Theor Chim Acta (1987) 71: 117-148

Theoretica Chimica Acta

Computational chemistry on the FPS-X64 scientific computers

Experience on single- and multi-processor systems

Martyn F. Guest[1], Robert J. Harrison[1], Johan H. van Lenthe[2], and Lambertus C. H. van Corler[2]

[1]Computational Science Group, SERC Daresbury Laboratory, Daresbury, Warrington WA4 4AD, UK
[2]Theoretical Chemistry Group, State University of Utrecht, Transitorium III, Padualaan 8, Utrecht-De Uithof, The Netherlands

(Received July 7, revised and accepted September 10, 1986)

Contents

1. Introduction

The purpose of this paper is to review the impact and use of the FPS-X64 scientific computers in computational chemistry, focusing attention on experience gained on both an FPS-164/MAX, installed at the Science and Engineering Research Council's Daresbury Laboratory, UK, and on the distributed system, composed of an IBM 4381-3 front end processor and 10 FPS-164 attached array processors, at the ECSEC facility in Rome, Italy.

The review is necessarily selective, and is divided into several sections. In Sect. 2 we outline some general characteristics of the FPS-X64 machines. We consider their performance in various computational chemistry kernels in Sect. 3, where the general strategy adopted in code implementation is outlined. In Sect. 4 we consider the implementation and performance on FPS X64 scientific computers of a typical *ab initio* program, GAMESS, and provide in Sect. 5 estimates of the cost-effectiveness of these machines in the context of supercomputers exemplified by the Cray-1S and CDC Cyber-205.

In Sect. 6 we consider the feasibility of migrating computational chemistry codes to the different architectures characteristic of the next generation of multiprocessors. Finally, in Sect. 7 we describe our initial attempts at adapting codes to run on the distributed system of multiple FPS processors at the ECSEC facility in Rome.

1.1. Cost-effective computing in chemistry

The introduction in 1976 of the VAX 11/780 by Digital provided arguably the greatest impetus to the use of minicomputers in large scale chemical computations, and led to the migration of many theoretical chemists from the less cost-effective alternative offered by the large scalar mainframes that typified the central computing facility of the 1960s and 1970s. Since 1976 use of the VAX-11/780 "superminicomputer" or its equivalents in chemical computations has proliferated to the extent of becoming a *de facto* standard. Yet even the new generation of superminis, from Data General, Digital, Gould/SEL, Prime, Harris, IBM and Perkin-Elmer [1], that provide speeds several times that of the VAX-11/780, cannot disguise the fact that such machines are often inadequate for more than routine applications. Indeed the superminis perform as much as two orders of magnitude slower than the supercomputers in widespread use, the Cray-1S and CDC Cyber-205.

The search for increased cost-effective performance is evident in the emergence of two classes of computer systems during the last decade, the supercomputer itself and superminis with attached processors (hereafter referred to as mini + AP). We consider in the sections below the relative merits of these two approaches and, by comparing the performance of chemical software on both Cray-1S and FPS-164/MAX, consider whether mini+AP configurations represent genuine alternatives to time-shared supercomputer access.

The pioneering work on the FPS-164 attached scientific computer was conducted by workers in the theoretical chemistry group at Argonne National Laboratory [2]. This work demonstrated that the FPS-164 provides performance comparable to that of modern mainframe computers at a cost comparable to that of a supermini. A feature of the Argonne work, however, and one that will continually be reflected in the following sections, is that achieving such performance requires considerable care in developing and implementing codes. Such an effect is, of course, well established in the regime of supercomputers typified by the Cray-1S [3-5].

Finally, mention should be made of the inevitable path that lies ahead in the search for improved performance, namely the exploitation of the parallelism inherent in scientific computation. Perhaps the most impressive demonstration in a chemical framework of migrating code from sequential to parallel is seen in the LCAP parallel system due to Clementi and co-workers at Kingston, New York [6], which has at present a theoretical performance of about 110 Mflops [6]. Indeed, execution times for various chemistry codes on the distributed system are comparable to those on a Cray-1S [7]. Such figures must, however, be considered in the light of no attempt having been made to either exploit the vector architecture of the Cray or to adapt the codes to the architecture of the FPS-164 [7]. Our primary aim here is somewhat different. We wish to consider the steps necessary to exploit the full performance of one AP, and to subsequently contrast that performance with supercomputer performance typified by the Cray-1S and Cyber-205. Consideration is given in Sect. 6 to the features of multiprocessor systems that are necessary in order to satisfy the requirements of computational chemistry, while our own experience in adapting codes to a parallel environment will be considered in Sect. 7.

2. The FPS-X64 scientific computers

In this section we briefly review some features of the FPS-X64 scientific computers, focusing attention on the FPS-164 and the machine configuration at Daresbury.

The FPS-164 attached processor is a highly parallel, pipelined processing unit. It has been designed to be a fast and inexpensive CPU with 64-bit floating point operations. The processor is commonly referred to as an array processor because of its ability to perform array operations quickly. The AP contains an independent CPU and its own memory and disk drives. The CPU on the FPS-164 runs at 5.5 million instructions per second, and several concurrent operations can take place on each instruction cycle, so that peak performance is about 11 megaflops. While allowing for the distinction between peak and realised performance, the AP has proved effective for the specialised "number crunching" that constitutes a large part of scientific and engineering computer applications [2, 8].

The machine may be upgraded to an FPS-164/MAX, providing the potential to include vector features through the addition of special purpose boards to augment performance, particularly on matrix operations. Each MAX board adds 22 Mflops to peak attainable performance. The present machine configuration at Daresbury includes three such boards, and thus has a theoretical performance of 77 Mflops.

In February 1985 FPS announced the release of a new product, the FPS-264. While maintaining 100% software compatability with the FPS-164, the reduction in the machine clock cycle, from 182 to 53 ns, leads to the machine running at least 3.45 times faster than its predecessor when executing software that does not involve disk or transfers to the front end, cache misses, interleave spins or refresh. While the machine architecture allows for the same type of memory-mapped schemes used in the MAX implementation on the FPS-264 as on the 164, a MAX version of the 264 remains unannounced given the requirement for ECL gate arrays of a density not presently available.

2.1. Hardware

The machine logic is 64-bit with some VLSI components, but the overall technology is not state of the art. The 64 bit instruction word is capable of initiating up to 10 tasks within each machine cycle (182 ns). The asynchronous and pipelined design allows initiation of a task on a given unit on every machine cycle even though a previous task is not finished. The parallel design allows simultaneous initiation of tasks on more than one unit in the same machine cycle. For example, an add, a multiply, a memory fetch, and compare can all be initiated with one instruction word.

The machine at Daresbury comprises a main memory with 1.5 MWords (12 MBytes) of storage, and a table memory. The main memory is the primary storage unit, containing program source and data. The table memory, an auxiliary storage medium, needs 2 machine cycles to complete a read or write request, compared with 3 cycles for the main memory and is partitioned into a read-only portion (TMROM, 8 KWords), containing certain constant tables, and a random access portion (TMRAM, 16 KWords) which can be used via assembly language or through FPS supplied math library routines.

2.2. Strategy for using the scientific computer

The AP may be used in two distinct ways. In the AP EXecutive (APEX) mode, a host FORTRAN program calls subroutines that run on the AP. In contrast the

System Job Executive (SJE) mode allows a complete program to be run entirely on the AP, and involves significantly less interaction and reliance on the host. In this latter mode, the SJE software is run partially on the AP and partially on the host. Typically in the SJE mode of operation the user can interact with the SJE software on the host, load the executable module and data to the AP, create other files on the disk subsystem, initiate execution and wait for the completion of the job. Note that all usage of the Daresbury machine described in this paper has relied exclusively on SJE mode, while the parallel implementation described in Sect. 7 made extensive use of APEX mode (see Appendix II).

3. Computational chemistry

It is in conjunction with the remarkable progress in computer technology over the last two decades that chemical theorists are now making the type of contributions to science envisaged by Dirac more than fifty years ago. Looking ahead, we may be confident that within the next decade theoretical/computational chemistry will be an equal partner with the traditional fields of organic, inorganic and physical chemistry. Clearly, computers are and will continue to be part of the arsenal of instrumentation available to molecular science.

The *ab initio* computation of the electronic structure of molecules is a subject with a history at least as long as that of electronic computers themselves. Since 1960, the subject has shown a particularly rapid evolution, with the outcome that many reasonably efficient computer programs are available on scalar machines. These codes share at least two traits: they all represent a considerable investment in man-years (say between 5 and 30) and all are relatively large (say between 20 000 and 150 000 lines of FORTRAN), as befits a complicated problem. The advent of the large scale vector processors (e.g. Cray-1, CDC Cyber-205) and their smaller brothers (e.g. FPS-164 and the more recent Convex C1) led to substantial efforts [2–5] to alter the existing codes to conform with the requirements of these new machines. The aim throughout was to achieve a code design which was reasonably transportable from one vector (or array or scalar) processing computer to another, with the capability of driving these processors at somewhat near their maximum power. Central to this effort was the realisation that five of the most important steps in a typical quantum chemistry calculation may be structured around the matrix multiply operation (MMO), yielding a relatively machine independent structure, given that the MMO is capable of driving all existing computers at their maximal power. A summary of the characteristics of electronic structure calculations, and an outline of the MMO-based algorithms involved has been given previously [2–3].

3.1. Code implementation strategy

Our experience in the development of optimised Quantum Chemistry (QC) programs on vector processors typified by the Cray-1 and Cyber-205 suggests that optimum performance can only be achieved by resorting to assembly language constructs for many of the vector kernels involved, e.g. the matrix multiply operation (MMO), SCATTER, transposition etc. This partly reflects a lack of "richness" in typical QC software, where for each fetch and store very little floating point arithmetic occurs. This use of assembly code permits, for example, account to be taken of the segmented nature of the scalar and vector functional units, and enhances the MMO on the Cray-1S from some 37 Mflop to an asymptotic performance of 147 Mflop (see below).

The Cray-1S is perhaps not the ideal machine for comparing FORTRAN and assembler performance. Store access conflict problems together with chain-slot loss left much to be desired for the FORTRAN user. Although these effects have been largely remedied on the XMP-1, reliance on FORTRAN still leaves the user unable to take advantage of hiding scalar control activities under vector operations. The situation is less clear on the multiprocessor XMP-*n*, where store access conflict problems in the case of a general code implemented at the large granularity level would be exacerbated by the generation of redundant store/fetch operations.

While the 205 is potentially an ideal FORTRAN machine, similar problems remain because of deficiencies in the current FORTRAN compiler. Many of the typical loops in QC software involve items from argument lists, and as such inhibit automatic vectorisation using the FORTRAN 200 Compiler, given the requirement for a vector length "known" to be less than 65536. Since use of the "UNSAFE vectorisation" option commonly led to miscompiled code, the strategy of building a META library of FORTRAN callable routines was adopted. Such a philosophy is in line with using the mathematical subroutine library of the FPS-164, which includes FORTRAN callable subroutines written in optimised assembly language to perform, amongst others, vector and matrix operations. The typical improvement figures arising from use of this library have been documented by Dunning and co-workers [2], and are reflected in the timings quoted in Table 1 for various standard vector operations.

The above discussion raises the obvious question as to what order of operation should this reliance on factored routines be instigated. It would seem clear that factoring of code should occur at the N^2/N^3 level, e.g. MMO, matrix square, diagonalisation, etc.; i.e. where there is a clear potential for taking advantage of the interplay between loop lengths. Indeed the prospects for large scale granularity

Table 1. APMATH64 versus FORTRAN (OPT = 3) CPU times (seconds) for various standard vector operations, involving 1000 calls with a vector length (VL) of 50

Mode	Vector routine				
	SDOT	SAXPY	SCALE	COPY	ADD
Out-of-line FORTRAN	0.040	0.051	0.042	0.038	0.067
In-line FORTRAN	0.032	0.041	0.031	0.031	0.060
APMATH/APAL	0.027	0.037	0.026	0.025	0.037
Break-even VL over FORTRAN	40	35	20	20	10
VL for 90% full speed	90	100	70	70	80

may well be at this level. Factoring at order N, however, is more debatable; in cases where one cannot guarantee the magnitude of N (i.e. where the compiler will use a given algorithm regardless), or where one is dealing with non-vectorised code, then again factoring may appear beneficial, to accommodate alternative optimisation techniques e.g. loop folding. Factoring at the N-level is not, however, in general recommended, except perhaps where all of the loop may be represented by a single CALL to the library utility. Obviously the larger the library, the more likely this condition would be satisfied. An attempt to widen the range of the Basic Linear Algebra Subroutines (BLAS) might encourage such factorisation.

As an example of the clear advantages to be gained from factoring routines at the N^2/N^3 level, we present in Table 2 the results of a matrix diagonalisation benchmark intended to supplement the previous analysis conducted by Dunning and co-workers [2]. We consider a similar benchmark, based on diagonalising a series of real symmetric matrices, with rank 10, 20, 30, . . . , 100, using 64-bit floating point arithmetic. Again the CPU time was measured for the diagonalisation of each size matrix and a weighted sum of these CPU times used as the benchmark execution time. Each matrix of rank N contributed to this sum as $(100/N)^3 \times$ CPU [2]. While the previous analysis was restricted to the EISPACK RS routine, we consider below the performance on the Cray-1S, FPS-164 and Cyber-205 (2-pipe) of nine diagonalisation routines available in various mathematical libraries and quantum chemistry codes:

(i) The library routine RS available in APMATH64 on the FPS-164 and in SCILIB on the Cray-1S (optimised FORTRAN).

Table 2. Matrix diagonalisation benchmark. Total CPU time (seconds) as a weighted function of the time necessary to diagonalise ten real symmetric matrices (see text)

		CPU time				
			FPS-164			
Routine	Source/ library	Cray-1S	OPT(1)	OPT(3)	OPT(4) ONETRIP	Cyber-205
	Householder					
RS	APMATH64/ SCILIB	6.8		32.6		
EIGRS	IMSL	10.2	117.2	79.9	78.0	24.4
HQRII	MOPAC	14.4	96.2	45.9	43.2	21.6
HOUSE		—		24.1		
F02ABF	NAG	6.0		73.3		16.2
SDIAG2		8.3	107.7	76.2	74.0	23.7
	Jacobi					
JACOBI	ATMOL	20.8	139.2	114.0	111.9	32.1[a]
ERDUW	BERKELEY	27.6	160.1	122.1	121.5	33.3
JACO	DISCO	116.9	675.6	432.3	416.1	126.3

[a](Meta 15.7 s) FORTRAN 200, cycle L640A

(ii) EIGRS, an unoptimised FORTRAN version of RS from the IMSL library of routines [9].
(iii) F02ABF, from the NAG library [10].
(iv) HQRII [11], as implemented in the semiempirical MOPAC program [12].
(v) HOUSE, a locally written APAL version of HQRII employing table memory in the tridiagonalisation and MAX boards in the back transformation.
(vi) SDIAG2, as implemented in the MUNICH system of programs.

These routines are all based on the Householder QR method. We also consider the performance of the following Jacobi diagonalisers:

(vii) JACOBI, from the ATMOL system of programs [13].
(viii) JACO, the diagonalisation routine from the direct-SCF program DISCO [14].
(ix) ERDUW, as taken from the Berkely System of Quantum Chemistry codes.

As noted by Dunning, a call to the APMATH library RS routine improves the FPS-164 time by a factor of 2.5 over the FORTRAN equivalent code (EIGRS), although the performance of the FORTRAN coded HQRII on the FPS should be noted, 75% of the speed of the APAL routine and almost twice as fast as the next best FORTRAN code (F02ABF). HQRII is also the most efficient FORTRAN routine on the host AS-7000, but is the slowest Householder routine on the Cray-1S.

Improvements with pipelining on the FPS are rather disappointing, with the factor of 2.2 achieved by HQRII far greater than that of the other routines. In fact FORTRAN on the FPS is only marginally better than on the AS-7000, with factors between 1.1 and 1.4, a further reminder that replacing FORTRAN subroutines with calls to the standard APMATH routines provides an essential step in realising the full potential offered by the FPS-164. We are unable to explain the exceedingly poor performance of the Jacobi diagonaliser in DISCO, a feature on all machines.

Finally we consider the performance of our MAX-compatible routine HOUSE operating on larger matrices. For a matrix of rank 240 HOUSE requires 8.5 s, to be compared with 28.7 s for the APMATH EIGRS routine. For such matrices the FPS-164/MAX is approximately 92 times faster than the ubiquitous VAX-11/780.

3.2. Disk I/O

Quantum chemistry is, of course, a subject with a notoriously high demand on I/O devices, both in terms of I/O rate and data capacity. The computational chemist will always be faced with the headache of trying to reduce I/O charges, and the optimum way of achieving this is often a function of both machine and installation. However, the design of any I/O system for quantum chemistry must always feature asynchronous activities with the maximum size of data transfer permitted, and by implication will require a large amount of available memory for coordinating this activity. On the Cyber-205 our present I/O system, utilising the maximum unit of data transfer (12K words using small page transfers), acts to improve the I/O efficiency of certain stages of computation (and subsequent cost) by a factor of ten or more. The development of an efficient I/O system for the FPS-164 has been outlined in [2].

We note here a similar behaviour to that found on the Cyber when performing asynchronous I/O to the D64 and FD64 disk subsystems on the FPS. Table 3 presents the elapsed times and associated transfer rates observed in migrating 40 MBytes of data asynchronously to an FD64 800 MBytes disk, as a function of the unit of data transfer. Increasing this unit from 0.5K to 20K words is seen to improve the I/O efficiency by a factor of 7, although even with 20K word transfers performance falls short of the 1.8 MBytes/s theoretical maximum. We are currently unable to explain this rather poor performance, which may well be a local characteristic of the Daresbury set-up.

3.3. Role of the MMO

As has been mentioned previously the electronic structure problem can be formulated such that the MMO

$$R = AB$$

where perhaps A or B (but not both) is sparse, is of fundamental importance for five steps in a typical calculation

(i) 2-electron integral evaluation over Gaussian functions [15, 16]
(ii) Hartree–Fock construction of a non-correlated wavefunction [17]
(iii) Transformation of the 2-electron integrals from an atomic to a molecular orbital basis [18]
(iv) The construction of a correlated wavefunction using the DIRECT-CI technique [4, 19]
(v) The multiconfiguration SCF construction of a correlated wavefunction [20].

Details of an optimised MMO routine (MXMB) written in assembly language for the Cray-1S, using the "outer product" formalism and capable of utilising

Table 3. Disk I/O benchmark total elapsed times (seconds) and associated I/O rates for the transfer of 40 MBytes of data, as a function of the unit of transfer

Unit of transfer (KWords)	Total elapsed time (s)	Transfer rate (MBytes/s)
0.5	280	0.14
1	140	0.28
2	105	0.38
4	70	0.58
5	57	0.70
10	44	0.86
20	40	1.00

any sparsity in A or B, have been given previously [3]. Execution rates in excess of 100 Mflop are possible on the Cray with matrix dimensions (vector length, VL) as low as 24, the routine achieving an asymptotic performance of 147 Mflop when a dimension of 64 is used (or any multiple thereof), correlating with a hardware characteristic of the Cray. The performance of a straightforwardly coded vectorised FORTRAN version is in marked contrast, being asymptotically only 37 Mflop with very large matrix dimensions (see Fig. 1 of [3b]). Note that the direction of vectorisation in the sparse MMO depends upon which matrix is assumed sparse, so that one should vectorise by rows or columns of the result matrix, depending upon whether *A* or *B*, respectively, is assumed sparse.

It is perhaps worth recalling at this point the Mflop rate for the hierarchy of preferred operations relevant to the machines under discussion:

	Cray-1S	Cyber-205 2-pipe	FPS-164
Recursive linked triad (MMO)	147	200	9.6
Recursive linked triadic	49	200	4
Recursive linked diadic	38	100	2.5

Effectively the full potential of the FPS-164 in the sparse matrix multiply may be obtained directly through use of the APMATH library routine SMMMV (see [2b]).

A consequence of the increased vector startup times on the Cyber-205 is, of course, the requirement for a significantly longer VL if optimal performance is to be achieved. Thus MMO rates of 100 Mflop on a 2-pipe 205 require a matrix dimension of 170, and illustrate the need for an alternative "long vector" algorithm for small matrices. Such an algorithm [21], comprising a hybrid scheme of dyadic operations, matrix transposition and bisection techniques outperforms the outer product algorithm by a factor of two for matrices of dimension 20 on a 2-pipe machine, where it rapidly achieves an asymptotic performance of 45 Mflop. Indeed this algorithm can outperform the hardware inner product order on a 4-pipe machine.

A comparison of the sparse MMO performance on the Cray-1S, FPS-164, Cyber-205 (2-pipe) and AS-7000 is given in Table 4. In this benchmark a series of MMOs 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 4 refer to 100 MMOs for rank 10 matrices, 90 for 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, and in each case a comparison is drawn between the performance of code written entirely in FORTRAN and that in assembly language.

For FORTRAN implementation, the sparse-MMO on the Cray-1S is 10.6 times faster than the FPS-164 when handling non-sparse matrices, and 9.1 times faster

Table 4. Sparse MMO benchmark. Total CPU times (seconds) for a series of sparse MMOs (R = AB, see text) implemented in FORTRAN and assembly language

		Sparsity in B-matrix	
		0%	50%
FORTRAN			
AS-7000[a]		103.5	52.9
FPS-164 (OPT = 3)[b]		60.4	32.8
Cyber-205 (OPT = DPRS)[f]		19.2	10.2
Cray-1S[c]		5.7	3.6
Assembly language			
AS-7000		99.0	49.4
FPS-164 (SMMMV)		17.4	11.7
FPS-164/MAX-3[d]	FULL (PDOT)	8.9	8.9
	FULL (PMMUL)	4.9	4.9
	SPARSE	6.4	4.9
Cyber-205 MXMB		2.9	1.6
Cray-1S	MXMB	1.4	0.7
	MXMA[e]	1.2	1.2

[a] FORTVS, level 1.4.1, with OPT = 3 and AUTODBL(DBL)
[b] APFTN64, F02-00
[c] COS 1.12
[d] See text
[e] SCILIB
[f] FORTRAN 200, cycle L640A

in the sparse matrix case. The FPS-164 is faster than the AS-7000 in both cases, by factors of 1.7 and 1.6 in the non-sparse and sparse case, respectively.

Comparing assembly language MMO to FORTRAN MMO, the APMATH library routine SMMMV is of order 2.8-3.5 times faster, and the Cray-1S MXMB 4.1-5.1 times faster with assembly language implementation, the factor depending on sparsity. Indeed these assembly language timings reflect the asymptotic performance of the machines, with the Cray-1S 16.7 times faster than the FPS-164 in the sparse-MMO.

The FPS-164 sparse MMO routine SMMMV is 4.5 times faster than FORTRAN implementation on the AS-7000 when operating on sparse matrices. Compared to supercomputer FORTRAN implementation, SMMMV is comparable in speed to the Cyber, and three times slower than the Cray-1S.

Our initial approach to utilising the MAX capabilities of the machine has centred on coding an efficient FPS/MAX sparse MMO routine, SPARSE. The code initially invokes an APAL routine to pack the sparse matrix in blocks prior to invoking a second routine to drive the MAX.

The sparse dot product runs at full speed on the MAX but cannot, due to extra overhead from unpacking the MD vector, use Table Memory. Thus the limiting speed of a sparse dot product on an FPS-164/MAX-n is 22n Mflops. A non-sparse algorithm would limit to (22n + 11) Mflops, although naive use of the APMATH-MAX basic library routines as the kernel of a FORTRAN MMO produces an unacceptable level of subroutine call overhead for all but very large matrices. The finite dimensions of MAX vector memory, MD scratch work space and conflicting requirements to reduce packing and loading overhead, further limit the peak theoretical performance of a sparse MAX matrix product to 54 Mflops on 3 boards. The achieved rate of the MAX sparse MMO in the benchmark of Table 4 reflects the performance envelope of the benchmark, which peaks at matrix rank of 80. A significantly larger vector length would be required to approach asymptotic performance. With 50% sparsity and matrix rank of 400, SPARSE is 4.5 times faster than the library routine SMMMV.

4. An *ab initio* computational chemistry system

The vector implementation of five of the steps in a typical Quantum Chemistry calculation leads to the efficient computation of the total energy of a given molecular species at a fixed nuclear geometry. Chemistry, however, is not concerned merely with the properties of a molecule at a single point, but with the more general characteristics of multi-dimensional potential energy surfaces, with a quantitative account of the making and breaking of chemical bonds crucial in the study of reaction mechanisms. Ideally we wish to move automatically, and systematically, on a surface from one stable molecular geometry, through one or more transition states, onto a product equilibrium geometry. Such a "walking process" became viable with the development of efficient methods for calculating gradients of the molecular energy [22], together with the evolution of robust and efficient algorithms for locating minima and transition states based on first- and, more recently, second-derivative information.

The complexity and sheer size of the programs required in such studies presents a formidable task for the computational chemist. Such a code must include, in addition to the optimised steps above, routines for the evaluation of the energy derivative for a broad class of wavefunctions of increasing complexity, involving computation of the derivatives of the one- and two-electron integrals. All such steps reside under control of optimisation routines designed to locate and characterise the stationary points on the potential surface in the minimum number of energy and energy-gradient evaluations.

These programs are potentially vast consumers of both machine cycles and the more general resources of memory, disk space etc. Much of the 300 hours of Cray-1 time allocated by SERC to users in the QC community in the period 1983–4 was consumed through use of these codes. It is estimated that the equivalent computations on the AS-7000 at Daresbury would have required at least 5000 h, effectively the entire machine. Some 20–30% of the current VP usage at the University of London Computer Centre (ULCC) and the University of Manchester Computer Centre (UMRCC) is taken up by Quantum Chemistry calculations.

Work in this area at Daresbury has concentrated on the GAMESS program (General Atomic and Molecular Electronic Structure System), a 200 000 line general purpose *ab initio* molecular electronic structure program for performing SCF- and MCSCF-gradient calculations [23]. The program utilises the Rys Polynomial or Rotation techniques to evaluate repulsion integrals over *s*, *p* and *d* type Cartesian Gaussian orbitals. Open- and closed-shell SCF treatments are available within both the RHF and UHF framework, with convergence controls provided through a hybrid scheme of level shifters and the DIIS method [24]. In addition generalised valence bond (GVB), CASSCF and more general MCSCF [20] calculations may be performed.

The analytic energy gradient is available for each class of wavefunction above. Gradients for *s* and *p* Gaussians are evaluated using the algorithm due to Schlegel [25], while gradients involving *d* Gaussians utilise the Rys Polynomial Method. The recent incorporation of gradient pseudopotential capabilities also promises to significantly extend the size of system amenable to study. Geometry optimisation is performed using a quasi-Newton rank-2 update method, while transition state location is available through either a synchronous transit [26] or trust region method [27]. Force constants may be evaluated by numerical differentiation. Large scale multi-reference CI calculations may be performed using the Direct-CI formalism.

A variety of wavefunction analysis methods are available, including population analysis, localised orbitals, graphical analysis and calculation of 1-electron properties.

4.1. Implementation of GAMESS on an FPS-164

As an example of the typical problems encountered in migrating code from processor to processor, we consider our implementation of the GAMESS package on the FPS-164. Some of the problems that arose in converting the Cray version of the code are outlined below (see also [2]):

(i) A potential problem is the use of non-standard data types - INTEGER*2, LOGICAL*1 - in common blocks and equivalence usage. Due account of these effects had been taken during Cray implementation of the code.
(ii) Use of extended DO-loops
(iii) Use of Hollerith data types instead of character type data. Most QC programs are written in FORTRAN-4, but nevertheless compile successfully with FORTRAN-77 compilers, with the aid of various language flag options etc. Yet it was felt timely, given the general requirements of APFTN64, to undertake the task of converting the entire code to a FORTRAN-77 standard, at least as far as character type data was concerned. This conversion took approximately 2 weeks to carry out, involving, for example, major changes to the free-format data input routines.
(iv) Use of dummy arrays that are not initialised on the most recent entry into the subroutine.

(v) The most serious problem encountered, and one that took several months to resolve, involved implementation of an efficient direct access asynchronous I/O system (see Sect. 3.2). Both the Cray and Cyber versions of the code rely on a multi-buffered I/O system based on the fundamental building block of 512 words (the Cray block). Typically multiple blocks are written under control of a single output statement (using, for example, the Q5 routines on the 205), but may be subsequently processed through multiple read commands. Attempts to conform to this structure using the asynchronous I/O facilities within APFTN64 revealed intolerable elapsed/wait times. The initial solution to this problem involved basing the I/O system on the FILE$ routines (vol. 3 of the Operating System Manual Set), a collection of FORTRAN callable routines providing far greater flexibility than their FORTRAN-77 counterparts.

(vi) The initial 1/2 MWord configuration on the Daresbury machine provided a potential constraint on the systems amenable to study. The amount of available memory has been optimised in two ways:

(a) In common with most QC codes, GAMESS features a large array which is partitioned and passed to subroutines in segments, the space requirements for each segment depending on the chemical system under investigation. Access to such an array on the FPS is achieved through use of the /SYS$MD/ common block and SYS$ADDMEM routine to define the first usable location in the program workspace.

(b) The space requirements of the code itself have been minimised by extensive use of the flexible OVERLAY features of APLINK64.

4.2. Performance of GAMESS on an FPS-164

In the present section we consider the performance of the GAMESS program on the FPS-164, Cray-1S and Cyber-205. Three separate calculations are considered. In Tables 5 and 6 we list the CPU times for the various steps in calculations on the TiH_4 molecule. The first calculation (see Table 5) involved the evaluation of the one- and two-electron integrals over a contracted Gaussian basis of 76 functions, followed by an SCF calculation. The optimised orbital space was then transformed to a 67 orbital basis (with the Ti 1s, 2s, 2p, 3s and 3p orbitals frozen),

Table 5. FPS-164 and Cray-1S performance comparison for a 2nd-order CI calculation on TiH_4 (see text) using GAMESS

	CPU time (seconds)		CPU ratio
Description	FPS-164	Cray-1S	FPS/Cray
Integral generation	305.2	54.6	5.6
SCF optimisation	90.1	23.0	3.9
Integral transformation	1071.0	98.8	10.8
CI diagonalisation[a]			
Cycle time (775742 csf)	1446.1	198.3	7.3

[a] Convergence achieved in 10 iterative cycles

and a full second-order CI wavefunction calculation performed, involving 8 valence electrons distributed in 10 orbitals ($4a_1$, $3t_2$, $4t_2$ and $5t_2$), yielding a total of 775, 742 configuration state functions (csf's), conducted within the direct-CI framework.

The integral generation step is 5.6 times faster on the Cray then on the FPS-164, and the SCF step 3.9 times faster. Both integral transformation and direct-CI steps show increased factors arising from the dominant role played by the sparse MMO. The factor of 11 in the transformation step would undoubtedly be repeated in the CI phase given an increase in the external space employed in the present calculation. The observed factor of 7.3 stems from an external space of just 57 functions. Note that on the current MAX implementation, the SCF and transformation times of Table 5 are reduced to 84.4 and 553.8 s respectively, with the SCF step now 3.7 times faster on the Cray-1S and the transformation step 5.6 times faster.

In the second calculation (Table 6) we consider the performance of the 2nd-order direct-MCSCF module [20]. The timings refer to a more modest basis of 46 functions, and are given for integral generation, SCF, symmmetry adaptation (via a pseudo-transformation) and a CASSCF calculation, with 8 electrons in 12 orbitals, yielding a total of 17 945 csf's. Referring to the CASSCF step, we find the Cray-1S to be 8 times faster than the FPS-164, and the Cyber-205 (1-pipe) only 2.2 times faster.

The final benchmark (Table 7) contrasts the performance of GAMESS with that of the CADPAC and ATMOL program suites in performing a routine 91 basis function 3-21G SCF calculation on the nitrobenzene molecule (C_{2v} symmetry). Considering GAMESS, we again find the integral generation step to be 5.0 faster on the Cray-1S than on the FPS-164, and the SCF step 5.2 times faster. GAMESS is well suited to this type of calculation, with the increased speed factors against CADPAC and ATMOL in integral evaluation derived from use of the rotated axis technique rather than Gauss–Rys quadrature. Two factors account for the increased performance in the SCF step. With the present DIIS implementation, GAMESS converged in 15 iterative cycles, whilst both ATMOL and CADPAC required manual intervention, arising from inappropriate starting vectors, and

Table 6. GAMESS machine performance comparison for a CASSCF calculation on the TiH_4 molecule (see text)

	CPU time (seconds)		
Description	FPS-164	Cray-1S	Cyber-205[a]
Integral generation	115		
SCF optimisation	23		
Symmetry adaptation	80	9	20
17,945 csf-MCSCF	2691	345	1184

[a] 1-pipe at SARA, Amsterdam

Table 7. FPS-164, Cyber-205 and Cray-1S performance comparison in an SCF calculation on the Nitrobenzene molecule (see text)

	CPU time (seconds)		
Program	FPS-164	Cray-1S	Cyber-205[a]
	2-electron integral evaluation		
GAMESS	199	40 (16)[b]	72
CADPAC	—	114	208
ATMOL	1436	297	375
	SCF wavefunction optimisation		
GAMESS	188	36	105
CADPAC	—	87	228
ATMOL	3200	248	600

[a] 2-pipe, Manchester
[b] Vectorised rotated axis integrals

needed significantly more iterations to achieve convergence. Both GAMESS and CADPAC use in default the P-supermatrix, while ATMOL uses integrals directly in the SCF step. Although these timings may thus appear somewhat biased, they were obtained using the facilities available within the standard version of each program on the parent machine, and clearly show that GAMESS has a major role to play in SCF calculations on large systems with *s*, *p* basis sets.

4.3. Performance of the FPS-264

To provide some idea of the improvement to be expected on the newly-released FPS-264, we include in Table 8 the overall timings obtained in optimising the geometrical structure of chromium tetranitrosyl, $Cr(NO)_4$, using a double zeta basis of 110 functions [28], The table shows the breakdown of this gradient

Table 8. Performance of the FPS-164 and FPS-264 in computational chemistry. A geometry optimisation of $Cr(NO)_4$ (see text)

	CPU times (seconds)	
Step	FPS-164	FPS-264
Input phase	3	1
Vector generation	11	3
1-electron integrals	104	30
2-electron integrals	4221	1192
SCF	2153	622
1-electron gradient integrals	559	167
2-electron gradient integrals	8845	2582
Wavefunction analysis	44	12
Other	12	3
TOTAL CPU seconds	15952	4612

optimisation into component parts, and contrasts performance on FPS-164 (release F1.0) and FPS-264.

The increased performance of the 264, by a factor of 3.5, suggests that the impact of this machine from Floating Point Systems on computational chemistry will be just as marked as its predecessor. Note that this benchmark involved migrating the executable load module directly from 164 to 264, and does not reflect the possible improvement to be obtained from 264 specific software.

4.4. Towards open-ended ab initio capabilities

At first sight the N^4 I/O and storage problem seems to present an insuperable constraint upon *ab initio* calculations. Although integral storage limits to an N^2 problem for very large molecules, one does not see this behaviour in practice except for extended systems, such as a long polymer chain. Even then the number of integrals remains prohibitive for this effect to become appreciable. Clearly a radical rethink of the conventional approach is required to permit the routine examination of molecular systems comprising, say, more than 30 heavy atoms. The novel direct-SCF algorithm due to Almlof [14] seems to provide a viable alternative, and we have implemented such techniques within the GAMESS package.

The philosophy of direct-SCF is not to store the integrals, but to calculate them as required on each SCF iteration, thus eliminating virtually all I/O from the SCF, at the expense of increased CPU requirements. The only constraint upon the dimension of a calculation, N, is the amount of CPU time available, and the dimension of real machine memory (presently $3N^2$ words, plus program source). An obvious prerequisite to an efficient implementation of this algorithm is a highly optimised integral package. The importance of this is readily demonstrated by comparing the iteration times of GAMESS (169 s) and the original program due to Almlof (576 s) in a small 91 basis function test calculation on the nitrobenzene molecule run on the FPS-164.

Two sets of calculations performed by us on the FPS-164 at Daresbury illustrate the applicability of the method to large systems, and suggest that extensive calculations may be conducted given a suitably configured, dedicated local facility, thus obviating the need for time-shared access on a large supercomputer facility (e.g. a Cray or multi-AP system).

Firstly, we have optimised, in a 300 function, STO-3G basis set, a sixty atom icosahedral carbon cluster (Buckminsterfullerene or footballene [29]), using direct-SCF and analytic gradients. This cluster is of current experimental interest and has been the subject of several semi-empirical calculations [30]. Our resulting geometry is in agreement with that of reference [30b]. The direct-SCF used 980 s per iteration, while evaluation of the one- and two-electron derivative integrals required 7500 s. Storage of the symmetry unique integrals in a conventional calculation would have required about 0.2 GByte of disk space. More significant is our 540 basis function split-valence 3-21G calculation upon the same cluster. The direct-SCF used 4500 s per iteration. A conventional treatment would have

Table 9. A computational physics and chemistry benchmark

		CPU times (seconds)			
Code	Subject area	FPS-164	FPS-264	Cray-1S	Time ratio FPS-164/Cray
GAMESS		1006	301	116	8.7
MULTI	Computational	2691	765	345	7.8
DISCO	chemistry, CCP1	1127		208	5.4
MOPAC		481	178	120	4.0
MDTEST	Molecular Dynamics CCP5	502	132	43	11.6
CASCADE	Lattice defects, CCP5	287	84	29	10.0
LMTO	Electronic structure of solids, CCP9	482		134	3.6

filled our present disk system (1.8 GBytes), and produced an iteration time of 2400 s (approximately 2000 s I/O transfer time plus 800 s CPU partially overlapped with the I/O).

5. Cost effectiveness of the X64 scientific computers

In the preceding sections we have outlined the implementation and performance of various computational chemistry codes. In the process of evaluating the performance of a computer, the ultimate criterion is centred on its ability to handle production work. In the present section we compare the performance of 7 codes taken from the various subject areas supported by the Collaborative Computational Projects (CCPs), these codes being in routine use on the Cray-1S at ULCC and on the FPS-164 at Daresbury. The timings of Table 9 refer to typical production jobs (see Appendix I).

Note that all CPU timings presented above refer to the FPS-164 itself, and were obtained prior to the FPS-164/MAX upgrade. The impact of this upgrade on the benchmark will be reported at a later date.

6. Parallel processing and computational chemistry

"Parallel processing" is exhibited in various ways in the present generation of scientific computers. Array processors use several arithmetic elements (adders, multipliers, arithmetic-logic units) to increase performance. Vector processors combine the arithmetic elements of array processors with pipelining techniques and high-speed electronic components for the hundreds of Mflops performance claimed for these machines. One attractive way to improve performance is to make use of parallel processing by embedding specialised co-processors in an existing high-speed machine, with the provision of software support to enable the user to express the parallelism where appropriate. This is exemplified in the FPS/164-MAX system, where the Matrix Algebra Accelerator (MAX) modules are integrated into the established FPS-164 architecture through memory mapping, and are directed by the CPU to perform any one of a fixed set of linear algebra operations. The MMO based algorithms (Sect. 4) of quantum chemistry are well placed to take advantage of such co-processors.

Before discussing our own experiences in the parallel implementation of quantum chemistry software in Sect. 7, we briefly outline the options available in parallel architectures, and consider the appropriateness of each from the standpoint of computational chemistry.

The phrase "parallel processing" will hereafter be used to refer to the use of multiple processors to speed up the execution of a single program, in contrast to methods designed to improve processing of a random job load (statistical parallelism, multiple streams of independent jobs; systolic parallelism, or pipelined subprograms). In assessing the ease of migrating any application code from a sequential to a parallel environment, we need consider the following:

(i) Can the computation be parallelised asynchronously in "large sections", i.e. at least at the subroutine level, where the parallel part involves decomposing the domain of the problem, and letting each processor work on a different part of the problem. This domain decomposition is "coarse-grain" parallelism, and would appear vital in realising the full potential of any multiprocessor system. Compilers will not be able to recognise this type of parallelism, at least in the near future, and intuitively coarse-grain parallelism will act to minimise communications between parallel processors.

(ii) How frequent and time consuming are the necessary communications between parallel processes? What is the operating system overhead in initiating such communications?

(iii) How evenly can the work be distributed between any number of processors? Such an even distribution is clearly required in achieving maximum efficiency.

(iv) How crucial is the architectural feature of each processor being connected to a common (or shared) memory?

6.1. *Extremes in multiprocessor philosophy*

In spite of the performance increases achieved through pipelining and vectorisation, supercomputers are reaching the limits of their capabilities. However sophisticated its design and however fast its components, a single processor supercomputer eventually reaches limits imposed by fundamental electrical properties – switching speeds and propagation delays.

The answer to improving supercomputer performance lies in concurrent architectures. Concurrency is a high level or global form of parallelism, denoting independent operation on a collection of simultaneous computing activities. A concurrent MIMD (multiple instruction-multiple data) machine thus uses loosely coupled, multiple, interacting processors to perform many operations at once. Concurrency contrasts with other forms of parallelism, such as pipelining and multiple functional elements. These forms imply some form of lock-step control, which ultimately limits the expandability and performance of a system. Concurrency allows expansion to a larger number of processors because of the flexibility afforded by distributed memory, distributed control and loose coupling.

Two of the approaches being taken to implementing parallel supercomputers are considered below.

6.2. Multiple vector or attached processors, MVAP

MVAP architectures are obtained by coupling two or more standard supercomputers (or attached processors) together, typically to a common fast memory, as has been done by Cray Research with the X-MP series. Cost considerations, together with the use of shared memory in these and similar systems, ultimately limit the number of processors that can be connected: communication overhead and Amdahl's law suggests a limit of 16 co-operating processors.

Partridge and Bauschlicher [6b] have considered on a two-CPU Cray X-MP the multiprocessor implementation of algorithms for (a) sparse symmetric matrix-vector product, (b) four index integral transformation, and (c) calculation of diatomic two-electron Slater integrals. They demonstrated the considerable degree of parallelism inherent in present algorithms that can be readily exploited, but suggest "considerable algorithmic development will be required for some steps (MCSCF and CI) to reduce network traffic, particularly on non-shared memory architectures".

The most impressive and comprehensive investigation of MVAP parallelism in computational chemistry is provided by the on-going experimental parallel supercomputer system, called LCAP (loosely coupled array of processors), developed by Clementi and co-workers at IBM Kingston, New York [6]. The initial LCAP system, LCAP-1, consisted of 10 FPS-164 scientific computers (seven attached to an IBM 4381 host and three to an IBM 4341), the FPS-164 processors being coupled to the IBM hosts through standard IBM 3 Mbyte/s channels. Numerous examples taken from quantum mechanics, molecular dynamics and Monte Carlo have demonstrated the high degree of parallelism obtainable in all three disciplines [31, 32]. Our own experience in adapting quantum chemistry codes to the corresponding LCAP system at the IBM Scientific Centre in Rome, Italy, is considered in Sect. 7 below.

The LCAP system at Kingston is continually being upgraded in response to the experiments conducted thereon [32]. The IBM 4381 and 4341 hosts have been replaced by an IBM 3081 and 4383, while each FPS-164 has been upgraded to an FPS-164/MAX-2. Attempts to increase the flexibility of LCAP include the provision of a bus connecting the APs and five shared bulk memories, forming two rings for further communication from AP to AP. The need for such extensions provides a further reminder that applications of either limited granularity or requiring heavy AP- to AP-data transfer may require considerable algorithmic development to be viable on non-shared memory architectures.

Workers at Kingston have recently instigated work on a second configuration, LCAP-2, comprising 10 FPS-264 processors using an IBM 3081 as front end processor. We await with interest a comparison of the performance of LCAP-2 with, say, a Cray X-MP/48.

6.3. Concurrent micro-processors, CMP

A more practical approach to implementing supercomputer capabilities uses a large number of today's most economical computational element, the microcomputer, made feasible by the rapid enhancements in performance and cost advances achieved through VLSI processors. Numerous university research programs have focused on micro-processor based approaches to large-scale computing e.g. the Cosmic Cube at MIT [33], the V2/64 at Waterloo [34], DADO and VFPP at Columbia [35, 36] and the MMCE at Carnegie Mellon. When the concepts of distributed memory, distributed control and connected networks are used with concurrent computing architecture, there are few practical limits to the number of processors that can be linked to form a supercomputer system. Considering the application of such CMP systems to quantum chemistry codes, we immediately find the following requirements:

(i) 64-bit floating point capabilities, with vector arithmetic pipelines available on each of the processors.

(ii) If, in the interests of scalability to a large number of processing nodes, we are to forsake use of shared memory, and thus migrate to distributed memory systems based on a limited form of interconnection, we clearly need to quantify at the outset the minimum high-speed store requirements of each node. Bearing in mind the crucial role of memory in achieving both vectorisation and efficient I/O performance, it is clear that an inadequate high-speed node memory may well prove the Achilles' heel in adapting quantum chemistry codes to CMP architectures.

(iii) MIMD architectures that use message passing rather than shared storage for communications between the nodes are alleged to provide increased efficiency compared to memory-sharing schemes. The choice of communication protocols by which messages and data are exchanged will clearly affect the performance of a concurrent algorithm: intuitively we require asynchronous message passing capabilities, for use of synchronous protocols would almost certainly lead to significantly slower performance.

(iv) Each node should be able to directly access its own disk storage. Provision of, say, 1 GByte of disk-store/node would undoubtedly act to lessen the need for very-high message passing capabilities between component nodes.

(v) A crucial consideration in assessing available options is that of cost. Most of the supercomputers available today (from manufacturers such as Cray, CDC, etc.) lay in the $5 million to $15 million price range. The ultimate aim of a CMP based system must be to bring the computer-power associated with the more traditional supercomputer architecture into the price range affordable by a single (university) group. In the search for cost-effectiveness, we shall, rather arbitrarily, confine our attention to systems costing 10% of this figure, i.e. in the \$0.5 million to \$1.5 million range.

(vi) The software investment to date in FORTRAN-based codes on both scalar and vector machines strongly suggests that FORTRAN remains the application programming language. While the definition of concurrent processing problems may be more easily accomplished through alternative languages, this would lead to massive conversion requirements and a total lack of continuity with past, present and future serial and MVAP machines. These are unacceptably high prices to pay.

(vii) If configuring a CMP system with the mode attributes outlined above proves cost prohibitive, then downgrading some of the nodes in heterogeneous fashion would still leave QC applications well placed to take advantage of the techniques developed in an MVAP framework.

6.3.1. Available CMP systems. Two commercially available systems promise to satisfy many of the requirements outlined above, the FPS Tesseract from Floating Point Systems, and the iPSC-VX from Intel. Both systems are based on the binary n-cube interconnect scheme developed by Seitz and Fox [33, 37]. The advantages of the hypercube topology include:

(a) It provides the option to expand to larger, more powerful systems as needs increase or VLSI-component technology improves. Both the FPS and Intel implementations provide the potential for open-ended architectures in contrast to other approaches, such as shared memory and buses, which are limited in the extent to which they can be expanded.
(b) The hypercube offers high communications efficiency and communications capabilities that closely match the needs of real problems.

Returning to the original criterion of price, we find that both the iPSC-VX/2^6 (peak performance of 424 Mflops (64-bit) or 1280 Mflop (32-bit)) and the FPS T-40 (peak performance of 512 Mflop for 64-bit arithmetic) lie comfortably within our suggested price range. Both offer some 50% of peak Cray X-MP/48 performance. Such comparisons based on peak rates are, of course, fatuous and it remains to be seen whether the tremendous potential offered by such CMP architectures can be translated into reality by the computational chemist.

7. A parallel implementation of quantum chemistry codes

In the following sections we describe our experience in adapting code to run on an MVAP system, outlining the implementation of four important steps in a quantum-chemical calculation (integral evaluation, SCF, 4-index transformation and direct-CI) on the parallel environment at the IBM European Center for Scientific and Engineering Computing (ECSEC) in Rome [38].

In Sect. 7.1 we outline the pertinent features of both the available hardware and software characterising the parallel environment, and in Sects. 7.2 and 7.3 provide an outline of the implementation of the four above mentioned steps. We try to assess the efficiency of parallelism for each step, and report some provisional timings, showing the amount of overhead to be expected. An attempt is then made to relate our experiences to that likely to be found on CMP architectures.

7.1. The parallel environment at ECSEC

The ECSEC facility consists of an IBM 4383 host computer, running under VM-CMS, and 10 attached FPS-164 processors each with 1 MWord of memory and 0.6 GByte of disk space. No communication is possible between the attached processors, so all communication between the FPS machines has to go through the IBM host, effectively in serial mode. The communication between IBM and FPS proceeds through 2 MByte/s I/O-channels.

Two different software environments are provided.

(a) The VM-EPEX approach, developed at the IBM T. J. Watson Laboratory in Yorktown [39]. This environment allows various virtual machines to run concurrently and to communicate and share data via a shared memory. Each virtual machine has also private memory and private disks. Since each may attach an FPS-164 processor, "real" parallelism may be obtained by migrating the work-load to the AP. The VM-EPEX system features shared memory and may thus be considered to be an approximation to a multi-processor system like the Cray X-MP, though the latter allows disks to be shared as well.

VM-EPEX offers a preprocessor, with parallel FORTRAN extensions under the control of directives such as:

@DO	Divide a DO-loop over the processes,
@serial begin	Defines a part of the code to be executed by one process only,
@barrier	Synchronises all processes,

as well as access to shared memory (@shared/Block/) and to the number of processes and the identification number of each process.

When no control is exerted all processes run in parallel the same code, but not necessarily the same data, yielding data-driven parallel processing. The shared memory is not protected against a simultaneous update of the same memory location by two processes, so in parallel adding into the same variable in shared memory may produce unpredictable results. Therefore the user has to guard against this, and the adding of the results from the various processes has to be performed by a single processor, i.e. serially. The scatter of data, e.g. at the end of a sort, may be performed in parallel though, since one may be sure that each memory location is only accessed once. Note that since shared memory is only available on the IBM host, which was in our case mostly running on 1 processor, no real parallelism is involved in the manipulation of shared memory anyway.

(b) The VMFACS [40] approach developed at Kingston, is more directly geared towards a LCAP setup. One Master virtual machine can divide the work over various slave virtual machines, that can attach in their turn an FPS-164. The IBM slaves may in fact be transparent to the user, so one may call the attached processors directly as slaves in the Master program. Here a preprocessor is available to translate instructions like:

C$ EXECUTE ON ...	Execute a subroutine on either a specific slave or all slaves, with the results collected again on the master.
C$ WAIT FOR ...	Synchronise all or certain processes
C$ SLIN ...	Allow for data-communication between master
C$ SLOUT ...	and slave in a way very similar to the FPS-APEX
C$ SLIO ...	IBM-FPS communication.

and one has access to the number of slaves and each slave can access its own identification.

In the master–slave environment, in contrast to VM-EPEX, the default is to run in serial mode only on the master and specific instructions are required to run in parallel. Since the master–slave communication may be "directly" between attached processor and host and the serial processing on the IBM is built in, this approach seems the more natural in a LCAP environment.

In the practical parallelisation of a program, however, there seems to be no difference in actual strategy. Since we want to be prepared for shared memory machines like the Cray X-MP, the VM-EPEX approach was chosen as the main implementation tool. Various aspects of host-AP control are presented in Appendix II.

7.2. Implementation strategy

In all stages of the quantum-chemical calculation three phases may be distinguished in the parallel implementation:

Phase 1. The workload is divided among the processes in either an arbitrary way (using @DO's) or in some carefully predefined manner. Each process produces its own (intermediate) result file which contains sufficient information to label the data in a unique way.

In the integral evaluation and the calculation stages of the 4-index transformation these intermediate files contain labelled integrals. Here since a major part of the calculation is actually performed in Phase 1, dividing the workload may be left to a simple scheduler, with the sole constraint being an equi-partition of the load over the available processors. In Direct-CI the division of the workload is defined by the calculation of the symbolic interaction between model-configurations. This is, in general, not a time consuming step and must be partitioned exactly to match the Fock-building parts of the integral sorter, and must therefore be divided in a rigorously deterministic way.

Phase 2. The second phase involves a data-driven calculation, which typically requires little change to the original sequential code. Each process goes over its own file and no coordination is required. Examples are (a) the SCF calculation, where a (partial) Fock-matrix is formed out of the density matrix and the partial

2-electron integral file, (b) the sorting stages of the 4-index transformation and, (c) the H-matrix-vector product in the Direct-CI employing the partial symbolic file.

Phase 3. The final phase is to synchronise the processes and gather and analyse the results. The adding and diagonalising of the Fock matrix, the final stage of the 4-index sort and the adding and analysing of the matrix-vector product in the Direct-CI fall into this category. This phase runs almost entirely in serial mode, because it is run on the host IBM, a significant portion involving the addition of the results of the various processes and analysing the result on the host.

7.3. Practical considerations

7.3.1. Integrals + SCF (Fig. 1). In order to provide an even distribution of work the first two DO-loops over shell blocks in the two-electron integral code were collapsed into a single loop over shell triangles. Using an @DO for this combined loop divides the integral calculation effectively over the processes, yielding partial integral files. Then the SCF calculation may proceed with the independent construction of the n partial-Fock matrices. The only overhead involved is the cost of adding the partial matrices, proportional to $n^2 \times np$, where n is the AO dimension and np is the number of processes, whereas the Fock matrix diagonalisation (n^3) is at present performed serially by one process only. Since the work divided is proportional to n^4 (the extra overhead and host-AP I/O amount only to n^2), and the data-driven phase is performed repeatedly, Hartree–Fock calculations prove to be particularly well suited to MVAP parallel processing.

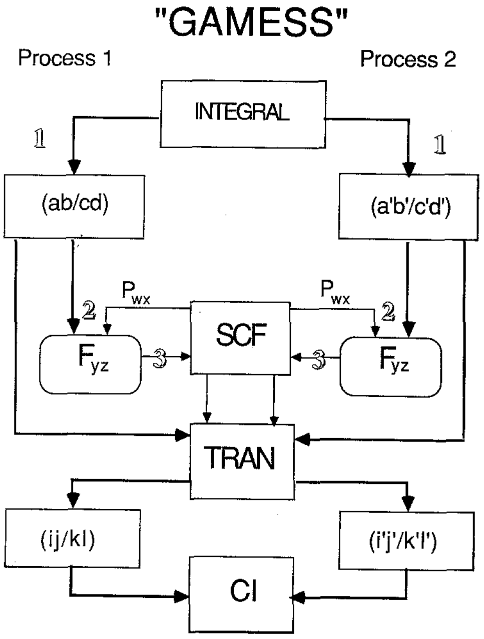

Fig. 1. Dataflow for GAMESS (integrals + SCF)

7.3.2. Integral transformation (Fig. 2). In the bucket sort, reading the partial integral file and writing the buckets to the sort file are the only parts running data-driven parallel (Phase 2). Then follows the reading of the "backchain" which, since we need to gather all integrals, has to be transported in serial mode to the host. This yields a significant overhead of the order of n^4 I/O operations and significant synchronisation overhead. When all integrals needed are in core the work may be divided (using @DO) over the processes, requiring an additional n^4 overhead in transporting the integrals back from host to APs. Since the work performed in parallel is of the order of n^5 and the I/O overhead and serial part is proportional to n^4, the 4-index may be expected to be a substantial bottleneck in the calculations, requiring very large dimensions to make the parallelisation gain prevail over the I/O overhead.

This problem may be completely removed if a shared disk is provided, as on the Cray X-MP, or efficient communication between the processors is available. In the former case one may divide the backchains over the processes, requiring negligible data transfer (e.g. only the starting block positions) and yielding completely parallel processing of the entire 4-index transformation without any overhead.

7.3.3. Direct-CI (Fig. 3). Here again the sorting is performed only partially in parallel again requiring host-AP I/O. However, this sort is usually not a substantial part of a CI calculation so the overhead stays moderate. The main time-consuming task in the CI calculation is the H-matrix-vector product which is performed using raw integrals and symbolic matrix elements. This part is highly vectorised

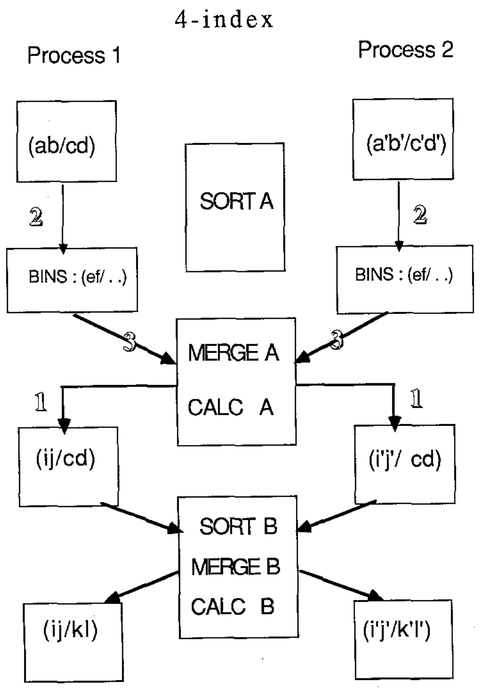

Fig. 2. Dataflow in the 4-index transformation

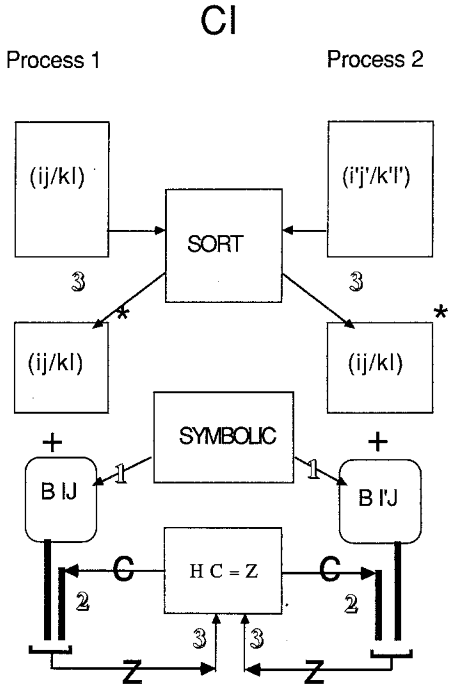

Fig. 3. Dataflow in direct CI

and almost completely driven from the symbolic matrix element list. Therefore this list generation was divided over the processes in a deterministic way. Since here the data-driven part is performed repeatedly and is by far the most time consuming part, with the overhead involving only the addition of the CI vector, the Direct-CI should be reasonably well adapted.

*7.3.4. **Preliminary results.*** We present in Table 10 the timings for a simulated parallel run on formic acid [41] using the GAMESS program. If one adopts the time of each step as the measure, the four processes are seen to run about three times as fast as the single processor case, whereas the overhead in CPU is only about 10%. In running the integrals + SCF really parallel on two FPS processors the elapsed time was ca. 1560 s compared to 2735 s on one processor, substantially more than the CPU times (cf. Table 10) showing that the IBM at the Rome ECSEC Center was too heavily loaded to provide realistic or competitive timings.

The amount of code requiring modification amounted to some 2%–5% of the total, usually only a few lines in any subroutine requiring change. A few routines needed significant modification, e.g. the builder of the diagonal of the Hamiltonian matrix in Direct assumed all diagonal elements to be present which is not true anymore on each process when running in parallel.

7.4. Role of parallelism in quantum chemistry

In the preceding sections we have outlined the current status and prospects for computational chemistry on both MVAP and CMP architectures. A point that

Table 10. A multi-processor benchmark. Total CPU times (seconds) for an SCF+SDCI calculation on formic acid (see text)

	Single processor[b]		Multiple IBM virtual processes				
	FPS	IBM-4383	1	2	3	4	max
2e integrals	330	338	71	137	70	69	137
SCF	385	325	127	183	94	73	183
4 index	324	913	230	239	226	231	239
Direct-CI	440	1593	605	422	359	327	605
Total		3176	1049	999	767	717	1164
Integral blocks[a]							
AO integrals		2246	430	959	485	377	
Transformed integrals		1861	478	497	455	435	

[a] 1 Block = 340 integrals + labels
[b] Timings from the current version of GAMESS on the FPS-164/MAX-3 are as follows: Integrals: 320 s; SCF: 78 s; 4-index: 172 s; Direct-CI: 340 s

should be stressed is that multi-processing on the coarse-grain level provides the obvious route to removing the inevitable scalar-bound portions of vector codes. While we would be very reluctant to embark on serious code implementation on any system without vector features on each node, it is undoubtedly true that for many disciplines multi-processing holds the promise of far greater performance than the present generation of vector supercomputers.

Experience on MVAP architectures has clearly shown the essential role of coarse-grain parallelism in achieving performance. While we may be accused of naivety in hoping to carry such techniques directly into the CMP regime, nevertheless an insistence on the adequate provision of both high speed memory on each node and high communication rates between nodes would seem crucial in deciding on the suitability of a specific CMP architecture. While the methods of computational chemistry lend themselves to MIMD parallel treatment, the present implementations of post Hartree-Fock methods point strongly towards shared memory architectures.

It is perhaps worth trying to put into context the advent of multi-processor capabilities in the light of advances made possible by vector machines. To date developments in some areas of quantum chemistry have kept pace with the advances in VP machine architecture. Thus the size of problem amenable to study by both MCSCF and CI techniques has increased by between two and three orders of magnitude in the last decade, a figure comparable to the increase in machine performance witnessed over the same period. The major impact of VPs has, however, been not so much the ability to perform state-of-the-art calculations, but more the ability to decrease response time to the demands of our experimental colleagues. This situation is especially true in an industrial environment.

There remains, however, the dilemma of the centralised VP. However prestigious a research group, it is never likely to control more than, say, one or two hours a day on such machines. The same situation will almost certainly apply on the present and future generation of MVAP machines. While the achievements of Clementi and co-workers on the LCAP architecture are extremely impressive, one should not lose sight of the obvious cost of such an installation. Based on peak rates alone, the cost/performance ratio for CMP systems would suggest an order of magnitude improvement over that realised by current supercomputer facilities.

An increasing requirement in the application of QC techniques to the field of computer-aided molecular modelling is that of the interactive work-station with high-speed graphics, typified by the Evans and Sutherland PS300. Such stations presently rely on machines such as the VAX-11/780 - the impact of CMP machines on personal workstations cannot be overestimated. It is difficult to visualise the availability of such facilities for an external user community on a centralised MVAP machine, given the current state of networking.

In an MVAP environment one needs to assess the cost-effectiveness of employing statistical parallelism (multiple streams of independent jobs) as against permitting a single job to control all machine resources. As demonstrated by Bauschlicher [6b], the communication overhead on an X-MP/24 is less than 10% on typical QC applications: if a specific application requires all of memory, it is in the interest of efficient machine utilisation for that job to use the full CPU power of all processors. One situation where there exists a clear requirement for parallelism is when a certain arbitrary resource is in short supply. Examples may be that the disk-space on one processor of an LCAP system is not sufficient to store the integrals for a given calculation. It is then essential to spread the job, and the integrals over various processors.

8. Summary

We have outlined the present use and impact of FPS-X64 scientific computers in computational chemistry, focusing attention on performance obtained on an FPS-164. There are several aspects of the present work which perhaps set it apart from that conducted at other FPS-164 installations:

(i) The machine is judged in terms of its performance against, not a VAX 11-780 or a superminicomputer, but against machines such as the Cray-1 and Cyber-205.

(ii) The user environment, involving both in-house and networked access at Daresbury, provides a far more demanding test of the machine than the more traditional "single user, single program" framework.

Experience to date suggests that the strength of the X64 products lies in the final CPU performance obtained, the various comparisons performed herein depicting factors for the FPS-164 of between 1/10 and 1/4 Cray-1S performance over a wide range of codes. The weakness of the mini+AP setup lies perhaps in the host-resident software, and the adverse effects of the rather primitive operating system on the AP.

Nevertheless, the question as to whether the mini+AP setup provides a viable alternative to time-shared supercomputer access would, at least in computational chemistry, appears to be answered affirmative.

In assessing the migration of computational chemistry codes to a parallel environment, we have considered the merits of both MVAP and CMP architectures. Our own experience in implementing Hartree-Fock and Direct-CI codes on a parallel MVAP environment suggest that the bottleneck to an efficient implementation may reside in the integral transformation step, at least on non-shared memory architectures.

MVAP systems imply a centralised facility, and the obvious shortcomings associated with such an environment, e.g. inflexible machine management, networking, etc. Based solely on peak performance, CMP architectures promise to realise the full potential of *ab initio* techniques in a real-time workstation environment for the non-computer motivated chemist. The role of the computational chemist in developing this scenario is crucial.

Acknowledgments. MFG and RJH wish to thank FPS (UK) for all their assistance provided to date. We thank F. Tarantelli (Perugia) for much help with IBM, FPS and the ECSEC environment, and S. Paternello (ECSEC) for valuable advice and special IBM subroutines. We are indebted to ECSEC for financial support. JHvL and LCHvC also thank IBM Holland (Dr. Ir. F. G. Insinger) for such support. We thank D. E. A. Pozzi for editing the manuscript.

Appendix I

A computational chemistry and physics benchmark (see Table 9)

Code	Calculation
GAMESS	2nd-order CI calculation on H_2O (TZVP basis; 53 937 csf)
MULTI	CASSCF calculation on TiH_4 (17 945 csf)
DISCO	Direct-SCF calculation on $HCONH_2$ (DZ basis)
MOPAC	Partial geometry optimisation of anthraquinone (no symmetry)
MDTEST	MD simulation of liquid argon (5000 time steps of 108 particle system)
CASCADE	Energy minimisation of Na+ defect in quartz
LMTO	Self consistent energy for hcp gadolinium

Appendix II

Using the APs under APEX64

The actual programming of and communication with the APs is performed by APEX64. The APEX64 system controls communication between the host and APs for parallel use. It enables the user to initialise and request a number of APs and release them again after use. This "real" parallel use of the APs was only achieved for the integral and SCF modules of GAMESS. Within the framework described above (Sect. 7.3), altering subroutines that were already running in simulated parallel mode was straightforward. For instance, the integral routine, consisting of a large nested loop structure, with an @DO as outer loop is divided into three parts, each calling a separate AP routine.

(i) The first routine call initialises the AP and transfers data from the host.
(ii) Within the @DO-loop host-computation of the integrals is replaced by a call to corresponding AP routines. The workload is distributed by the @DO loop, using the loop index as calling parameter.
(iii) Finally, after completion of the integral calculation on all processors, a third routine returns some data to the host, though the partial integral files are left on the AP. (Using only one AP routine with several entry points proved to be impossible.)

Using this construction one must be careful to use SAVE statements to preserve the status of the processors on transferring control between host and APs. Such statements must be introduced in all subroutines in which the SAVE'd parameters or common blocks appear. Failure to do so produces error diagnostics from APLINK.

Although it was not strictly necessary, the APs were attached for the total duration of the job.

Timings were difficult to obtain, but compared to the simulated parallel jobs the work load appeared to be far less evenly distributed. When running a large job and/or using a larger number of APs (say 8) unpredictable results were obtained. Roll-in/roll-out problems were encountered on several of the APs, leading to premature job abortion. On some occasions the results obtained were clearly in error. Insufficient time was available to investigate these problems further.

References

1. Killmon P (1983) Computer Design 22:167
2. a. Shepard R, Bair RA, Eades RA, Wagner AF, Davis MJ, Harding CB, Dunning TH Jr (1983) Int J Quant Chem 17; Bair RA, Dunning TH Jr (1984) J Comp Chem 5:44
   b. Bair RA (1984) FPS-164 matrix multiplication subroutine guide. Argonne National Lab
   c. Dunning TH Jr, Bair RA (1984) Advanced theories and computational approaches to the electronic structure of molecules, pl., NATO ASI Series. Reidel, Dordrecht Boston London
3. a. Guest MF, Wilson S (1981) in: Proc American Chem Soc Meeting, Las Vegas, August 1980. Wiley-Interscience, New York
   b. Saunders VR, Guest MF (1982) Comput Phys Commun 26:389
   c. Guest MF (1985) In: Supercomputer simulations in chemistry. Montreal
4. Saunders VR, van Lenthe JH (1983) Mol Phys 48:923
5. Ahlrichs R, Bohm HJ, Ehrnardt C, Scharf P, Schiffer H, Lischka H, Schindler M (1984) 6th Seminar on Computational Methods in Quantum Chemistry 31; Rohmer MM, Bernard M (1985) ASTERIX - a quantum chemistry package vectorised for the Cray-1. In: Supercomputer simulations in chemistry. Montreal
6. a. For recent applications in parallelism see the work of E. Clementi and co-workers, Department 48B, Kingston, NY
   b. Partridge H, Bauschlicher C (1985) Algorithms vs Architectures for Computational Chemistry. In: Austin Conference on Algorithms, Architectures and the Future of Scientific Computation, Austin
7. Clementi E, Corongiu G, Detrich JH (1985) Comput Phys Commun 37:287

8. Guest MF (1986) In: Computational physics and chemistry on an FPS-164/MAX, CCP/86/1
9. IMSL Program Library (1978)
10. NAG Fortran Library, Numerical Algorithms Group Ltd, 1984
11. Beppu Y (1982) Computers and chemistry 6
12. Stewart JP (1984) MOPAC - A general molecular orbital package. QCPE 455
13. Saunders VR, Guest MF (1976) ATMOL3, RL-76-000; Guest MF, Saunders VR (1974) Mol Phys 28:819
14. Almlof J et al. (1982) J Comp Chem 3:385; Almlof J, Taylor PJ (1984) In: Advanced theories and computational approaches to the electronic structure of molecules, p 107 NATO ASI Series. Reidel, Dordrecht
15. Dupuis M, Rys J, King HF (1976) J Chem Phys 65:111; King HF, Dupuis M (1976) J Comput Phys 21:144
16. McMurchie LE, Davidson ER (1978) J Comput Phys 26:218
17. Roothaan CCJ (1960) Rev Mod Phys 32:179
18. Yoshimine M (1969) IBM Technical Report, RJ-555, San Jose, USA
19. Siegbahn PEM (1980) J Chem Phys 72:1647
20. a. Werner H-J, Knowles PJ (1985) J Chem Phys 82:5053 and references therein
    b. Knowles PJ, Werner H-J (1985) Chem Phys Lett 115:259
21. Saunders VR (1985) VAMP on the Cyber-205
22. Pulay P (1969) Mol Phys 17:197; (1970) 18:473; (1971) 21:329
23. Dupuis M, Spangler D, Wendolowski J (1980) NRCC software catalog, vol 1, program no. QG01 (GAMESS); Guest MF, Kendrick J (1986) GAMESS user manual. An introductory guide, CCP1/86/1 Daresbury Laboratory
24. Pulay P (1982) J Comp Chem 3:556
25. Schlegel HB (1982) J Chem Phys 77:3676
26. Bell S, Crighton JS (1984) J Chem Phys 80:2464
27. Cerjan CJ, Miller WH (1981) J Chem Phys 75:2800
28. Guest MF, Hillier IH, Vincent M, Rosi M (1986) JCS Chem Commun: 438
29. Kroto HW, Heath JR, O'Brien SC, Curl RF, Smalley RE (1985) Nature 318:162
30. a. Haymet ADJ (1986) Chem Phys Lett 108:421; ibid, (1986) J Chem Soc 108:319; Haddon RC, Brus LE, Raghavachari K (1986) Chem Phys Lett 125:459
    b. Disch R, Schulman JM (1986) Chem Phys Lett 125:465
31. Corongiu G, Detrich JH (1985) IBM J Res Develop 29(4):422
32. Clementi E (1985) J Phys Chem 89:4426
33. Seitz CL (1985) The cosmic cube. Comm of the ACM 28-1:22
34. Ostlund NS, Whiteside RA (1985) A machine architecture for molecular dynamics: The systolic loop, Annals of the New York Academy of Science
35. Stolfo SJ, Shaw DE (1982) DADO: A tree-structured machine for production systems. AAAI 82. Carnegie Mellon University
36. Christ HN, Terrano AE (1984) A very fast parallel processor, IEEE Transactions on Computing C-33-4:344
37. Fox CG, Otto SW (1984) Algorithms for concurrent processors. Physics Today 37-5:50
38. The work described herein was carried out during two periods at ECSEC, initially by MFG and then JHvL and LCHvC, each stay of four weeks in duration. Unfortunately we have not been able to polish the work or obtain more definitive benchmarks as it remains impossible to use the ECSEC facility remotely from either the UK or the Netherlands
39. Darema-Rogers F, George DA, Norton VA, Pfister GF (1985) A VM parallel environment IBM research report RC11225. Yorktown Heights, NY
40. Chin S, Domingo L, Caltabiano R, Carnevali A, Detrich J (1985) Parallel computation on the loosely coupled array of processors: A guide to the precompiler. IBM Corporation, Kingston, NY
41. van Lenthe JH (1985) Supercomputer 5:33