ELSEVIER

31 January 1997

Chemical Physics Letters 265 (1997) 1-11

CHEMICAL PHYSICS LETTERS

A direct coupled cluster algorithm for massively parallel computers

Rika Kobayashi [a], Alistair P. Rendell [b]

[a] *Daresbury Laboratory, Warrington, Cheshire WA4 4AD, UK*

[b] *Supercomputer Facility, Australian National University, Canberra, ACT 0200, Australia*

Received 11 November 1996

Abstract

A closed-shell coupled cluster program specially designed to run efficiently on massively parallel computers is presented. The input/output bottleneck present in an earlier implementation has been circumvented by solving the coupled cluster equations in a direct manner. Sample calculations on glycine, cytosine and serine have been run on the CRAY T3D and T3E. The results show good scalability up to 256 processors.

1. Introduction

With the increasing power of modern-day computers more large and high-level ab initio calculations have become feasible. It is now possible to calculate energies and gradients for hundreds of atoms with thousands of basis functions at the self-consistent field (SCF) level [1,2], and correlated calculations have been achieved for hundreds of basis functions [3]. Implementations for such large-scale calculations fall into two categories: those targetting shared memory machines (symmetric multiprocessors or SMPs) and those targetting massively parallel processors (MPPs). The strategies required for programming in each environment are very different. At the highly correlated level the most notable work on SMPs is that of Koch and co-workers [4]; they have run single and double excitation coupled cluster (CCSD) calculations with more than 500 basis functions on a CRAY C90 by exploiting its large memory and disk. To our knowledge the only coupled cluster program for MPPs is the molecular orbital (MO)-driven code developed in part by one of us [5,6].

In this Letter we extend the previous work on MPPs and present an atomic orbital (AO)-driven CCSD(T) [7] algorithm for MPPs that has minimal input/output (I/O) requirements. This algorithm has been incorporated into the NWChem program package [8], making extensive use of the Global Array (GA) tools [9]. The GA tools constitute an efficient "shared memory" programming interface that greatly eases the handling and manipulation of distributed data items. Thus, whereas in the previous implementation [6] the double excitation amplitudes, for example, were stored in a distributed fashion and retrieved from remote processors by interrupt messages, they are now placed in a global array and manipulated using calls to the GA library. Encapsulating the complicated machine dependent interrupt programming in a library greatly enhances portability between platforms. Furthermore, an added benefit of using the GA tools is that they provide a variety of higher level functionality, e.g. a parallelised GA matrix multiplication routine.

PII S0009-2614(96)01387-5

Sample calculations using the new code are presented for glycine, cytosine and serine run on a CRAY T3D and CRAY T3E using up to 256 processors.

2. Theory and implementation

The coupled cluster method, in the form it is used today, was introduced to chemistry by Čížek and Paldus [10], and first implemented at the singles and doubles level by Purvis and Bartlett [11]. The theory behind the method has been discussed many times and so we will only sketch a brief outline here.

The coupled cluster wavefunction is written as an exponential of excitation operators acting on the Hartree–Fock reference,

$$|\Psi_{CC}\rangle = e^{T}|\Psi_0\rangle\,, \tag{1}$$

where T is a sum of singles, doubles, etc. excitation operators,

$$T = T_1 + T_2 + \ldots \tag{2}$$

that when truncated at the singles and doubles level gives rise to the CCSD method.

Substituting the wavefunction into the Schrödinger equation and projecting onto the subspace defined by the truncated excitation operator gives a set of coupled non-linear equations that are solved iteratively to yield the coupled cluster energy and amplitudes,

$$\langle\Psi_0|(\mathcal{H}-E_0)(1+T_1+T_2+\tfrac{1}{2}T_1^2)|\Psi_0\rangle = E_{CCSD}\,, \tag{3}$$

$$\langle\Psi_i^a|(\mathcal{H}-E_0)(1+T_1+T_2+\tfrac{1}{2}T_1^2+T_1T_2+\tfrac{1}{6}T_1^3)|\Psi_0\rangle = 0\,, \tag{4}$$

$$\langle\Psi_{ij}^{ab}|(\mathcal{H}-E_0)(1+T_1+T_2+\tfrac{1}{2}T_1^2+\tfrac{1}{2}T_2^2+T_1T_2+\tfrac{1}{6}T_1^3+\tfrac{1}{2}T_1^2T_2+\tfrac{1}{24}T_1^4)|\Psi_0\rangle = 0\,. \tag{5}$$

In terms of integrals and amplitudes we obtain the following form for the equations, taken from the paper by Scuseria et al. [12], on which we base our implementation. We assume canonical orbitals.

$$E_{CCSD} = \left[2(ia|jb) - (ib|ja)\right]\tau_{ij}^{ab}\,, \tag{6}$$

$$\begin{aligned}\sigma_i^a = {} & h_c^a t_i^c - h_i^k t_k^a + h_c^k\left(2t_{ki}^{ca} - t_{ik}^{ca} + t_i^c t_k^a\right)\\ & \quad \text{S1} \qquad \text{S2} \qquad\qquad \text{S3}\\ & + \left[2(ck|ai) - (ik|ac)\right]t_k^c + \left[2(ck|ad) - (dk|ac)\right]\tau_{ki}^{cd}\\ & \qquad\qquad \text{S4} \qquad\qquad\qquad \text{S5}\\ & - \left[2(ck|il) - (cl|ik)\right]\tau_{kl}^{ca}\,,\\ & \qquad\quad \text{S6}\end{aligned} \tag{7}$$

$$\begin{aligned}\sigma_{ij}^{ab} = {} & (ia|jb) + a_{ij}^{kl}\tau_{kl}^{ab} + b_{cb}^{ab}\tau_{ij}^{cd} + P_{ij}^{ab}\{g_c^a t_{ij}^{cb} - g_i^k t_{kj}^{ab}\\ & \quad \text{D1} \qquad \text{D2} \qquad \text{D3} \qquad\qquad \text{D4} \qquad \text{D5}\\ & + \left[(ia|bc) - (ik|bc)t_k^a\right]t_j^c - \left[(ai|jk) + (ai|ck)t_j^c\right]t_k^b\\ & \qquad\qquad \text{D6} \qquad\qquad\qquad\qquad \text{D7}\\ & + (j_{ic}^{ak} - \tfrac{1}{2}k_{ic}^{ka})(2t_{kj}^{cb} - t_{kj}^{bc}) - \tfrac{1}{2}k_{ic}^{ka}t_{kj}^{bc} - k_{ic}^{kb}t_{kj}^{ac}\}\\ & \qquad\qquad \text{D8} \qquad\qquad\quad \text{D9} \qquad\quad \text{D10}\end{aligned} \tag{8}$$

where P is a permutation operator defined as

$$P_{ij}^{ab}[\cdots]_{ij}^{ab} = [\cdots]_{ij}^{ab} + [\cdots]_{ji}^{ba} \tag{9}$$

and the intermediate quantities are

$$h_i^k = \left[2(kc|ld) - (kd|lc)\right]\tau_{il}^{cd}, \tag{10}$$

$$h_c^a = -\left[2(kc|ld) - (kd|lc)\right]\tau_{kl}^{ad}, \tag{11}$$

$$h_c^k = \left[2(kc|ld) - (kd|lc)\right]t_l^d, \tag{12}$$

$$g_i^k = h_i^k + \left[2(ik|cl) - (il|ck)\right]t_l^c, \tag{13}$$

$$g_c^a = h_c^a + \left[2(ac|dk) - (ad|ck)\right]t_k^d, \tag{14}$$

$$a_{ij}^{kl} = (ik|jl) + (ik|cl)t_j^c + (ck|jl)t_i^c + (kc|ld)\tau_{ij}^{cd}, \tag{15}$$

$$b_{cd}^{ab} = (ac|bd) - (ac|dk)t_k^b - (bd|ck)t_k^a, \tag{16}$$

$$j_{ic}^{ak} = (ai|ck) - (il|ck)t_l^a + (ad|ck)t_i^d - \tfrac{1}{2}(ck|dl)(t_{il}^{da} + 2t_i^d t_l^a) + \tfrac{1}{2}\left[2(ck|dl) - (dk|cl)\right]t_{il}^{ad}, \tag{17}$$

$$k_{ic}^{ka} = (ik|ac) - (ik|cl)t_l^a + (dk|ac)t_i^d - \tfrac{1}{2}(dk|cl)(t_{il}^{da} + 2t_i^d t_l^a), \tag{18}$$

$$\tau_{ij}^{ab} = t_{ij}^{ab} + t_i^a t_j^b. \tag{19}$$

Throughout this work we will use the convention that $i, j, k, l...$ label the occupied orbitals, $a, b, c, d...$ label the virtual orbitals, $p, q, r, s...$ label the whole range of molecular orbitals and $\alpha, \beta, \gamma, \delta...$ label the atomic orbitals. The number of occupied, virtual and atomic orbitals are denoted as n_o, n_v and n respectively.

Typically, the coupled cluster amplitude equations dominate the cost of a CCSD calculation, and so it is important to program them efficiently. With respect to operation count the most expensive terms are those labeled D2, D3, D8, D9 and D10 in Eq. (8), and the intermediates a, j and k that go towards forming them. These scale as n^6. With respect to storage requirements the most severe is associated with the two-electron integrals that have three or four virtual orbital indices. In the previous implementation [6] these were written to disk and read during each iteration of the CCSD. Of the other four indexed quantities, those with four occupied indices were replicated in local memory (i.e. the memory associated with a single processor), and those with one or two virtual indices were distributed across the global memory of the machine (i.e. the sum of the memory of all the processors). Processing the integrals with three and four virtual orbital indices gave rise to a major I/O bottleneck that limited the scaling of the previous code. Although this experience was specific to the Intel hypercube, the issue of scalable parallel I/O remains a major problem for MPPs. Hence in this work we have decided to circumvent the problem by evaluating those terms involving integrals with three and four virtual orbital indices in a "direct" fashion.

The only term that involves integrals with four virtual orbital indices is D3. This is also the costliest of the n^6 terms scaling as $n_o^2 n_v^4$. Our new AO-driven scheme for this term is given in Scheme 1. In this the four outer loops correspond to the atomic shell indices that define a batch of integrals. Parallel tasks are distributed according to the first two shell indices. As each batch of integrals is computed it is multiplied with previously back transformed τ amplitudes to generate a partially AO indexed "sigma" vector. When all the integrals are complete the partially AO indexed sigma vector is transformed to an intermediate S that corresponds to $S_{ij}^{pq} \equiv (p\ c|q\ d)\tau_{ij}^{cd}$. The first part of D3 is then formed by taking selected elements of S, while the remaining part is obtained by multiplying the relevant parts of S with the t_1 amplitudes. The storage requirements for this algorithm are a local array of size maxbfsh^4, where 'maxbfsh' is the maximum number of basis functions in a

back transform $\tau_{oo}^{\nu\nu}$ to $\tau_{oo}^{\alpha\beta}$ and t_o^ν to t_o^α

do α_{sh} = 1,nsh
do β_{sh} = 1,α_{sh}

if ((α_{sh},β_{sh}) *is my task) then*

do γ_{sh} = 1,nsh
do δ_{sh} = 1,γ_{sh}

generate a.o. integrals $(\alpha\beta|\gamma\delta)$, $(\alpha\gamma|\beta\delta)$

form $\sigma_+ = \frac{1}{2}\left[(\alpha\gamma|\beta\delta) + (\alpha\delta|\beta\gamma)\right]\left[\tau_{ij}^{\gamma\delta} + \tau_{ij}^{\delta\gamma}\right]$
form $\sigma_- = \frac{1}{2}\left[(\alpha\gamma|\beta\delta) - (\alpha\delta|\beta\gamma)\right]\left[\tau_{ij}^{\gamma\delta} - \tau_{ij}^{\delta\gamma}\right]$

accumulate $\sigma_{ij}^{\alpha\beta} = \frac{1}{2}\left(\sigma_+ + \sigma_-\right)$ into global memory
accumulate $\sigma_{ij}^{\beta\alpha} = \frac{1}{2}\left(\sigma_+ - \sigma_-\right)$ into global memory

enddo
enddo

end task

enddo
enddo

synchronise

transform $\sigma_{ij}^{\alpha\beta}$ to σ_{ij}^{ab}
transform $\sigma_{ij}^{\alpha\beta}$ to S_{ij}^{pq} : $S_{ij}^{pq} \equiv (p\ c|q\ d)\ \tau_{ij}^{cd}$
Form $\sigma_{ij}^{ab} = S_{ij}^{ab} - S_{ij}^{ak} * t_k^b - S_{ij}^{kb} * t_k^a$

Scheme 1. Loop structure used for evaluating the D3 term in Eq. (8).

shell, and a couple of global arrays of size $n_o^2 n^2$. To minimise the n^6 cost associated with this term we have taken symmetric/antisymmetric combinations of the integrals and amplitudes [12] and restricted the loops to $\alpha \geqslant \beta$, $\gamma \geqslant \delta$, $i \geqslant j$. The operation count is $\frac{1}{4} n_o^2 n_v^4$, but as a consequence of this loop structure we require the integrals to be computed four times the minimal list. In Section 4 we note the relative cost of the integral generation to the rest of the CCSD procedure for a variety of test cases.

The remaining "problem" terms only involve integrals with three virtual orbital indices. These are treated in a similar fashion to, and simultaneously with, the formation of S. For clarity, however, we illustrate the algorithm used for these terms in a separate scheme (Scheme 2). Specifically, using the same integral generation loops as for the S intermediate we contract the AO integrals with back transformed singles amplitudes to form two more intermediates X and Y, that are again partially AO indexed. When all the integrals have been generated the AO indices on X and Y are transformed to the MO basis to give terms corresponding to $X_{ij}^{pq} \equiv (a\ p|j\ q)\ t_i^a$ and $Y_{ij}^{pq} \equiv (a\ j|p\ q)\ t_i^a$. The formation of X and Y requires two local arrays of size maxbfsh$^2 n_o n$ and two global arrays of size $n_o^2 n^2$.

We note that the S, X and Y intermediates can be used in many other parts of the sigma equations, with elements read from global memory as required:

$$a_{ij}^{kl} = (ik|jl) + X_{ji}^{lk} + X_{ij}^{kl} + S_{ij}^{kl}\,, \tag{20}$$

$$h_i^k = 2S_{il}^{kl} - S_{il}^{lk}\,, \tag{21}$$

$$h_c^k = 2X_{lk}^{lc} - X_{ll}^{kc}\,, \tag{22}$$

$$g_i^k = h_i^k + 2X_{li}^{lk} - X_{li}^{kl}\,, \tag{23}$$

back transform τ_{oo}^{uv} to $\tau_{oo}^{\alpha\beta}$ and t_o^v to t_o^α

do α_{sh} = 1,nsh
do β_{sh} = 1,α_{sh}

if ((α_{sh},β_{sh}) *is my task*) *then*

do γ_{sh} = 1,nsh
do δ_{sh} = 1,γ_{sh}

generate a.o. integrals $(\alpha\beta|\gamma\delta)$, $(\alpha\gamma|\beta\delta)$

form $X_i^{\alpha\beta\gamma} = (\alpha\gamma|\beta\delta)t_i^\delta$
form $Y_i^{\alpha\beta\gamma} = (\alpha\beta|\gamma\delta)t_i^\delta$

enddo
enddo

transform $X_i^{\alpha\beta\gamma}$ to $X_{ij}^{\alpha\beta}$
transform $Y_i^{\alpha\beta\gamma}$ to $Y_{ij}^{\alpha\beta}$

end task

enddo
enddo

synchronise

transform $X_{ij}^{\alpha\beta}$, $Y_{ij}^{\alpha\beta}$: $X_{ij}^{pq} \equiv (a\ p|j\ q)\ t_i^a$
$Y_{ij}^{pq} \equiv (a\ j|p\ q)\ t_i^a$

Scheme 2. Loop structure used for evaluating the X and Y intermediates.

$$g_c^a = h_c^a + 2Y_{kk}^{ac} - X_{kk}^{ac}, \tag{24}$$

$$\mathrm{S4} = 2X_{ki}^{ka} - X_{ki}^{ak}, \tag{25}$$

$$\mathrm{S5} = 2S_{ki}^{ka} - S_{ki}^{ak}, \tag{26}$$

$$\mathrm{D6} = X_{ji}^{ba} - X_{ji}^{bk}t_k^a, \tag{27}$$

$$\mathrm{D7} = [(ai|jk) + X_{ji}^{ka}]t_k^b, \tag{28}$$

$$\mathrm{J3} = X_{ik}^{ac}, \tag{29}$$

$$\mathrm{K3} = Y_{ik}^{ac}. \tag{30}$$

Once all the terms involving integrals with three and four virtual orbital indices have been computed the remaining terms just involve intermediates, amplitudes and integrals with at most two virtual indices. These can be evaluated via multiplications local to a processor (e.g. for h_a^c, J1–J3, K1–K3, S1–S3, S6, D1, D4, D5) or globally across processors (e.g. for D2, D8–D10) by invoking the parallel GA matrix multiplication routine. Note that J1–J5 and K1–K4 refer to the respective terms of the j and k intermediates.

Being able to manipulate whole matrices in global memory has the advantage of allowing us to reduce the operation count associated with terms D8, D9 and D10. These terms scale as $n_o^3 n_v^3$ and involve the intermediates j and k whose J4, J5 and K4 terms also all scale as $n_o^3 n_v^3$ giving an overall cost of $6n_o^3 n_v^3$. D10, however, is similar to D9 so we only need to form one in a global array with the other being obtained by a simple index permutation. Furthermore, for D8 instead of forming j (at cost $2n_o^3 n_v^3$) and k (at cost $n_o^3 n_v^3$) separately we can accumulate $j - \frac{1}{2}k$ directly into global memory.

$$\begin{aligned} \mathrm{J4} + \mathrm{J5} - \tfrac{1}{2}\mathrm{K4} &= -\tfrac{1}{2}(ck|dl)\left(t_{il}^{da} + 2t_i^d t_l^a\right) + \tfrac{1}{2}\left[2(ck|dl) + (dk|cl)\right] t_{il}^{ad} + \tfrac{1}{4}(dk|cl)\left(t_{il}^{da} + 2t_i^d t_l^a\right) \\ &= \tfrac{1}{4}\left[2(ck|dl) - (dk|cl)\right]\left[2t_{il}^{ad} - t_{il}^{da} - 2t_i^d t_l^a\right] . \end{aligned} \tag{31}$$

Thus the cost has been reduced from 6 to $4n_o^3 n_v^3$.

3. The perturbative triples correction

Within our CCSD implementation we have also programmed the perturbative triples correction given by Raghavachari et al. [7]. This correction, denoted (T), is an estimate from perturbation theory of the contribution from triple excitations to the coupled cluster energy and is evaluated using the optimised cluster amplitudes at the end of a CCSD calculation. The computational cost of the triples calculation scales as n^7, making it considerably more expensive than the CCSD calculation. However, the triples are non-iterative and only require two-electron integrals with at most three virtual orbital indices. In this work we have used the "*aijkbc* algorithm" of Rendell et al. [13]. The important advantage of this algorithm is that it permits the three virtual indexed integrals to be formed and processed in blocks depending on how much global memory is available, the disadvantage is that the n^7 computational cost is 4 times the minimum.

The triples calculation proceeds by forming blocks of three virtual indexed integrals of the form $(ao|vv)$ and $(av|ov)$, where the range of the virtual index a is determined by the available global memory. Given this batch of integrals intermediates of the form: $f(b,c) = \sum_d (bd|ai) t_{kj}^{cd} - \sum_l (ck|jl) t_{il}^{ab}$ are evaluated locally on each processor. A contribution to the triples energy is obtained by taking products of these intermediates and accumulating the result. When this is complete a new batch of three virtual integrals is computed and the procedure is repeated until all three virtual two-electron integrals have been computed. The algorithm is illustrated in Scheme 3. Parallelisation is achieved both in the formation of the batches of three virtual indexed integrals (in a similar manner to that described by Wong et al. [3]) and in the computation of the triples energy.

4. Sample calculations

We have parallelised every aspect of the CCSD(T) code as much as possible. The main body of the CCSD code (Figs. 1 and 2) is distributed over nsh^2 tasks, where 'nsh' is the number of shells. The perturbative triples correction (Scheme 3) is also distributed over nsh^2 tasks in the computation of blocks of three virtual indexed integrals, and then over $n_o n_a$ tasks in processing these integrals (where n_a is the number of virtual orbitals that can be treated in a given batch). Three molecules of varying size have been used to assess the performance of our code: glycine, cytosine and serine. The geometries for these have been taken from the Cambridge Crystallographic Database [14], and the basis sets used are the correlation consistent sets of Dunning [15]. We note that Cartesian polarisation functions were used throughout, that symmetry was not used and that all orbitals were included in the correlation procedure. Our initial calculations used the CRAY T3D at the Edinburgh Parallel Computing Centre. The CRAY T3D is based on DEC 21064 chips (capable of 150 Mflops peak performance) connected in a 3D torus. At the time of this work the machine had 256 processors each with 8 MW of local memory.

We first consider glycine [16]. Locally imposed time limits restricted the CCSD calculation to a minimum of 16 processors and the (T) to 64 processors. The times for one CCSD iteration and the triples calculation are given in Table 1 and plotted in Fig. 1. Between 16 and 256 processors the CCSD speeds up by a factor of 11.4, while between 64 and 256 processors the triples speeds up by a factor of 3. This corresponds to a parallel efficiency of 71% and 75% respectively. Considering that this is a relatively small test case the results are surprisingly good.

Determine range of index "a" according to available memory
Parallel direct transformation to form $(ao|vv)$, $(av|ov)$

do $a = a_{lo}, a_{hi}$
  get t_{oo}^{av} from global memory
  get integrals $(av|vo)$, $(ao|vv)$ from global memory
  do j = 1,nocc
    get t_{jo}^{vv} from global memory
    get integrals $(jv|oo)$, $(jo|vo)$ from global memory

if ((a,j) is my task) then

    do i = 1,nocc
    do k = 1,i
    do b, c = 1,nvir
      form $f_{1n}(b,c) = \sum_d (bd|ai) t_{kj}^{cd} - \sum_l (ck|jl) t_{il}^{ab}$
      form $f_{2n}(b,c) = \sum_d (ad|bi) t_{kj}^{cd} - \sum_l (ck|jl) t_{il}^{ba}$
      form $f_{3n}(b,c) = \sum_d (bd|ai) t_{jk}^{cd} - \sum_l (cj|kl) t_{il}^{ab}$
      form $f_{4n}(b,c) = \sum_d (ad|bi) t_{jk}^{cd} - \sum_l (cj|kl) t_{il}^{ba}$
      similarly form f_{1t}, f_{2t}, f_{3t}, f_{4t} as $i \leftrightarrow k$
      evaluate contribution to the triples energy from products of f

end task

end all loops

Repeat as necessary for next batch of $(ao|vv)$ and $(av|ov)$ integrals

Scheme 3. Loop structure used for "direct" evaluation of the perturbative triples correction to the CCSD energy.

Table 1
Wall clock time (in s) obtained on the Cray T3D for various parts of a CCSD(T) calculation on glycine ($C_2O_2NH_5$) using the cc-pvdz basis set and with no frozen orbitals ($n_o = 20$, $n_v = 80$)

	Number of processors				
	16	32	64	128	256
CCSD – one iteration					
main AO block	330.7	169.6	87.2	46.6	26.4
– integral generation	167.6	86.3	41.6	20.6	10.6
– formation of S	49.6	24.7	12.8	6.4	3.3
– formation of X and Y	53.6	25.6	13.3	6.8	3.7
remaining sigma terms	44.8	24.4	14.7	9.8	6.1
total	381.8	197.6	104.0	57.5	33.5
(T)					
integrals+transformation			69.9	37.5	65.0
total			1010.4	527.1	329.7

SCF energy: −282.793141 E_h. MP2 energy: −283.635433 E_h. CCSD energy: −283.661272 E_h. CCSD(T) energy: −283.689261 E_h.

A breakdown of the timings associated with various parts of the code is also given in Table 1. These times have been recorded by inserting timing calls in the relevant portions of the code and are the results reported by processor 0. The results will vary somewhat from processor to processor as the code is load balanced with respect to the integral computation. Looking at the results for the CCSD iteration, as expected, the majority

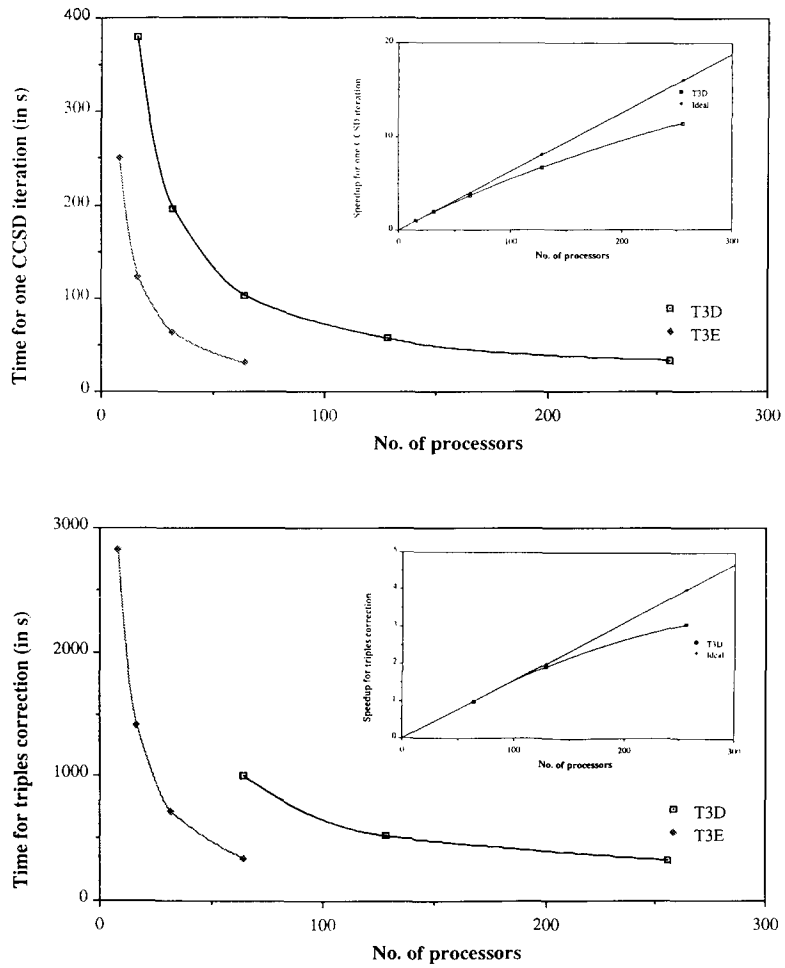

Fig. 1. Scaling of one CCSD iteration and (T) for glycine. Insets: speedup curves (normalised to the lowest number of processors used).

(>85%) of the time goes into the main AO driven block that calculates the S, X and Y intermediates. Within this roughly 50% of the time is associated with the computation of integrals, about 15% with the computation of the partially AO indexed S intermediate and another 15% with the X and Y intermediates. For a large calculation we would expect the formation of S, which scales as $n_o^2 n_v^4$, to dominate the computation, but this is not the case here. After the main AO integral block most of the remaining time is spent in the formation of the other sigma vector terms. Although this involves some work which scales as n^6 this part of the calculation accounts for less than 15% of the total time.

In comparison with the CCSD, the integral generation in the triples calculation is insignificant, accounting for less than 10% of the total time. For this small system it was only necessary to compute the three virtual indexed integrals once. In going from 64 to 128 processors the integral generation time decreases, but from 128 to 256 it increases. This reflects the small size of the calculation in that the transformation component inherent in the generation of these integrals is becoming communication bound for a large number of processors [3].

Results for the slightly larger cytosine [17] test case are presented in Table 2 and Fig. 2. Between 64 and 256 processors the CCSD shows a parallel efficiency of about 88%, while between 128 and 256 processors the triples are over 90% efficient. For this larger system the time required for the integral generation in the CCSD

Table 2
Wall clock time (in s) obtained on the Cray T3D for various parts of a CCSD(T) calculation on cytosine ($C_4ON_3H_5$) using the cc-pvdz basis set and with no frozen orbitals ($n_o = 29$, $n_v = 116$)

	Number of processors		
	64	128	256
CCSD – one iteration			
main AO block	491.4	253.5	134.4
– integral generation	169.9	87.4	43.7
– formation of S	116.0	57.6	28.9
– formation of X and Y	92.9	47.2	24.9
remaining sigma terms	103.6	56.6	33.7
total	605.0	315.5	171.2
(T)			
integrals+transformation		215.5	207.1
total		5399.7	2897.8

SCF energy: −392.595550 E_h. MP2 energy: −393.806612 E_h. CCSD energy: −393.836324 E_h. CCSD(T) energy: −393.884859 E_h.

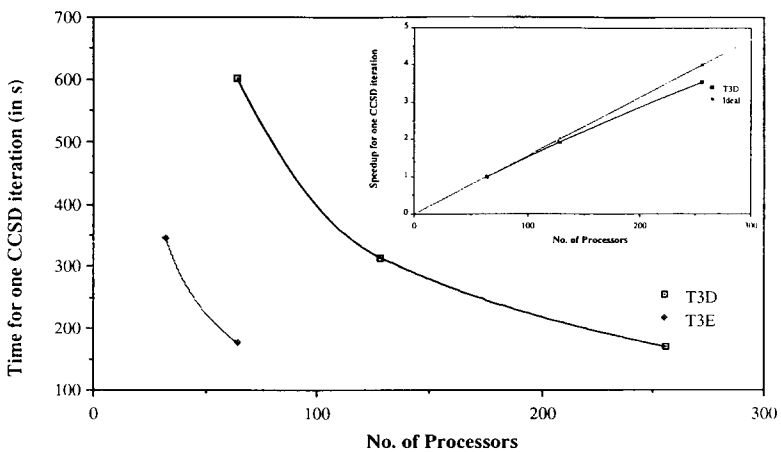

Fig. 2. Scaling of one CCSD iteration for cytosine Inset: speedup curve (normalised to the 64 processor time).

is now less than 30%, compared with over 40% for glycine. The integral generation time is now only slightly greater than the time associated with the S intermediate (the dominant $n_o^2 n_v^4$ term).

Our largest test case is serine [18]. The basis set used was Dunning's cc-pvtz basis set without the f functions on the heavy atoms and d functions on the hydrogens. This gives 238 basis functions with 28 occupied orbitals and 210 virtuals. Only 256 processors were used, since global memory requirements prevented the calculation from being performed on 128 or fewer processors. This is not surprising as a quantity of size $n_o^2 n^2$ (of which we store several) requires roughly 45MW of memory, and there is only 8MW of memory per processor that must be shared between local and global data items and the memory requirements of the operating system. For this calculation there are just over 17.2 million independent amplitudes. Timings for the calculation are given in Table 3, the perturbative triples correction was omitted as being too time consuming. One CCSD iteration was found to take about 15 minutes elapsed time, with about 23% of this time associated with the integral generation and 21% with the formation of the S intermediate.

The next generation MPPs promise faster processors and more memory per processor. During the course of

Table 3
Wall clock time (in s) obtained on 256 processors of the Cray T3D for various parts of a single CCSD iteration on serine ($C_3O_3NH_7$) using a modified cc-pvtz basis set [a] and with no frozen orbitals ($n_o = 29$, $n_v = 116$)

	Time
main AO block	715.8
– integral generation	205.3
– formation of S	188.3
– formation of X and Y	139.0
remaining sigma terms	144.4
total	872.7

[a] The f functions on C, N and O and d functions on H were not included. SCF energy: −396.752464 E_h. MP2 energy: −398.162457 E_h CCSD energy: −398.185896 E_h.

Table 4
Timings (in s) obtained using a Cray T3E for the glycine and cytosine benchmarks [a]

	Number of processors			
	8	16	32	64
glycine				
total CCSD - one iteration	249.9	123.7(381.8)	63.4(197.6)	30.8(104.0)
total (T)	2829.1	1418.0	709.7	341.8(1010.4)
cytosine				
total CCSD - one iteration			345.2	177.1(315.5)

[a] Times obtained on the T3D are given in parentheses.

this work we were able to gain access to such a machine, a CRAY T3E located at Cray Research Inc. The Cray T3E has a similar topology and programming model to the T3D, but has faster processors (DECchip 21164 with 600 Mflops peak performance), a faster interconnect (480Mbytes/s versus 300Mbytes/s on the T3D) and potentially more memory per node. The machine available to us had 64 processors each with 16MW of memory. Calculations were run for glycine and cytosine. The results are given in Table 4 together with the corresponding T3D times when available. The results consistently show a 3–4 times speedup over the T3D while maintaining the same scalability. Extrapolating these findings to serine implies that one iteration on a 256 processor T3E would take roughly 4–5 min. Assuming 15 iterations are required to converge the calculation it is clear that large-scale CCSD calculations are fast becoming routine. Our goal of performing CCSD calculations with 500 basis functions will probably soon be viable if the molecule has at least one element of symmetry and the core orbitals are frozen. Whether we would be able to perform the (T) calculation for such a system is doubtful, due probably to time constraints rather than memory!

5. Conclusion

We have successfully implemented an AO-driven CCSD(T) algorithm specially designed to run on MPPs. Sample calculations have been presented for glycine, cytosine and serine run on a CRAY T3D with up to 256 processors. The results show that the program runs quite effectively in parallel, scaling well up to 256 processors. The main limitation of this algorithm is memory, restricting our largest case to 238 basis functions. Preliminary calculations run on the Cray T3E show a 3–4 fold performance increase over those run on the T3D. This shows promise for the next generation MPPs to make large-scale coupled cluster calculations routine.

Acknowledgements

Our thanks go to Dr. Howard Pritchard and Dr. John Carpenter of CRAY Research Institute, who funded this work, for helpful discussions and access to their machines. Thanks also to the people involved at Daresbury Laboratory and Pacific Northwest Laboratory for their help, and to Professor N.C. Handy for critical reading of the manuscript and helpful comments.

References

[1] G.E. Scuseria, Chem. Phys. Lett. 243 (1995) 193.
[2] M. Challacombe, E. Schwegler and J. Almlöf, J. Chem. Phys. 104 (1996) 4685.
[3] A.T. Wong, R.J. Harrison and A.P. Rendell, Theor. Chim. Acta 93 (1996) 317.
[4] H. Koch, A. Sánchez de Merás, T. Helgaker and O. Christiansen, J. Chem. Phys. 104 (1996) 4157.
[5] A.P. Rendell, T.J. Lee and R. Lindh, Chem. Phys. Lett. 194 (1992) 84.
[6] A.P. Rendell, M.F. Guest and R.A. Kendall, J. Comput. Chem. 14 (1993) 1429.
[7] K. Raghavachari, G.W. Trucks, J.A. Pople and M. Head-Gordon, Chem. Phys. Lett. 157 (1989) 479.
[8] NWChem program package, developed at Pacific Northwest Laboratory, PO Box 999, Mail Stop K1-96, Richland, WA 99352.
[9] J. Nieplocha, R.J. Harrison and R.J. Littlefield, Global Arrays; A Portable Shared Memory Programming Model for Distributed Memory Computers, Supercomputing '94 (IEEE Computer Society Press, Washington, DC, 1994).
[10] J. Čížek and J. Paldus, Int. J. Quant. Chem. Symp. 5 (1971) 359.
[11] G.D. Purvis and R.J. Bartlett, J. Chem. Phys. 76 (1982) 1910.
[12] G.E. Scuseria, C.L. Janssen and H.F. Schaefer, J. Chem. Phys. 89 (1988) 7382.
[13] A.P. Rendell, T.J. Lee, A. Komornicki and S. Wilson, Theor. Chim. Acta 84 (1993) 271.
[14] Cambridge Structural Database, compiled and distributed by the Cambridge Crystallographic Data Centre, Cambridge.
[15] T.H. Dunning Jr, J. Chem. Phys. 90 (1989) 1007.
[16] L.F. Power, K.E. Turner and F.H. Moore, Acta Crystallogr. Sect. B 32 (1976) 11.
[17] R.J. Mc Clure and B.M. Craven, Acta Crystallogr. Sect. B 29 (1973) 1234.
[18] T.J. Kistenmacher, G.A. Rand and R.E. Marsh, Acta Crystallogr. Sect. B 30 (1974) 2573.