Journal of Computer-Aided Molecular Design, 5 (1991) 171–185
ESCOM

J-CAMD 116

The implementation of ab initio quantum chemistry calculations on transputers

M.D. Cooper* and I.H. Hillier

Department of Chemistry, University of Manchester, Manchester M13 9PL, U.K.

Received 10 August 1990
Accepted 23 October 1990

Key words: Parallel processing; Transputer; Hartree–Fock calculations; GAMESS

SUMMARY

The RHF and geometry optimization sections of the ab initio quantum chemistry code, GAMESS, have been optimized for a network of parallel microprocessors, Inmos T800-20 transputers, using both indirect and direct SCF techniques. The results indicate great scope for implementation of such codes on small parallel computer systems, very high efficiencies having been achieved, particularly in the cases of direct SCF and geometry optimization with large basis sets.

The work, although performed upon one particular parallel system, the Meiko Computing Surface, is applicable to a wide range of parallel systems with both shared and distributed memory.

INTRODUCTION

The value of ab initio molecular orbital calculations in modelling many aspects of electronic structure having both academic and industrial importance is now well established. Such calculations at the restricted Hartree–Fock (RHF) level are now largely routine and have been implemented in many quantum chemistry packages (e.g. GAUSSIAN, GAMESS, CADPAC) available on a variety of computer hardware. There is, however, a constant search for more cost-effective ways of carrying out these computationally intensive studies driven, to a large extent, by the need to study an increasing range of large molecular systems.

During the last few years improvements in VLSI technology have resulted in the appearance of relatively inexpensive, fast microprocessors which, combined with cheap memories and discs, have led to the appearance of super-workstations capable of performing calculations that would previously not have been possible without access to large mainframe computers. With the arrival of such powerful microprocessors a major change has occurred in the design of small computer

* Present address: Department of Computer Science, University of Manchester, Manchester M13 9PL, U.K.

0920-654X/$ 5.00

systems with the introduction of computers which exploit microprocessor parallelism. Such systems offer, with the investment of appropriate time and effort in the programming, enormous amounts of processing power at relatively low cost and in the form of systems which need minimal maintenance and consume very little power.

These systems allow the introduction of processor parallelism to old and large codes with the minimum amount of recoding by replacing the serial versions of those parts of the program that can benefit most from the parallelism, usually the most time-consuming sections, by parallel versions which run on many processors. The rest of the code remains in the original, serial form running on a single 'master' processor with the remainder of the processors inactive during the serial sections of the code. The parallel or slave processors are initialised at the start of the parallel cycles with any data that they may require and are then left to perform their part of the parallel cycle. When the parallel section is completed the results can be collected back to the master processor which may or may not have participated in the parallel section. While this will not appeal to the parallel processing purist, who may feel that the only way to get the full benefit from distributed parallel processing is to rewrite the code completely in a parallel form, with careful selection of those sections to be parallelised excellent performance improvement can be achieved by this method of 'stubs' with the minimum amount of code being re-written.

In this paper we describe an implementation of closed-shell RHF calculations, including geometry optimization, on one particular parallel system, the Meiko Computing Surface [1], which comprises a parallel network of microprocessors, the Inmos T800-20 transputer [2]. We have access to a small Computing Surface, containing 5 T800-20 transputers, in house and to a somewhat larger system, with up to 24 transputers, in the Manchester Computer Centre (MCC). While we have based the implementation upon a particular quantum chemistry code, GAMESS [3], the techniques we have used and the conclusions reached are probably applicable to most serial ab initio codes.

AB INITIO RHF SCF CALCULATIONS

The time-consuming steps in carrying out RHF SCF calculations are well-known, being the evaluation of the two-electron integrals (often in supermatrix form) and the repeated construction of the Fock matrix (**G**). In the work reported herein, we use mainly two molecular examples; nitrobenzene in a 3-21G basis (91 basis functions) and morphine in an STO-3G basis (124 basis functions).

We have investigated the parallelisation of two techniques for carrying out the SCF calculation. In the conventional method (filed SCF) the two-electron integrals are stored (in memory or on disc) and are used at each cycle of the SCF procedure to construct **G**. In the direct SCF method [4], the two-electron integrals are not stored but are calculated at each cycle of the SCF stage, thus drastically reducing the storage needed but increasing the amount of computer time required. In the RHF calculation of nitrobenzene on an FPS M64/60, using the conventional SCF method, the calculation of the two-electron integrals and the **G** matrix construction accounted for 87% of the total elapsed time, this figure rising to 98% for the morphine calculation. Clearly, for calculations of any size, the best results will be achieved by the introduction of efficient parallelism to these two sections of the GAMESS code.

We now describe some general features of the implementation of the GAMESS code on the Meiko Computing Surface.

THE HARNESS AROUND THE FORTRAN

Meiko Fortran is not capable of being run on a 'naked' transputer and so Fortran programs are compiled as library subroutines to be called from within an occam program, a 'harness', which also executes concurrent processes to provide the run-time environment for the Fortran. The Fortran program performs all communication with the external environment via a set of four occam channels (for screen, keyboard and file input and output) which are defined within the occam harness and then passed to the Fortran as parameters of the subroutine call. Extra channels, defined by the programmer, can also be passed into the Fortran to allow explicit communication with other sequential processes executing either concurrently or in parallel with the Fortran.

In our implementation the serial sections of the code are performed by a complete version of the GAMESS code, running on a 'master' processor. This Fortran was originally to be supported by a relatively simple harness, written in occam2, which would allow access to the host discs. It soon became apparent, however, that the access times for the files on the host discs were excessive due to the very slow connection between the computing surface and the host machine, a μVAX II. While this was not significant for the command and output text files, the GAMESS random access work-files are heavily used throughout the calculation and so the rate at which they can be accessed is crucial. This problem was solved by the inclusion of an extra processor in the network which supports a large amount of memory and also supports hardware to allow access to a hard disc unit. The standard Meiko 'mass-store' board, which has a single T800 transputer, 8 Mbytes of RAM and a controller to perform DMA access to the hard disc via a SCSI bus, was installed together with a 300-Mbyte hard disc unit. The mass-store board runs a second program, written entirely in occam2, which configures the hard disc as a device for the storage of a maximum of about 65636 4-kbyte blocks of the two-electron integrals file. The rest of the files, being very much smaller, are all stored entirely in memory on the mass-store board. The removal of the file access delays to the host discs makes a significant difference to the time taken for the conventional SCF calculations as can be seen from Table 1. Using the fact that the integrals file is always accessed sequentially and only either for reading or writing on a single pass, it is possible to introduce a set of buffers to allow the pre-fetching and post-saving of blocks of integrals from/to the hard disc unit. In this way the delays associated with the fetch of blocks from the hard disc can be completely removed. The disc accesses are then performed at the same speed as if the entire file were held in memory on the storage processor. Some example timings, for an RHF calculation on $(VS_4)^{3-}$ performed by various serial versions of GAMESS, are shown in Table 1.

TABLE 1
TIMES[a] FOR RHF ENERGY EVALUATION ON VS_4^{3-} (55 BASIS FUNCTIONS)

	μVAX II		T800 transputer using host discs	T800 transputer, mass-store and disc	T800, mass-store, and buffered disc
	CPU	Wall			
2-electron integrals	1622	1708	985	525	524
SCF cycles (32)	1220	1723	11507	710	457
Total	2973	3594	12826	1295	1042

[a]All are elapsed times except for the μVAX where both elapsed and CPU times are given.

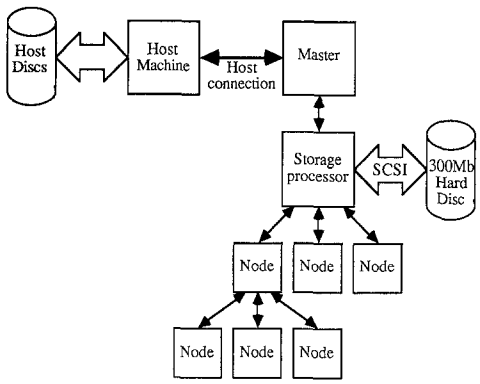


Fig. 1. Hardware structure for the parallel version of GAMESS.

The form of the parallelised version of GAMESS will, therefore, be a master processor which runs a full version of the GAMESS code, a storage processor which supports all work-file accesses via RAM disc and hard disc and a network of slave processors that are only active during the parallel sections of the code. The fact that the network of slaves requires access to the storage processor for saving and reading blocks of two-electron integrals determines that the tree structure should be as shallow as possible in order to reduce the number of inter-processor links between the slaves and the storage processor. This fact, combined with the restriction that the transputer has only four links, forces the network of slaves to be a ternary tree below the storage processor as shown in Fig. 1.

Each of the slave processors (the nodes) contains a Fortran program built up from the sections of code taken from the serial version together with routines to perform the initialization of any variables using data received from the master at the start of the parallel section. The 'sub-GAMESS' Fortran program is supported by an occam2 harness to handle all communication required throughout the tree of processors. The harness is made up of many concurrent processes which allow the receipt and transmission of messages, from both the Fortran and the superior and subordinate nodes, with the minimum of interruption of execution of the Fortran program. A schematic diagram of the main processes on a node is shown in Figs. 2A and B.

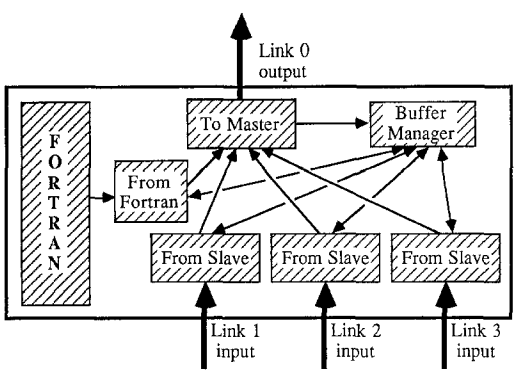


Fig. 2A. Major processes involved in upward message passing through a node.

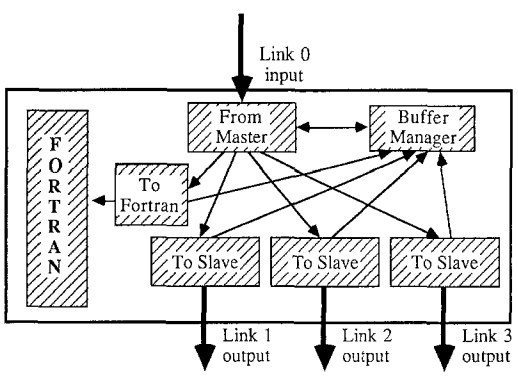


Fig. 2B. Major processes involved in downward message passing through a node.

The node has separate processes to handle receipt and transmission from/to each of the links and from/to the Fortran. The processes use a different set of buffers for messages being passed up and down the tree, and there are enough buffers for a message to be in progress in both directions along every link at once. In this way the delays associated with the passing of messages through the network and their interference with the execution of the Fortran program on each node of the tree can be kept to a minimum.

In order to efficiently parallelize the integral file generation and the **G** building, the tree of processors must be able to access the integrals file on the hard disc with the minimum delay. To achieve this, parallel access to the disc buffers was made possible by the introduction of a buffer manager and separate receivers and senders for each of the three branches of the tree, below the storage processor, and for the master processor (which is active during the parallel cycles). The form of the major processes on the storage processor, during the parallel cycles, is then as shown in Fig. 3.

With this set of programs, every processor in the network has access to the integrals file with the minimum of delay; each block or request for a block from the tree is passed through the individual nodes to one of the slave controllers on the storage processor where it is dealt with as appropriate by either placing the block into one of the manager's buffers, for later saving to disc, or by the sending back of the next available block to the requesting slave. Using this harness, then, we can now consider the parallelization of the individual sections of the GAMESS Fortran.

TWO-ELECTRON INTEGRAL GENERATION IN PARALLEL

The two-electron integrals are generated by two separate sets of routines in the GAMESS code. The first generates 'low accuracy' integrals by the rotated axes algorithm [5] and is usually used for those integrals involving only s and p functions. The second uses the Gauss–Rys method [6] in order to generate 'high accuracy' integrals and is usually only used where the integral involves one

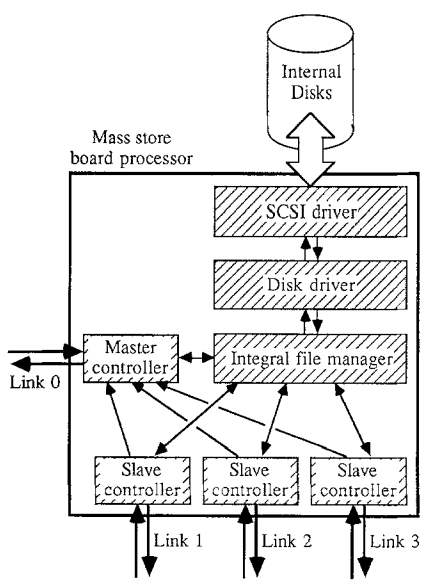


Fig. 3. Major processes active on the storage processor during parallel execution.

or more d functions. Both integral generation routines are based on four nested loops over the indices of the four Gaussian contractions in the integral. To avoid generating each of the unique integrals more than once the loops have the form shown in Fig. 4.

The integrals are stored, together with their associated indices, in blocks of fixed size of 4-kbytes at a density of 340 integrals per block. Since the indices are incorporated in the block, the orderings of the blocks and of the integrals within each block are not significant and can be changed as desired. During the generation of the integrals from each index set, the integrals are stored in RAM until there are enough to fill a block at which time it is appended to the integrals file. There are many ways in which the integral generation can be split between the individual processors by allowing each processor to range over particular sets of the indices. The solution used is shown, schematically, in Fig. 5. Here INITIAL is an integer, unique to the processor, which is it's number in the parallel network and NPROC is the number of compute processors active during the parallel cycles. Thus, on each processor, the set of indices {I, J, K, L}, is ignored unless that processor's counter is zero after decrementing. If the counter is zero then it is reset and all the integrals that can be produced from the index set are generated and stored. Thus the m^{th} processor in the parallel network of n processors will perform all integrals associated with the m^{th}, $(m+n)^{th}$, $(m+2n)^{th}$,... index sets produced by the loops.

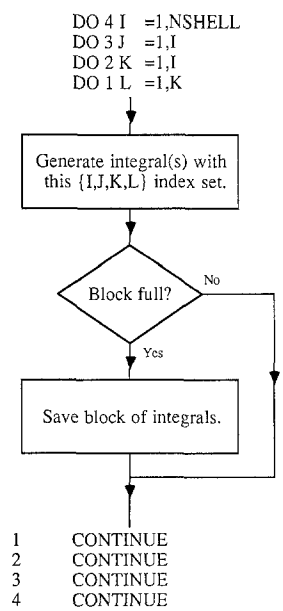


Fig. 4. Loop structure for integral generation.

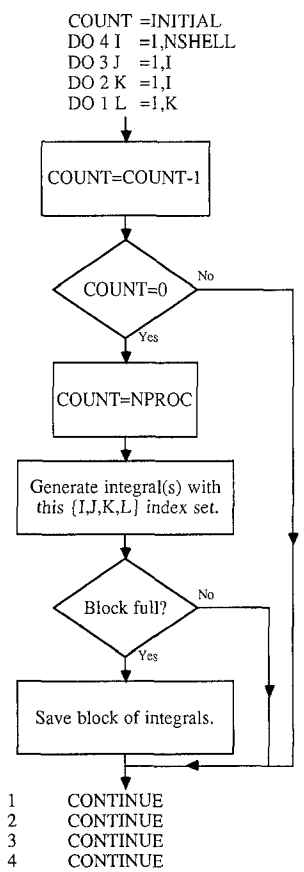


Fig. 5. Loop structure for parallel integral generation.

While splitting the integrals between the processors in this way ensures that $\frac{1}{n}$th of the possible index sets will be considered by each processor, the amount of calculation associated with each of the index sets is not the same due to the mixture of s, p and d functions present in the basis set. While this must result in some imbalance in the load on each of the processors it should not be significant since the number of index sets is usually very much larger than the number of processors and the sets containing each function are scattered between processors. In fact very little evidence of such load imbalance was observed, with the processors finishing within a very few seconds of one another in all of the calculations which have been performed.

The storage of integral file blocks occurs in the same way for each individual parallel processor as in the serial case, with a block being sent for storage as soon as the processor has generated enough integrals to fill it. The Fortran simply sends the block into one of the buffers associated with the harness before returning to the generation of further integrals. The harness then passes the block back, via the tree structure, to the storage processor where it is stored as the next block of the integrals file. Since the link transfer of the block is performed in parallel with other CPU activities, the Fortran program is only delayed for the duration of the transfer of the block to the harness via an occam channel. This is effectively a RAM copy and so degrades the speed of the Fortran much less than passing it directly to the link.

Since the load balancing is performed purely on the basis of the number of computational tasks to be performed, it is important that the master processor should operate as efficiently as the rest of the nodes in the network. In order for it to do this there must be a similar buffering system between the master Fortran and its link to the storage processor. While this will result in some small delay to all the file reads performed during the serial sections of the code, this is negligible compared with the delay which would occur in the parallel integral generation, as the slave processors finished their tasks before the master, and may even be compensated for by the removal of the delays in the writes to files from the serial code sections.

PARALLEL CONSTRUCTION OF THE TWO-ELECTRON CONTRIBUTION TO THE FOCK MATRIX

G, the two-electron contribution to the Fock matrix, is constructed by reading the integrals from the file and, after multiplying by the appropriate elements of the density matrix, adding the integrals into the appropriate elements of the Fock matrix. The integrals file is read and processed sequentially until the terminal block, a block containing a zero integral count, is received. A schematic diagram for this is shown in Fig. 6.

Since the individual elements of the Fock matrix are independent of one another and the integrals are just added into it, the process of **G** matrix construction can be split between processors by simply allowing each individual processor to request blocks of integrals and use them to form a 'sub-matrix' from the integrals contained in whichever blocks it receives. When the file is exhausted, the sub-matrices can then be summed on the master processor to form the final version. Thus the serial task, of order N^4 where N is the number of basis functions, of the matrix construction is converted to n tasks, each of order $\frac{N^4}{n}$, performed in parallel and n serial tasks, of order N^2, the matrix summation.

Each of the parallel processors contains a version of the code for the processing of the integrals file very similar to that in the serial version with the file reads replaced by a request for the next

block of integrals. This request is passed, via the occam harness, to the storage processor which then sends the next available block of the integrals file to the requesting processor. Since the actions of the harness are concurrent with those of the Fortran it is possible to separate the two Fortran actions, block request and receipt, and allow the Fortran to 'pre-request' the next block of integrals as soon as the last has been received. In this way the delays associated with the fetching of the next block can be kept to a minimum. The schematic flow is then as shown in Fig. 7.

In the case of the slave processors, the requested integral block arrives on the processor in one of the buffers belonging to the harness and is then passed, when required, into the Fortran via an occam channel. Since the blocks are transferred into the buffer, via a transputer link, in parallel with the Fortran's processing of the last block, the delays associated with obtaining the next integral block should be greatly reduced. As in the case of the integral file generation, the master processor must be buffered between the link to the storage processor and the input channel to the Fortran process in order to remove the link-transfer delay in block requests. This increases the time taken to perform each file access during the serial sections of the code but is more than compensated for by the improvement in the speed of operation of the master processor during the parallel operation.

RESULTS FOR CONVENTIONAL SCF PROCEDURE

The timings for the parallel version of the GAMESS code, for various transputer tree structures of up to four parallel processors, performing an SCF calculation on nitrobenzene are shown in Tables 2 and 3. In all cases the file handling was performed by a dedicated storage processor having a 300-Mbyte hard disc for two-electron integral storage and 7 Mbytes of RAM for storage of

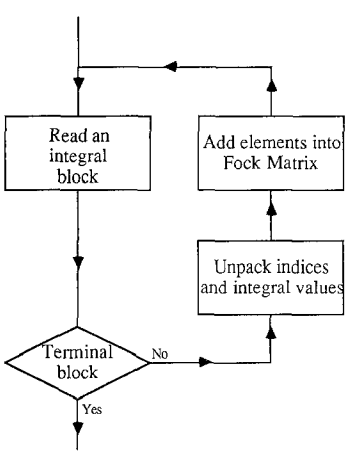


Fig. 6. Flow during serial **G** construction.

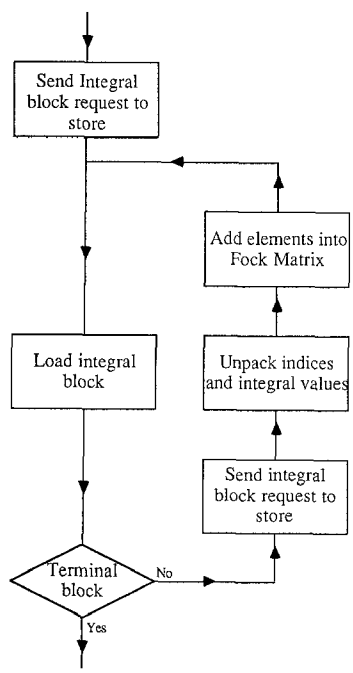


Fig. 7. Flow for each compute processor during parallel **G** construction.

TABLE 2
ELAPSED TIMES FOR NITROBENZENE (3-21G) RHF CALCULATION USING A SINGLE COMPUTE PROCESSOR AND MASS-STORE BOARD WITH NO BUFFERED SAVING OR PRE-REQUESTING OF INTEGRAL FILE BLOCKS FROM/TO THE HARD DISC UNIT

	Supermatrix	Non-supermatrix
Two-electron integrals	837	679
G matrix construction (1 cycle)	97	178
Total for calculation (23 cycles)	3951	5657

all other files. In the diagrams of Table 3 [M] denotes the master processor, [S] the dedicated storage processor, and [N] the various nodes of the tree structure each of which supports a slave version of the Fortran. Here the timings given are for the generation of the integrals file, an average for the generation of the **G** matrix and the total time for the run. The percentage figures are measures of the parallel efficiency of the code as given by the following equation:

$$\text{Efficiency} = \frac{T_s}{T_n \times n} \tag{1}$$

where T_n is the time taken to perform the section when using n compute processors operating in parallel and T_s is the time for the serial version of the code. The first percentage figure has T_s for a completely serial version of the code with no buffered saving of blocks and no pre-request of blocks. The timings for this serial version are shown in Table 2. The second percentage figure has T_s taken from the first structure shown of the parallel version (in Table 3.1) which, while it has only the master processor active, does perform buffered saving and pre-requesting of blocks. The efficiencies of greater than 100% in the first column of each of the Tables are due to the inclusion of block pre-fetch by the compute processors. Those in the second column are more difficult to explain since each of the processors should be operating with equal speed. In fact there is an imbalance in the speed at which the different Fortran programs are executed with the larger, master program processing the integral file blocks at a noticeably slower rate than the smaller slave programs. This can only be put down to a difference in the way that the programs operate with the smaller program using either registers or the on-chip memory more effectively than the larger one.

These timings show that the parallelisation of the integral generation has been very successful with an almost linear decrease in the time taken as the number of processors is increased. They also show that the occam harness is operating efficiently in transporting the blocks of integrals to the storage processor since there is virtually no variation in the time required as the shape of the tree is changed to increase it's depth. In the case of the **G** matrix construction, however, while the improvement is excellent as the first two slave processors are added, no further improvement is achieved from the third. An examination of the delays in the block retrieval time has shown that the average delay associated with the loading of the next block of integrals by each of the compute processors, rises significantly when the second slave is added and is roughly doubled again with the addition of the third. Thus, all benefit from the third slave processor is lost due to increased delays in block reception. These increases are due to the slow transfer rate of blocks from the hard disc to the storage processor. They occur despite the use of pre-loading of blocks to the buffers, as described in the section on the harness, and so can only be overcome by the replacement of the

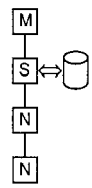
TABLE 3
ELAPSED TIMES AND EFFICIENCIES FOR NITROBENZENE (3-21G) BY FILED RHF ENERGY EVALUATION USING VARIOUS TREE STRUCTURES

	secs	eff.%	eff.%
Integrals	810	103	-
G build	58	167	-
Total run	3035	130	-

Table (3.1a) Supermatrix

	secs	eff.%	eff.%
Integrals	659	103	-
G build	150	119	-
Total run	5000	113	-

Table (3.1b) Non-supermatrix

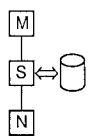


	secs	eff.%	eff.%
Integrals	278	100	97
G build	27	120	72
Total run	1782	74	57

Table (3.4a) Supermatrix

	secs	eff.%	eff.%
Integrals	233	97	94
G build	50	119	100
Total run	2275	83	73

Table (3.4b) Non-supermatrix

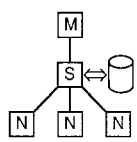


	secs	eff.%	eff.%
Integrals	409	102	99
G build	27	180	107
Total run	1909	104	79

Table (3.2a) Supermatrix

	secs	eff.%	eff.%
Integrals	340	100	97
G build	74	120	101
Total run	2938	96	85

Table (3.2b) Non-supermatrix

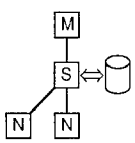


	secs	eff.%	eff.%
Integrals	211	99	96
G build	27	90	54
Total run	1717	58	44

Table (3.5a) Supermatrix

	secs	eff.%	eff.%
Integrals	177	96	93
G build	37	120	101
Total run	1933	73	65

Table (3.5b) Non-supermatrix

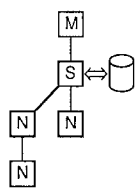


	secs	eff.%	eff.%
Integrals	280	99	96
G build	27	120	72
Total run	1782	74	57

Table (3.3a) Supermatrix

	secs	eff.%	eff.%
Integrals	233	97	94
G build	49	121	102
Total run	2267	83	74

Table (3.3b) Non-supermatrix

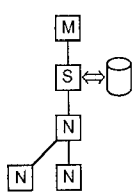


	secs	eff.%	eff.%
Integrals	209	100	97
G build	27	90	54
Total run	1713	58	44

Table (3.6a) Supermatrix

	secs	eff.%	eff.%
Integrals	175	97	94
G build	37	120	101
Total run	1935	73	65

Table (3.6b) Non-supermatrix

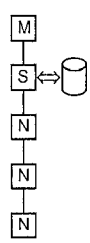


	secs	eff.%	eff.%
Integrals	210	100	96
G build	27	90	54
Total run	1718	57	44

Table (3.7a) Supermatrix

	secs	eff.%	eff.%
Integrals	176	96	94
G build	38	117	99
Total run	1943	73	64

Table (3.7b) Non-supermatrix



	secs	eff.%	eff.%
Integrals	208	101	97
G build	27	90	54
Total run	1716	58	44

Table (3.8a) Supermatrix

	secs	eff.%	eff.%
Integrals	174	98	95
G build	38	117	99
Total run	1943	73	64

Table (3.8b) Non-supermatrix

disc by a faster (and more expensive) disc unit. In the case of the non-supermatrix calculation the speed up remains almost linear for larger numbers of processors than in the supermatrix case but, ultimately, the same problem will occur. Extrapolating the non-supermatrix times to larger

numbers of processors indicates that the disc-I/O rate will be exceeded when seven or eight T800s are used with a fastest time of about 19 s per **G** matrix construction cycle. This time is proportionally smaller than the fastest supermatrix time due to the smaller integrals file produced when non-supermatrix integrals are used.

While the disc access times have placed a limit on the number of processors that can be applied to the **G** matrix construction section of the code, the improvement over the serial version of the code is still quite good and the 5-processor version (including the storage processor) is very much faster than the μVAX II system hosting the computing surface. The elapsed time for the μVAX II to perform the same calculation, as a single task, is 16138 s — more than nine times as long as the fastest transputer network shown.

The I/O problem we have identified can only be solved by either the addition of faster discs or a number of separate, autonomous discs such that no disc unit becomes saturated by the available compute processors. In a system where CPU power is so cheap, however, an alternative strategy is to increase the number of processors and perform the **G** construction by the use of direct SCF, an entirely CPU-limited process, which should prove at least as efficient as the integral generation has been found to be. The storage processor can still be used for the other GAMESS files, accessed during the serial sections of the code, but the hard disc and its controller are no longer required.

DIRECT RHF MATRIX BUILDING

The use of direct SCF, with the two-electron integrals re-generated each time they are required, has a great advantage over filed SCF in that it avoids the need for large amounts of disc space for the integrals file and so allows the performance of calculations of any size provided that the matrices (density matrix and Fock matrix) can be held in memory. The disadvantage is that the amount of CPU time required to generate **G** at each cycle is sharply increased which, despite the removal of the delays associated with file access, will increase the overall time for the calculation. For multi-tasking machines where the I/O and CPU activity for the various active tasks can be interleaved, the switch to direct SCF results in a drastic reduction in task throughput but on single-task machines, such as transputer networks, the file-delays are very significant and so the increase in the elapsed time should be less dramatic.

Direct SCF is a facility already present in the GAMESS code using the algorithm described by Almlöf et al. [4]. The integral routines simply call procedures to add the integrals into **G** rather than those which save them to disc and so parallel direct SCF is a relatively simple extension to the filed version previously described. The parallel routines generate the same subset of integrals as in normal SCF and build a partial **G** matrix from them. These partial matrices are then summed onto the main processor, as for filed SCF **G** building. Results for this parallel version are given in Tables 4A and B. The percentages are the parallel efficiencies, as given by Eq. 1, relative to the time for a single processor as also given in the tables.

These results show that the **G** matrix construction by direct SCF has parallelised extremely well with the efficiency of that section decreasing only gradually as the number of processors is increased. The decrease is due to the set up and matrix summation times for the network and to the small amount of code that is duplicated on each processor, during the parallel section; these become more significant as the total time for the section is decreased. The decrease in efficiency is

TABLE 4A
ELAPSED TIMES FOR DIRECT SCF CALCULATION OF NITROBENZENE IN A 3-21G BASIS

	No. of T800s								
	1	4		8		12		16	
	s	s	%	s	%	s	%	s	%
G generation	13812	3504	99	1875	92	1327	87	1072	81
Rest of SCF	848	848	–	848	–	848	–	848	–
Total for run	14731	4424	83	2794	66	2246	55	1991	46

TABLE 4B
ELAPSED TIMES FOR DIRECT SCF CALCULATION OF MORPHINE IN AN STO-3G BASIS

	No. of T800s								
	1	4		8		12		16	
	s	s	%	s	%	s	%	s	%
G generation	77101	19430	99	9900	97	6678	96	5144	94
Rest of SCF	1684	1684	–	1684	–	1684	–	1684	–
Total for run	79391	21659	92	12129	82	8907	74	7373	67

more pronounced in the total time for the calculation due to the larger amount of serial code present in the matrix manipulations applied to the Fock matrix. The efficiency of the total time for the calculation improves with the number of basis functions due to the fact that the duration of the **G** build operation is proportional to N^4 (N being the number of basis functions) whereas the matrix operations vary with N^3 and so the fraction of the cycle time that is spent in parallel action increases with the number of basis functions. Another factor affecting the efficiency is the nature of the basis functions used. Basis sets, such as STO-3G, where all of the basis functions have the form of a contraction of three Gaussians, will exhibit greater parallel efficiency than a calculation using a 3-21G set since these latter functions have less primitives per contraction than the STO-3G basis. Hence, for the STO-3G basis, the integral evaluation will be a greater part of the total time for the same number of primitive Gaussians.

These data can be extrapolated to larger numbers of processors and, by applying a logarithmic fit to the morphine **G** build times we can say that, for 32 processors, the time for the morphine job should be about 4260 s in total. This gives an overall efficiency of about 58%. The reduction in the parallel efficiency with increased number of processors provides a limit to the number that can be effectively applied to the calculation. The actual number that is applied will be determined by many factors such as required speed and the financial resources available but, ultimately, the procedure cannot be performed faster than the serial time present in the SCF cycles.

GEOMETRY OPTIMIZATION

One of the major uses for the RHF process is the optimization of molecular geometries to find either the minimum-energy structure or the structure that corresponds to the transition state between two minima. GAMESS uses quasi-Newton methods to perform these optimizations using analytical first-order derivatives of the energy and numerically estimated second-order derivatives. For calculations of any size, the most computationally intensive step in this process is the time taken to generate the first-order derivatives of the molecular energy with respect to the nuclear coordinates.

The evaluation of the gradients, the first-order derivatives of the molecular energy, involves, again, the evaluation of a large number of integrals having a similar form to those required in the SCF process. The major difference here is that, even for the largest calculations, the time spent in the evaluation of the derivatives of the one-electron contribution to the energy is not insignificant compared with that spent in the derivatives of the two-electron contribution, as was the case in the RHF process. Thus both the one- and the two-electron derivatives must be parallelised in order to obtain good overall efficiencies in the optimization cycles.

The two-electron integral derivative generation routines have a similar form to that of the two-electron integral routines used in the SCF calculation and so the same techniques can be used to split the individual tasks between the processors as described in the section on parallel generation of two-electron integrals, with the prospect of an even greater degree of success due to the more computationally intensive integrals being evaluated. The one-electron gradients are computed by three separate routines, each of which performs the integrals associated with a particular term in the gradients. One routine generates the terms involving the derivatives of the overlap integrals; these are relatively simple to calculate and occupy only a small portion of the one-electron integral derivative time. This routine was not parallelised since the increase in speed would not be significant after set-up times are taken into account. The remaining two routines, which were parallelised, evaluate the derivatives of the integrals over the one-electron Hamiltonian. These routines involve nested loops over the basis function indices and so similar methods were employed to divide the integral tasks over the parallel processors.

The results for various numbers of processors, performing the same optimizations as described above, are given in Tables 5A and B. These timings demonstrate that this section has parallelised extremely well with very high efficiencies for the morphine calculation. The increase of efficiency in the two-electron integral derivative routines, over that obtained in the SCF integrals, is due to the increased amount of calculation required to evaluate each of the integrals involved. The smaller amount of serial time involved ensures that the efficiency for the optimization step remains high even for quite large numbers of processors. The efficiencies in the one-electron gradients are significantly lower than those for the two-electron gradients, for two reasons. First, only two of the three routines used have been parallelised, and second, the fact that they take far less time to perform than the two-electron means that the set-up time for the network, being similar in both cases, is far more significant. These effects are particularly noticeable in the nitrobenzene calculation where the one-electron gradients take very little time to calculate and so the fixed set-up time for the two parallel cycles soon begins to dominate the total.

We can, again, extrapolate these data to larger numbers of processors by a logarithmic fit to estimate that the time taken for the morphine gradients would be about 440 s when run using 32 T800 transputers which would give an overall efficiency for the cycle of about 82%.

TABLE 5A
ELAPSED TIMES FOR ONE CYCLE OF OPTIMIZATION OF NITROBENZENE IN A 3-21G BASIS

	No. of T800s								
	1	4		8		12		16	
	s	s	%	s	%	s	%	s	%
1-electron deriv.	112	42	67	30	47	26	36	26	27
2-electron deriv.	657	172	95	91	90	65	84	52	79
Cycle time	774	219	88	126	77	96	67	82	59

TABLE 5B
ELAPSED TIMES FOR ONE CYCLE OF OPTIMIZATION OF MORPHINE IN AN STO-3G BASIS

	No. of T800s								
	1	4		8		12		16	
	s	s	%	s	%	s	%	s	%
1-electron deriv.	1633	441	93	241	85	176	77	142	72
2-electron deriv.	9943	2501	99	1266	98	857	97	651	95
Cycle time	11582	2948	98	1512	96	1038	93	799	91

CONCLUSIONS

The project has proved largely successful with a final version of the code that can perform direct RHF optimizations on large molecules with a very high degree of parallel efficiency. The number of transputers that can be used, before the efficiency drops below 50%, increases with the number of basis functions from about 10–12 in the case of nitrobenzene (91 basis functions at SV3-21G) to about 35 for morphine (124 basis functions at STO-3G) and is even higher for larger numbers of basis functions. The current versions of Meiko quad-compute boards, having 4 Mbytes of RAM, can allow a maximum of more than 300 basis functions in the calculation for which we can estimate that over 100 processors could be efficiently applied. To improve the efficiency of the total calculation it would be necessary to reduce the time for the serial sections of code used during the SCF cycles by parallelising the matrix manipulations performed upon the completed Fock matrix. Operations of this type, while well suited to vectorization, are not easily performed on distributed memory parallel machines and rarely give good efficiencies on the quite large numbers of processors being applied here. Thus it is, perhaps, better to concentrate on the easily parallelised sections of the code and wait for a hardware solution to become available such as, perhaps the new Transputer + Intel i860 board which allows the superior processing performance of the i860 processor to be exploited by programs while using the transputer for interprocessor communications.

TABLE 6
TOTAL ELAPSED TIMES FOR MORPHINE BY DIRECT SCF

	16 × T800	μVAX II	FPS M64/60	SG 240GTX	Cray X-MP
G build	5144[a]	~250700	7749	10269	1915
SCF	1684	~5600	82	233	10
Total run	7373	~257000	7888	10575	2027

[a]Parallel code.

While it is important to discuss the code in terms of the method and efficiency of its parallelism, ultimately, the important factor is how well its speed compares with other machines commonly used to perform calculations of this type. The timings shown in Table 6 are totals for an energy evaluation, by direct RHF, run on various widely used single-processor machines compared with 16 T800s. The transputer timings could, of course, be further improved by the addition of more processors.

While the code has been developed on the Meiko Computing Surface using transputers, the basic structure of the parallelism is not restricted to the transputer and the direct RHF and gradient operations should be readily portable to any multi-processor distributed memory system. It should also be possible to port it to shared memory systems provided that the problems of memory-access conflicts can be overcome. The simplest way in which this can be done is by furnishing each shared-memory processor with a separate submatrix to act upon by replacing the Fock-matrix array with a set of arrays of the same size, i.e. by making them operate as a distributed memory system. The implementation of the filed SCF would present greater problems in that parallel accesses to the integrals file which, on the Computing Surface, are performed by the occam2 tree structure, would be difficult to simulate on a shared memory system where no such communication protocol can exist. This is perhaps no great drawback since the intention of this project has been to use the enormous amounts of cheap CPU power now available to free the SCF procedure from the restriction of limited and expensive disc space available for the storage of the two-electron integrals file. This we have achieved making possible the performance of extremely large and time-consuming calculations on small parallel super-workstations.

ACKNOWLEDGEMENTS

We thank Glaxo Group research for support of this project, Mr. A. Grant for his help with the Meiko Computing Surface in the Manchester Computer Centre and Dr. M.F. Guest for helpful discussions.

REFERENCES

1 Meiko technical information, Meiko Scientific, 640 Aztec West, Bristol, U.K.
2 Transputer Reference Manual, published by Prentice-Hall in association with INMOS, 1988.
3 Guest, M.F., GAMESS: User's Guide and Reference Manual, SERC (Daresbury) document, 1989.
4 Almlöf, J., Faegri, K. and Korsell, K., J. Comp. Chem., 3 (1982) 385.
5 Pople, J.A. and Hehre, W.J., J. Comp. Phys., 27 (1978) 161.
6 Dupuis, M., Rys, J. and King, H.F., J. Chem. Phys., 65 (1976) 111.