Theor Chim Acta (1993) 84: 289–299

Theoretica Chimica Acta

An efficient implementation of the direct-SCF algorithm on parallel computer architectures*

Martin Feyereisen and Rick A. Kendall**

Molecular Science Software Group, Theory, Modeling and Simulation Program, Molecular Science Research Center, Pacific Northwest Laboratory, Richland, WA 99352, USA

Received October 1, 1991/Accepted January 14, 1992

Summary. The development and implementation of a parallel direct self consistent field (SCF) Hartree–Fock algorithm, with gradients and random phase approximation solutions is presented. Important details of the structure of the parallel version of DISCO and preliminary results for calculations using the Concurrent Supercomputing Consortium Intel Touchstone Delta parallel computer system are reported. The data show that the algorithms are efficiently parallelized and that throughput of a one processor CRAY X-MP is reached with about 16 nodes on the Delta. The data also indicate sequential code which was not a bottleneck on traditional supercomputers, can become time critical on parallel computers.

Key words: Direct-SCF algorithm – DISCO

1. Introduction

The development of *ab initio* electronic structure computational schemes has been closely coupled with advances in computer hardware. As an example, the Direct-SCF approach was developed when the processing speed of computers surpassed their input and output (I/O) storage abilities [1]. In recent years, computationally intensive portions of programs have been extensively modified to take advantage of the tremendous processing speed possible with vector processing supercomputers [2–4]. Currently, the computing ability of computers constructed of several processing units (parallel computers), each with moderate performance when individually compared to a supercomputer, has surpassed the power of conventional supercomputers consisting of a few very powerful processors. Most existing programs were designed around the concept of serial execution, and thus, perform poorly when simply ported to a parallel machine.

* This work was performed under the auspices of the Office of Basic Energy Sciences, Division of Chemical Sciences, U.S. Department of Energy, under contract DE-AC06-76RLO 1830 for Pacific Northwest Laboratory which is operated by Battelle Memorial Institute for the U.S. Department of Energy.

** *Present address:* Cray Research Inc., 655E Lone Oak Dr. Eagan, MN 55121, USA

Particularly, programs which require I/O often perform poorly on parallel machines, since these machines have limited I/O capabilities when compared to their formidable processing power. This discrepancy between the processing speed of parallel computers and their I/O performance capabilities makes them plausible candidates for exploiting direct methods, where processing requirements are large and I/O requirements are negligible. In the paper, we will discuss our efforts to implement algorithms to take advantage of the capabilities of parallel or distributed computers having hundreds of processors.

2. Implementation

We have implemented parallel versions of computationally intensive sections of the Direct-SCF program DISCO. These subroutines construct matrices from combining the two-electron integrals over a Gaussian basis with other variables (such as one-electron density elements). The computation of these two-electron integrals (Mulliken notation):

$$(ij|kl) = \int\int \varphi_i^*(1)\varphi_j^*(1) \left|\frac{1}{r_{12}}\right| \varphi_k(2)\varphi_l(2)\, d1\, d2 \tag{1}$$

typically require greater than 95% of the time in a direct calculation. Most applications require the entire set of two-electron integrals, which increases by the fourth power of the size of the basis. There are three places where two-electron integrals are required in the program DISCO:

1. the Fock matrix $\boldsymbol{F}$, which for closed-shell systems is given by:

$$F_{ij} = h_{ij} + \sum_{k,l=1}^{N} D_{kl}\{(ij|kl) - \tfrac{1}{2}(ik|jl)\} \tag{2}$$

where $\boldsymbol{h}$ is the one-electron Hamiltonian, $\boldsymbol{D}$ is the one-particle density matrix, and N is the number of basis functions.

2. the nuclear gradient g, defined by:

$$gx_m = \frac{\partial E}{\partial x_i} = \frac{\partial V_{NN}}{\partial x_m} + \sum_{i,j=1}^{N} \left\{D_{ij}\frac{\partial h_{ij}}{\partial x_i} - Q_{ij}\frac{\partial S_{ij}}{\partial x_m}\right\} - \tfrac{1}{2}\sum_{i,j,k,l=1}^{N} D_{ij}D_{kl}\frac{\partial\{(ij|kl) - \tfrac{1}{2}(ik|jl)\}}{\partial x_m} \tag{3}$$

where $\boldsymbol{S}$ is the overlap matrix of the basis, $\boldsymbol{Q}$ is the energy-weighted density matrix, and X_m is the mth Cartesian component of nuclear center X. The derivatives of the two-electron integrals can be expressed in terms of related two-electron integrals.

3. the random phase approximation (RPA) solution vector product $\begin{pmatrix}\boldsymbol{P}\\ \boldsymbol{Q}\end{pmatrix}$, defined by [5]:

$$\begin{aligned} P_{il} &= \sum_{k,j=1}^{N} \{Z_{kj}^{Z}\{2(il|kj) - (ik|jl)\} + Y_{kj}^{c}\{-2(il|kj) + (lk|ji)\}\} \\ Q_{il} &= \sum_{k,j=1}^{N} \{Y_{kj}^{c}\{2(il|kj) - (ik|jl)\} + Z_{kj}^{c}\{-2(il|kj) + (lk|ji)\}\} \end{aligned} \tag{4}$$

where $\begin{pmatrix} \boldsymbol{Z} \\ \boldsymbol{Y} \end{pmatrix}$ is the RPA solution vector. For efficiency, the above quantities are computed in an 'AO-driven' fashion, where once an integral is computed, its contribution to all appropriate matrix elements is considered. All three matrices are linear with respect to the contribution made by a two-electron integral. Therefore, the work of computing the two-electron integrals and combining them with the appropriate matrix elements to form the desired matrix products could be distributed among several processors and the results summed at the finish. It is advantageous that the work required for processing integrals scales as the fourth power in the number of basis functions (N), while the matrices formed are of size N^2 ($\boldsymbol{F}$, $\boldsymbol{P}$, and $\boldsymbol{Q}$) or N ($\boldsymbol{g}$: the nuclear gradient is constructed as a $3N$ quantity to allow for the option of 'floating' basis functions). The large amount of cpu work required to compute the matrix elements with respect to their size (approximately N^2 or N^3) coupled with the ability to independently compute their contributions make it possible to effectively parallelize the construction of $\boldsymbol{F}$, $\begin{pmatrix} \boldsymbol{P} \\ \boldsymbol{Q} \end{pmatrix}$, and $\boldsymbol{g}$.

Two-electron integrals are most often computed a group at a time for computational efficiency. The DISCO program computes integrals in blocks defined by shells of groups, where the shell refers to the set of angular functions and group refers to the functions of same angular momentum having the same nuclear center. As an example, using the 3-21G* basis the groups of integrals (SP|SD) would contain $(3 \times 1) \times (2 \times 3) \times (3 \times 1) \times (1 \times 6)$, or 324 integrals. For a computation involving first row elements with a double-zeta quality basis, the typical integral block would contain approximately 500 integrals. Between 300 to 500 floating point operations (flops) are required to compute each integral.

Integral prescreening is used to determine which integrals are nonzero, and whenever possible, avoid their computation. The density matrix is used to screen integrals at the group level. If the largest density element to be combined with the integral group is greater than a certain threshold, the group is computed and further screened by their radial prefactors [1].

It is possible to construct a 'superatom' basis, where basis functions from different centers could be combined into larger groups [6]. Using superatoms, integral blocks often contain several thousand integrals. This can be very advantageous for computing integrals on a vector-processing computer, where processing speed is proportional to the vector length of the shell components of the integral block. While the processing speed increases substantially using superatoms, the memory requirement for computing integrals also increases, and the benefits of integral prescreening, important for Direct-SCF efficiency, degrade with the use of superatoms. Consequently, computing integrals using superatom blocks is normally only preferred on large memory vector-processing supercomputers.

Table 1 shows pseudocode for the algorithm used to construct the Fock matrix in DISCO. The skeleton Fock matrix is constructed from the 'petite' list [7] then 'symmetrized' to form the full Fock matrix. The algorithms to construct the two-electron portion of the nuclear gradient and the RPA solution product are similar. They require a different petite list than the Fock matrix, and therefore, have different loop restrictions. The same subroutine is used to compute the two-electron integrals in the construction of all three matrices. The derivative integrals have different prefactors than the regular integrals, and these prefactors are added when a flag is set by the calling routine.

Table 1. Pseudocode for serial construction of the Fock matrix

a	Zero $\boldsymbol{F}$
b	Loop over atoms I, J, K, and L
c	Loop over shells i on atom I
d	Loop over shells j_s on atom J
e	Loop over shells k_s on atom K
f	Loop over shells l_s on atom L
g	Loop over symmetry operators $\hat{R}$, $\hat{S}$, and $\hat{T}$
h	Define integral block $(ij\|kl) = (i\hat{R}(j_s)\|\hat{T}(k_s\hat{S}(l_s))$
i	Apply density prescreening, if $D_{\max} < t$ goto k
j	Compute $(ij\|kl)$, Combine $(ij\|kl)$ with $\boldsymbol{D}$ and add to $\boldsymbol{F}$
k	End all loops
m	Symmetrize $\boldsymbol{F}$
n	Add $\boldsymbol{h}_1$ to $\boldsymbol{F}$

Maximum parallel efficiency is achieved when all processors carry out the same amount of work (e.g., a set of load-balanced tasks). There are three commonly used methods for parallelizing the computation of a lsit of unrelated tasks. One approach is to have all processors scan the list and carry out only a portion of the tasks. This is typically done by indexing the tasks, assigning each processor a number and having the processor complete tasks whose processor number is a modulo of the task index. One of many ways to code this modulo counter is:

```
nproc = nnodes( )                    !return number of processes
me = nodeid( )                       !returns process id 0 to nproc – 1
ipcount = (me – 1)                   !initialize counter
do
  ipcount = ipcount + 1              !increment counter
  if(mod(ipcount,nproc).eq.0) then
     "do work"
  endif
enddo
```

Using this approach, all processors are responsible for the same number of tasks. Only when all tasks are about the same size, all processors have the same computational power, and there are a large number of tasks, is the work load balanced. Another approach, called the master/slave topology, assigns one processor (master) to allot tasks to the other processors (slaves). Typically the master will attempt load balancing by assigning new tasks to idle slaves or slaves as they become idle. The third technique, which uses a counter shared among all processes, is a hybrid approach of the first two. The shared counter methodology uses the loop structure of the modulo method. Instead of matching the task id to the process number, the match is made with the shared counter and then the shared counter is incremented. This allows the "master" process to be involved in the "real" work and allows dynamic load balancing. This algorithm requires some sort of multitasking or interrupt handling.

There are advantages and disadvantages to each of these task scheduling techniques. The master/slave topology and the shared counter technique require communication between the master and slaves while the modulo topology does not. If communication time is expensive with respect to the task computation time, the modulo topology is probably favored. On the other hand, if the processors do not have the same power (such as a distributed network of heterogeneous workstations with or without an interactive load), or the task sizes are typically quite large, but vary substantially in size, the master/slave topology or the shared counter approach is probably favored. The two-electron routines in DISCO have been parallelized using the master/slave concept. For each two-electron routine, the master loops over the list of integral blocks and assigns the task of computing an integral block to a particular slave. The slave is then responsible for computing the integrals and their post-processing to form their contribution to the slave's portion of the Fock matrix. Along with other information, slaves receive density prescreening information used to further prescreen integrals based on radial prefactors [1].

The parallelized construction of the Fock matrix will be discussed, but the ideas are the same for the nuclear gradient and RPA vector formation. For the construction of the Fock matrix, in order to minimize communication between processors, each slave constructs its own copy of the Fock matrix from the subset of integral blocks it was requested to compute by the master. Therefore, each slave is required to hold both the entire $\boldsymbol{F}$ and $\boldsymbol{D}$ matrices in its memory. Consequences of this requirement will be addressed in the Discussion section. The pseudocode for the parallelized Fock buildup is shown in Table 2. Note that the modifications required for the master portion are minor. The master must distribute the density matrix to all slaves (a^*). The master must find an idle slave to process a block of integrals (j^*). After all looping has been completed, the partial Fock matrices are summed and returned to the master (l^*). The code required for the slave routine is minimal.

The communication between the master and the slaves, along with starting up and killing the slave processes was carried out using the TCGMSG [8] message passing toolkit. While TCGMSG supports asynchronous communications on some hardware platforms, we chose to only use synchronous calls to maintain code compatibility across a wide range of systems [9]. The required synchronous message passing for some hardware may be removed in future versions of TCGMSG. The compatibility over many systems has the advantage in that development and debugging of the algorithms were conducted on local sun workstations, whereas the results discussed in this paper were carried out remotely on the Intel Touchstone Delta system.

Initial runs on the Delta indicated that communication costs between the master and slaves were degrading performance when only one block of integrals was assigned as a task to the slaves. The program was modified to include ten blocks of integrals per task, effectively cutting the communication time by a factor of ten. This is simply an effective increase in the granularity of the problem. All results given in this paper were carried out using ten blocks of integrals per task. The decrease in communication time by grouping blocks of integrals in tasks is expected to be partially offset by decreased load balancing and the optimum task size will be system and problem dependent. Further studies on optimum task size are planned.

Table 2. Structure for parallel construction of the Fock matrix

MASTER CODE	
a^*	Send $\boldsymbol{D}$ to slaves
b	Loop over atoms I, J, K, and L
c	Loop over shells i on atom I
d	Loop over shells j_s on atom J
e	Loop over shells k_s on atom K
f	Loop over shells l_s on atom L
g	Loop over symmetry operators $\hat{R}$, $\hat{S}$, and $\hat{T}$
h	Define integral block $(ij\|kl) = (i\hat{R}(j_s)\|\hat{T}(k_s\hat{S}(l_s))$
i	Apply density prescreening, if $D_{\max} < t$ goto k
j^*	Find slave to compute $(ij\|kl)$
k	End all loops
l^*	Sum all $\boldsymbol{F}$
m	Symmetrize $\boldsymbol{F}$
n	Add $\boldsymbol{h}_1$ to $\boldsymbol{F}$
SLAVE CODE	
a^*	Receive $\boldsymbol{D}$ from master and zero $\boldsymbol{F}$
j_1^*	receive $(ij\|kl)$ request from master
j_2^*	if requested to quit goto l^*
j_3^*	Compute $(ij\|kl)$, Combine $(ij\|kl)$ with $\boldsymbol{D}$ and add to $\boldsymbol{F}$
j_4^*	goto j_1^*
l^*	Sum $\boldsymbol{F}$ and send to master

3. Results

Initial calculations on the Delta machine were carried out on the molecule imidazole ($C_3N_2H_4$) using the cc-pvdz, aug-cc-pvdz, and cc-pvtz correlation consistent basis sets [10, 11]. The total numbers of contracted basis functions for these calculations were 95, 161, and 235, respectively. All Delta times were obtained in a stand-alone environment. We expect the timings to increase from a multiuser environment due to performance degradation. The time (in seconds) required to construct the Fock matrix using varying numbers of nodes is given in Table 3. For comparison, the time required for the same basis using DISCO on a single processor of a CRAY X-MP is also given. All CRAY timings were obtained a multiuser environment unless otherwise noted. Previous studies have indicated about a 10% increase of time for running DISCO in a multiuser environment on a CRAY versus a dedicated environment.

The timings on the CRAY X-MP reflect running the calculation using superatoms while the timings for the Delta were obtained without using superatoms. The CRAY X-MP is typically three times faster when using superatoms [6], while the Delta would be expected to be a factor of two slower when using superatoms. Calculations previously performed on similar systems with the CRAY X-MP (using its hardware performance monitor) indicate that approximately twice as many operations (ops) are needed to construct the Fock matrix using superatoms as opposed to not using them (due to less effective integral

Table 3. Time[a] to construct Fock matrix for imidazole with different bases on different number of nodes

Nodes	cc-pvdz	aug-cc-pvdz	cc-pvtz
2	442.6	3470.5	11917.1
16	62.1	232.9	797.7
64	16.4	56.9	193.3
90	12.9	41.0	138.6
128	11.0	29.5	99.0
256	9.7	16.5	54.8
320	9.8	14.3	47.4
400	9.1	13.0	42.2
512	9.0	12.0	38.7
(1)X-MP[b]	47.7	223.2	664.6

[a] time in seconds on Intel Delta Machine
[b] Time in seconds on one processor of a CRAY X-MP4-64

prescreening). But on the CRAY X-MP, the longer vector lengths associated with using superatoms increases the performance of the algorithm by about a factor of six, netting a performance increase of a factor of three. On pipelined scalar machines such as the Delta, there is little if any increase in performance using longer vectors, and the additional ops required to construct the Fock matrix using superatoms are not balanced by an effective decrease in execution time. Comparison of timings between the CRAY X-MP and the Delta indicate that about sixteen nodes of the Delta are required for the effective throughput of a CRAY X-MP. Using 512 nodes of the Delta increases throughput for the larger two bases to about 18 times that of one CRAY X-MP processor. Figure 1 shows the speedup curves for the three bases along with the ideal limit of speed using the master/slave parallel topology (i.e., speedup is equal to the number of slaves) and the speedup based on the Amdahl's law model for the cc-pvtz basis:

$$\text{Speedup} = \frac{\text{Time(1 node)}}{\text{Time}(N \text{ nodes})} = \frac{T_s + T_p}{T_s + \dfrac{T_p}{N}} \tag{5}$$

where T_s is the time required for the serial portion of the routine and T_p is the time required for the parallelizable portion. For the cc-pvtz case, T_s was measured to be five seconds. Three seconds are spent in the routine which symmetrizes the 'skeleton' Fock matrix, and the remainder is spent computing expectation values (i.e. two-electron energy). The speedup for the smallest basis (cc-pvdz) is rather poor, reaching only a maximum of about fifty (50) for 512 processors. On the other hand, the speedups for the two larger bases are linear up to about 200 nodes and increase up to a factor of 300 for 512 nodes. The speedup agrees well with the predicted model up to 128 nodes. Above 128 nodes, the actual speedup is significantly less than predicted.

Deficiencies in the speedup model include: (i) an increase in time associated with broadcasting the density matrix to the slave nodes and summing up the Fock contributions from the slaves, which should increase with respect to the number of the nodes; (ii) the time needed to start up the nodes and wait for them

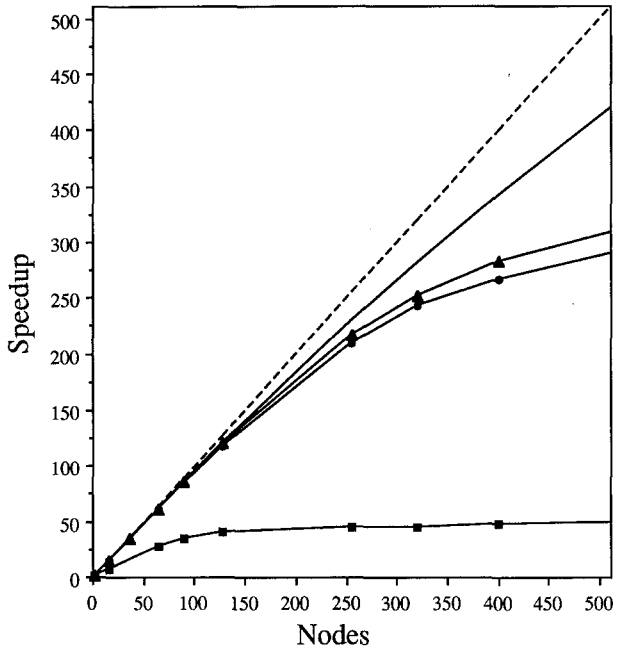

Fig. 1. Speedup curves for constructing Fock matrix of imidazole —■— CCDZ —●— ACCDZ —▲— CCTZ ——— Ideal* ——— Model* (*See text for complete description)

to finish after all tasks have been issued, which should scale linearly with respect to the number of nodes; (iii) inefficiencies in computing from poorer load balancing with the increase of slave processors; (iv) and the partial overlap of the serial time (such as master/slave communication) with the parallel time.

Studies were carried out to investigate some of these factors. Out of the 39 sec used to construct the Fock matrix with 512 nodes, five seconds were required to complete the serial portion of the routine (T_s). Two seconds were required to broadcast the density matrix to the slaves and sum up the Fock matrix (Table 2 master code a* and l*). 24 sec were required for the master to complete the required loops to generate the integral tasks (Table 2 master code b–i). Of the remaining eight seconds used, five were required for the communication calls between the master and slave (Table 2 master code j*), and three were idle time spent by the master waiting for an available slave (Table 2 master code j*). For the actual timings to adhere to Amdahl's law, the sequential time the master spends generating tasks and communicating with slaves would have to completely overlap with the time spent by the slaves doing the parallel work. Only the two seconds required for the broadcast and global sum could not overlap with the parallel time, but these two seconds do not account for the discrepancy between the actual times and the predicted ones. Rather, inefficiencies in the Fock matrix buildup are likely the result of the master not being able to keep all of the slaves busy. This could be removed by either having the master assign larger tasks, or having a sublevel of masters, each assigning tasks to their own pool of slaves.

Table 4 shows the timings and speedups for computing the two-electron portion of the nuclear gradient for the imidazole molecule using the cc-pvtz basis (235 contracted basis functions). The speedup is also given for the model in Eq. (5). For logistical reasons, it was not possible to construct the gradient on only

Table 4. Time[a] to construct two-electron portion of nuclear gradient for imidazole with cc-pvtz basis on different number of nodes.

Nodes	Time	Speedup (Actual)	(Model)[b]
2	112,395.0[c]	1.0	1.0
16	7,494.9	15.0	15.0
36	3,213.0	35.0	35.0
64	1,785.7	62.9	63.0
90	1,264.8	88.9	88.9
128	887.0	126.7	126.7
256	443.6	253.4	253.9
320	356.0	315.7	317.2
400	286.6	392.2	396.3
512	224.1	501.5	506.5

[a] Time in seconds on Intel Delta Machine
[b] See text for complete description
[c] time estimated (see text)

two nodes, so this time was estimated using the model to replicate the actual times for up to 64 processors. Fitting to the model predicted a total time of 112,395 sec with a T_s of 1.95 sec. For these computations, the predictive speedups based on the Amdahl model are highly accurate. The time required to compute the derivative integral blocks' contribution to the two-electron portion of the nuclear gradient is an order of magnitude greater than the associated time required for the Fock matrix. This allows for a better overlap between the serial portion of the master time associated with generating tasks with the parallel time required of the slaves to compute the tasks. Additionally, the slaves are less likely to be idle if they have larger tasks.

Final calculations were carried out on the 38 atom organic molecule ($C_{17}O_3H_{18}$, see Fig. 2) bis(2,5-dimethyl phenyl)carbonate mentioned by Lüthi et al., in their paper on the parallelization of Direct-SCF for shared-memory machines [12]. They were able to decrease the time required to construct the Fock matrix from two hours on one processor of a dedicated CRAY Y-MP/8-128 to 15 minutes using all eight processors in parallel for a speedup of 7.91. For a comparison, the same computation required 500 sec on 256 nodes of the Delta and 356 sec on 512 nodes.

We have tested the parallel implementation of the RPA polarizabilities, but do not have results to discuss in this paper. The expected speedups should be similar to those seen for the construction of the Fock matrix. Performance analysis for the RPA code will be published elsewhere [9].

4. Conclusions

The above implementation requires complete copies of both the Fock matrix (nuclear gradient or RPA vector), and the density matrix (RPA solution vector) in memory on each node. Currently, each node of the Delta has 16 megabytes of

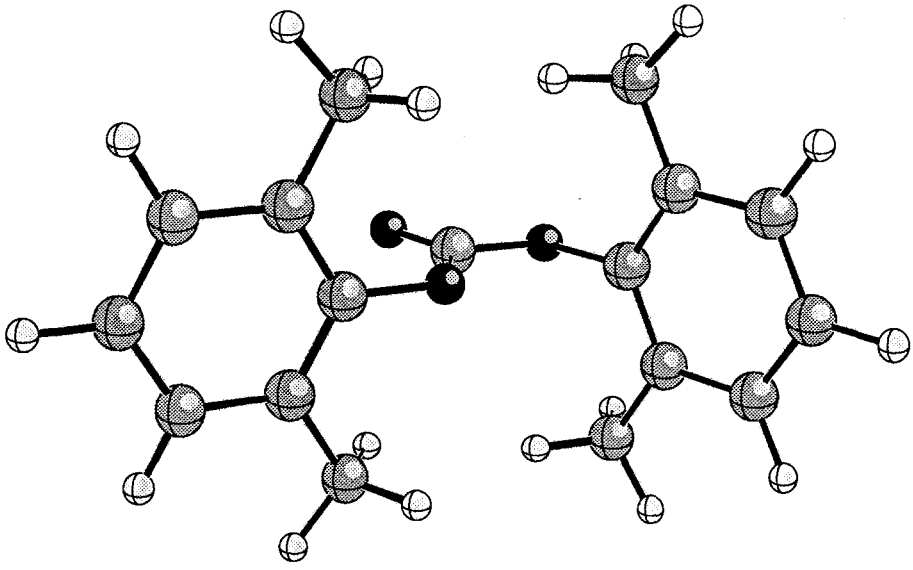

Fig. 2. Bis(2,6-dimethyl phenyl)carbonate

physical memory. This allows us to carry out calculations on systems up to about 400 basis functions. By partitioning ***F*** and ***D*** across several nodes, it would be possible to investigate much larger systems. A drawback of this approach is either the added communication required to send ***F*** and ***D*** elements between nodes, or leaving the AO-driven regime and redundantly compute integrals in a Fock matrix driven manner (i.e. compute all the integrals necessary for the Fock matrix elements on the praticular node).

Our studies indicate that computationally intensive sections of existing programs can be readily modified to take advantage of the processing power available on parallel machines in the regime of a few hundred processors. Currently, the routines we have parallelized are most efficient for studies on larger chemical systems, where the computational costs are severe and use of parallel machines is desirable.

While Amdahl's law oversimplified the speedup for the construction of the Fock matrix, it emphasizes the problem of efficiently using large numbers of processors. The time required to compute the one-electron portion of the nuclear gradient for imidazole with the cc-pvtz basis was 645 sec on the Delta, and has become the bottleneck for computing the nuclear gradient on a large number of processors. Likewise, the total time per SCF iteration was 78 sec, which is only a speedup of 150 for 512 nodes. In order to gain overall efficiency for a large number of processors, large portions of the existing serial code, which on a single processor is computationally insignificant, will have to be modified.

Acknowledgements. This research was performed in part using the Intel Touchstone Delta System of the Concurrent Supercomputing Consortium. Access to this facility was provided by the United States Department of Energy. We would also like to acknowledge Sharon Brunett and Dr. Heidi

Lorenz-Wirzba from Caltech for allowing us early access to the Delta via the "friendly users" program. We would also like to thank Drs Ron Shepard and Robert Harrison for many useful comments on the manuscript.

References

1. Almlöf J, Faegri K, Korsell K (1982) J Comput Chem 3:385
2. Siegbahn PEM (1984) Chem Phys Lett 109:417
3. Olsen J, Roos BO, Jørgensen P, Jensen HFA (1988) J Chem Phys 89:2185
4. Knowles PJ, Werner HJ (1988) Chem Phys Lett 143:514
5. Feyereisen M, Nichols J, Oddershede J, Simons J (1992) J Chem Phys 96:2978
6. Feyereisen M (1990) Univ of Minnesota Ph.D. Thesis 67
7. Dupuis M, King HF (1977) Int J Quantum Chem 11:613
8. Harrison RJ (in press) Int J Quantum Chem
9. Feyereisen M, Kendall RA, Nichols J, Dame D, Golab J (in preparation)
10. Dunning THJr (1989) J Chem Phys 90:1077
11. Kendall RA, Dunning TH Jr, Harrison RJ (accepted J Chem Phys)
12. Lüthi HP, Mertz JE, Feyereisen M, Almlöf J (1992) J Comput Chem 13:160