Theor Chim Acta (1993) 84: 271–287

Theoretica Chimica Acta

© Springer-Verlag 1993

Evaluation of the contribution from triply excited intermediates to the fourth-order perturbation theory energy on Intel distributed memory supercomputers

Alistair P. Rendell[1], Timothy J. Lee[2], Andrew Komornicki[3,*], and Stephen Wilson[4]

[1] SERC Daresbury Laboratory, Warrington WA4 4AD, UK

[2] NASA Ames Research Center, Moffett Field, CA 94035, USA

[3] Polyatomics Research Institute, Mountain View, CA 94043, USA

[4] SERC Rutherford Appleton Laboratory, Chilton, Oxfordshire, OX11 0QX, UK

Received October 1, 1991/Accepted January 3, 1992

Summary. Three previously reported algorithms for the evaluation of the fourth-order triple excitation energy component in many-body perturbation theory have been compared. Their implementation on current Intel distributed memory parallel computers has been investigated. None of the algorithms, which were developed for shared memory computer architectures, performed well since they lead to prohibitive IO demands. A new algorithm suitable for distributed memory machines is suggested and its implementation on two Intel i860 supercomputers is described. A high level of parallelism is obtained.

Key words: Fourth-order triple excitation energy – Many-body perturbation theory – Intel distributed memory parallel computers

1. Introduction

Over the last decade there has been significant interest in the role of triple excitations in a many-body perturbation theory expansion for the electron correlation energy and in the related coupled-cluster theory [1–36]. Early calculations [3–8, 10] established that the fourth-order triple excitation energy component is: (i) frequently just as important as the other fourth-order energy components and should, therefore, be included in any treatment which aims to go beyond third-order; (ii) observed to be particularly sensitive to the quality of the basis set employed and, therefore, requires the use of large systematically constructed basis sets; (iii) particularly important for multiply bonded systems; (iv) capable of extending the range of applicability of expansions based on a single determinantal reference function.

The purpose of this paper is to explore potential algorithms for the evaluation of the fourth-order energy component involving triply excited intermediate states on distributed memory parallel processing computers, such as the Intel

* *Mailing address*: 1101 San Antonio Road, Suite 420, Mountain View, CA 94043, USA

i860 GAMMA and DELTA machines. For all of the algorithms included in this study, each of the processors may be performing different operations at the exact same time since, e.g., matrix multiplies being performed on each processor will in general consist of matrices of different sizes. Therefore, we limit our current study to MIMD distributed memory parallel computers since it will be much more difficult to map conventional quantum chemical algorithms onto a SIMD architecture.

Algorithms [9] devised to evaluate the fourth-order many-body perturbation theory triple excitation energy component scale as n^7, where n is the number of basis functions. The evaluation of other fourth-order energy components leads to algorithms which scale as n^6 or less and thus a full fourth-order calculation is dominated by the triple excitation component. In many-body perturbation theory calculations the fourth-order triple excitation energy component is sometimes heuristically neglected [37] or approximated using a truncated set of virtual orbitals [18]. Cullen and Zerner [38] and Almlöf [39] have suggested novel algorithms which scale as n^6.

For calculations based on the coupled-cluster expansion (for a recent review see reference [40]), in which certain classes of diagrammatic terms are summed to high-order, the computational difficulties are compounded. It should be noted, however, that the first coupled-cluster calculation to include contributions from triply excited intermediate states was reported nearly twenty years ago by Paldus, Čižek and Shavitt [40], although to obtain a tractable algorithm these authors employed a minimum basis set. Whereas the cluster expansion based on all connected single and double excitation amplitudes (CCSD) gives rise to an algorithm which scales as n^6, when triple excitations are included an algorithm scaling as n^8 is obtained. It is, therefore, not surprising that a number of procedures for estimating the effects of triple excitations in coupled-cluster calculations have been developed. Lee et al. [12] simply added the fourth-order perturbation theory energy component arising from triply excited intermediate states to the correlation energy obtained from a CCSD calculation. Raghavachari [13] made a perturbative evaluation of the fourth-order energy associated with singly and triply excited intermediate states based on the wave function obtained from a coupled-cluster expansion involving only the connected double excitation amplitudes. Urban et al. [14] followed a similar procedure but employed the amplitudes obtained from a CCSD wave function. More recently a similar procedure, denoted CCSD(T) [21], has been shown to give very good results for a number of "difficult" systems [22–24]. All of these calculations can be carried out with algorithms which are modifications of those used for fourth-order many-body perturbation theory and which, therefore, scale as n^7.

Recent years have seen the proliferation of new computer designs that employ parallel processing in one form or another in order to enhance performance. The advent of concurrent computation is already having a significant impact on molecular electronic structure calculations (for a recent review see [42]). In electron correlation studies, the linked diagram theorem of many-body perturbation theory effectively decouples a many-electron system involving a large number of electrons into a series of smaller problems each of which can be treated concurrently during a calculation. A number of algorithms for multiprocessor computers with shared memory have been proposed for the evaluation of the fourth-order energy component associated with triply excited intermediate states [28–32, 34–36]. In Sect. 2, we review and compare these algorithms. In Sect. 3, an algorithm suitable for implementation on a distributed memory computer is described and its performance characteristics on the Intel i860 GAMMA and DELTA machines are presented. In Sect. 4 we compare the performance measured on the Intel machines with those observed on other contemporary supercomputers. Section 5 contains our conclusions.

2. Algorithms for the evaluation of the fourth-order correlation energy component associated with triply excited intermediate states

In this section we consider three algorithms for the evaluation of the fourth-order energy associated with triply excited intermediate states. All three algorithms have already been considered for parallel implementations on multiprocessor shared memory computers. However, no realistic calculations have been performed to explore the potential of these methods on distributed memory parallel computers. The first algorithm is based on a diagrammatic partitioning of the fourth-order triples energy and has recently been demonstrated by Moncrieff et al. [30] to run very efficiently and at near peak performance on multiple processors of a CRAY Y-MP/8. The second algorithm was developed by Rendell et al. [29] within the context of a triples correction to the CCSD energy and computes the total fourth-order triples energy. This method has also been implemented on multiple processors of a CRAY Y-MP/8 and showed similar performance characteristics to the diagrammatic approach of Moncrieff et al. The final algorithm was developed by Dupuis et al. [19] for the calculation of the total fourth-order perturbation theory energy. A brief discussion concerning the applicability of this latter method to a multiprocessor environment has been given by Watts and Dupuis [20], although to the best of our knowledge this has yet to be implemented.

Each of the three algorithms considered involves a sum over three occupied orbital indices (*ijk*) and three virtual orbital indices (*abc*), together with an additional sum over an occupied or virtual orbital index. The algorithms differ in the order in which the *ijk* and *abc* summations are performed and the extent to which the triple excitation energy is analysed in terms of its diagrammatic components. These two factors determine: (i) the total number of floating point operations; (ii) the memory requirements; (iii) the efficiency of implementation on a vector processing machine; (iv) the efficiency of implementation on a parallel processing machine; (v) the input/output (IO) requirements for calculations which use disk storage for the two-electron integrals. To distinguish between the three algorithms we shall label them by the order in which the *ijk* and *abc* summations are carried out, thus the *bcjkia* algorithm of Moncrieff et al. performs the summation over index *a* first (the inner fortran loop) and index *b* last (the outer fortran loop).

2.1. The bcjkia algorithm of Moncrieff et al.

The *bcjkia* algorithm of Moncrieff et al. [28, 35] forms the part of a concurrent computation Many-Body Perturbation Theory (*cc*MBPT) program that evaluates the energy associated with each of the 16 fourth-order diagrams that involve triply excited intermediate states (*cc*MBPT-4t). Such a diagrammatic analysis can be important; for example, for the beryllium ground state it is possible to perform a detailed angular momentum analysis of the different diagrammatic components using graphical methods [32, 33].

Assuming real orbitals, fourth-order diagrammatic many-body perturbation theory gives 12 unique terms involving triply excited intermediates. Although these terms have been given previously [9] they are repeated here [Eqs. (1)–(12)] for discussion purposes:

$$E_4(A_T) = -\tfrac{1}{2}\sum_{\rm ijk}\sum_{\rm abc} f_{\rm ijk;abc} \times f_{\rm ijk;acb}/D^{\rm abc}_{\rm ijk}, \tag{1}$$

$$E_4(B_T) = -\tfrac{1}{2}\sum_{\rm ijk}\sum_{\rm abc} f_{\rm ijk;abc} \times f_{\rm ikj;abc}/D^{\rm abc}_{\rm ijk}, \tag{2}$$

$$E_4(C_T) = -\tfrac{1}{2}\sum_{\rm ijk}\sum_{\rm abc} g_{\rm ijk;abc} \times g_{\rm ijk;abc}/D^{\rm abc}_{\rm ijk}, \tag{3}$$

$$E_4(D_T) = -\tfrac{1}{2}\sum_{\rm ijk}\sum_{\rm abc} g_{\rm ijk;abc} \times g_{\rm ikj;abc}/D^{\rm abc}_{\rm ijk}, \tag{4}$$

$$E_4(E_T) = \sum_{\rm ijk}\sum_{\rm abc} f_{\rm ijk;abc} \times f_{\rm ikj;acb}/D^{\rm abc}_{\rm ijk}, \tag{5}$$

$$E_4(F_T) = \tfrac{1}{4}\sum_{\rm ijk}\sum_{\rm abc} f_{\rm ijk;abc} \times f_{\rm ijk;abc}/D^{\rm abc}_{\rm ijk}, \tag{6}$$

$$E_4(G_T) = \tfrac{1}{4}\sum_{\rm ijk}\sum_{\rm abc} g_{\rm ijk;abc} \times g_{\rm ijk;abc}/D^{\rm abc}_{\rm ijk}, \tag{7}$$

$$E_4(H_T) = \sum_{\rm ijk}\sum_{\rm abc} g_{\rm ijk;abc} \times g_{\rm ikj;acb}/D^{\rm abc}_{\rm ijk}, \tag{8}$$

$$E_4(I_T) = -\tfrac{1}{4}\sum_{\rm ijk}\sum_{\rm abc} f_{\rm ijk;abc} \times g_{\rm ikj;acb}/D^{\rm abc}_{\rm ijk}, \tag{9}$$

$$E_4(J_T) = \sum_{\rm ijk}\sum_{\rm abc} f_{\rm ijk;abc} \times g_{\rm ijk;abc}/D^{\rm abc}_{\rm ijk}, \tag{10}$$

$$E_4(M_T) = \tfrac{1}{2}\sum_{\rm ijk}\sum_{\rm abc} f_{\rm ijk;abc} \times g_{\rm ijk;acb}/D^{\rm abc}_{\rm ijk}, \tag{11}$$

$$E_4(O_T) = \tfrac{1}{2}\sum_{\rm ijk}\sum_{\rm abc} f_{\rm ijk;abc} \times g_{\rm ikj;abc}/D^{\rm abc}_{\rm ijk}, \tag{12}$$

where

$$f_{\rm ijk;abc} = \sum_{d} ([\rm id|jb] - [ib|jd])([da|kc] - [dc|ka])/D^{\rm db}_{\rm ij}, \tag{13}$$

$$g_{\rm ijk;abc} = \sum_{\rm l} ([\rm jb|la] - [ja|bl])([il|kc] - [ic|kl])/D^{\rm ab}_{\rm jl}, \tag{14}$$

$$D^{\rm ab\ldots}_{\rm ij\ldots} = \epsilon_{\rm i} + \epsilon_{\rm j} + \cdots - \epsilon_{\rm a} - \epsilon_{\rm b} \cdots, \tag{15}$$

indices ijk . . . /abc . . . are used to denote occupied/virtual spin orbitals, ϵ are the orbital energies and the two-electron integrals are in charge density notation. Spin integration leads to 9 terms for the f [Eq. (13)] and g [Eq. (14)] intermediates. Using the italic indices *ijk . . . /abc . . .* for occupied/virtual spatial orbitals, we have for example for the f intermediates:

$$f^{\alpha\alpha\alpha;\alpha\alpha\alpha}_{ijk;abc} = T_1 + T_2 + T_3 + T_4, \tag{16}$$

$$f^{\alpha\alpha\alpha;\beta\alpha\beta}_{ijk;abc} = T_1 + T_3, \tag{17}$$

$$f^{\alpha\alpha\alpha;\beta\beta\alpha}_{ijk;abc} = T_2 + T_4 \tag{18}$$

$$f^{\alpha\beta\beta;\alpha\alpha\alpha}_{ijk;abc} = T_1 + T_2, \tag{19}$$

$$f^{\alpha\beta\beta;\beta\alpha\beta}_{ijk;abc} = T_1, \tag{20}$$

$$f^{\alpha\beta\beta;\beta\beta\alpha}_{ijk;abc} = T_2, \tag{21}$$

$$f^{\beta\alpha\beta;\alpha\alpha\alpha}_{ijk;abc} = T_3 + T_4, \tag{22}$$

$$f^{\beta\alpha\beta;\beta\alpha\beta}_{ijk;abc} = T_3, \tag{23}$$

$$f^{\beta\alpha\beta;\beta\beta\alpha}_{ijk;abc} = T_4, \tag{24}$$

where the secondary intermediates $\boldsymbol{T}$ are defined as follows:

$$T_1 = \sum_d [id|jb][da|kc]/D^{db}_{ij}, \tag{25}$$

$$T_2 = -\sum_d [id|jb][dc|ka]/D^{db}_{ij}, \tag{26}$$

$$T_3 = -\sum_d [ib|jd][da|kc]/D^{db}_{ij}, \tag{27}$$

$$T_4 = \sum_d [ib|jd][dc|ka]/D^{db}_{ij}. \tag{28}$$

Inspection of Eqs. (1)–(12) reveals that for a given *ijkabc* index combination it is necessary to have access to:

$$f_{ijk;abc}, f_{ijk;acb}, f_{ikj;abc}, f_{ikj;acb}, \tag{29}$$

as well as the corresponding permutations of the $\boldsymbol{g}$ intermediates, before all the diagrammatic contributions can be evaluated. These four f and $\boldsymbol{g}$ permutations are also those required if the summations in Eqs. (1)–(12) are restricted according to:

$$\sum_{ijk}\sum_{abc} \rightarrow \sum_{b>c}\ \sum_{j>k}\ \sum_{i}\sum_{a}\ . \tag{30}$$

This restricted summation will decrease the number of floating point operations by a factor of four. Hence for a fixed *bcjk* index combination the four different permutations [Eq. (29)] of the f and $\boldsymbol{g}$ intermediates are evaluated for all *ia* indices, denoted $F^{bc;jk}(ia)$, $F^{cb;jk}(ia)$, $F^{bc;kj}(ia)$ and $F^{cb;kj}(ia)$. This is achieved at near optimal efficiency on a CRAY Y-MP/8 using matrix multiplications to obtain the secondary $\boldsymbol{T}$ intermediates [Eqs. (25)–(28)], e.g.:

$$T^f_1 = \sum_d [da|kc][id|jb]/D^{db}_{ij} = \sum_d A1^{k,c}(a,d)B^{j,b}(d,i), \tag{31}$$

$$T^g_1 = \sum_l [il|jb][la|kc]/D^{ac}_{lk} = \sum_d B^{k,c}(a,l)A2^{j,b}(l,i), \tag{32}$$

(where the denominator is incorporated into the quantity $\boldsymbol{B}$) and summing these to produce the spin free f and $\boldsymbol{g}$ intermediates shown in Eqs. (16)–(24). The final algorithm, as shown in Fig. 1, contains four matrix multiplications to form the spin free f intermediates for a given *bcjk* index permutation and four more to obtain the equivalent $\boldsymbol{g}$ intermediates. These matrix multiplications are repeated four times for the different permutations of *bcjk* shown in Eq. (29). Without restricting the summations according to Eq. (30), the number of floating point operations in the part of this algorithm which scales as n^7 is $16(n_o^3n_v^4 + n_o^4n_v^3)$,

```
inode = 0
do 100 b = 1, nvirt
  read $[bv|vo]$, $[vv|bo]$, $[bo|vo]$
  do 100 c = 1, b
    read $[cv|vo]$, $[vv|co]$, $[co|vo]$
    do 100 j = 1, nocc
      do 100 k = 1, j
        inode = inode + 1
        if (mod(inode,nodes).ne.me) go to 100
        |
        200 continue
        Form $F^{bc;jk}(ia)$
        $t_1^f = \sum [da|kc][id|jb]/D_{ij}^{db} \equiv$ mxm(v:v:o) Eq. 25
        $t_2^f = -\sum [dc|ka][id|jb]/D_{ij}^{db} \equiv$ mxm(v:v:o) Eq. 26
        $t_3^f = -\sum [da|kc][ib|jd]/D_{ij}^{db} \equiv$ mxm(v:v:o) Eq. 27
        $t_4^f = \sum [dc|ka][ib|jd]/D_{ij}^{db} \equiv$ mxm(v:v:o) Eq. 28
        Formation of spin free intermediates
        $f_{ijk;abc} = \sum t_n^f$ Eqs. (16–24)
        |
        Form $G^{bc;jk}(ia)$
        $t_1^g = \sum [il|jb][la|kc]/D_{lk}^{ac} \equiv$ mxm(v:o:o)
        $t_2^g = -\sum [jl|ib][ka|lc]/D_{lk}^{ac} \equiv$ mxm(v:o:o)
        $t_3^g = -\sum [il|jb][la|kc]/D_{lk}^{ac} \equiv$ mxm(v:o:o)
        $t_4^g = \sum [jl|ib][ka|lc]/D_{lk}^{ac} \equiv$ mxm(v:o:o)
        Formation of spin free intermediates
        $g_{ijk;abc} = \sum t_n^g$
        |
        REPEAT 200 for $f_{ijk;acb}$, $f_{ikj;abc}$, $f_{ikj;acb}$
        |
        Sum $f^{jk;bc}(ia)/g^{jk;bc}(ia)$ to $E_{pt4}^T(me)$ Eqs. (1–12)
        |
100 continue
|
global sum over nodes to give $E_{pt4}^T$
```

Fig. 1. The *bcjkia* (*cc*MBPT-4t) algorithm

where n_o/n_v is the number of occupied/virtual orbitals. Employing the restricted summation reduces this to $4(n_o^3 n_v^4 + n_o^4 n_v^3)$. The memory requirements for the storage of the intermediates scales as $n_o n_v$.

Moncrieff et al. obtained a parallel version of the *bcjkia* algorithm by assigning tasks defined by different *bcjk* indices to different processors. This was accomplished on machines such as the CRAY Y-MP and IBM 3090 by the technique of "dynamic load balancing" using "global indices" under the control of a "lock" [32, 43]. The kernel of each of the tasks assigned to a single processor is a series of matrix multiplications. Each processor accumulates 12 partial sums corresponding to the different diagrammatic contributions given in Eqs. (1)–(12). These partial energies are summed across the different processors once all *bcjk* index combinations have been processed. Since parallelism is utilized at the $n_o^2 n_v^2$ level, given $n_o^2 n_v^2$ processors the limiting computational rate would be that of a process of order n^3.

The *cc*MBPT-4t code developed by Moncrieff et al. has been employed in a study of macro-tasking in a multi-job environment on a CRAY Y-MP/8 computer. It has been demonstrated that there is no degradation in the performance of the system as the proportion of multitasked jobs is increased [31]. Further work has demonstrated that on a heavily loaded CRAY X-MP/4 system a multiprocessed job running at low scheduling priority using the dynamic load balancing technique employed in the *cc*MBPT-4t program is an effective "scavenger", consuming almost all potentially idle cycles [44].

For large cases an external storage device will be required for the two-electron integrals. Since the largest memory requirements in the computation occurs for the $[vv|vo]$ integrals some means of paging these from disk should be implemented. Assuming that memory is available to hold quantities of length $n_o n_v^2$, the $[vv|vo]$ integrals can be accessed according to the b and c indices given in Eq. (30). This requires four buffers of length $n_o n_v^2$ to hold the $[vv|bo]$, $[bv|vo]$, $[vv|co]$ and $[cv|vo]$ integrals which are read from disk as detailed in Fig. 1. Given no restrictions on the indices b and c it is obvious from Fig. 1 that the $[vv|vo]$ integral list will have to be read $2(n_v + 1)$ times. Should it be necessary, the $[ov|ov]$ integrals can also be paged into memory according to the b or c index simultaneously with the $[vv|vo]$ integrals. The minimum feasible memory requirement of the method would appear to be four buffers of length $n_o n_v^2$, two of length $n_o^2 n_v$ for the $[ov|ov]$ integrals, an array of size $n_o^3 n_v$ for the $[oo|ov]$ integrals and several work arrays of length n^2.

2.2. *The ijkabc algorithm of Rendell et al.*

Rendell et al. [29] have presented an algorithm which yields the total fourth-order triples energy component, E_{pt4}^T, rather than the individual diagrammatic components. The expression for E_{pt4}^T used by these authors is given in terms of spin restricted orbitals as:

$$E_{pt4}^T = \tfrac{1}{3}\sum_{ijk}\sum_{abc}(4W_{ijk}^{abc} + W_{ijk}^{bca} + W_{ijk}^{cab})(W_{ijk}^{abc} - W_{ijk}^{cba})/D_{ijk}^{abc}, \tag{33}$$

where

$$W_{ijk}^{abc} = P_{ijk}^{abc}\left(\sum_d^{vir}[bd|ai][kc|jd]/D_{kj}^{cd} - \sum_l^{occ}[ck|jl][ia|lb]/D_{il}^{ab}\right), \tag{34}$$

and P_{ijk}^{abc} is a permutation operator defined by:

$$P_{ijk}^{abc}\begin{pmatrix}abc\\ijk\end{pmatrix} = \begin{pmatrix}abc\\ijk\end{pmatrix} + \begin{pmatrix}acb\\ikj\end{pmatrix} + \begin{pmatrix}cab\\kij\end{pmatrix} + \begin{pmatrix}cba\\kji\end{pmatrix} + \begin{pmatrix}bca\\jki\end{pmatrix} + \begin{pmatrix}bac\\jik\end{pmatrix}. \tag{35}$$

The algorithm for evaluating Eq. (33) used by Rendell et al. requires that quantities of size n_v^3 be held in memory, so that for a given ijk index of W all abc were formed, denoted $W^{ijk}(abc)$. As evident from Eq. (34) and the definition of the permutation operator given in Eq. (35), this requires 6 pairs of matrix multiplications of the form:

$$W^{ijk}(abc) \leftarrow \sum_d A1^k(ac,d)B^{ji}(d,b) - \sum_l B^k(ac,l)A2^{ji}(l,b), \tag{36}$$

which can be vectorized over the compound virtual index ac. Since W_{ijk}^{abc} is symmetric with respect to permutation of the index pairs ia, jb and kc it is only

```
do 100 i = 1,nocc
  read $[vv|vi]$
  do 100 j = 1,i
    read $[vv|vj]$
    do 100 k = 1, j
      read $[vv|vk]$
      Set $W^{ijk}(abc) = 0$
        |
      200 continue
        $W^{ijk}(abc) \leftarrow \sum [bd|ai][kc|jd]/D^{cd}_{kj} \equiv$ mxm(v$^2$:v:v) Eq. (34)
        $W^{ijk}(abc) \leftarrow \sum [ck|jl][ia|lb]/D^{ab}_{il} \equiv$ mxm(v$^2$:o:v) Eq. (34)
        Sort $W^{ijk}(abc)$ for next contribution
        |
      REPEAT 200 with $\begin{pmatrix} acb \\ ikj \end{pmatrix}, \begin{pmatrix} cab \\ kij \end{pmatrix}, \begin{pmatrix} cba \\ kji \end{pmatrix}, \begin{pmatrix} bca \\ jki \end{pmatrix}, \begin{pmatrix} bac \\ jik \end{pmatrix}$ Eq. (35)
        |
      Sum $W^{ijk}(abc)$ to $E^T_{pt4}$ Eq. (33)
        |
100 continue
```

Fig. 2. The *ijkabc* algorithm

necessary to form $W^{ijk}(abc)$ such that $i > j > k$. The final algorithm of Rendell et al. is outlined in Fig. 2.

Without restricting the indices i, j and k, the presence of the 6 pairs of matrix multiplications to form W will make the number of floating point operations in the part of the algorithm which scales as n^7 $6(n_o^3 n_v^4 + n_o^4 n_v^3)$. Restricting $i > j > k$ reduces this to $(n_o^3 n_v^4 + n_o^4 n_v^3)$, which is a factor of 4 less than the *bcjkia* (*cc*MBPT-4t) algorithm.

Use of multiple processors on a CRAY Y-MP/8 was achieved by using a parallel matrix multiply routine and parallelizing the sorting operations and final energy summation over n_v orbitals. This algorithm would therefore appear to tend towards a computational ratio of n^6 in the limit of n_v processors.

As discussed by Rendell et al., the ideal minimum memory requirement for their algorithm is three buffers of length n_v^3 to store integral lists $[vv|vi]$, $[vv|vj]$ and $[vv|vk]$, and two other scratch arrays of the same length. The $[vv|vo]$ integrals are read from disk approximately n_o^3 times. This memory requirement can be further reduced to only two buffers of length n_v^3, but at the expense of increased IO requirements.

2.3. *The abcijk algorithm of Dupuis et al.*

The algorithm recently described by Dupuis et al. also calculates the total fourth-order triple excitation energy component. In this algorithm the intermediates $W^{abc}(ijk)$ and not $W^{ijk}(abc)$ are stored in memory, i.e. the summations are performed in the order *abcijk*. This algorithm therefore has a much smaller memory requirement, $\sim n_o^3$, than that of Rendell et al., but involves the same number of floating point operations. However for small values of n_o the algorithm is less suitable for vector processing as the vector loop in Eq. (36) is only of order n_o^2.

3. Parallel algorithms for distributed memory computers

The algorithms described in the previous section have been implemented on "conventional" supercomputers, machines consisting of a small number of powerful processors sharing a common memory. In this section we consider the problem of implementation on "novel" architecture machines with a larger number of (usually less powerful) processors each with its own local memory. Such machines are typified by the Intel i860 hypercube, the so-called GAMMA machine, and the more recent Intel i860 DELTA machine.

These machines consist of a system of tightly coupled processing elements (processor with associated memory) each performing separate tasks and communicating via message passing. Disk access is available through a concurrent file system in which each processing element, or node, can "simultaneously" access the same file. The relevant specifications of the distributed memory computers employed in the present study are collected in Table 1.

We are interested in performing calculations on large systems, which may contain, for example, $n_v \sim 500$ and $n_o \sim 50$ and have C_s or C_{2v} symmetry. On the Intel i860 GAMMA/DELTA machines used here there are 8/16 Mybtes of memory on each node (although such machines can support up to 32 Mbytes of memory per node). This being the case, it would appear prudent not to assume that quantities of the size n_v^3 can be held on each node; a more realistic maximum would be $n_o n_v^2$. Consequently, the *bcjkia* and *abcijk* algorithms which meet this requirement would appear potentially suited to implementation on the Intel machines. On the other hand, the method of Rendell et al. does not lend itself to a distributed memory architecture without incurring a large amount of internode communications. Ideally the Rendell et al. algorithm should distribute the formation of the $W^{ijk}(abc)$ intermediates over the various nodes, but this would necessitate storing quantities of length n_v^2 on each node.

In view of the factor of 4 difference in the n^7 floating point operation count between the *bcjkia* and *abcijk* algorithms, we began by investigating the performance of the *abcijk* scheme on the Intel i860 GAMMA machine. This algorithm is illustrated in Fig. 3. In cases where the size of the calculation is sufficiently small, all integrals can be held on each node and there is no need to use external disk storage. The results obtained using between 1 and 32 nodes are given in

Table 1. Specifications of the Intel i860 distributed memory computers employed in the present experiments

	GAMMA	DELTA
Installation	SERC Daresbury Laboratory[a]	California Institute of Technology
Topology	Hypercube	Mesh
No of Nodes	32	16 × 32 (512)
Memory per Node	8 Mbytes	16 Mbytes
Cycle time	40 MHz	40 MHz
Peak Performance per Node	40 Mflops	40 Mflops
Peak Performance	1.28 Gflops	20.48 Gflops

[a] For further details see Ref. [45]

Table 2. The performance of the *abcijk* parallel algorithm obtained when all integrals are held in memory on each node of the Intel i860 GAMMA machine[a]

No. of processors	Wall clock time (sec)	Speed increase
1	139.6	1.00
2	69.9	2.00
4	35.0	3.99
8	17.6	7.93
16	8.8	15.79
32	4.5	31.24

[a] Number of occupied/virtual orbitals per symmetry [4, 3, 2, 1/10, 12, 14, 16]; a total of 10 occupied and 52 virtual orbitals

Table 2. As expected, the results indicate a near perfect scaling of computation time with the number of nodes.

When the size of the basis set increases it is no longer possible to hold all integrals on each node and it is necessary to page through (at least) the $[vv|vo]$ integrals stored on the concurrent file system. This can conveniently be achieved by having four buffers of length $n_o n_v^2$ for the blocks of integrals $[av|vo]$, $[vv|ao]$, $[bv|vo]$ and $[vv|bo]$ which are read from disk as shown in Fig. 3. The final procedure will effectively read the $[vv|vo]$ integrals $2(n_v + 1)$ times, which is identical to the IO requirements of the out-of-core version of the *bcjkia* procedure discussed in Sec. 2.1. The results obtained for the same calculation reported in Table 2, but now reading the $[vv|vo]$ integrals from disk, are given in Table 3.

```
do 100 a = 1,nvirt
read $[av|vo]$ and $[vv|ao]$
do 100 b = 1,a
read $[bv|vo]$ and $[vv|bo]$
do 100 c = 1,b
Set $W^{abc}(ijk) = 0$
|
200 continue
$W^{abc}(ijk) \leftarrow \sum [bd|ai][kc|jd]/D^{cd}_{kj} \equiv \mathrm{mxm}(o^2{:}v{:}o)$
$W^{abc}(ijk) \leftarrow \sum [ck|jl][ia|lb]/D^{ab}_{il} \equiv \mathrm{mxm}(o^2{:}o{:}o)$
Sort $W^{abc}(ijk)$ for next contribution
|
Repeat 200 with $\binom{acb}{ikj}, \binom{cab}{kij}, \binom{cba}{kji}, \binom{bca}{jki}, \binom{bac}{jik}$ Eq. (35)
|
Sum $W^{abc}(ijk)$ to $E^T_{pt4}$ Eq. (33)
|
100 continue
```

Fig. 3. The *abcijk* algorithm

Table 3. The performance of the *abcijk* parallel algorithm obtained when the $[vv|vo]$ integrals read from the concurrent file system using synchronous IO on the Intel i860 GAMMA machine[a]

No. of processors	Max IO time (sec)	Max CPU time (sec)	Wall clock time (sec)	Speed increase[b]
1	238.7	155.4	394.1	1.00
2	147.5	72.1	219.5	1.80
4	136.1	36.1	172.1	2.29
8	107.2	18.1	125.3	3.15
16	124.4	9.4	133.4	2.95

[a] Number of occupied/virtual orbitals per symmetry [4, 3, 2, 1/10, 12, 14, 16]; a total of 10 occupied and 52 virtual orbitals
[b] Based on the wall clock time

It is immediately apparent that even on 1 node the process is totally IO bound, and while there is some gain from using multiple nodes the asymptotic computational rate is quickly reached. Furthermore, the relative ratio of the CPU (central processing unit) to IO time suggests that no great improvements can be made by using asynchronous IO, but rather that the inherent IO requirements of the algorithm substantially exceed the current capabilities of the Intel i860 GAMMA machine. Although future releases of the operating system and associated improvements in the IO subsystem may lead to improved IO transfer rates, it would be necessary to increase the IO transfer rate by at least one order of magnitude before this algorithm would become viable.

A possible solution to this problem may exist if all $[vv|vo]$ integrals can be held on the combined memory of all nodes. A systolic loop could then be envisaged in which blocks of integrals of length $n_o n_v^2$ are passed around the loop in sequence with the indices a and b shown in Fig. 3. Since the communication speed on the Intel is greater than the IO transfer rate, this would theoretically give improved performance. However, the number of occupied and virtual orbitals would strongly dictate the minimum number of nodes that must be used to perform a calculation and this is probably not a desirable feature, e.g. it may require 100 nodes to perform a calculation containing 100 virtual orbitals.

As mentioned above and discussed in Sect. 2.1, an out-of-core version of the *bcjkia* algorithm has an identical IO requirement for the $[vv|vo]$ integrals as the *abcijk* procedure. Thus, despite the potential for being easily implemented on the Intel i860 machines, it is also liable to suffer from poor IO characteristics. To achieve a highly scalable method suitable for current Intel computers it would seem imperative that the IO requirements be reduced to an absolute minimum. From the formula for E_{pt4}^{T} given in Eq. (33) it is difficult to see how this can be improved, although for the *bcjkia* algorithm this is not the case. Examination of Eqs. (1)–(12) reveals that the only indices to remain constant in all the different permutations of the f and g intermediates are i and a. For the f intermediate the indices i and a derive from integrals $[iv|ov]$ and $[av|vo]/[vv|ao]$ respectively and from $[oa|ov]$ and $[io|ov]/[oo|iv]$ integrals for the g intermediate. Consequently if the summations in Eqs. (1)–(12) are reordered such that:

$$\sum_{ijk}\sum_{abc} \rightarrow \sum_{a}\sum_{i}\sum_{j>k}\sum_{bc}, \tag{37}$$

it is possible to read the $[vv|vo]$ integrals from disk in sequence with the outer index of the summation. The $[vv|vo]$ integrals are effectively read from disk only twice (in the form $[av|vo]$ and $[vv|ao]$). Ideally the other integrals ($[ov|ov]$ and $[vo|oo]$) would be stored in memory on each node. However, as shown above these can be read from disk in sequence with either index a or i, or alternatively distributed over the global memory of the machine and retrieved as required. An algorithm which reads all integrals from disk is illustrated in Fig. 4. We term this the *aijkbc* algorithm. The memory requirement on each node is three buffers of length $n_o n_v^2$ and three of length $n_o^2 n_v$ as well as some n^2 buffers.

```
                inode = 0
              do 100 a = 1,nvirt
        read $[av|vo]$, $[vv|ao]$, $[ao|vo]$
              do 100 i = 1,nocc
        read $[io|ov]$, $[oo|iv]$, $[vi|vo]$
              inode = inode + 1
   if (mod(inode,nodes).ne.me) go to 100
                     |
              do 100 j = 1,nocc
               do 100 k = 1, j
                     |
                200 continue
              Form $F^{ijk;a}(bc)$
    $t_1^f = \sum [da|kc][id|jb]/D_{ij}^{db} \equiv$ mxm(v:v:v)
    $t_2^f = -\sum [dc|ka][id|jb]/D_{ij}^{db} \equiv$ mxm(v:v:v)
    $t_3^f = -\sum [da|kc][ib|jd]/D_{ij}^{db} \equiv$ mxm(v:v:v)
    $t_4^f = \sum [dc|ka][ib|jd]/D_{ij}^{db} =$ mxm(v:v:v)
      Formation of spin free intermediates
        $f_{ijk;abc} = \sum t_n^f$ Eqs. (16–24)
                     |
              Form $G^{ijk;a}(bc)$
    $t_1^g = \sum [il|jb][la|kc]/D_{lk}^{ac} \equiv$ mxm(v:o:v)
    $t_2^g = -\sum [jl|ib][ka|lc]/D_{lk}^{ac} \equiv$ mxm(v:o:v)
    $t_3^g = -\sum [il|jb][la|kc]/D_{lk}^{ac} \equiv$ mxm(v:o:v)
    $t_4^g = \sum [jl|ib][ka|lc]/D_{lk}^{ac} \equiv$ mxm(v:o:v)
      Formation of spin free intermdeiates

              $g_{ijk;abc} = \sum t_n^g$
                     |
         REPEAT 200 for $f_{ikj;abc}$
                     |
Sum $f^{jk;bc}(ia)/g^{jk;bc}(ia)$ to $E_{pt4}^T(me)$ Eqs. (1–12)
                     |
                100 continue
                     |
   global sum over nodes to give $E_{pt4}^T$
```

Fig. 4. The *aijkbc* algorithm

The n^7 floating point operation count of this new algorithm is the same as that of the original diagrammatic *cc*MBPT-4t approach, but the IO requirements are much improved. Parallelism of the code inside the outermost loop would ensure that each node operates on a distinct part of the $[vv|vo]$ integral file, however this could lead to an inefficient load balance if the number of nodes is not a factor of n_v, and would also yield a limiting computational rate of n^6. We have therefore parallelized the code inside the next loop (as shown in Fig. 4) which potentially leads to a rate limiting computational rate of n^5. While it would be possible to parallelize inside the fourth (k) loop of Fig. 4, this could lead to several processors needing to read the same part of the integral files at the same time and degrade the overall performance.

In Table 4 we present timings for the new algorithm on different numbers of nodes for a calculation involving 4 occupied and 112 virtual orbitals. In these calculations, not only were the $[vv|vo]$ integrals paged into memory, but also the $[ov|ov]$ and $[oo|ov]$ integals. The results show substantially better IO performance than those presented in Table 3. On a single processor and using synchronous IO approximately 207 seconds is required to read the integrals from disk. However, this can be reduced to about 19 seconds if asynchronous IO is used. Although as the number of nodes increases the asynchronous IO times exhibit a somewhat random behavior, the results are not as unacceptable as the values found in Table 3. The variation in the IO time is a consequence of conflicts with other users wishing to perform IO. For the CPU time alone, a speed increase of 222 is observed when employing 256 CPUs relative to one CPU, but this factor is reduced to 183 when the IO time is included.

The results in Table 4 for $n_o = 4$ and $n_v = 112$ also illustrate the effects of inefficient load balancing. As shown in Fig. 4, the code is parallel at the $n_v n_o$ level, which in this case corresponds to 448 (4×112) tasks. Therefore, if the number of nodes is not a factor of 448 there will be an inefficient load balance.

Table 4. The performance of the *aijkbc* parallel algorithm obtained on the Intel i860 DELTA machine reading the $[vv|vo]$, $[vo|vo]$ and $[vo|oo]$ integrals from the concurrent file system using asynchronous IO. There are 4 (112) occupied (virtual) orbitals and no symmetry giving rise to inefficient load balancing (see text for details)

No. of processors	Max IO time (sec)	Max CPU time (sec)	Wall clock time (sec)	Speed increase[a]
1	18.9[b]	12218.7	12237.6	1.00
4	7.4	3060.1	3067.3	3.99
16	10.4	765.8	775.2	15.79
36	3.7	357.4	360.5	33.95
64	3.8	192.6	196.1	62.40
100	6.1	137.4	142.6	85.81
144	14.4	109.9	123.8	98.85
196	10.9	82.5	92.7	132.01
256	12.0	55.0	66.7	183.47

[a] Based on the wall clock time

[b] The equivalent IO time using synchronous IO is approximately 207 seconds

Table 5. The performance of the *aijkbc* parallel algorithm obtained on the Intel i860 DELTA machine reading the $[vv|vo]$, $[vo|vo]$ and $[vo|oo]$ integrals from the concurrent file system using asynchronous IO. There are 4 (128) occupied (virtual) orbitals and no symmetry giving rise to efficient load balancing (see text for details)

No. of processors	Max IO time (sec)	Max CPU time (sec)	Wall clock time (sec)	Speed increase[a]
16	5.5	1734.2	1739.5	1.00[b]
32	53.7	866.7	919.7	1.89
64	5.6	433.4	438.8	3.96
128	9.3	216.9	225.8	7.70
256	16.2	108.8	124.8	13.94

[a] Based on the wall clock time
[b] Note that the 16 node time is the reference point for these ratios

The results obtained using 4, 16 and 64 nodes are ideally load balanced, and based on the CPU time alone the speed up (3.99, 15.96, 63.44) is very good. For the other results, the number of nodes is not a factor of the number of tasks, and since the number of nodes is tending towards the number of tasks the effect on the speed up is quite drastic.

In Table 5 we report results obtained for $n_o = 4$ and $n_v = 128$, a case for which the number of nodes is always an exact factor of the number of tasks. The timing obtained on 256 nodes shows a speed up of 15.9 relative to the 16 node timing and based solely on the CPU time. This result is to be compared with 13.9 for the $n_o = 4$ and $n_v = 112$ calculation.

4. Discussion

The results presented in Table 5 indicate an 87.1% increase in the efficiency of the computation for the 256 processor case with respect to the 16 processor case. With this level of parallelism a 256 processor machine could deliver 8.9 Gflops given a rate of execution of 40 Mflops per node.

The total wall clock times required to carry out calculations with $n_o = 4$, $n_v = 112$, 128 on the 6 processor IBM 3090/600J VF, the 8 processor CRAY Y-MP/8128 and the 256 processor Intel i860 DELTA machine are compared in Table 6. The rate of execution achieved on the CRAY Y-MP for the largest case was 2.284 Gflops [35] which should be compared with the theoretical peak performance of 2.667 Gflops. The peak performance of the 256 processor Intel i860 DELTA machine is 3.84 times that of the CRAY Y-MP/8128. However, for our largest calculation the wall clock time measured on the CRAY Y-MP/8 is 73% of that obtained for the Intel machine. Although a significant level of parallelism has been built in to the code the rate of execution observed on each of the Intel nodes falls well below the peak rate. Each of the tasks assigned to a given node involves a part which scales as n^7 involving a series of multiplications and a part which scales as n^6 which is required to assemble the contributions to the different diagrammatic components. On the Intel machine, this

Table 6. Comparison of the total wall clock times observed in the present experiments with previous work

Machine	No. of processors	Wall clock time (sec) $n_v = 112$	$n_v = 128$
IBM 3090/600J VF[a]	6	513.1	839.4
CRAY Y-MP/8128[b]	8	56.4	91.4
Intel i860 DELTA[c]	256	66.7	124.8

[a] Ref. [36]
[b] Ref. [30]
[c] Present work

second part is found to demand about 2/3 of the time required for each task, a situation which does not persist on the CRAY Y-MP where effective vectorization is obtained. Using the data given by Moncrieff et al. [35] for the CRAY Y-MP/8128 and the wall clock times given in Table 6 for the $n_v = 128$ case, we estimate the rate of execution on 256 nodes of the Intel i860 DELTA machine to be 1.67 Gflops. This estimated computational rate could be improved by restructuring the n^6 component of the code specifically for the i860, and by future releases of the i860 FORTRAN-compiler which will better exploit the capability of i860.

5. Conclusions

We have compared three different algorithms, originally developed for shared memory parallel processing architectures, for the evaluation of the fourth-order triple excitation energy component in many-body perturbation theory and investigated their implementation on distributed memory parallel computers. The algorithm of Moncrieff et al. evaluates the energy corresponding to each of the diagrammatic terms. The other algorithms of Rendell et al. and Dupuis et al. yield the total triple excitation energy. The algorithms, which can be labelled by the order in which summation over three occupied orbital indices and three virtual orbital indices are carried out, differ in: the number of floating point operations they involve (the algorithm of Moncriefl et al. requires 4 times the number arising in the other two), the memory requirements (the algorithm of Rendell et al. requires the storage of $\sim n_v^3$ intermediates) and the IO demands when two-electron integrals are stored on disk. None of the algorithms developed for shared memory architectures performed well when implemented on a distributed memory machine. All three algorithms were found to lead to prohibitive IO demands. However, it was discovered that the algorithm of Moncrieff et al. could be modified to minimize the IO requirements. This new algorithm and its implementation on two Intel i860 machines has been described. A high level of parallelism has been obtained, but the rather poor performance of each node resulted in a disappointing overall rate of computation.

A more general conclusion evident from this investigation is that direct or "superdirect" algorithms, or hybrid schemes, will undoubtedly find favor as more sophisticated distributed memory parallel computers are developed. That is, algorithms in which a minimum of disk access is used in trade for a more CPU intensive load will be essential since currently, advances in processing speed are increasing much more rapidly than advances in IO performance. For the triples component of the fourth-order perturbation theory energy a hybrid scheme whereby the $[vv|vo]$ integrals are computed as needed and the $[ov|ov]$ and $[oo|ov]$ integrals are stored on disk and read in as needed would probably show much better scalable performance with respect to the number of nodes than any of the algorithms described here. We will be investigating such an algorithm and the results of that study will be reported in due course.

Acknowledgments. This work was presented at the Argonne/Battelle Pacific Northwest Laboratories workshop on "Parallel Computers in Chemical Physics" at Willowbrook, Illinois, 17–19 July 1991. APR and TJL would like to thank Robert Harrison (Argonne National Laboratory) and Ray Bair (Battelle Pacific Northwest Laboratory) for invitations to attend the workshop. We would also like to acknowledge Pacific Northwest Laboratory and the Concurrent Supercomputing Consortium for providing access to the Intel Touchstone DELTA supercomputer, located at the California Institute of Technology. We also thank Ray Bair, Rick Kendall and Martin Feyereisen of the Pacific Northwest Laboratory for assistance in implementing the code on the DELTA.

References

1. Wilson S (1978) in: Saunders VR (ed) Correlated wave functions. Proc Daresbury Lab Study Weekend. SRC Daresbury Laboratory
2. Wilson S, Silver DM (1979) Int J Quantum Chem 15:683
3. Wilson S, Saunders VR (1979) J Phys B At Mol Phys 12:L403; (1980) *ibid* 13:1505
4. Wilson S (1979) J Phys B At. Mol Phys 123:L657; (1980) *ibid* 13:1505
5. Guest MF, Wilson S (1980) Chem Phys Lett 72:49
6. Wilson S, Guest MF (1980) Chem Phys Lett 73:607
7. Frisch MJ, Krishnan R, Pople JA (1980) Chem Phys Lett 75:66
8. Krishnan R, Frisch MJ, Pople JA (1980) J Chem Phys 72:4244
9. Wilson S, Saunders VR (1980) Comput Phys Commun 19:293
10. Wilson S, Guest MF (1981) Molec Phys 43: 1331
11. Noga J (1983) Comput Phys Commun 29:117
12. Lee YS, Kucharski SA, Bartlett RJ (1984) J Chem Phys 81:5906
13. Raghavachari K (1985) J Chem Phys 82:4607
14. Urban M, Noga J, Cole SJ, Bartlett RJ (1985) J Chem Phys 83:4041
15. Urban M, Cernusak I, Kello V, Noga J (1987) in: Electron correlation in atoms and molecules, Meth Comput Chem 1:117
16. Noga J, Bartlett RJ (1987) J Chem Phys 86:7041
17. Scuseria GE, Schaefer HF (1988) Chem Phys Lett 152:382
18. Adamowitz L, Bartlett RJ (1988) Phys Rev A37:1
19. Dupuis M, Mougenot P, Watts JD, Hurst GJB, Villar HO (1989) in: Clementi E (ed) MOTECC modern techniques in computational chemistry. Escom, Leiden
20. Watts JD, Dupuis M (1989) IBM Technical Report KGN-197, August 16, 1989
21. Raghavachari K, Trucks GW, Pople JA, Head-Gordon M (1989) Chem Phys Lett 157:479
22. Lee TJ, Rendell AP, Taylor PR (1990) J Chem Phys 92:489
23. Lee TJ, Scuseria GE (1990) J Chem Phys 93:489
24. Scuseria GE, Lee TJ (1990) J Chem Phys 93:5851
25. Bartlett RJ, Watts JD, Kucharski SA, Noga J (1990) Chem Phys Lett 165:513
26. Baker DJ, Moncrieff D, Wilson S (1990) in: Evans RG, Wilson S (eds) Supercomputational science. Plenum Press, NY

27. Lee TJ, Rice JE (1991) J Chem Phys 94:1215
28. Baker DJ, Moncrieff D, Saunders VR, Wilson S (1991) Comput Phys Commun 62:25
29. Rendell AP, Lee TJ, Komornicki A (1991) Chem Phys Lett 178:462
30. Moncrieff D, Saunders VR, Wilson S (submitted) Int J Supercomputer Appln
31. Moncrieff D, Saunders VR, Wilson S (1991) Parallel Computing 17:773
32. Wilson S (1992) in: Wilson S, Dierchsen GHF (eds) Methods in computational molecular physics. Plenum Press, NY
33. Mårtensson-Pendrill AM, Wilson S (in preparation)
34. Wilson S, Moncrieff D (submitted) Supercomputer
35. Moncrieff D, Saunders VR, Wilson S (submitted) Comput Phys Commun
36. Moncrieff D, Saunders VR, Wilson S, Rutherford Appleton Laboratory Report RA-91-064
37. Bartlett RJ, Shavitt I, Purvis II G (1979) J Chem Phys 71:281
38. Cullen JM, Zerner MC (1982) Theoret Chim Acta 61:203
39. Almlöf J (1991) Chem Phys Lett 181:319
40. Paldus J (1992) in: Wilson S, Dierchsen GHF (eds) Methods in computational molecular physics. Plenum Press, NY
41. Paldus J, Čižek J, Shavitt I (1972) Phys Rev A5:50
42. Wilson S (ed) (1989) Concurrent computation in chemical calculations. Meth Comput Chem 3, Plenum Press, NY
43. Saunders VR (1990) in: Evans RG, Wilson S (eds) Supercomputational science. Plenum Press, NY
44. Saunders VR, Wilson S (in press) Parallel Computing
45. Guest MF, Sherwood P, van Lenthe JH, Theoret Chim Acta (this issue)