networks. Tci12 is not accessible for running user code but acts as a file server.

Compilation options
`xlf -O3 -qhot -qnosave -qarch=pwr2`

### 8.1.8 Cray J932-4096

The Rutherford Appleton Laboratory 32-processor Cray J90 was used. The Cray J932-4096 is a shared memory, vector computer manufactured by Cray Research Inc. Each processor has a clock cycle of 10 nsec and chained add and multiply vector pipes giving 200 Mflop per node and an overall performance of 6.4 Gflops with a memory bandwidth capable of sustaining peak performance. The 4 Gbyte main memory is a "flat" high speed memory for maximum vector performance, unlike earlier Cray computers the J90 has a small cache memory for instructions and scalar data only.

Compilation options
`cf77 -Oscalar3 -Ovector3 -Otask3`

### 8.1.9 PowerPC model 250T

Tci17 is a 66 MHz PowerPC RISC/6000 model 250T (approx. 50 MFlops).

Compilation options
`xlf -O3 -qhot -qnosave -qarch=ppc`

## 8.2 Appendix II: Message Passing and Communications

### 8.2.1 Message Passing Environments

*R.J. Allan*

Several message-passing harnesses are available on the SP2. These in turn call the Communication Subsystem (CSS) library routines which handle the communications between nodes. There are two separate implementations of this library, the Internet Protocol (IP) and the User Space (US). The IP version uses the Internet Protocol while the US version gives the best performance because it is designed for the high performance switch.

The current configuration of the system also allows for shared or dedicated access to the nodes CPU and its high performance switch adaptor. Clearly if there is more than one process running on a node this will impact the execution rate, because more memory will be used and access patterns will change. There is also more process switching and running too many processes per node can (and does) significantly degrade performance. Processes include AIX kernel processes so none of the timing results in this document are free from context switching uncertainties (see Hockney and Carmona [21]). We tested the effects of reproducing the timed runs on a busy system using MPL. Other tests were done on a quiet system using IP or US protocols.

The results have been characterised by a best-fit straight line using a least squares procedure. This yields the gradient and intercept of the line taking both cpu time and elapsed time. We report only elapsed time here, as it is likely to be the only measure of interest. Parameters extracted from the least-squares routine are a and b and the elapsed time is related to

$t = a + b \times n$

where n is the number of bytes of data transmitted.

t is measured in microseconds, a in microseconds (the startup latency) and b in microseconds per byte. These parameters are related to those of Hockney and Carmona [21] as follows:

$r_\infty = 1/b$; the asymptotic rate in MB/s and

$n_{\frac{1}{2}} = a \times r_\infty$; the message length at which this half this rate is achieved.

We show below results of performance for the various message-passing harnesses tested These include – MPL, MPI, PVM3, PVMe, TCGMSG, GA Tools, HPF and BSP.

### MPL

MPL is IBM's native message-passing environment which is of course the fastest way to transmit data between SP2 processors. We were able to achieve a scaling of

$t = 86 + 0.0284 \times n$ microsecs equivalent to an asymptotic rate of 35.2 MB/s on the US network.

We show in Figure 8.1 typical variation of the times obtained on the IP network on a loaded system.

### MPI

The MPI software on the Daresbury SP2 is a prototype, rather than IBM's own release which we expect shortly. We nevertheless achieved a scaling of

$t = 114 + 0.02876 \times n$ microsecs giving an asymptotic speed of 34.77 MB/s.

### PVM3

There is no native PVM3 implementation on either the SP2 or the T3D. We consider that PVM has been useful in supporting the exploitation of heterogeneous clusters of workstations but is not the

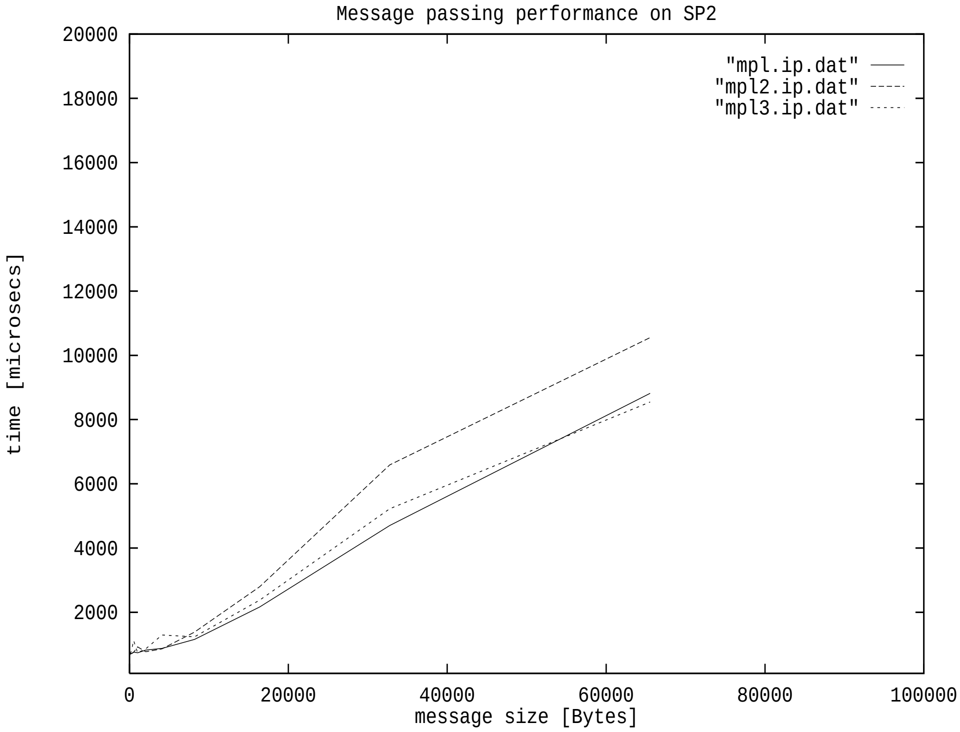



Figure 8.1: Performance of the ip and us networks for a dedicated system

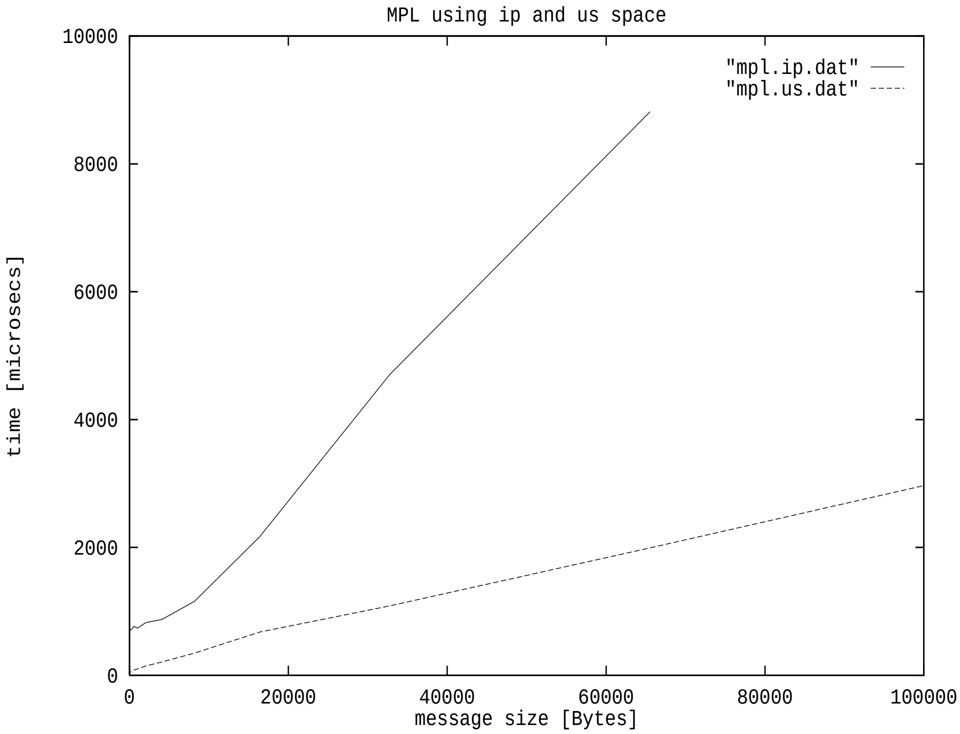



Figure 8.2: Performance of the IP network on a shared system cf. US network

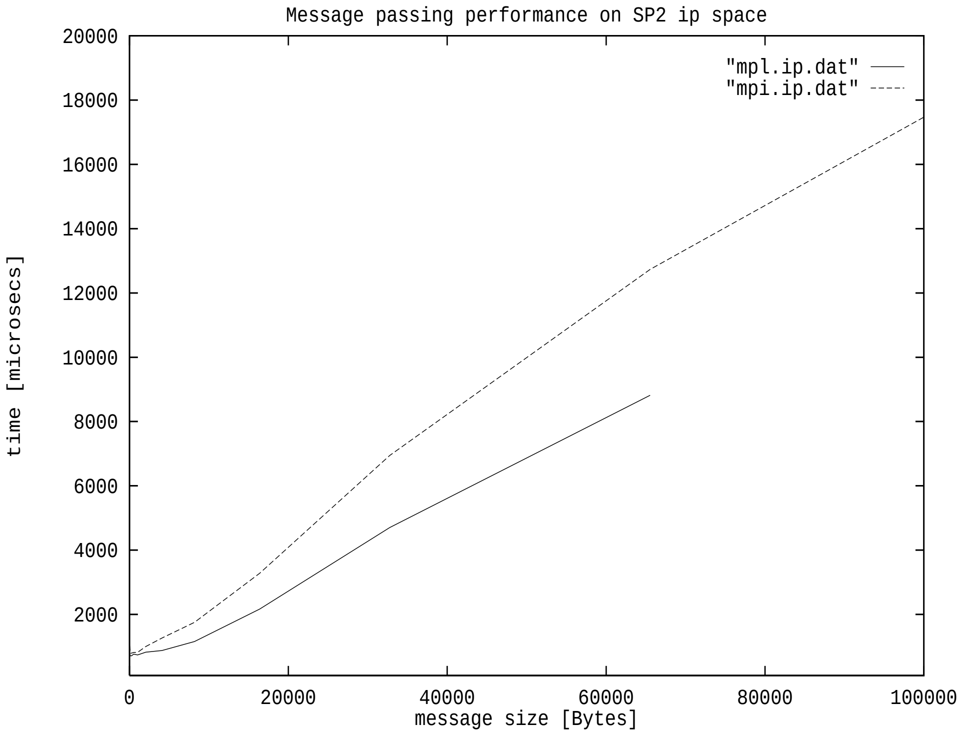



Figure 8.3: Performance of MPL and MPI in the IP space

most satisfactory way to use MPP supercomputers. We would rather advocate the use of MPI as this "international standard" becomes more mature.

### PVMe

PVMe is an "optimised" version of PVM2 provided as an interim measure by IBM Italy. It is not portable and we have not encouraged its use. It is likely to be replaced by IBM's implementation of MPI soon and we therefore recommend this.

The option PVMe_MEMCPY=power2 was used to improve performance, which yielded

$t = 535 + 0.0293 \times n$ microsecs or 34.14 MB/s.

### TCGMSG

This message-passing harness was written by Robert Harrison et al. of Battelle Pacific Northwest National Laboratory, USA (previously at Daresbury Laboratory and Argonne National Laboratory). It is a very robust harness available on many platforms. However it is likely that it will be superseded in the future by MPI as the latter becomes more mature through vendors' support.

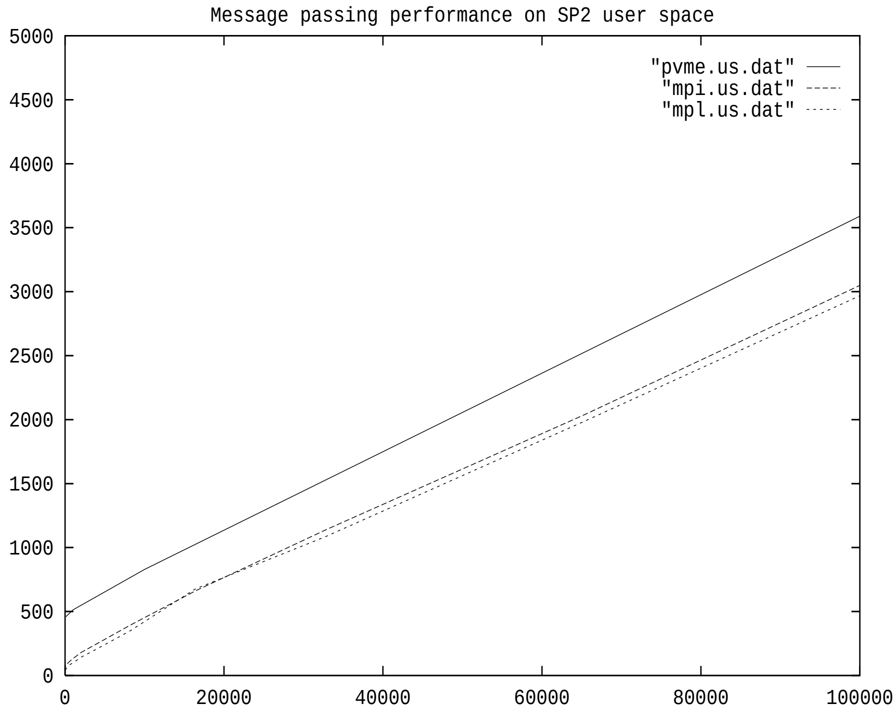



Figure 8.4: Performance of PVMe, MP and MPI in the us space

### 8.2.2 Data Parallel and Virtual Shared Memory Environments

Information on data parallel environments will be included in a future updated version of this report. They are the subject of current investigation as explained below. The environments mentioned here include examples of 'single-ended communications'. It is almost certain that this mode will become the norm in all future software, since it permits asynchronicity because only one processor engenders communication to memory on another one. This ultimately means that you don't need to know where the data is stored, and a compiler can automatically distribute arrays as in HPF (currently with a little help).

#### SHMEM

SHMEM is the Cray implementation of so called "single-ended" communications allowing asynchronous access to data stored on another processor, in effect a form of "virtual shared" memory. Not only is this the fastest form of communication on the Cray T3D, permitting asynchronous access to static data on all processors, but it is a rôle model for future message-passing systems. We have advised our consortium collaborators to use this model but, unfortunately, it is not portable yet. A separate report is available showing performance.

### HPF

Study of the utility of HPF for a number of applications in science and engineering is beginning. The CRAFT model on the Cray T3D has already been used by at least one consortium (the TMCODE for turbulence transition modelling). On other systems a variety of software has been tested. ADAPTOR (from GMD, Berlin) [86] is available on the Intel iPSC/860. The N.A. Software HPF Mapper and Fortran-90 compiler is available on Sun workstations on the Daresbury LAN, and recently became available on the SP2 for appraisal. IBM's own HPF compilation system is also expected shortly and will be made available for use. We currently have a beta-test license to evaluate the VAST-HPF source converter from Pacific Sierra Inc. and are testing it on the SP2 and T3D on a sample application. Similarly the Portland Group HPF compiler is available on the T3D.

An interesting report was written by T.Brandes [88] and by Getov et al. [87].

### GA-Tools

The GA-Tools developed by Robert Harrison and Rick Littlefield at Battelle Pacific Northwest National Laboratory, USA [39], are available on both the T3D, SP2 and other systems at Daresbury. They provide an easy interface to 'virtual shared memory' for the storage and asynchronous access of large matrices. Some global operations are provided too.

The software currently requires the TCGMSG message-passing harness, which is also available, but will be ported to MPI in the future.

A number of applications, particularly in the area of *ab initio* quantum chemistry already make heavy use of GA-Tools and we see this kind of programming as a possible future standard. At the moment it is partly viewed as an interim between the message-passing model and HPF permitting a good level of optimisation through explicit control of data locality by the programmer.

A rather thorough report exists benchmarking most functionality of the GA-Tools on a Cray T3D, Intel Paragon and IBM SP1 and SP2. This was written by Howard Pritchard [93].

The software is available from *anonymous@ftp.pnl.gov* .

### BSP

The BSP library was written by Richard Miller et al. of the Oxford computing Laboratory [90]. It provides an implementation of the ideas of Leslie Valiant – his bulk synchronous programming model [89]. The library has only three subroutines which permit asynchronous single-ended communications to be performed and the application to be synchronised in supersteps.

At least one application in computational engineering (by Kevin Parrott et al.) uses this model very successfully on the T3D.

**Other Software**

We have investigated other software which attempts to implement virtual shared memory or some other high level of global operations. A larger programming exercise was also attempted at Daresbury using both Fortran-77 and C++. This was the PARLANCE project (Parallel Library and Numerical Computing Environment). It provided us with valuable experience of the requirements and difficulties of supporting such packages.

A separate report detailing all the packages we have investigated is available either in hard copy form from the authors

R.J.Allan and P.Lockey *Parallel Application Software on High Performance Computers: A survey of parallel software packages of potential interest in scientific applications* 2nd edition (Daresbury Laboratory, 1996)

or at WorldWide Web URL *http://www.dl.ac.uk/TCSC/HPCI/reports/* .

## 8.3 Appendix III: Linear Algebra and Mathematical Libraries

### 8.3.1 Basic Linear Algebra Subprograms (BLAS)

*P.A. Walmsley, I.J. Bush and P. Lockey*

We note that a fuller report on this work is in preparation and detailed results can be obtained from the authors.

I.J.Bush, P.Walmsley and R.J.Allan*Parallel Application Software on High Performance Computers: Basic Linear Algebra Subprograms* (Daresbury Laboratory, 1996)

**Introduction**

Many scientific and engineering applications perform numerically intensive calculations relying heavily on vector and/or matrix operations. As a result the Basic Linear Algebra Subprograms library (BLAS), the Engineering Scientific Subroutine Library (ESSL) and the FORTRAN-77 equivalent were benchmarked on a single node on the IBM SP2 as well as on the CRAY T3D and INTEL iPSC/860.

Note that the BLAS library is not tuned for the POWER2 architecture and all real codes would use libessl.a. The libblas.a is only available for historical reasons.

Two types of performance measurements were taken:

- Calling the routines without previously referencing the vectors or matrices, i.e. the data is not initialised and is therefore assumed not to be in cache.
- Initialise the vectors and matrices before calling the routines, i.e. some/all the data is assumed then to be in cache.

In both cases a call to each routine was made once only. Details of the specific routines tested and the performance on the three machines is described below.

### BLAS 1 Benchmarks

Two of the most commonly used routines, DDOT and DAXPY, were tested, along with the equivalent FORTRAN-77 code. DDOT computes the dot product of two vectors where as DAXPY updates a vector with the product of a scalar and another vector.

Level 1 routines make little reuse of data and their performance is therefore limited by optimising the use and bandwidth between cache and registers.

The benchmarks were carried out for a range of double precision vectors from 256 to 128K elements.

**Table 3.1.1a - DDOT Results for uninitialised Data**

| Vector Size | Performance (MFlops) | | | | | | |
|---|---|---|---|---|---|---|---|
| | SP2 | | | T3D | | iPSC/860 | |
| | BLAS | ESSL | FORTRAN | BLAS | FORTRAN | BLAS | FORTRAN |
| 256 | 10.20 | 3.56 | 10.15 | See Note 1 | | 5.99 | 3.12 |
| 512 | 10.83 | 5.68 | 10.90 | ‘‘ | | 6.87 | 3.29 |
| 1024 | 11.16 | 8.25 | 11.30 | ‘‘ | | 7.74 | 5.56 |
| 2048 | 11.24 | 10.40 | 11.51 | ‘‘ | | 8.58 | 8.56 |
| 4096 | 11.62 | 12.16 | 11.47 | ‘‘ | | 11.76 | 11.59 |
| 8192 | 11.41 | 12.43 | 11.64 | ‘‘ | | 14.30 | 14.15 |
| 16384 | 11.08 | 12.98 | 11.00 | ‘‘ | | 15.97 | 15.88 |
| 32768 | 10.54 | 12.27 | 10.75 | ‘‘ | | 17.03 | 16.95 |
| 65536 | 10.65 | 12.60 | 10.57 | ‘‘ | | 17.59 | 17.27 |
| 98304 | 10.47 | 12.17 | 10.63 | ‘‘ | | 17.57 | 17.51 |
| 131072 | 10.60 | 12.56 | 10.64 | ‘‘ | | 17.65 | 17.72 |

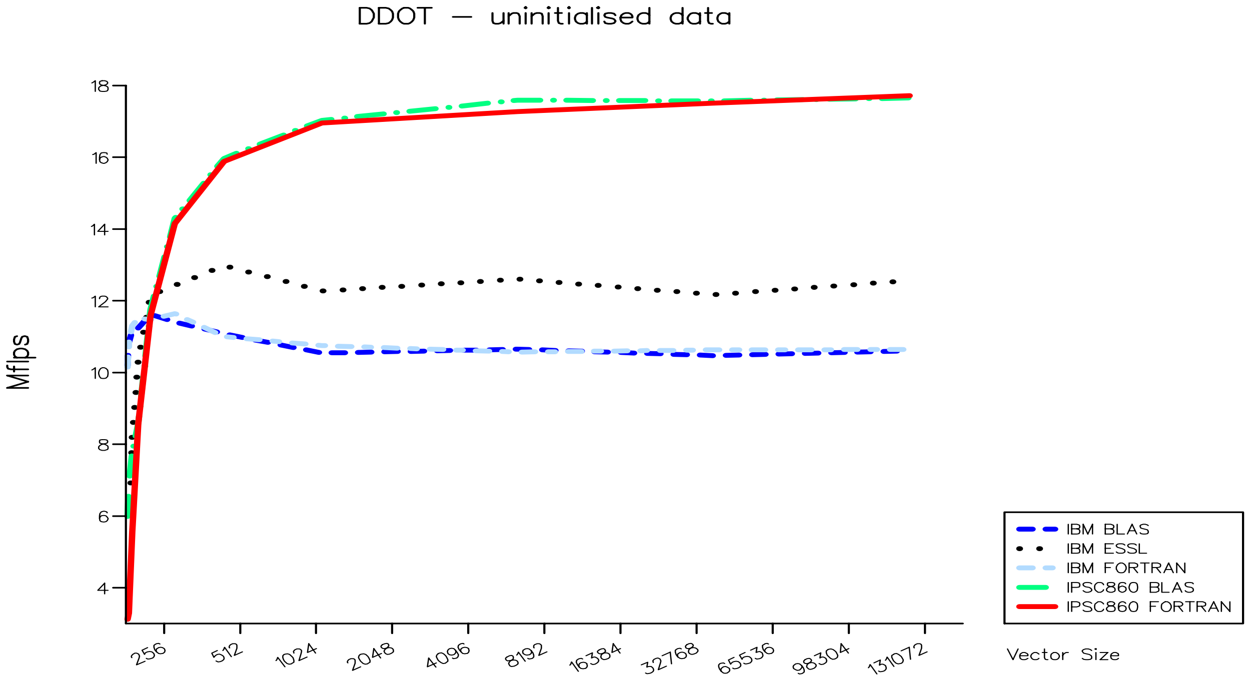



**Table 3.1.1b - DDOT Results for initialised Data**

| Vector Size | Performance (MFlops) | | | | | | |
|---|---|---|---|---|---|---|---|
| | SP2 | | | T3D | | iPSC/860 | |
| | BLAS | ESSL | FORTRAN | BLAS | FORTRAN | BLAS | FORTRAN |
| 256 | 48.81 | 4.75 | 53.69 | 15.93 | 12.00 | 7.20 | 11.58 |
| 512 | 56.14 | 9.29 | 59.65 | 21.33 | 12.42 | 7.63 | 13.99 |
| 1024 | 61.58 | 17.45 | 62.93 | 25.12 | 12.53 | 6.94 | 15.59 |
| 2048 | 63.98 | 31.81 | 65.07 | 25.86 | 12.55 | 8.28 | 17.14 |
| 4096 | 64.40 | 48.63 | 64.95 | 31.11 | 10.83 | 11.43 | 17.60 |
| 8192 | 44.78 | 37.32 | 49.29 | 32.19 | 9.22 | 14.08 | 18.05 |
| 16384 | 31.43 | 33.94 | 33.20 | 32.85 | 9.03 | 15.86 | 16.98 |
| 32768 | 34.36 | 40.44 | 33.84 | 33.17 | 9.22 | 16.44 | 18.07 |
| 65536 | 34.86 | 44.69 | 35.40 | 33.38 | 9.19 | 17.27 | 17.84 |
| 98304 | 34.03 | 45.47 | 35.12 | 33.09 | 9.18 | 17.54 | 18.00 |
| 131072 | 35.40 | 46.42 | 35.61 | 33.45 | 9.18 | 17.60 | 17.96 |

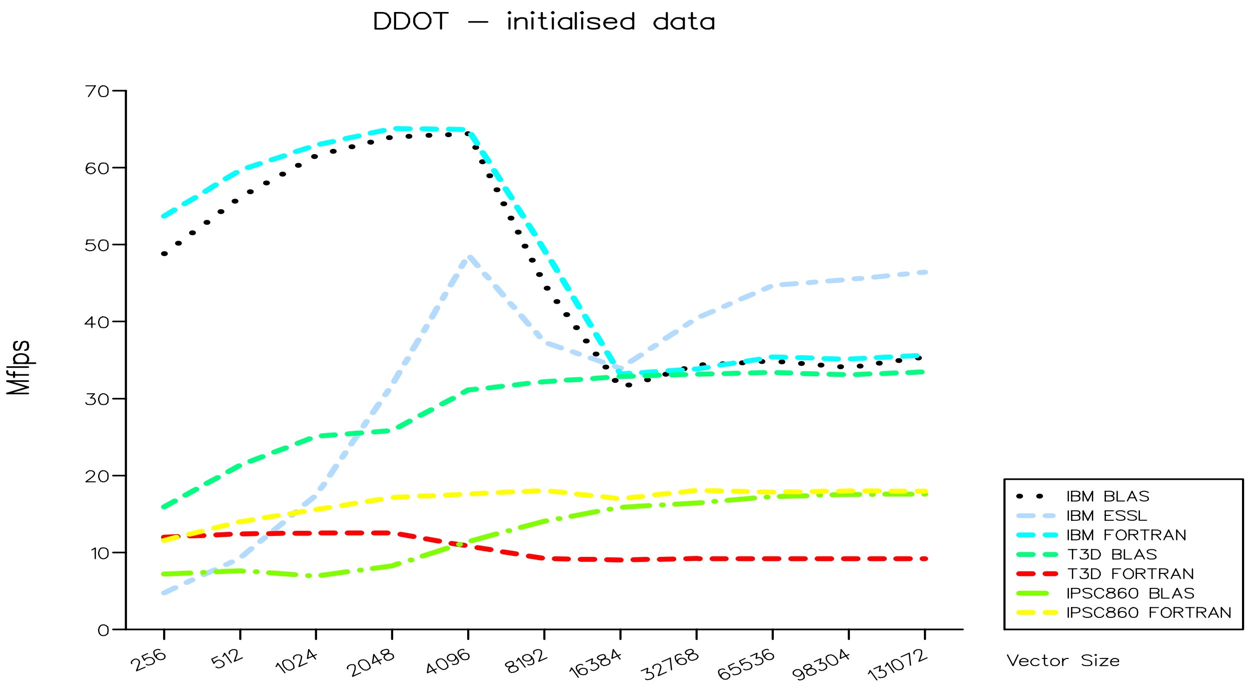



**Table 3.1.1c - DAXPY Results for Uninitialised Data**

| Vector Size | Performance (MFlops) | | | | | | |
|---|---|---|---|---|---|---|---|
| | SP2 | | | T3D | | iPSC/860 | |
| | BLAS | ESSL | FORTRAN | BLAS | FORTRAN | BLAS | FORTRAN |
| 256 | 11.16 | 3.68 | 11.64 | See Note 1 | | 4.71 | 1.74 |
| 512 | 10.98 | 4.34 | 11.53 | ‘ ‘ | | 4.93 | 3.00 |
| 1024 | 12.06 | 8.09 | 12.40 | ‘ ‘ | | 5.40 | 4.68 |
| 2048 | 12.36 | 10.04 | 12.50 | ‘ ‘ | | 6.50 | 6.42 |
| 4096 | 12.56 | 11.98 | 12.81 | ‘ ‘ | | 8.00 | 7.88 |
| 8192 | 12.32 | 12.39 | 12.63 | ‘ ‘ | | 8.96 | 8.92 |
| 16384 | 11.75 | 12.14 | 12.21 | ‘ ‘ | | 9.96 | 9.55 |
| 32768 | 11.99 | 12.71 | 11.82 | ‘ ‘ | | 9.95 | 9.73 |
| 65536 | 11.64 | 12.03 | 11.57 | ‘ ‘ | | 10.05 | 9.94 |
| 98304 | 11.68 | 12.41 | 11.59 | ‘ ‘ | | 10.09 | 10.05 |
| 131072 | 11.51 | 12.27 | 11.47 | ‘ ‘ | | 10.15 | 10.08 |

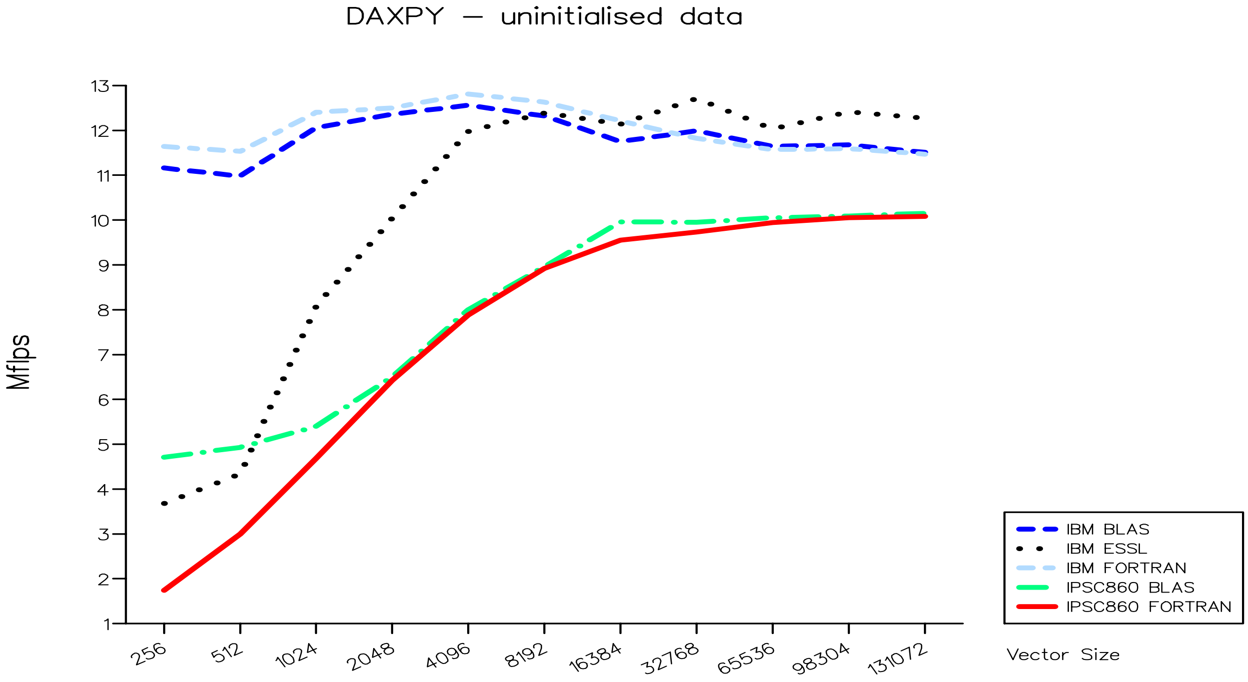
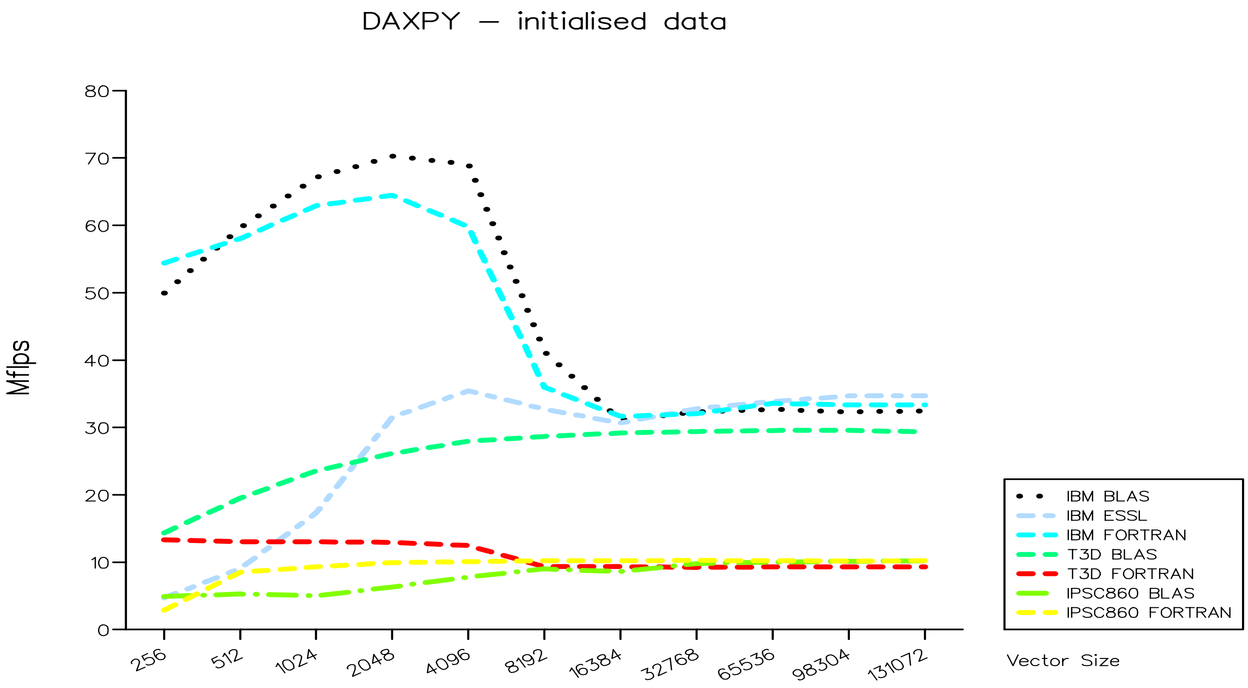



**Table 3.1.1d - DAXPY Results for Initialised Data**

| Vector Size | Performance (MFlops) | | | | | | |
|---|---|---|---|---|---|---|---|
| | SP2 | | | T3D | | iPSC/860 | |
| | BLAS | ESSL | FORTRAN | BLAS | FORTRAN | BLAS | FORTRAN |
| 256 | 49.94 | 4.69 | 54.37 | 14.34 | 13.31 | 4.91 | 2.85 |
| 512 | 59.65 | 9.05 | 58.04 | 19.49 | 13.05 | 5.27 | 8.50 |
| 1024 | 67.11 | 17.32 | 62.92 | 23.56 | 13.01 | 5.01 | 9.31 |
| 2048 | 70.27 | 31.55 | 64.46 | 26.15 | 12.94 | 6.32 | 9.93 |
| 4096 | 69.06 | 35.42 | 59.81 | 27.94 | 12.46 | 7.80 | 10.08 |
| 8192 | 41.21 | 32.72 | 35.96 | 28.67 | 9.34 | 8.99 | 10.22 |
| 16384 | 31.09 | 30.65 | 31.64 | 29.17 | 9.34 | 8.59 | 10.22 |
| 32768 | 32.29 | 32.79 | 32.04 | 29.40 | 9.23 | 9.79 | 10.26 |
| 65536 | 32.69 | 33.85 | 33.58 | 29.55 | 9.29 | 10.01 | 10.21 |
| 98304 | 32.31 | 34.68 | 33.34 | 29.57 | 9.29 | 10.13 | 10.19 |
| 131072 | 32.44 | 34.72 | 33.35 | 29.33 | 9.29 | 10.19 | 10.20 |




**BLAS 2 Benchmark** The matrix vector routine, DGEMV, was benchmarked for this level of BLAS, along with the equivalent FORTRAN-77 code. DGEMV multiplies a vector by a matrix and stores the result in another vector.

**Table 3.1.2a - DGEMV Results for Uninitialised Data**

| Matrix Size | Performance (MFlops) | | | | | | |
|---|---|---|---|---|---|---|---|
| | SP2 | | | T3D | | iPSC/860 | |
| | BLAS | ESSL | FORTRAN | BLAS | FORTRAN | BLAS | FORTRAN |
| 10X 10 | 0.03 | 1.07 | 40.92 | See Note 1 | | 2.64 | 2.73 |
| 16X 16 | 0.07 | 2.27 | 11.13 | ‘‘ | | 5.69 | 7.35 |
| 50X 50 | 0.67 | 12.84 | 20.70 | ‘‘ | | 17.90 | 14.32 |
| 100X100 | 2.53 | 21.41 | 21.05 | ‘‘ | | 25.06 | 20.02 |
| 128X128 | 3.69 | 22.23 | 20.58 | ‘‘ | | 28.72 | 22.07 |
| 500X500 | 14.72 | 20.64 | 18.27 | ‘‘ | | 26.12 | 28.06 |
| 512X512 | 15.63 | 24.01 | 18.61 | ‘‘ | | 26.08 | 28.09 |
| 600X600 | 16.31 | 20.82 | 19.13 | ‘‘ | | 26.22 | 26.93 |
| 700X700 | 17.11 | 20.83 | 19.19 | ‘‘ | | 26.26 | 27.39 |
| 900X900 | 17.88 | 20.91 | 19.25 | ‘‘ | | 26.53 | 27.80 |
| 1024X1024 | 17.74 | 23.63 | 17.42 | ‘‘ | | 26.11 | 28.02 |

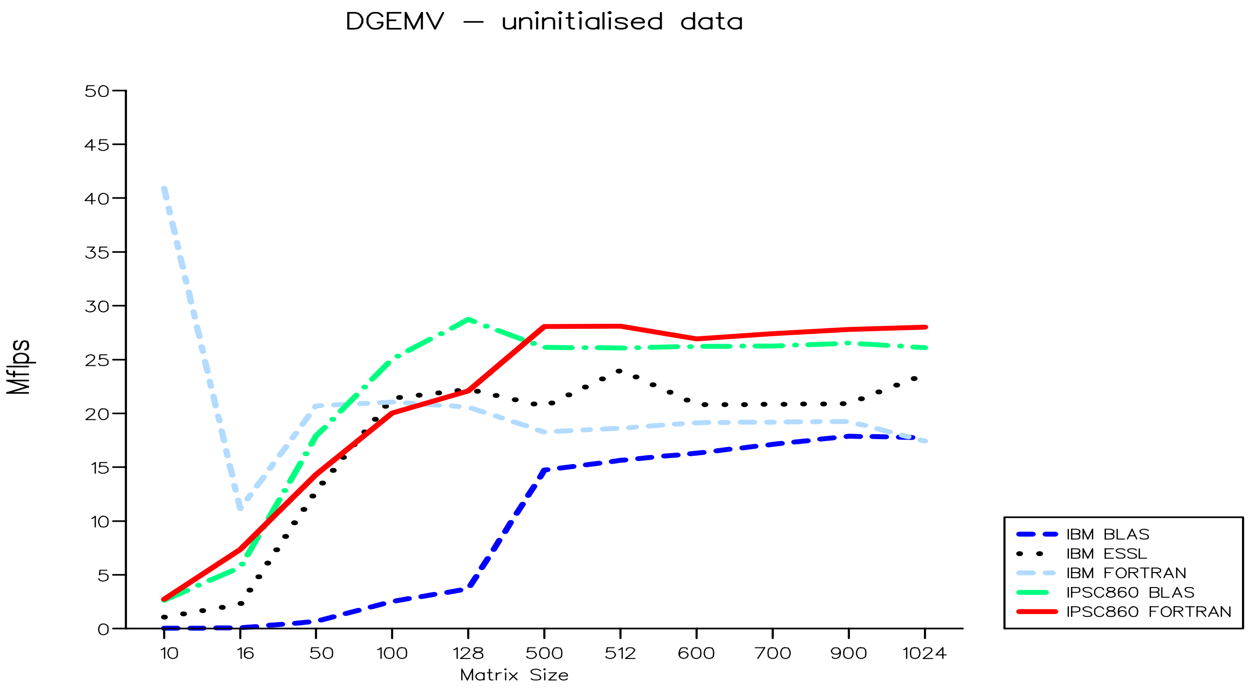
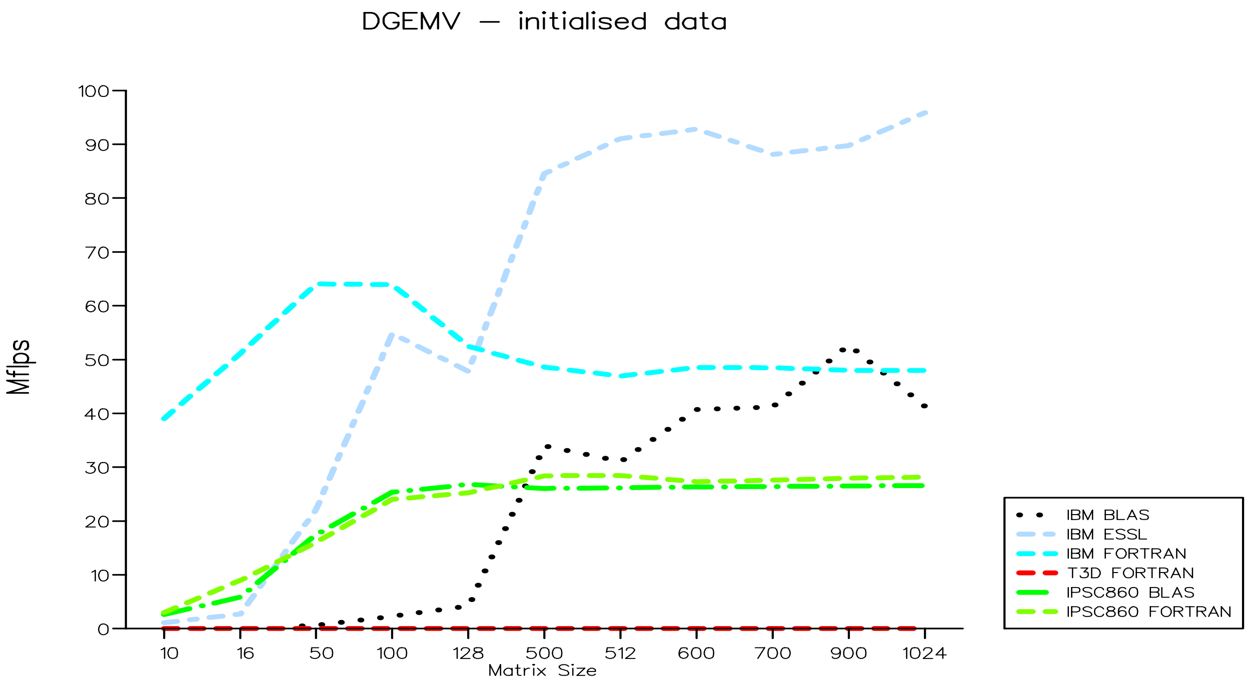



**Table 3.1.2b - DGEMV Results for Initialised Data**

| Matrix Size | Performance (MFlops) | | | | | | |
|---|---|---|---|---|---|---|---|
| | SP2 | | | T3D | | iPSC/860 | |
| | BLAS | ESSL | FORTRAN | BLAS | FORTRAN | BLAS | FORTRAN |
| 10X10 | 0.03 | 1.10 | 39.02 | 6.25 | | 2.62 | 2.98 |
| 16X16 | 0.07 | 2.69 | 51.13 | 10.79 | | 5.85 | 8.94 |
| 50X50 | 0.61 | 22.15 | 64.04 | 31.77 | | 17.47 | 16.03 |
| 100X100 | 2.30 | 54.83 | 63.94 | 44.06 | | 25.34 | 24.01 |
| 128X128 | 4.20 | 47.80 | 52.46 | 48.72 | | 26.79 | 25.23 |
| 500X500 | 33.95 | 84.58 | 48.58 | 48.07 | | 26.05 | 28.40 |
| 512X512 | 31.08 | 91.06 | 46.93 | 54.96 | | 26.16 | 28.47 |
| 600X600 | 40.69 | 92.83 | 48.53 | 47.61 | | 26.32 | 27.31 |
| 700X700 | 41.21 | 88.12 | 48.48 | 47.57 | | 26.39 | 27.57 |
| 900X900 | 52.69 | 89.77 | 47.99 | 47.58 | | 26.50 | 27.96 |
| 1024X1024 | 41.27 | 95.87 | 47.97 | 54.72 | | 26.60 | 28.16 |




**BLAS 3 Benchmark** DGEMM, which computes the product of two matrices, was used for this benchmark along with the equivalent FORTRAN-77 code.

Tables 3.1.3a show the results for uninitialised data and 3.1.3b for initialised data.

**Table 3.1.3a - DGEMM Uninitialised Data**

| Matrix Size | Performance (MFlops) | | | | | | |
|---|---|---|---|---|---|---|---|
| | SP2 | | | T3D | | iPSC/860 | |
| | BLAS | ESSL | FORTRAN | BLAS | FORTRAN | BLAS | FORTRAN |
| 10X10 | 0.20 | 5.86 | 106.86 | See Note 1 | | 7.61 | 3.18 |
| 20X20 | 1.51 | 42.15 | 100.46 | ‘‘ | | 15.48 | 12.15 |
| 50X50 | 20.04 | 142.78 | 144.68 | ‘‘ | | 29.32 | 16.12 |
| 64X64 | 36.55 | 170.21 | 135.35 | ‘‘ | | 32.54 | 21.34 |
| 100X100 | 89.69 | 194.41 | 135.45 | ‘‘ | | 34.48 | 24.03 |
| 128X128 | 112.61 | 179.80 | 126.58 | ‘‘ | | 34.40 | 25.12 |
| 200X200 | 135.89 | 207.27 | 146.35 | ‘‘ | | 35.84 | 26.64 |
| 256X256 | 143.80 | 210.13 | 21.92 | ‘‘ | | 35.82 | 27.19 |
| 500X500 | 155.66 | 226.30 | 155.87 | ‘‘ | | 36.83 | 28.47 |
| 512X512 | 154.55 | 227.99 | 21.82 | ‘‘ | | 36.52 | 28.53 |
| 700X700 | 158.99 | 233.04 | 155.59 | ‘‘ | | 36.68 | 27.51 |

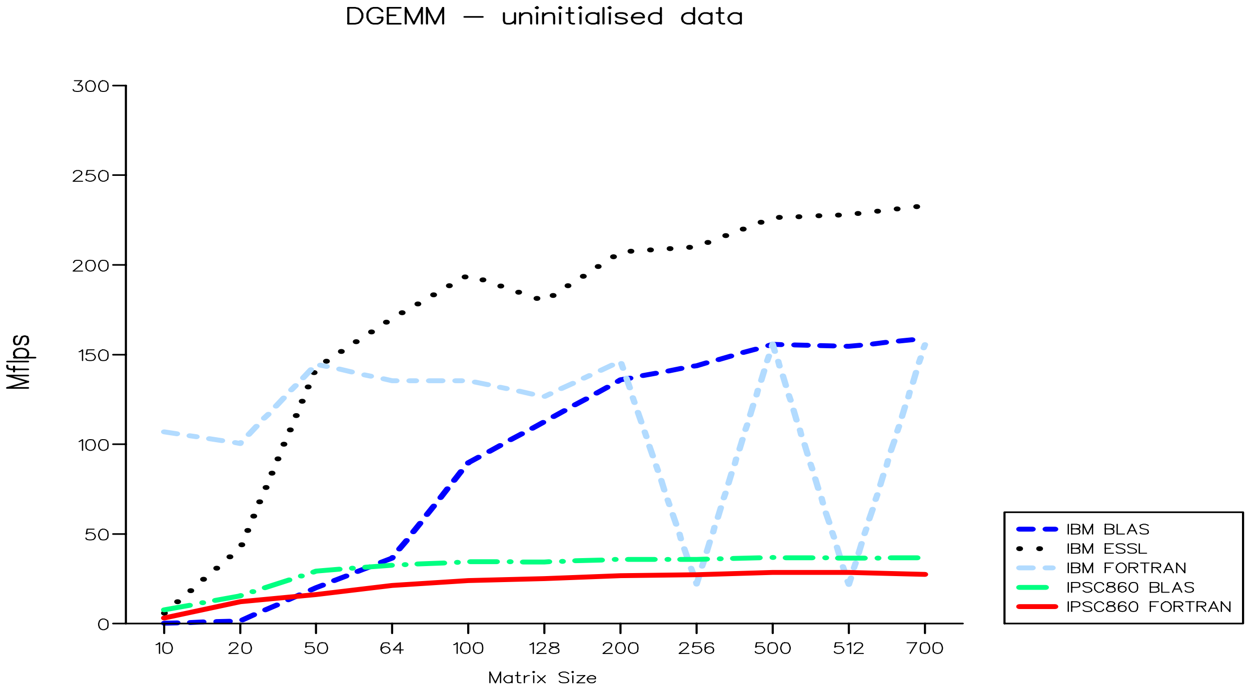
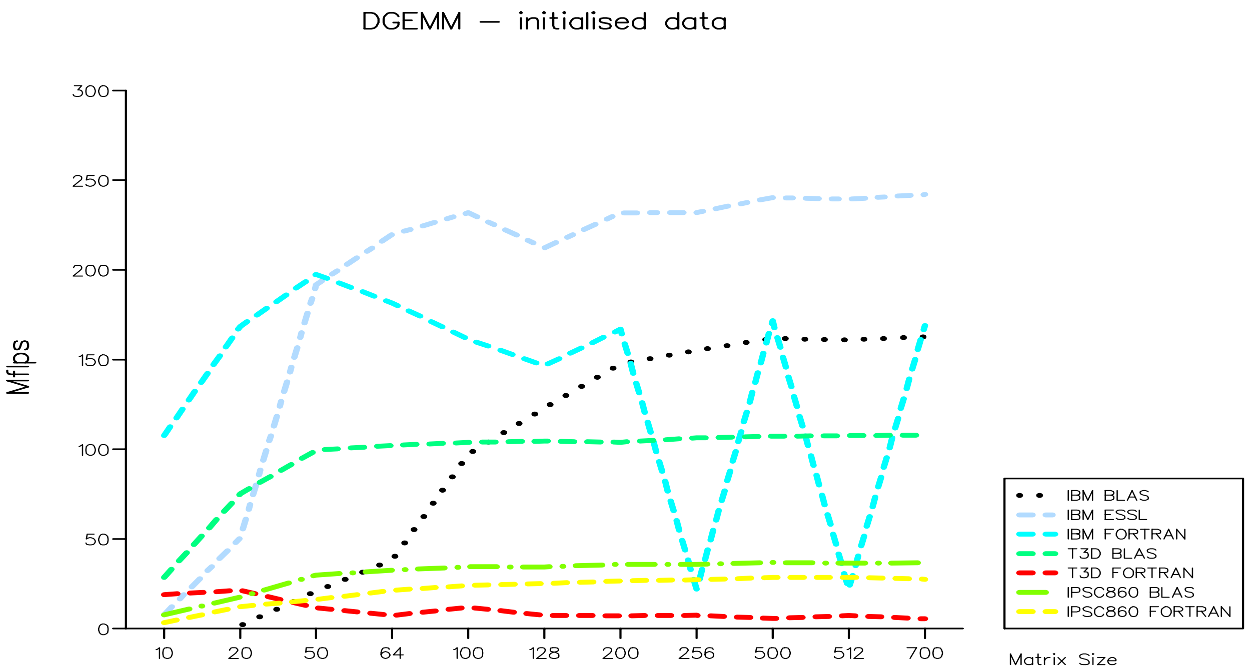



**Table 3.1.3b - DGEMM Initialised Data**

| Matrix Size | Performance (MFlops) | | | | | | |
|---|---|---|---|---|---|---|---|
| | SP2 | | | T3D | | iPSC/860 | |
| | BLAS | ESSL | FORTRAN | BLAS | FORTRAN | BLAS | FORTRAN |
| 10X10 | 0.20 | 7.66 | 107.55 | 28.53 | 18.97 | 7.76 | 3.24 |
| 20X20 | 1.57 | 50.53 | 168.40 | 75.09 | 21.36 | 17.58 | 12.21 |
| 50X50 | 20.97 | 191.71 | 197.51 | 99.45 | 11.66 | 29.89 | 16.17 |
| 64X64 | 38.40 | 219.67 | 181.50 | 102.17 | 7.30 | 32.56 | 21.31 |
| 100X100 | 97.32 | 231.89 | 161.35 | 103.79 | 11.91 | 34.47 | 24.05 |
| 128X128 | 122.98 | 212.29 | 146.53 | 104.51 | 7.36 | 34.39 | 25.10 |
| 200X200 | 147.59 | 231.77 | 166.93 | 103.91 | 7.15 | 35.82 | 26.63 |
| 256X256 | 155.16 | 231.98 | 21.98 | 106.32 | 7.40 | 35.83 | 27.20 |
| 500X500 | 161.76 | 240.33 | 171.64 | 107.26 | 5.67 | 36.85 | 28.46 |
| 512X512 | 160.92 | 239.39 | 21.91 | 107.55 | 7.27 | 36.50 | 28.55 |
| 700X700 | 162.72 | 242.02 | 168.89 | 107.91 | 5.48 | 36.67 | 27.51 |




**Notes**

Note 1 - Cray T3D will not run code for uninitialised data

Note 2 - For all tests double precision and a unit stride to access the vectors was used

Note 3 - V1.2 of the CLASSPACK Basic Math Library and V4.0 of the Portland compiler were used on the Intel iPSC/860.

Note 4 - V?? of the BLAS library and version cf77_6.1 of the Fortran compiler were used on the T3D.

Note 5 - V2.0 of the BLAS library and V3.2 of the Fortran compiler were used on the IBM SP2.

Note 6 - The Engineering Scientific Subroutine Library (ESSL), used on the SP2, was the serial version. The benchmarks were done using the library tuned for the POWER2 processor (esslp2).

### 8.3.2 Various MXMB routines

*N.M. Harrison and R.J. Allan*

**Fortran**

Four test codes are available which are optimised for different systems.

- mxmb.f : Simplest possible implementation (how good is your compiler)

- mxmb_risc.f : Outer loop unrolled for cached RISC - eg: RS6000
- mxmb_vect.f : vector processors - inner loop unrolled
- ibm_mxm/ : IBM's optimised code for R6000

### Assembler

Hand written assembler code are also implemented on some systems.

- mxmb_ymp.s : CRAY YMP
- mxmb_c90.s : CRAY C90
- mxmb_rs6000.s : IBM RS6000
- mxmbn_rs6000.s mxmb_370.s : IBM mainframe 370

### Libraries

mxmb_blas.f : implemented on standard BLAS library

### Results

Only results from the mxmb_risc and mxmb_vect tests are presented here in depth. We however summarise at the end by comparing the results of mxmb_vect mxmb_risc mxmb_blas and mxmb_assem (hand-coded assembler) on the SP2 wide node.

Table 8.1: Matrix Multiplication: mxmb_vect

| dim | 530h | AXP400 | SGIchal/L | HP750 | HP755 | IBM360 | IBM370 | SP2 wide |
|---|---|---|---|---|---|---|---|---|
| 10 | 10.8 | 32.8 | 12.5 | 16.1 | 27.2 | 16.2 | 20.3 | 23.8 |
| 20 | 16.9 | 56.0 | 15.0 | 24.1 | 40.3 | 25.4 | 31.7 | 40.0 |
| 30 | 19.4 | 48.1 | 15.3 | 27.3 | 45.0 | 29.1 | 36.4 | 52.6 |
| 40 | 21.6 | 35.8 | 15.9 | 29.4 | 49.5 | 32.4 | 40.5 | 62.4 |
| 50 | 22.4 | 28.1 | 14.4 | 29.6 | 50.9 | 33.6 | 42.0 | 66.7 |
| 60 | 23.5 | 31.6 | 12.6 | 30.7 | 52.8 | 34.5 | 44.1 | 76.4 |
| 70 | 23.9 | 28.5 | 12.3 | 30.8 | 53.1 | 35.9 | 44.8 | 80.0 |
| 80 | 24.6 | 31.8 | 12.4 | 31.8 | 54.1 | 36.9 | 46.1 | 78.8 |
| 90 | 24.1 | 28.6 | 12.4 | 33.0 | 54.1 | 36.2 | 45.2 | 81.0 |
| 100 | 20.6 | 32.2 | 12.5 | 33.4 | 54.6 | 30.9 | 38.6 | 83.3 |

The above results were generated on the SP2 wide node using compiler option `xlf -O`. It was noted that changing the option to `xlf -O3 -qhot -qarch=pwr2` resulted in poorer performance for the low dimensions but increase in performance for the larger dimensions in the mxmb_vect test.

The libesslp2.a library was used in the final test. Clearly it is working well as reported in the chapter on BLAS.

Table 8.2: Matrix Multiplication: mxmb_risc

| dim | 530h | AXP400 | SGIchal/L | HP750 | HP755 | IBM360 | IBM370 | SGI/XL | SP2 wide |
|---|---|---|---|---|---|---|---|---|---|
| 10 | 19.2 | 39.2 | 16.9 | 18.9 | 39.1 | 28.8 | 36.0 | 25.4 | 50.0 |
| 20 | 28.3 | 44.7 | 19.2 | 26.6 | 57.6 | 42.5 | 53.1 | 28.8 | 76.9 |
| 30 | 33.0 | 41.5 | 19.6 | 31.4 | 66.0 | 49.5 | 61.9 | 29.5 | 90.8 |
| 40 | 35.7 | 29.3 | 14.3 | 33.4 | 71.8 | 53.6 | 66.9 | 29.8 | 99.8 |
| 50 | 37.3 | 26.8 | 14.5 | 36.1 | 74.9 | 56.0 | 69.9 | 26.6 | 100.0 |
| 60 | 38.5 | 26.6 | 13.4 | 36.6 | 77.1 | 57.8 | 72.2 | 22.7 | 110.4 |
| 70 | 39.4 | 26.3 | 13.4 | 37.8 | 78.9 | 59.1 | 73.9 | 22.1 | 120.1 |
| 80 | 39.7 | 25.3 | 13.7 | 38.0 | 79.1 | 59.6 | 74.4 | 22.0 | 113.8 |
| 90 | 38.7 | 24.4 | 14.0 | 38.7 | 80.5 | 58.1 | 72.6 | 22.2 | 121.5 |
| 100 | 29.9 | 25.9 | 14.1 | 38.5 | 80.5 | 44.9 | 56.1 | 22.2 | 117.6 |

Table 8.3: Comparison of MXMB Code Variants on SP2 Wide Node

| dim | mxmb_risc | mxmb_vect | mxmb_assem | mxmb_vect |
|---|---|---|---|---|
| 10 | 50.0 | 23.8 | 111.1 | 90.9 |
| 20 | 76.9 | 40.0 | 125.0 | 200.0 |
| 30 | 90.8 | 52.6 | 124.9 | 199.8 |
| 40 | 99.8 | 62.4 | 124.8 | 249.6 |
| 50 | 100.0 | 66.7 | 125.0 | 250.0 |
| 60 | 110.4 | 76.4 | 141.9 | 198.7 |
| 70 | 120.1 | 80.0 | 120.1 | 192.1 |
| 80 | 113.8 | 78.8 | 128.0 | 256.0 |
| 90 | 121.5 | 81.0 | 132.5 | 243.0 |
| 100 | 117.6 | 83.3 | 133.3 | 222.2 |