Theor Chim Acta (1993) 84: 423–441

Theoretica Chimica Acta

Parallelism in computational chemistry

I. Hypercube-connected multicomputers

M. F. Guest[1], P. Sherwood[1], and J. H. van Lenthe[2]

[1] S.E.R.C. Daresbury Laboratory, Daresbury, Warrington WA4 4AD, UK

[2] State University of Utrecht, Padualaan 14, Utrecht-De Uithof, The Netherlands

Received October 1, 1991/Accepted February 18, 1992

Summary. An account is given of experience gained in implementing computational chemistry application software, including quantum chemistry and macromolecular refinement codes, on distributed memory parallel processors. In quantum chemistry we consider the coarse-grained implementation of Gaussian integral and derivative integral evaluation, the direct-SCF computation of an uncorrelated wavefunction, the 4-index transformation of two-electron integrals and the direct-CI calculation of correlated wavefunctions. In the refinement of macromolecular conformations, we describe domain decomposition techniques used in implementing general purpose molecular mechanics, molecular dynamics and free energy perturbation calculations. Attention is focused on performance figures obtained on the Intel iPSC/2 and iPSC/860 hypercubes, which are compared with those obtained on a Cray Y-MP/464 and Convex C-220 minisupercomputer. From this data we deduce the cost effectiveness of parallel processors in the field of computational chemistry.

Key words: Computational chemistry application software – Hypercube-connected multicomputers

1 Introduction

Parallel computers have the potential to perform numerical calculations much more cost-effectively than conventional supercomputers. The reduced cost of computational power arising from the introduction of such machines opens the door to a far more widespread and extensive adoption of numerically intensive methods, given that the recognised programming investment required is forthcoming. In this paper we consider the impact specifically of hypercube-connected multicomputers on computational chemistry, describing our own work in migrating code to the Intel iPSC/2 and iPSC/860 hypercubes. Practical algorithms are described which address the two major problems in parallel computing: the need both to achieve load balancing, with an even distribution of work across the network of processors, and to minimise delays associated with migrating data between processors.

We present in Sect. 2 a brief overview of the second generation systems of hypercube architecture from Intel Scientific Computers. In Sects. 3 and 4 we consider the techniques employed in implementing a variety of computational chemistry applications on the machine, describing both quantum chemistry (Sect. 3) and macromolecular energy refinement (Sect. 4). In presenting performance figures in both areas, we show timings achieved in benchmarking the GAMESS [1] molecular electronic structure code and AMBER [2] molecular refinement code.

The iPSC/2 and iPSC/860 used in this study are located at the SERC Daresbury Laboratory. The timings on the Cray Y-MP/464 were obtained at the SARA Supercomputer Centre, Amsterdam under a grant from the Stichting Nationale Computer Faciliteiten.

2 The iPSC/2 and iPSC/860 hypercubes

Both concurrent processors at Daresbury comprise 32 multiple instruction multiple data (MIMD) nodes placed on a hypercube with direct-connect channels. Each node of the iPSC/2 consists of an Intel 80386 processor (4 Mips in 32-bit arithmetic), with 4 Mbyte of memory. Enhanced floating point performance is achieved through a Weitek 1167 SX scalar accelerator (0.6 Mflop 64-bit or 1 Mflop in 32-bit arithmetic). Vector processing on the nodes is available through a VX-vector accelerator (6.6 Mflop in 64-bit or 20 Mflop 32-bit). The vector board has 1 Mbyte associated memory which can be addressed, making a total of 5 Mbyte node memory. This dual memory can, however, often lead to serious restrictions in vector processing on the cube. Migrating extrinsically vectorised code is often difficult given the 1 Mbyte restriction, while the need to continually move data between the SX and VX memory impacts sharply on performance.

This memory constraint is significantly reduced on the iPSC/860 (or so-called GAMMA machine), where each node has a minimum of 8 Mbytes of memory, with 4 nodes of the Daresbury machine having 16 Mbytes. Each processor node now contains an Intel i860 microprocessor chip [3] which is clocked at 40 MHz and has an effective peak rating of 40 Mflop for the 64-bit arithmetic characteristic of scientific and engineering computation. Developments of the i860-multicomputer are at the heart of the DARPA-funded 'Touchstone Project', which includes within its set of milestones the development of a prototype 2000-node machine and 200 Mbytes/sec communications capabilities.

At present, however, inter-node communications on both the iPSC/860 and iPSC/2 are handled through the same DMA (Direct Memory Access) node-controller. This provides a maximum 10.7 Mbyte/sec asynchronous communications network on eight bidirectional channels, each node having 7 channels for the network plus one used for I/O. Messages can be switched over a route specified by their header information without cpu intervention. In contrast to the first-generation iPSC/1 a cube may, from an application viewpoint, be considered as an ensemble of fully connected nodes, since there is now little more delay in long-distance messages than in neighbour-neighbour transfers. An obvious limitation, however, is apparent when considering the iPSC/860; as we shall see, while the cpu capability of each node has increased by at least an order of magnitude, the effective communication bandwidth is the same as its predecessor. Clearly we may expect only the most coarse grained of application codes to be capable of fully exploiting the iPSC/860.

Each cube is controlled by its own front-end processor, the system resource manager (SRM), which is an Intel System 310 microcomputer with 8 Mbyte memory. The SRM is linked directly to the cube through direct connect hardware, with access to other UNIX workstations provided over Ethernet and TCP/IP networks. With a UNIX System V operating system, the SRM compiles, links, debugs, allocates and communicates with the hypercube. In order to reduce loading on the SRM, Intel now provide cross-compilers, and 'remote hosting' software which enables users to run programs from other UNIX machines. Remote hosting is at present possible at Daresbury from either a Sun workstation or the Convex C-220. This capability of dividing the workload between a UNIX engine and the multicomputer provides a valuable resource given hosting by a powerful UNIX processor whose memory and CPU capabilities far exceed that of any single node of the multicomputer. It enables a balanced solution whereby the well defined highly parallelised compute-intensive parts of the application are allocated to the cube, while any significant serial or memory demanding component, together with pre- and post-analysis associated with the application, are performed on the host. This dual approach has been used to advantage in various application areas when linking the iPSC/2 with the Convex C-220. The corresponding iPSC/860–Convex C-220 combination is less attractive, given the increased memory and CPU performance of the i860 nodes.

Each iPSC node carries its own multi-tasking operating system, called NX/2 (Node eXecutive/2). An effective subset of UNIX, NX/2 provides multiple tasks with private address space, dynamic memory allocation, UNIX I/O, round robin scheduling of processes, and supports the DECON parallel debugger. The UNIX common object file format is supported and routines may be called from Fortran. NX/2 provides the program's interface to the Direct-Connect communication network, with support provided for the sending and receiving of both synchronous and asynchronous messages.

In addition to the hypercube-connected set of compute nodes, both iPSC/2 and iPSC/860 address the requirements of high capacity, fast access mass storage through the Concurrent I/O System (CIO), an I/O subsystem that features an attached set of I/O nodes. Each I/O node has full access to the hypercube interconnect of the computational nodes, has 4 Mbytes or more of memory and a SCSI bus which connects to one or more high capacity drives. The I/O subsystem at Daresbury (connected to the iPSC/860) comprises four I/O nodes at present, each with two 574 Mbytes disks, permitting parallel random access to 4 Gbytes of data from any of the hypercube nodes.

The software environment surrounding the hypercube, commonly referred to as the 'Concurrent Workbench', provides a range of facilities. On the iPSC/2 compilers for Fortran-77, C, and concurrent Lisp are available, with VMS and Berkeley UNIX 4.2 Fortran extensions. Vector processing tools include the VAST2 vectorising pre-processor. VAST2 accepts Fortran-77 and -8X constructs and generates VecLib calls. It vectorises DO and IF loops, provides help and warnings, optimises performance, and provides a high-level interface to VecLib, a library of micro-coded vector routines, comprising the majority of the level-1 BLAS [4]. Fortran-77 and C compilers are available on the iPSC/860 (from The Portland Group), with optimised BLAS routines provided in the library from Kuck and Associates.

The Concurrent File System (CFS) software consists of specialised processes which run on the I/O nodes, and file management code which runs in conjunction with the application on compute nodes. The UNIX programming interface

provides a single file system view, beneath which CFS manages parallelism and concurrent access to files. The standard for the I/O calls is the same UNIX System V running on the SRM, with support provided for both synchronous and asynchronous Fortran I/O. Finally, a parallel symbolic debugger, DECON, is provided for code development on the hypercube. DECON is designed for message-passing analysis and repair; it can inspect and modify variables by name, set break and trace points, set conditional break and trace points, single-step on statements and subroutines, list source code, create aliases and macros, and provides help facilities.

Software is hosted by the SRM and remotely accessed from any UNIX workstation. Note that the iPSC/2 was installed at Daresbury in October 1988, and the iPSC/860 in July 1990.

3 Quantum chemistry software

The algorithms employed in quantum chemistry are, of course, computationally intensive and increased performance, through both algorithmic developments and the exploitation of new computer architectures, is continually being pursued to improve both accuracy and the size of molecular system amenable to treatment [5]. Of particular importance is the ability to locate minimum energy structures and transition states, allowing the study of reaction mechanisms. While expensive, such calculations are now routine following the development of efficient techniques for evaluating the gradient of the energy [6].

The molecular quantum chemistry program, GAMESS, which we shall use and discuss, approximates solutions to the Schrödinger equation within both the Hartree–Fock and post Hartree–Fock frameworks. On-going development of the UK version of the code (GAMESS-UK) is carried out at Daresbury, with the program currently available on a wide range of supercomputers, superminis and superworkstations. The program provides for *s*, *p*, *d* and *f*-type cartesian Gaussian orbitals, with open- and closed-shell SCF treatments available within both RHF and UHF frameworks. These treatments are augmented by generalised valence bond (GVB), complete active space SCF (CASSCF), and more general MCSCF calculations. *Ab initio* core potentials are provided in both semi-local and non-local formalism for valence-only molecular orbital treatments. The analytic energy gradient is available for each class of wavefunction above. Geometry optimisation may be performed in either internal or cartesian space, using a quasi-Newton rank-2 update method, while transition state location is available through either a synchronous transit, trust region or 'hill-walking' method. Force constants may be evaluated by either analytic methods (see below) or by numerical differentiation. Configuration interaction estimates of the correlation energy may be generated through conventional-CI (using table-driven selection algorithms), Direct-CI and Full-CI treatments.

Considerable effort has been targeted to increasing both the range of computational methods available and the size of molecular system amenable to treatment. Recent additions to the code include both coupled Hartree–Fock calculations of molecular polarizabilities and perturbations due to nuclear displacements, allowing for the analytic computation of all the dipole and quadrupole moment derivatives of a molecule. Analytic second derivatives (force constants) of the energy, analytic calculations of polarizability derivatives and the calculation of infrared and Raman intensities are now possible. Møller–Ples-

set MP2 and MP3 perturbation theory calculations (with MP4 capabilities under development), include the analytic calculation of gradients, polarizabilities, dipole moment derivatives and force constants at the MP2 level. In addition to the Green's function OVGF and TDA methods for calculating ionisation spectra, an RPA (Random Phase Approximation) module has been implemented primarily for use in estimates of molecular excitation spectra.

A range of techniques for treating large molecules is also provided. In addition to effectively open-ended Direct-SCF capabilities (enabling calculations up to 1000 basis functions), GAMESS now includes a direct-MP2 module for correlation estimates of large molecules. The computation of accurate electronic wavefunctions for large molecules is assisted through hybrid molecular orbital/molecular mechanics and molecular dynamics methods. This involves coupling *ab initio* quantum mechanical (QM) calculations with molecular mechanical (MM) and dynamical (MD) calculations within GAMESS. The areas of application include molecular systems whose size precludes *ab initio* treatment, treating systems in which chemical bonds are broken or formed, e.g. transition states, and handling electronically excited states. As well as incorporating molecular mechanics as a vehicle for handling large molecules, a wide range of semiempirical features will shortly be made available.

In addition to the functionality outlined above, a variety of graphical analysis tools (targeted to superworkstation usage) and an interface into front-end model building capability is provided by the DISPLAY package developed at Daresbury.

3.1 Sparse matrix multiply benchmarks

Before considering the hypercube implementation of GAMESS, we present some statistics based on the sparse matrix multiply operation (MMO) to provide some idea of iPSC/2 and iPSC/860 node performance. The MMO operation is central to the efficient operation of modern quantum chemistry codes on vector processors [7], it being possible both to extract near peak performance for this kernel and to formulate many QC steps around this operation. A comparison of the single processor sparse MMO performance on a variety of machines, including Supercomputers (Cray X-MP/416, IBM 3090-600E/VF and Convex C-3840 (18 nsec clock)), Superminis (Convex C-220, FPS M64/60 and Alliant FX2808), workstations (IBM RS6000, Hewlett Packard 9000/7xx series and Apollo DN10020, DEC Station 5000/200, Silicon Graphics INDIGO and 4D/220, 4D/320 and 4D/420, Stardent 1520 and 3020, Solbourne 4000 and Sun SPARC-Station 2/GS and 4/370) and Novel Architecture machines (Intel iPSC/2 (SX and VX node) and iPSC/860, and both T800-20 transputer and i860-based Meiko Computing Surfaces), is given in Table 1. In this benchmark a series of MMOs ($R = A \times B$) involving matrices of rank 10, 20, 30, ..., 100 were performed. Each MMO was performed a number of times, this number being inversely proportional to the rank, so that the summed CPU times of Table 1 refer to 100 MMOs of rank 10 matrices, 90 of rank 20 matrices, and so on up to 10 MMOs for matrices of rank 100. Figures are presented for both 'full' (0% sparse) and 50%-sparse B matrices, with the performance figures referring to code written entirely in Fortran.

A striking feature of the benchmark is that a single SX node of the iPSC/2, while performing at approximately the same speed as the T800 20 MHz Trans-

Table 1. Sparse MMO benchmark. Total CPU times (sec) for a series of sparse MMOs ($R = AB$, see text) implemented in Fortran

Machine	Sparsity in B-matrix 0%	50%
VAX 8350	1005.6	530.4
T800-20	448.6	231.2
iPSC/2 SX-node	445.9	230.3
iPSC/2 VX-node	105.0	105.0
Meiko MK086 i860	35.7	18.4
iPSC/860 RX-node	21.2	11.0
Solbourne S4000	74.4	38.4
SUN 4/370	57.4	30.6
HP/Apollo DN10020	44.2	22.8
Silicon Graphics 4D/220	39.4	20.1
DEC S5000/200	33.1	17.1
Silicon Graphics INDIGO	31.0	14.3
Silicon Graphics 4D/320	27.9	14.4
SUN SPARCstation 2/GS	27.5	14.4
Stardent 1520	26.6	17.0
Silicon Graphics 4D/420	23.7	12.1
Stardent 3020	14.7	9.1
IBM6000 Model 320	14.6	7.1
IBM6000 Model 530	9.3	4.9
Hewlett Packard 9000/720	8.5	4.8
Hewlett Packard 9000/730	6.7	3.5
Alliant FX2808 (1CE)	17.2	9.3
FPS-M64/60	17.1	8.9
Convex C-220	10.6	6.5
IBM 3090-600-E/VF	11.3	6.0
Convex C-3840	5.2	3.3
Cray X-MP/416	2.8	1.8

puter, is nevertheless some 20 times slower than the RX-node of the iPSC/860. The i860 performance is double that of the Apollo DN10000 and Silicon Graphics 4D/220, and approximately 50% of that on the IBM RS6000 Series 530 and HP 720. Note that the IBM RS6000 Series 320 is the same speed as the Stardent 3020, twice the speed of the SGI 4D/320 and DEC 5000, and between 4–5 times the speed of the SUN and Solbourne machines.

One additional feature of the MMO benchmark not apparent in Table 1 is the significantly enhanced performance found on *all* machines when comparing assembly language MMO to Fortran MMO. Improvement factors of 3.2 (Cray X-MP), 4.2 (IBM 3090/VF), 3.1 (Convex C-220) and 3.4 (FPS-M64/60) with assembly language implementations point to the crucial role of optimised library routines. A corresponding factor of 3.8 is found when using the Kuck Library on the i860. It is certainly the case that a good deal of the i860 potential remains undelivered in present releases of the Fortran compiler, which provide a ceiling of 4–6 Mflop in scalar code and little more in vector. The 20–30 Mflop achievable in hand-coded level-3 BLAS seems possible at present only through

Table 2. Sparse MMO benchmark. Total CPU times (sec) for a series of similarity transformations ($H = Q^{\dagger}HQ$, see text) using both scalar and vector algorithms

Machine	Algorithm	
	Scalar	Vector
VAX 8350	7285.8	9834.8
T800-20	5176.0	4439.0
iPSC/2 SX-node	3809.6	4456.1
Meiko MK086 node	240.1	262.2
iPSC/860 RX-node	118.8	60.1
SOLBOURNE S4000	711.9	750.2
SUN 4/370	615.9	609.6
Dec S5000/200	368.3	445.1
Silicon Graphics 4D/220	344.8	440.3
SUN SPARCstation 2/GS	333.3	394.0
Silicon Graphics INDIGO	285.0	356.4
Silicon Graphics 4D/320	245.5	329.5
HP/Apollo DN10020	273.8	245.9
Stardent 1520	236.9	252.8
Silicon Graphics 4D/420	200.1	273.8
Stardent 3020	160.4	144.2
IBM6000 Model 320	139.0	73.3
Hewlett Packard 9000/720	111.2	60.4
IBM6000 Model 530	96.3	46.9
Hewlett Packard 9000/730	89.9	48.4
IBM6000 Model 540	66.9	38.4
Alliant FX2800 (1CE)	243.1	86.3
Alliant FX2800 (3CE)		39.5
FPS-M64/60	141.2	50.8
CONVEX C-220	124.1	60.1
CONVEX C-3840	55.1	25.4

library specification. The availability and degree of functionality of library software are, we believe, important issues when considering cost-effective performance. This effect is evident from the second benchmark which, given in Table 2, involves performing a series of similarity transforms ($Q^{\dagger}HQ$) using both a scalar and vector algorithm. In the latter case we utilise the BLAS library routine DGEMM (where available) for performing the requisite MMOs, in the former case the dot product BLAS routine, DDOT. The i860 is seen to exhibit impressive performance, running the same speed as the Convex C210 and Hewlett Packard HP 720, and significantly faster than the IBM Model 320. The latter is now twice the Stardent 3020, between 3.5–6.5 times the SGI 4D/320, DEC 5000 and SPARCStation 2 and 9–10 times the SUN 4/370 and Solbourne.

3.2 Direct-SCF calculations

The conventional quantum mechanical SCF procedure is carried out in two steps, the first involving the evaluation of integrals over basis functions. The

most time consuming of these integrals has the form:

$$(ij/kl) = \int\int \phi_i(1)\phi_j(1)1/r_{12}\phi_k(2)\phi_l(2)\,\partial\tau_1\,\partial\tau_2 \tag{1}$$

For a system with n basis functions there are $n^4/8$ of these integrals, leading to a computational complexity for this step of $O(n^4)$. The second step is the calculation of the SCF energy and molecular orbitals. This iterative procedure requires the repeated $O(n^4)$ construction and $O(n^3)$ diagonalisation of a Hamiltonian matrix from the pre-calculated integral list. The Hamiltonian, or so-called Fock, matrix has the form:

$$F_{ij} = H_{ij} + \sum_k \sum_l P_{kl}[(ij/kl) - 1/2(ik/jl)] \tag{2}$$

where H is a one-electron matrix and P the density matrix formed from the SCF vector. In the direct-SCF procedure [8] the integral list is recalculated every SCF iteration, removing the need to store the two-electron integrals. While the calculation of the integrals and their subsequent incorporation in the Fock matrix may be expected to dominate the computation, given their $O(n^4)$ dependence, there are several $O(n^3)$ steps occurring in the SCF procedure. These include matrix multiplication (associated with both the back-transformation of the eigenvectors and the implementation of the DIIS procedure [9] for acceleration of convergence), matrix orthogonalisation and the diagonalisation of the Fock matrix.

The size of system amenable to *ab initio* treatment has increased significantly with the development of direct-SCF algorithms, given the removal of the historical bottleneck of disk storage associated with the $O(n^4)$-integral lists. Furthermore the direct-SCF/gradient technique is well suited to parallel processing since the evaluation of the energy- and derivative-integrals may be readily partitioned over the component nodes of a multicomputer, with minimal communication overheads. Our initial programming strategy for the iPSC/2 thus focused on the direct-SCF gradient capabilities of GAMESS, aiming to provide a parallelised code (80000 lines of Fortran) capable of performing RHF/UHF/GVB calculations for geometry optimisations and transition state characterisations. The implementation involved parallelising both the one- and two-electron integral plus derivative integral sections of the code (for both the rotated axis [10] and Gauss–Rys quadrature [11] methods) within the framework of the direct-SCF method. Standard features of the program, including convergence aids (DIIS), saddle point algorithms and pseudopotential capabilities (with gradients) were retained in the parallel code.

The rotated axis integral code due to Pople and co-workers [10] is now over a decade old and, although completely scalar code, still competes with recent vectorised integral programs for integrals over contracted *sp* shells. In order to minimise overall memory requirements, we adopted the scalar version of the code for the parallel implementation. Note that while the code has been vectorised in an extrinsic fashion to yield speed increases of anything from two to eight [12], the memory required prohibited adopting this version on the iPSC/2. In essence each node runs the same integral code, looping over all symmetry distinct integrals, with the work divided using a straightforward partitioning of the workload through:

$$MOD(\textbf{\textit{ijkl, nodes}}).\ eq.\ \textbf{\textit{mynode}} \tag{3}$$

```
IJKL=0
DO I,J,K,L  ! 4-fold loop over unique shells
   IJKL=IJKL+1
   IF ( MOD (IJKL,NODES). EQ. MYNODE ) THEN
   ! Compute and store shell information
   DO II=1,NGI  ! Loop over primitives in shell I
      DO JJ=1,NGJ  ! and for shell J
         DO KKLL=1,NGK*NGL  ! Loop over K and L primitives
            ! Compute 70 components in scalar fashion
         END DO
         ! Form 256 integral contributions
      END DO
   END DO
   ! Rotate 256 integrals into molecular frame
   ! Add contributions to partial-Fock Matrix, HNODE
   END IF
END DO
! Generate total Fock-Matrix
! form node contributions, HNODE
CALL GDSUM (HNODE, LENGTH, SCRATCH)
```

Fig. 1. Parallelised direct-SCF structure

where ***ijkl*** is a running index over the two-electron integrals, ***nodes*** the total number of nodes available, and ***mynode*** an integer labelling the particular node in question. Thus an integral is evaluated if the integral index modula the total number of nodes is equal to the node identification number. The parallelised node structure of the direct-SCF code is sketched in Fig. 1, using as an example the integral routine SP1111 which computes integrals of the form $(sp\ sp \mid sp\ sp)$, though what is shown is directly applicable to the other routines. Note that exactly the same method for partitioning the workload can be applied to the calculation of the derivative integrals.

Each node thus calculates a unique sub-set of integrals, adding the integral contributions to its own copy of the Fock matrix. The resulting set of n partial Fock matrices are then combined, with the resulting total Fock matrix distributed to all the nodes using the global communication utility (GDSUM) available on the cube. Orthogonalisation and diagonalisation of the total Fock-matrix are at present performed in serial fashion on each node and are being parallelised [13] to improve overall performance.

One feature of the practical details of the work is worthy of mention here. While the direct-SCF method removes the major I/O component associated with the two-electron integral lists, there still remains a disk requirement for handling restart information and the storage of various scratch arrays. Prior to installation of the CFS, this requirement was handled by using the memory of the Host System Resource Manager (SRM) as a virtual disk for the cube, since I/O to the hard disk of the SRM proved too inefficient for this purpose.

Timings from a geometry optimisation run on a 32-node partition of the iPSC/2 (SX-nodes) and iPSC/860, and on the Convex C-220 using the GAMESS direct-SCF capabilities are given in Table 3. The example provides a performance comparison for a 6-31G SCF/gradient geometry optimisation of nitrobenzene, featuring a total of 91 basis functions. Considering initially the iPSC/2, the

Table 3. Performance figures (sec) for a 6-31G SCF/gradient geometry optimisation of nitrobenzene

Step	Dependence	Convex C-220 CPU time (sec)	iPSC/2-SX Elapsed time (sec)	iPSC/860 Elapsed time (sec)
2-electron integrals	$O(n^4)$	7991	3353	280
2-electron gradient integrals	$O(n^4)$	2666	907	87
SCF (diag., MMO)	$O(n^3)$	118	6167	486
Total		10775	10427	853

overall timings suggest that a 32-node iPSC/2 performs at approximately the same speed as the Convex C-220. What is perhaps more illuminating is the degradation in performance caused by the serial $O(n^3)$ steps; while the parallelised integral and derivative integral $O(n^4)$ steps are running some 2–3 times faster than the Convex, the serial steps have grown to dominate the computation. This is not surprising when taken in the context of the MMO benchmarks of Table 1, where the C-220 is seen to be some 40 times faster than a single SX node of the iPSC/2. Attempts to vectorise the serial steps reduced the 32-node $O(n^3)$ timing from 6167 to 3379 seconds, and hence the overall time to 7622 seconds. This problem can, however, only be alleviated by parallelising the diagonalisation and matrix multiply operations, given the relatively slow single processor speed of the iPSC/2. Such attempts for the MMO reduced the 32 SX-node $O(n^3)$ timing from 6167 to 2661 seconds, and the overall time to 6921 seconds.

Turning to the iPSC/860, we find the 32-node hypercube to be performing some 30 times faster than the Convex in the $O(n^4)$ steps, with the overall performance, however, downgraded to 12 times the Convex because of the serial $O(n^3)$ code. The iPSC/860 timings compare favourably with those recorded on the Y-MP (1841 seconds for the three steps of Table 3), suggesting the 32-node hypercube is performing some twice the speed of the single-processor Cray.

We show in Table 4 two further sets of timings from direct-SCF calculations on the trinitrotoluene and morphine molecules, both conducted in a 6-31G basis set (154 and 227 basis functions respectively). Timings are presented for the Convex C-220, single-processor Cray Y-MP and iPSC-860, the latter as a function of the number of nodes. We note that in the largest case, the 32-node iPSC/860 is performing some 3 times the single-processor Y-MP, and of the order of 20 times the Convex C-220. The timings also reveal a considerably better scaling as a function of the number of i860 nodes in the larger morphine calculation. The serial $O(n^3)$ code in the trinitrotoluene calculation takes some 300 seconds in total, and is the dominant part of the calculation on 32-nodes. In the no-symmetry morphine case, the $O(n^4)$ step remains dominant, requiring some 1500 seconds in the 32-node case, compared to the $O(n^3)$ timing of 700 seconds.

Finally, we note that implementation of the conventional SCF procedure is straightforward given the CIO. In this case each node writes its own partial list of two-electron integrals to a separate file on the I/O subsystem, and proceeds to

Table 4. Total times for 6-31G direct SCF calculations of trinitrotoluene and morphine (sec)

Machine	Trinitrotoluene (154 GTOs)	Morphine (227 GTOs)
iPSC/860 (Nodes)		
1	3929	
2	2120	
4	1214	10517
8	761	5543
16	557	3352
32	489	2276
Convex C-220	4879	45000
CRAY Y-MP	616	6622

process just this list in each cycle of the iterative SCF process. Again each node will be responsible for generating in parallel a partial Fock matrix, which are subsequently combined using the global GDSUM routine, and diagonalised in serial fashion on each node.

3.3 Integral transformation

The iPSC Concurrent I/O subsystem (CIO), permitting parallel access by the compute nodes of the hypercube to a given file on the subsystem, provides a key feature in implementing standard quantum chemistry codes with a sizable I/O requirement. The integral transformation, or so called '4-index' transformation is needed to transform the two-electron integrals from the original orbital basis, ϕ to the basis of the molecular orbitals (MO) ψ as calculated in the preceding SCF calculation:

$$\psi_p = \sum_i C_{p,i}\phi_i \tag{4}$$

The integrals (pq/rs) are thus computed as:

$$(pq/rs) = \sum_{i,j,k,l} C_{p,i}C_{q,j}C_{r,k}C_{l,s}(ij/kl) \tag{5}$$

Through suitable sorting of the integrals so that for a given i, j all (ij/kl) integrals are present in core the 4-index transformation may be arranged via a series of matrix-multiplications at a total cost of n^5 floating point operations and two sorting steps of n^4 integrals each. The starting point for the transformation is the set of m-partial integral files associated with each node residing on the concurrent file system. The parallel version of the code remains based on the Yoshimine algorithm [14] involving, through the formation of partial sums to reduce the

computational complexity, the following phases:

• Sorting the atomic integrals $[(ij/kl)]$ so that for a given ij pair index, all kl are available. A disk-bin sort method is employed, with each node sorting its own partial integral file and generating its own sortfile.

• The ij pair indices are divided over the nodes. Each node then reads the required integrals from its own, and the $(m-1)$ partial sortfiles generated by the other processors. The key to this implementation is the ability of the I/O subsystem to support simultaneous file access by multiple nodes.

• The transformation of integrals to semi-transformed form involving $M = N(N+1)/2$ matrix multiplications (MMOs) of the form $Q^{\dagger}AQ$ (where Q is the eigenvector array, A the matrix of integrals of common index pair ij) is then distributed evenly over the nodes, each node performing M/m MMOs.

Our experience on the iPSC/2 suggests that the 4-index transformation is well suited to distributed memory MIMD machines given shared disk capability. Shown in Table 5 are the timings, as a function of the number of scalar (SX) nodes, achieved on a 46 basis function transformation. Although the scalability is not as impressive as the integral generation, this may be largely attributed to the limited size of the example, when the n^5 cpu dependence of the transformation will not be dominant. Considering the iPSC/860 performance, we show in Table 6 timings from both the Sorting and Calculation phases of an integral transformation featuring a larger 90 basis function calculation.

Table 5. Timings (sec) for integral generation and integral transformation as a function of iPSC/2 node configuration (see text)

iPSC/2 Node configuration	Integral generation Time (sec)	Integral transformation Time (sec)
1-NODE	162	1058
2-NODE	81	586
4-NODE	42	308
8-NODE	22	228
16-NODE	13	136

Table 6. Timings (sec) for integral transformation (90 GTOs) as a function of iPSC/860 node configuration (see text)

iPSC/860 Node configuration	SORT Time (sec)	CALC Time (sec)	TOTAL Time (sec)
1-NODE	250	1167	1417
2-NODE	149	673	822
4-NODE	96	385	481

3.4 Direct-CI calculations

Turning to the post Hartree–Fock methods, we focus attention on the configuration interaction (CI) method. In CI, solutions to the n-electron Schrödinger equation are sought in the space of n-electron configurations constructed as spin adapted antisymmetric products of orbitals obtained in the SCF-calculation. In order to reduce the size of the problem only configurations differing by at most 2 orbitals with respect to one or more reference configurations are selected. The resulting linear variational problem (size 10^5–10^7) is solved iteratively requiring the evaluation of a CI-matrix CI-vector product every iteration; this product (Z):

$$Z = HC \tag{6}$$

where:

$$H(\mu, \nu) = \sum_{ij} A_{ij}(\mu, \nu)(i/j) + \sum_{ijkl} B_{ijkl}(\mu, \nu)(ij/kl) \tag{7}$$

is calculated directly from integrals, coupling coefficients A and B depending on the spin-couplings μ, ν and CI-vector elements (C) [15].

An efficient implementation requires the integrals to be sorted involving an effort $O(n^4)$. The HC matrix-vector product may then be obtained through a series of matrix multiplications at a total cost of $O(n^6)$. Our initial efforts have focused on porting the Direct-CI program due to Saunders and Van Lenthe [16] to the hypercube. This code is essentially a model configuration symbolic-driven program, where only the all-external part is integral-driven. There are three major components to consider:

1. Integral sorting;

2. Symbolic matrix element generation for model-configurations for each integral type. This involves evaluating the coupling coefficients for each pair of model-configurations and includes forming complete matrix elements for vacuum states;

3. The calculation of $HC = Z$, the CI matrix-vector product, using information from 1 and 2.

3.4.1 Parallel implementation. The division of the workload in the parallel implementation is defined by the calculation of symbolic interactions between model-configurations. Model symbolic matrix element generation is characterised by a double-loop over model-configurations, with the parallel division employed over the outer loop. Such a strategy enables generation of the Z-vector to be performed completely in parallel, driven by the parallelism imposed on the symbolic generation.

The implementation is centered around two categories of file residing on the CIO system, *local* files and a single *common* file. The latter category is only accessed in write mode by the root (or node-0), generally after some global operation, but may be read by all nodes of the cube. It contains frequently accessed data which is not dependent as such on the configurations e.g. the (ab/cd), (ia/bc), (ij/ab), (ia/jb), (ij/ka) integrals. *Local files* have a similar role to the partial AO integral files used in conventional SCF and partial MO integral files generated in the transformation modules. Each processing node will

uniquely access its own *local* file, which contains symbolic information peculiar to the node, arising from the parallel symbolic matrix element generation. The fully internal matrix elements $(ijkl)$ are included in this set. The following steps are performed:

1. The *node* partial MO integral files are sorted and merged (using the global GDSUM routine) for each integral type, prior to output by node-0 to the *common file* (all except for the (ij/kl) integrals). In addition, the (ij/kl) integrals and those required for the Fock-matrices are also sent to the other nodes.

2. The parallel model symbolic matrix element generation produces a symbolic matrix element (PSME) list on each node's *local file.*

3. The PSME list, together with the required integrals and CI coefficients are processed, producing a partial HC vector on each *local file.* Each node thus has access to a complete copy of the C vector, a complete set of integrals (both on the *common file*) and forms its own contribution to the Z-vector.

4. The partial Z-vectors are combined, again using the global routines, and written to the *common file.* The CZ vector-vector products needed may be divided among the nodes (not the rate determining step). A new solution is then determined, with the perturbation vector written to the *common file.* The iterative procedure then returns to step 3 above.

Some additional detail on the present implementation is given below:

• The Fock-matrix contributions, which are pre-summed in the integral-sorting stage using the occupations of the configurations, represent a special case. They could be treated as normal (effective) 1-electron integrals and written to the *common file*, whereupon a given node at Z-vector generation time would simply select those required. At present we generate them in parallel on each node, requiring the Fock-integral generation to be precisely synchronous with the corresponding symbolic generation.

• A complete copy of the Z-vector is written to each *local file.* Since the division of the workload is by interaction (i.e. symbolic), any part of the C- or Z-vectors may be needed. Note, for example, that an interaction between an $N-2$ state a and $N-2$ state b is used to update Z_a (using C_b) and Z_b (using C_a). Thus while the integral storage is just that in the serial code, the parallel implementation suffers from having to keep multiple copies of the Z-vector on disk during its construction.

• After Z-vector generation, the partial Z-vectors are globally summed and re-divided over the nodes, when the Davidson diagonalisation may proceed.

• In the Davidson diagonalisation, the C- and Z-vectors may be distributed over the nodes, with each node housing a partial vector. In practice only dot-products of C and Z (to form the projected H-matrix), and of C and the previous C (to orthogonalise the perturbation vector) are required. The global summation routine is then used to sum the partial dot-products. The advantages of this scheme are that the disk-storage for the vectors is just that of the single-node case, the required dot-products are completely parallel, and efficient use is made of node memory.

• Since no symbolic term is generated for the (ab/cd) $N-2/N-2$ interaction, some additional work is required in parallelising this interaction.

• No effort as yet has been made to optimise I/O efficiency, although we note that for large cases the calculation scales well in terms of the number of MMO operations performed per I/O operation.

The present implementation relies heavily on the distributed I/O capabilities of the iPSC, and only in the Davidson makes efficient use of the combined node memory available. Its major advantage is good node scalability, and a structure that retains code portability.

4 Molecular mechanics and molecular dynamics software

In this section we describe a parallel implementation of part of the AMBER package (Assisted Model Building with Energy Refinement) due to Kollman and co-workers [2]. We have concentrated on two representative applications of the AMBER force field, the molecular mechanics (MM) energy minimisation of a structure by conjugate gradient and/or steepest descent techniques and the calculation of free energy differences by molecular dynamics (MD) simulations.

4.1 Parallelisation strategy

A large part (typically greater than 90%) of the computation time for MM and MD applications is spent in the evaluation of contributions to the energy and forces on the atoms arising from non-bonded and H-bonded atom pairs. This process is usually split into two stages – the generation of a list of pairs of atoms which are close enough to warrant force and energy evaluation (the pair- or neighbour-list), and the energy and force calculation for each entry in this list. For large systems the former is more time-consuming, and considerable savings in computational cost arise from the fact that the pair-list does not need to be updated for every cycle of a MM minimisation or MD simulation (it is typically updated every 10–100 cycles).

Parallelisation can easily be achieved by dividing the pair-list over a number of processors, and summing the contributions to the energy and forces at the end of each cycle. The implementation we have chosen involves running a modified version of the original (MM or MD) codes [2] on one node (referred to as the master), and a small (slave) program on the remaining nodes. Initially the slave program was responsible only for the generation of a partial pair-list and associated energies and forces. We subsequently included code to perform the proper and improper torsion angle terms on the slave nodes as well, since the time required to calculate these terms on the master node often proved significant after the non-bonded contacts had been distributed over 8 or more nodes.

Contributions to the energies and forces from the remaining terms in the energy expression (e.g. bond stretch and angle bend) are performed on the master. After collecting the results from the rest of the processors (by a call to a global summation routine), the master node can then update the atom positions, and send the new coordinates to the slave nodes.

An alternative scheme (which we have not yet implemented) involves running a modified version of the whole code on every node. Each processor is responsible for a subset of the force and energy terms, but, after global summation, the next set of atom positions are computed independently on every node. The

advantage of this scheme is that the communication of the coordinates from master to slaves is avoided. However, a large amount of topology data and executable code is present on all the slave nodes, reducing the memory available for pair-list storage, and thus the size of system that may be studied. However, this scheme may well prove preferable to the master-slave implementation in the future as larger node memories become available.

To achieve load balancing, it is important to distribute the pair-list in such a way that a similar number of atom pairs are handled by each slave node. In practice this is achieved by allocating all i–j pairs ($i \leqslant j$) for a set of atoms i with $i_1 \leqslant i \leqslant i_2$ to a single node, and determining the atom indices i_1 and i_2 for each node. This task is performed by the master node immediately prior to each pair-list generation on the slave nodes. The division of the atom list in this way requires knowledge of the number of i–j pairs associated with each atom i. This is initially obtained by performing a 'dummy' pair-list evaluation on the master node (in which the pair-list is not stored but the pairs associated with each atom are counted), and then updated after each pair-list evaluation on slave nodes by passing the appropriate data back to the master.

4.2 Molecular mechanics

In this section we present sample timings for the parallel implementation of the MINIM module on the Intel iPSC/2 and iPSC/860 multicomputers. Timings for a single processor of a Convex C200 are given for comparison. Table 7 contains elapsed timings as a function of number of nodes for a single cycle of minimisation of the protein alphalytic protease, (1755 atoms, 229,898 non-bonded interactions).

Results for a larger example (which because of memory limitations on the master node could not be run on the iPSC/2) are given in Table 8. The system under study in this case was the enzyme thermolysin (6391 atoms, 1,029,628 non-bonded interactions).

A comparison of the timings for the iPSC/2 with those for the iPSC/860 clearly show the effect of the increase in processing speed without a change in the communication times. The result is rather poor scalability of the processing rate with number of nodes, and the problem is most severe for the smaller example, since the ratio of the number of atoms to the number of non-bonded contacts is smaller here. A consideration of the effects of increasing the communications rate is given below.

Table 7. Timings (sec) for a cycle of molecular mechanics minimisation of protein alphalytic protease as a function of node configuration (see text)

Node configuration	iPSC-2	iPSC-860
2-NODE (1 master, 1 slave)	42.22	2.25
4-NODE (1 master, 3 slaves)	14.9	0.90
8-NODE (1 master, 7 slaves)	6.97	0.53
16-NODE (1 master, 15 slaves)	4.39	0.41
32-NODE (1 master, 31 slaves)	2.94	0.41
Convex C-210	2.24	

Table 8. Timings (sec) for the molecular mechanics minimisation of protein thermolysin as a function of iPSC/860 node configuration (see text)

Node configuration	Time (sec) minimisation
2-NODE (1 master, 1 slave)	9.67
4-NODE (1 master, 3 slaves)	3.66
8-NODE (1 master, 7 slaves)	2.03
16-NODE (1 master, 15 slaves)	1.54
32-NODE (1 master, 31 slaves)	1.29
Convex C-210	7.57

4.3 Free energy difference calculations

The GIBBS module of the AMBER package computes the difference in free energy between two species by MD simulation. The procedure relies on the fact that although the absolute value of the free energy for a given system cannot readily be determined by a single simulation, it is possible to estimate the change in free energy associated with a small perturbation of the system. By gradually mutating one system into another (by smoothly changing the force field parameters) and summing the free energy changes associated with each mutation step, it is possible to estimate the free-energy difference between the species. Clearly any MD algorithm in which the non-bonded contributions to the energy and forces dominate the computation time can be parallelised by the algorithm described above and illustrated for the MM calculation. We show in Table 9 elapsed timings from the domain implementation as a function of number of nodes for 1 cycle of molecular dynamics for the mutation of cytosine to iminocytosine in a bath of 263 water molecules (804 atoms, 100,778 non-bonded interactions).

As for the MINIM module, the iPSC/860 figures indicate that the performance on larger hypercubes (more than 8 nodes) is limited by the cost of the communications *vide infra.*

Table 9. Timings (sec) for a single cycle of molecular dynamics for the mutation of cytosine to minocytosine (see text)

Node configuration	iPSC-2	iPSC-860
2-NODE (1 master, 1 slave)	32.84	2.62
4-NODE (1 master, 3 slaves)	12.21	1.13
8-NODE (1 master, 7 slaves)	7.28	0.72
16-NODE (1 master, 15 slaves)	5.08	0.58
32-NODE (1 master, 31 slaves)	3.18	0.43
Convex C-210	1.21	

Since the results are obtained from a statistical analysis of the simulations, it is also possible to parallelise MD computations by performing an independent simulation on each processor, and combining the results from each run in the statistical analysis. This approach has the advantage that communication between processors is minimal, but as the pair-list is not distributed it is not suitable for large systems if memory on the nodes is limited. Another drawback in some applications is the fact that long-term fluctuations may not manifest themselves in a shorter MD run.

4.4 Enhanced communications

It was noted above that the scalability of the computation rate with number of nodes is limited by the cost of communications. We have thus chosen to consider the consequences for performance of a ten-fold increase in communications speeds, this being the improvement available in the next generation of Intel multicomputers (e.g. the DELTA prototype at CalTech). As an illustration, we consider the larger of the two examples (thermolysin) given above for the MINIM program, running on a 32 node partition of the iPSC/860. The time for each cycle, as given in Table 8 is 1290 ms. A breakdown of this cycle time is given in Table 10, together with projected figures based on a ten-fold increase in communications speeds.

Clearly the higher communications rates result in a much more acceptable scaling of performance, with communications corresponding to about 20% of the job cost.

5 Conclusion

We have outlined the present use and impact of the iPSC/2 and iPSC/860 hypercube-connected multicomputers in computational chemistry, describing code implementation and subsequent performance in both quantum chemistry and macromolecular modelling (MM and MD). Experience to date suggests that the strength of the iPSC machines lies in their ease of use, standard UNIX and Fortran environment and the high degree of functionality associated with the Concurrent File System. The CFS has, for example, provided the means for removing the perceived bottleneck of the integral transformation step in quantum chemistry applications on distributed memory machines [14].

Table 10. Breakdown of the cycle time for the thermolysin minimisation (milliseconds) and projected times given a 10-fold increase in communications speeds

Operation	Time (iPSC/860)	Time (projected)
Send coordinates to nodes	300	30
Parallel computation	330	330
Global summation	630	63
Compute new coordinates	30	30
Total	1290	453

Experience on the i860-based Intel iPSC/860 has proved encouraging. The problems experienced with the iPSC/2 in adopting a coarse-grain approach, namely the somewhat restrictive node memory, the dual nature of the vector board memory, and the degrading effect of serial code, have been significantly alleviated with the iPSC/860. It must be said however that the straightforward migration of the direct-connect message passing hardware from the iPSC/2, and in a similar fashion the CIO system, has rendered the machine extremely unbalanced compared to its predecessor, highlighting the need for direct algorithms to truly exploit the present configuration. Having issued that proviso, we do believe that the i860-based multicomputer will prove at least an order of magnitude more cost-effective in computational chemistry than the recognised minisupercomputer alternatives, given continued enhancements to the present i860 FORTRAN compiler, and the provision of a broader library of optimised software.

References

1. Dupuis M, Spangler D, Wendoloski J (1980) NRCC Software Catalog, Vol 1, Program No QG01 (GAMESS); Guest MF, Kendrick J (1986) GAMESS Users Manual, SERC Daresbury Laboratory, CCP1/86/1; Guest MF, Sherwood P (1990) GAMESS Users Guide and Reference Manual. Revision A.1, SERC Daresbury Laboratory
2. Singh UC, Weiner PK, Caldwell JW, Kollman PA (1986) AMBER (UCSF Version 3.0) Dept Pharmaceutical Chem, Univ of California, San Francisco
3. Kohn L, Margulis N (1989) IEEE Micro, 15
4. Lawson C, Hanson RJ, Kincaid D, Krogh F (1979) ACM Trans Math Software 5:308
5. Guest MF, Harrison RJ, van Lenthe JH, van Corler LCH (1987) Theoret Chim Acta 71:117; Clementi E, Gorongiu G, Detrich JH, Kahnmohammadbaigi H, Chin S, Domingo L, Laaksonen A, Nguyen NL (1985) Physica 131B:74
6. Putlay P (1969) Molec Phys 17:197; (1970) 18:473; (1971) 21:329; Schlegel HB (1982) J Chem Phys 77:3676
7. Guest MF, Wilson S (1981) in: Proc American Chemical Society Meeting, Las Vegas, August 1980. Wiley-Interscience, NY; Saunders VR, Guest MF (1982) Comp Phys Comm 26:389; Guest MF (1985) in: Supercomputer Simulations in Chemistry, Montreal
8. Almlöf J, Faegri K, Korsell K (1982) J Comput Chem 3:385; Haser M, Ahlrichs R (1989) J Comp Chem 10:104
9. Pulay P (1980) Chem Phys Lett 73:393; (1982) J Comp Chem 3:556
10. Pople JA, Hehre WJ (1978) J Comp Phys 27:161
11. Dupuis M, Rys J, King HF (1976) J Chem Phys 65:114
12. Harrison RJ, Guest MF (1988) CCP1 Newsletter 12:37
13. Littlefield RJ, Maschoff KJ (1992) J Par Distr Computing (in press)
14. Yoshimine M (1969) IBM Technical Report RJ-555 San Jose, USA
15. Siegbahn PEM (1980) J Chem Phys 72:1647
16. Saunders VR, van Lenthe JH (1983) Molec Phys 48:923