Theor Chim Acta (1997) 95: 13–34

Theoretica Chimica Acta

An integral direct, distributed-data, parallel MP2 algorithm

Martin Schütz, Roland Lindh

Department of Theoretical Chemistry, Chemical Centre, P.O. Box 124, University of Lund, S-22100 Lund, Sweden

Received May 7, 1996/Final revision received September 19, 1996/Accepted September 19, 1996

Summary. A scalable integral direct, distributed-data parallel algorithm for four-index transformation is presented. The algorithm was implemented in the context of the second-order Møller–Plesset (MP2) energy evaluation, yet it is easily adopted for other electron correlation methods, where only MO integrals with two indices in the virtual orbitals space are required. The major computational steps of the MP2 energy are the two-electron integral evaluation $\mathcal{O}(N^4)$ and transformation into the MO basis $\mathcal{O}(ON^4)$, where N is the number of basis functions, and O the number of occupied orbitals, respectively. The associated maximal communication costs scale as $\mathcal{O}(n_\Sigma O^2 V N)$, where V and n_Σ denote the number of virtual orbitals, and the number of symmetry-unique shells. The largest local and global memory requirements are $\mathcal{O}(N^2)$ for the MO coefficients and $\mathcal{O}(OVN)$ for the three-quarter transformed integrals, respectively. Several aspects of the implementation such as symmetry-treatment, integral prescreening, and the distribution of data and computational tasks are discussed. The parallel efficiency of the algorithm is demonstrated by calculations on the phenanthrene molecule, with 762 primitive Gaussians, contracted to 412 basis functions. The calculations were performed on an IBM SP2 with 48 nodes. The measured wall clock time on 48 nodes is less than 15 min for this calculation, and the speedup relative to single-node execution is estimated to 527. This superlinear speedup is a result of exploiting both the compute power and the aggregate memory of the parallel computer. The latter reduces the number of passes through the AO integral list, and hence the operation count of the calculation. The test calculations also show that the evaluation of the two-electron integrals dominates the calculation, despite the higher scaling of the transformation step.

Key words: Parallel – Møller–Plesset – Four-index transformation

1 Introduction

In recent years, advances in computer technology together with substantial improvements in quantum chemical algorithms have enabled *ab initio* electronic structure calculations on chemical systems of increasing complexity. In 1989 Price et al. [1] reported a self-consistent field (SCF) study on $C_{63}H_{113}N_{11}O_{12}$,

a derivative of the immuno-suppressive drug cyclosporine using a 3-21G basis set, involving 1000 basis functions. SCF and MP2 calculations of similar or even larger size have been carried out also for fullerene and some of its derivatives, although making use of the high symmetry of these chemical systems [2–5]. Today, SCF calculations including several thousands of basis functions [6–8], and coupled cluster calculations with some hundred basis functions [9–11] are feasible. Common to all of these *ab initio* algorithms which aim at large-scale problems is that they are *integral direct* in the sense that the electron repulsion integrals (ERIs) are reevaluated whenever needed, rather than computed once, stored on disk and read from disk when required. The use of integral direct methods is motivated by the following observations:

(i) The limiting factor precluding many applications, imposed by the use of conventional (non-integral direct) methods is the disk space, required to store the ERIs ($\mathcal{O}(N^4)$ quantity with N denoting the number of basis functions).

(ii) Conventional methods inflict a heavy input/output (I/O) load on the machine. Regarding the hardware development within the last decade, the advances achieved in the design of faster CPUs are much more impressive than the improvements, accomplished for I/O systems. This is especially true for local workstation (clusters) and massively parallel processing (MPP) supercomputers.

(iii) Ongoing improvements in integral evaluation methods [12] such as integral prescreening, reexpansion of Gaussian products in auxiliary basis sets, and splitting of Coulomb and exchange part with subsequent semiclassical treatment of the Coulomb part are reducing the extra costs of evaluating the ERIs multiple times.

Integral direct methods were first used in SCF theory ("direct SCF" approach by Almlöf et al. [13], but have since been extended to methods like multiconfigurational SCF [MCSCF] [14, 15], many-body perturbation theory [MBPT(2)] (the cheapest method which includes dynamic correlation of electrons) [16, 17], MBPT(2) gradients [18] and coupled cluster methods [9, 10]. In contrast to the SCF method, where the ERIs over atomic orbitals (AOs), i.e. the basis functions) are immediately contracted to the Fock matrix in AO basis, and only AO integrals are needed, correlated methods including MCSCF are usually formulated in terms of ERIs over molecular orbitals (MOs). This means in practice, that an integral transformation of the ERIs in AO basis to the MO basis is required as an intermediate step. A full 4-index transformation, carried out as four-quarter transformations has a flop count that scales as $\mathcal{O}(N^5)$ with the number of basis functions N, and has $\mathcal{O}(N^4)$ storage requirements. At a first sight these memory requirements for integral transformation seem to rule out any integral direct implementation of a correlated method. Fortunately enough, however, most correlated methods can be reformulated in terms of AO ERIs and a reasonably small subset of MO integrals. Such MO integral subsets typically have two indices restricted to the *occupied* orbital space of dimension O, which is usually much smaller than N. For example, the computation of the MBPT(2) energy only requires the exchange integrals $(ia|jb)$, while for direct MCSCF theory the Coulomb $(ij|pq)$ and exchange $(ip|jq)$ MO integral lists are needed (Mulliken notation with i, j denoting occupied, a, b virtual, and p, q any orbitals). The memory necessary to hold such a subset of MO integrals is then $\mathcal{O}(O^2N^2)$.

In order to improve the performance and capacities of *ab initio* electronic structure codes beyond current limits and to tackle grand challenge-class problems, it has become increasingly evident during the last few years, that it is necessary to exploit the inherent parallelism in existing algorithms and to maximize such parallelism by abandoning and replacing certain parts of the algorithms, or even

developing new algorithms from scratch. The class of parallel computer architectures most suitable for computational chemistry is the Multiple Instruction Multiple Data (MIMD) category with multiple independent instruction threads that operate on multiple, distinct data items (compared to the Single Instruction Multiple Data (SIMD) class of machines with only a single instruction thread). MIMD computers can further be subdivided with respect to their memory arrangement: For *shared-memory* systems the common memory lies in the address space of each processor, while for *distributed-memory* systems each processor has fast access to its own *local* memory only, and access to *non-local* memory requires explicit *interprocessor communication.* Portable *message-passing libraries* (with MPI (message-passing interface) [19] as the emerging standard) are providing support for these communications in distributed-memory systems; however all interprocessor communication events have to be programmed explicitly into the application code and the application programmer has to shoulder the burden to set up a consistent and efficient communication scheme and to avoid e.g. deadlock situations or data inconsistencies. The programming of shared-memory computers is in general much easier, since each processor can have access to all data items of the algorithm without any need for cooperation with other processors, although some care has to be taken in order to avoid data access conflicts and to maintain data integrity. On the other hand, the access of very many processors on a common, shared memory imposes a serious bottleneck. Therefore, virtually all MPP systems are either explicitly of the distributed-memory type (e.g. IBM SPX, Intel Paragon), or provide virtual shared memory (e.g. Cray T3D), what means that each processor has its own local memory, but there is hardware support at various levels for accessing the virtual, global address space. There are no genuine MPP shared-memory systems available these days, which provide *uniform* fast memory access. As a consequence, for good performance the application programmer should distinguish between local (fast) and non-local (slow) memory access in his code. This is nothing new, since in order to write efficient sequential code for ordinary scientific workstations the programmer also has to be aware of on-cache and off-cache memory access. MPP machines just augment these two different memory classes by a third one (non-local memory), where access to is especially slow. The use of message-passing libraries automatically enforces a distinction between local and non-local memory. For those parallel algorithms, which are based on *replicated-data* schemes (each processor owns its own copy of each data item) the usage of message passing is a convenient choice to code up the necessary communication framework. For other algorithms however, which require a *distributed-data* scheme explicit message passing can become cumbersome.

The majority of the present distributed parallel direct SCF codes replicate the Fock and density matrices [20–28], which minimizes communication costs and enables dynamic load balancing in a simple way. Such codes are well suited for clusters of processors interconnected by slow communication links (ethernet, wide-area networks) and with time-dependent load situations [25, 27]. However, replicated-data schemes are inherently non-scalable, since the maximum problem size still is limited by the local memory of a single processor, rather than the aggregate memory of all processors together. Recently, some very efficient implementations of truely scalable *distributed-data* SCF algorithms have been discussed in the literature [29, 6–8], where the Fock and density matrices were fully distributed over all processors of an MPP machine.

The development of parallel, integral direct algorithms for correlated methods calls automatically for distributed-data algorithms, due to the hefty memory

requirements of direct integral transformation. There have been a number of parallel implementations of integral transformation and correlated methods, recently reviewed by Harrison and Shepard [30]; however, most of these implementations were still disk based and not integral direct. Márquez and Dupuis recently reported an integral direct, distributed-data parallel implementation of the MP2 energy, based on explicit message passing [31]. This algorithm however suffers from rather high-memory requirements ($\mathcal{O}(ON^3)$) and communication costs ($\mathcal{O}(N^4)$). Due to these memory requirements the number of (occupied) MOs in the first transformation index that can be dealt with in a single pass over the integral list is normally very small and a large number of passes is required already for medium-sized problems (about 150 basis functions). The parallel algorithm benefits mainly from the enlarged aggregate memory when multiple processors are used, reducing the number of necessary integral passes. This is especially evident for their "super direct" version, where the most expensive part of the calculation, i.e. the evaluation of the ERIs is replicated on each node, avoiding the costly $\mathcal{O}(N^4)$ communication of the AO integrals, what makes the algorithm non-scalable with respect to the number of processors. Speedups were reported only for the relatively modest number of 4 processors. A direct, distributed-data parallel implementation of the *orbital-invariant* MBPT(2) theory was recently reported by Bernholdt and Harrison [32]. This method involves the computation of the half-transformed exchange integrals $(iv|j\sigma)$ and the iterative solution of the amplitude equation for the double-excitation amplitudes $t_{ij}^{v\sigma}$ in a mixed MO/AO representation, which greatly increases the spatial locality of the basis functions spanning the virtual space. The dominant computational costs are ascribed to the construction of the exchange operators (i.e. ERI evaluation and transformation) and the highest communication costs ($\mathcal{O}(O^3N^2)$) occur while solving the amplitude equations. Formidable speedups were reported on an MPP system with up to 160 computational nodes.

In the present paper, we describe a scalable, distributed-data parallel implementation of direct integral transformation for a restricted set of MO indices. This algorithm is currently implemented in a conventional MP2 code (conventional in the sense that a diagonal Fock matrix is used), and we describe specifically this MP2 implementation. However, we want to emphasize here that our primary goal is the development of a scalable algorithm for direct integral transformation, a building block for parallel implementations of integral-direct CASSCF and CASPT2 algorithms; work in this direction is already in progress [33]. The maximal communication costs of our integral-direct MP2 code are $\mathcal{O}(M_\Sigma O^2N^2)$, ($M_\Sigma$ denotes the number of symmetry-unique shells), whereas the computational step of highest complexity, i.e. the transformation of the first index is $\mathcal{O}(ON^4)$. The largest integral blocks, i.e. those which contain the integrals after three-quarter transformations and the fully transformed MO integrals, are distributed over all computational nodes. The communication framework is implemented on top of *global arrays* [34, 35], a communication library which supports *one-sided access* to two-dimensional, distributed arrays by the use of interrupt-driven message passing on the IBM SPX, and provides also mechanisms to take nonuniform memory access into account, at the level of the application program. In order to exploit molecular symmetry and to obtain maximal vectorization already in the first transformation step, the AO integrals are symmetry-adapted prior to the transformation ($\rightarrow$ *SO integrals*). All subsequent integral transformation steps then are efficient vector operations on SO integral symmetry subblocks, each identified by three irreducible representations (irreps) of the molecular point group.

Very recently and independently from our work, Wong et al. (WHR) developed an efficient, distributed-data parallel direct four-index transformation scheme [36], based on similar ideas as is the present algorithm. However, there are significant differences to our algorithm. Moreover, no attempts were made to exploit molecular symmetry.

In the following section we first outline briefly the computational problem and describe the sequential form of our algorithm, which was implemented in different variants. In Sect. 3 we discuss the data and task distribution of the parallel algorithm. In Sect. 4, preliminary data on the performance of the algorithm is presented and analysed.

2 The sequential algorithm

2.1 Computational problem

The MP2 contribution to the correlation energy for a closed-shell system can be written in spin-free formalism as

$$E^{(2)} = \sum_{i,j,a,b} \frac{(ia|jb)^2 + \frac{1}{2}[(ia|jb) - (ib|ja)]^2}{\varepsilon_i + \varepsilon_j - \varepsilon_a - \varepsilon_b} \tag{1}$$

where i, j and a, b denote occupied and virtual, canonical MOs, respectively, and ε_i, ε_j, ε_a, ε_b the corresponding eigenvalues of the Fock matrix. The MO exchange integrals $(ia|jb)$ are computed from a transformation of the AO or SO ERIs over restricted MO index ranges, i.e.

$$(ia|jb) = \sum_{\mu,\nu,\lambda,\sigma} C_{\mu i} C_{\nu a} C_{\lambda j} C_{\sigma b} (\mu\nu|\lambda\sigma), \tag{2}$$

where, in the present context, μ, ν, λ and σ denote SOs, and the MO coefficient matrix $\boldsymbol{C}$ transforms from the SO to the MO basis. In order to keep the flop count as low as possible the integral transformation usually is carried out in four subsequent steps, with one index transformed after the other. In the following we will refer to the transformation of the 1st, 2nd, 3rd and 4th index as the Q1, Q2, Q3 and Q4 step, respectively. Accordingly, the Q1 *block* contains the integrals $(i\nu|\lambda\sigma)$, the Q2 *block* the integrals $(ia|\lambda\sigma)$, and so on, while the *SO block* comprises the untransformed SO integrals $(\mu\nu|\lambda\sigma)$. Using this notation, the Q1 step, scaling as $\mathcal{O}(ON^4)$ with the problem size, dominates the integral transformation. The evaluation of the SO ERIs requires $\mathcal{O}(N^4)$ floating point operations (flops). Antisymmetrization and summation according to Eq. (1) are all $\mathcal{O}(O^2N^2)$ and not rate limiting.

The main obstacle in direct integral transformation schemes is the vast amount of memory, which is necessary to hold the partially transformed ERIs. For example, the Q1 block after full completion of the Q1 step requires $\mathcal{O}(ON^3)$ memory, as in the algorithm presented by Márquez and Dupuis [31]. In direct transformations with two indices limited to the occupied orbital space as for MP2 energy calculations, this can be reduced to $\mathcal{O}(O^2N^2)$, if the Q2 and Q3 steps already are carried out, *before* all SO integrals for the Q1 step are generated [16, 17]. Furthermore, if the available memory still is exceeded, it is possible to segment the first MO index range with the limiting case of a single i and the corresponding memory requirement of $\mathcal{O}(ON^2)$. However, this implies *multiple passes through*

the integral list (worst case: O), thus it is desirable to have as large segments of i as possible.

2.2 Sequential implementations

The integral direct MP2 algorithm presented here does not fully utilize the permutational symmetry of the ERIs: $\mu\nu \leftrightarrow \lambda\sigma$ is not exploited. For a sequential integral direct implementation the higher memory requirements of such algorithms as proposed by Werner and Meyer [37] would imply (i) multiple passes over the integral list, and (ii) considerable loss in vectorization for the integral transformation steps. Furthermore, these algorithms are not well suited for parallel implementation as discussed by Wong et al. [36].

The skeleton of our integral direct MP2 algorithm, as outlined in Fig. 1, is similar to the one described by Head-Gordon et al. [17]: The outermost loop is segmenting the first MO index range in as large chunks as possible (given by the available memory) and determines the number of passes through the integral list. The next four nested-loop structures each run over symmetry-unique shells. An SO ERI batch then is defined by a symmetry-unique shell quadruple and comprises the evaluation of all AO ERIs belonging to those shell quadruples, generated from the symmetry-unique ones by application of the double-coset representatives [38]. For the computation of the AO ERIs routines from the SEWARD integral generator

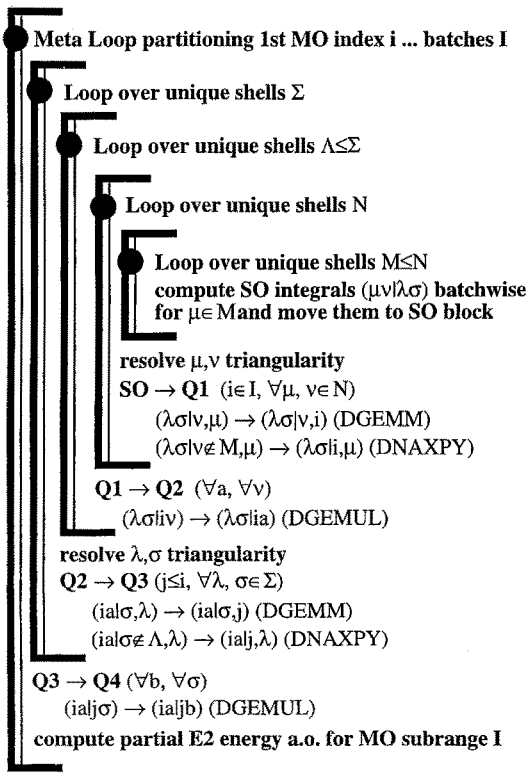

Fig. 1. Nested loop structure of the sequential MP2 algorithm (variant $[(\mu \leftrightarrow \nu), (\lambda \leftrightarrow \sigma)]$, which exploits the permutational symmetry of both pair entities). The outermost loop is segmenting the first MO index range in as large chunks as possible (which is given by the available memory) and determines the number of passes through the AO integral list. The next four nested loop structures each run over symmetry-unique shells

Table 1. Memory requirements for the different ERI blocks during the different stages of the direct integral transformation. α, β, γ denote irreps, N^α and S_K^α are the total number of SOs and the number of SOs within the symmetry-unique shell K, which belong to irrep α. I^α and O^α denote the number of occupied MOs in the first (ev. segmented) and third index, respectively, while V^α stands for the number of virtual MOs, all corresponding to irrep α. The second entry for the SO and Q2 block corresponds to the algorithms with segmented Q1 or Q3 step. The order of the flop count of the related step and the locality of the data for the parallel implementation are also indicated

IBlk.	Shells fixed	Req. mem	Flop count	Locality
SO	3 ; 4	$\sum_{\alpha\beta\gamma} N^{\alpha\otimes\beta\otimes\gamma} S_\Sigma^\alpha S_\Lambda^\beta S_N^\gamma$; $\sum_{\alpha\beta\gamma} S_M^{\alpha\otimes\beta\otimes\gamma} S_\Sigma^\alpha S_\Lambda^\beta S_N^\gamma$	$\mathcal{O}(N^4)$	Local
Q1	2	$\sum_{\alpha\beta\gamma} N^{\alpha\otimes\beta\otimes\gamma} I^\gamma S_\Sigma^\alpha S_\Lambda^\beta$	$\mathcal{O}(ON^4)$	Local
Q2	1 ; 2	$\sum_{\alpha\beta\gamma} N^{\alpha\otimes\beta\otimes\gamma} V^\beta I^\gamma S_\Sigma^\alpha$; $\sum_{\alpha\beta\gamma} S_\Lambda^{\alpha\otimes\beta\otimes\gamma} V^\beta I^\gamma S_\Sigma^\alpha$	$\mathcal{O}(OVN^3)$	Local
Q3	0	$\sum_{\alpha\beta\gamma} N^{\alpha\otimes\beta\otimes\gamma} O^\alpha V^\beta I^\gamma$	$\mathcal{O}(O^2VN^2)$	Global
Q4		$\sum_{\alpha\beta\gamma} V^{\alpha\otimes\beta\otimes\gamma} O^\alpha V^\beta I^\gamma$	$\mathcal{O}(O^2V^2N)$	Global

[39] are used. As in Ref. [17] the Q2 and Q3 steps are performed before the Q1 step is completed, yet we prefer to generate all SO ERI batches for a fixed, symmetry-unique shell triplet before the corresponding Q1 step is performed, and furthermore to produce all half transformed integrals for a single fixed, symmetry-unique shell index before the corresponding Q3 step is carried out. This increases the vector length at the cost of extra memory for the SO and Q2 blocks. However, since the memory, allocated by the SO, Q1 and Q2 blocks in the previous steps, all is reused for the Q4 block during the Q4 step, which consumes the largest amount of memory anyway (the Q3 + Q4 blocks are involved here), this is of no further consequences. Table 1 shows the memory requirements for the individual ERI blocks.

We have implemented several distinct variants of the algorithm. The first one, which is schematized in Fig. 1, takes advantage of the triangularity of the individual pair entities, i.e. it utilizes $\mu \leftrightarrow \nu$ and $\lambda \leftrightarrow \sigma$ permutational symmetry of the SO integral $(\mu\nu|\lambda\sigma)$. A single Q1 or Q3 step for an individual symmetry subblock then consists of one DGEMM (matrix multiply/add) and $S_K(N - S_K)O$ DAXPYs (matrix scale/add) operations (with N, S_K and O having the same meaning as in Table 1), while the Q2 and Q4 steps are simple matrix multiplications (DGEMULs). As already mentioned above, $\mu\nu \leftrightarrow \lambda\sigma$ permutational symmetry cannot be exploited in the present transformation scheme: because the shell indices Σ and Λ are fixed the Q1 block contains no integrals $(\mu\nu|i\sigma)$ or $(\mu\nu|\lambda i)$ with μ, $\nu \notin \Sigma, \Lambda$. Hence, in a single pass through the integral list the non-redundant ERIs are effectively computed twice. For further reference in this paper we introduce here the label $[(\mu \leftrightarrow \nu), (\lambda \leftrightarrow \sigma)]$ for this variant of the algorithm.

If $\mu \leftrightarrow \nu$ permutational symmetry is abandoned, the DGEMM and DAXPY operations of the Q1 step can be replaced by a single DGEMUL, which results in improved vectorization of this step, although at the expense of evaluating the non-redundant ERIs effectively four times. Since the Q1 step comprises $\mathcal{O}(ON^4)$ flops, while ERI evaluation is "only" $\mathcal{O}(N^4)$, this is not unattractive at a first sight, especially for systems with a large number of atoms, and we use the label $[(\mu \not\leftrightarrow \nu), (\lambda \leftrightarrow \sigma)]$ for this version of the algorithm.

Furthermore, since the Q2 step does not include any DAXPY operations, the Q3 step can be carried out immediately after the Q2 step for the subrange $\lambda \in \Lambda$ (Fig. 2). This reduces the vector length in the Q3 step, but also reduces the size of

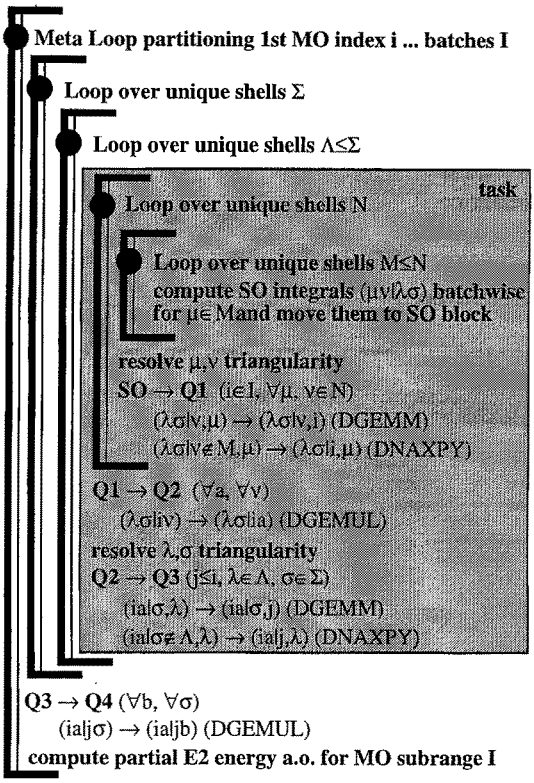

Fig. 2. Nested loop structure of the sequential MP2 algorithm (variant $([(\mu\leftrightarrow\nu), (\lambda\leftrightarrow\sigma)])_{\text{seg(Q3)}}$, which exploits the permutational symmetry of both pair entities, and performs a segmented Q3 step, i.e. over $\lambda \in \Lambda$, rather than $\forall\lambda$). The outermost loop is segmenting the first MO index range in as large chunks as possible (which is given by the available memory) and determines the number of passes through the AO integral list. The next four nested loop structures each run over symmetry-unique shells. The shaded region corresponds to a parallel task, which is defined by the symmetry-unique shell doublet $(\Sigma, \Lambda \leqslant \Sigma)$

the Q2 block (cf. Table 1), what is essential for the distributed parallel algorithm, as pointed out in Sect. 3. This version, which exploits triangularity of both pair entities, is denoted as $[(\mu\leftrightarrow\nu), (\lambda\leftrightarrow\sigma)]_{\text{seg(Q3)}}$.

Table 2 compiles timing results of the three different variants of the algorithm for calculations on *s*-tetrazine ($C_2H_2N_4$, D_{2h} symmetry, 406 primitives, 94 contracted functions, 30 correlated electrons), and phenanthrene ($C_{14}H_{10}$, C_{2v} symmetry, (672 primitives, 216 contracted functions, 94 correlated electrons). The corresponding CPU times for the disk-based MP2 algorithm are also included, for comparison. For the phenanthrene calculation, using a workspace of 12 MW (double precision), two integral passes were necessary and only the timings of the first one are given. From Table 2 the following is evident:

(i) The total time for integral direct MP2 energy evaluation is entirely dominated by the generation of the ERIs (t_{int}), i.e. 99% and 98% for *s*-tetrazine and phenanthrene, respectively. Hence, the large prefactor of the $\mathcal{O}(N^4)$ flop count for ERI evaluation outweighs the $\mathcal{O}(ON^4)$ dependence of the Q1 step. A similar behaviour was reported by Sæbø and Almlöf for their implementation [16].

(ii) The ratios $t_{\text{int}}(\text{direct})/t_{\text{int}}(\text{disk-based})$ are as expected, i.e. somewhat less than 2 for the $[(\mu\leftrightarrow\nu), (\lambda\leftrightarrow\sigma)]$ implementations, and somewhat less than 4 for the $[(\mu\not\leftrightarrow\nu), (\lambda\leftrightarrow\sigma)]$ version.

(iii) The total time spent for the four transformation steps is dominated by the Q1 step (t_{Q1}). The gain in vectorization achieved when $\mu\leftrightarrow\nu$ permutational symmetry is abandoned, decreases t_{Q1} by a factor of about 2. However, since t_{int} dominates

Table 2. CPU times (in s) for the different sequential versions of the integral direct MP2 algorithm, obtained on an IBM RS/6000 590. The CPU times are split into the individual contributions of SO ERI evaluation (t_{int}), and the Q1, Q2, Q3, Q4 transformation steps, respectively. Timings from calculations on *s*-tetrazine (406 primitives, 94 contracted functions, 30 correlated electrons)[a] and phenanthrene (672 primitives, 216 contracted functions, 94 correlated electrons)[b] are given. The corresponding CPU times for disk based MP2 energy evaluation are also given, for comparison

	s-tetrazine	Phenanthrene
Symmetry	D_{2h}	C_{2v}
N/O	94/15	216/47
Passes	1	2
$[(\mu\leftrightarrow\nu), (\lambda\leftrightarrow\sigma)]$		
t_{int}	684.9	9960.2
t_{Q1}	5.4	159.6
t_{Q2}	0.4	15.3
t_{Q3}	0.4	18.4
t_{Q4}	0.0	5.3
$[(\mu\not\leftrightarrow\nu), (\lambda\leftrightarrow\sigma)]$		
t_{int}	1226.6	19362.2
t_{Q1}	2.3	93.5
t_{Q2}	0.3	15.7
t_{Q3}	0.2	26.4
t_{Q4}	0.0	7.0
$[(\mu\leftrightarrow\nu), (\lambda\leftrightarrow\sigma)]_{seg(Q3)}$		
t_{int}	684.4	10166.5
t_{Q1}	5.1	143.7
t_{Q2}	0.4	15.0
t_{Q3}	0.8	62.3
t_{Q4}	0.0	7.2
Disk based		
t_{int}	387.6	5772.6
t_{Q1-4}	4.16	417.9

[a] ANO contracted as H(8*s*4*p*/2*s*1*p*), and C,N(14*s*9*p*4*d*/3*s*2*p*1*d*) [44] (real spherical representation)
[b] ANO contracted as H(7*s*/2*s*), and C(10*s*6*p*3*d*/3*s*2*p*1*d*) [45] (real spherical representation)

the whole calculation, the $[(\mu\leftrightarrow\nu), (\lambda\leftrightarrow\sigma)]$ variants are definitely more attractive. (iv) Performing a segmented Q3 step immediately after the Q2 step rather than to complete the Q2 step for $\forall\lambda$; $\sigma\in\Sigma$ and to execute the Q3 step for the whole λ range, increases t_{Q3} by a factor of about 2–3. However, since t_{Q3} is insignificant anyway, this is not harmful.

For the two chemical systems considered in Table 2 the first iteration of a direct SCF calculation takes 358 and 4614 CPUs, respectively. The differences in

execution time between direct SCF and disk-based integral generation are mainly due to better prescreening rates in the case of the direct SCF, since densities can also be taken into account for the calculation of the screening criterion. Moreover, since the two-electron part of the Fock matrix is constructed directly in the AO basis, there is no need for symmetry adaptation of the ERIs. Anyway, the MP2 energy is obtained for an additional fraction of the cost of the preceding direct SCF calculation.

2.3 Molecular symmetry

The efficiency and memory requirements of two-electron integral transformation benefit substantially from exploitation of molecular symmetry. As already mentioned above, our codes *symmetry-adapt* the AO ERIs in a preceding step, and the Q1, Q2, Q3, Q4 steps transform from SO to MO indices. The SO, Q1, Q2, Q3 and Q4 blocks are decomposed into individual 3*L-subblocks*, each uniquely defined by three symmetry labels L_i, $i = 1, \ldots, 3$, formed from the irreps Γ_i, $i = 1, \ldots, 3$ of the molecular point group $G \subseteq D_{2h}$, which belong to those indices, not being transformed in the next step: The first label L_1 is defined as the direct product $\Gamma_1 \otimes \Gamma_2 \otimes \Gamma_3$ of all three irreps Γ_i, $i = 1, \ldots, 3$, and is equal to the irrep, which represents the index that is transformed in the next step. The second label L_2 is formed as the direct product $\Gamma_1 \otimes \Gamma_2$ of only two irreps, while the last one L_3 is equal to the first irrep Γ_1 (cf. Fig. 3).

The distinct 3L-subblocks are then transformed independently by DGEMULs or DGEMMs and DAXPYs, addressing contiguous memory within the corresponding source and target 3L-subblocks, as indicated in Fig. 3 for the Q1 step. This allows for efficient use of cache memory without any need for intermediate buffering. The maximum number of 3L-subblocks is g^3, where g is the

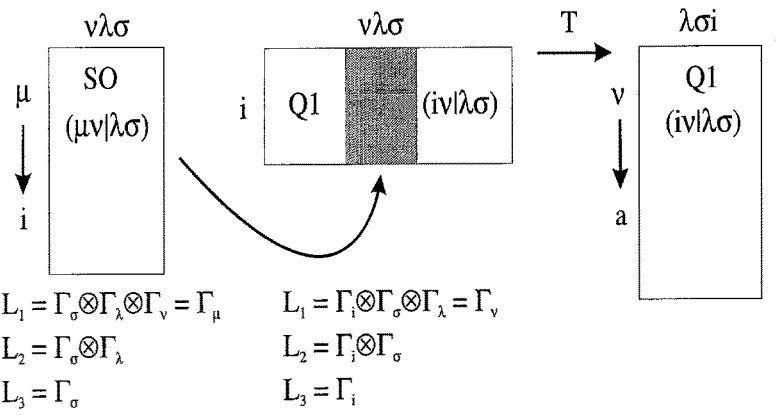

Fig. 3. Schematic representation of the Q1 step for a SO 3L-subblock. The SO and the corresponding target Q1 3L-subblock both are defined by three labels L_1, L_2, L_3, formed from the irreps of those SOs (MOs), which are not transformed in the current step. During the transformation, contiguous memory (shaded area) in the target Q1 3L-subblock is addressed, allowing for efficient use of cache memory. After the transformation, the Q1 3L-subblock is transposed, in order to expose ν as the fastest index for the Q2 step

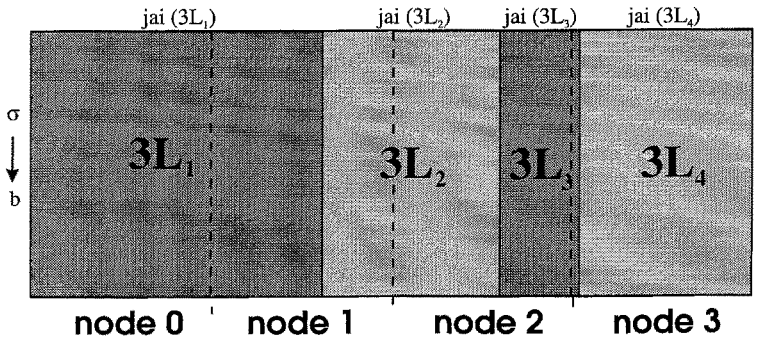

Fig. 4. Schematic representation of a 1L-subblock, which is distributed evenly over different nodes (global array). The 1L-subblock, defined by the single irrep L_1, contains all 3L-subblocks with the same leading dimension N_{L_1}. The boundaries of the 3L-subblocks usually do not coincide with the boundaries of the local patches of the 1L-subblock

order of the group G. Individual 3L-subblocks with the same direct product $L_1 = \Gamma_1 \otimes \Gamma_2 \otimes \Gamma_3$ all have the same leading dimension N_{L_1} and are gathered into a single 1L-subblock with symmetry label L_1 and leading dimension N_{L_1}. The maximum number of such 1L-subblocks. is g. Individual 3L-subblocks form column blocks within the corresponding 1L-subblocks (cf. Fig. 4). This grouping of 3L-subblocks is of no further importance for the sequential codes, but is used for the distribution of the data in the parallel algorithm, as discussed below.

2.4 Integral prescreening

For generally contracted ANO-type basis sets [40], it is advantageous to prescreen ERIs in the uncontracted AO basis, since all contracted functions of a shell share a common set of Gaussian exponents. Hence, prescreening in the contracted basis would imply that even if most of the contracted ERIs of a given shell quadruple could be neglected, still *all* primitive ERIs would remain to compute. The size of an individual ERI can be estimated using the Cauchy–Schwarz inequality

$$|(\alpha\beta|\gamma\delta)| \leqslant |(\alpha\beta|\alpha\beta)^{1/2}|(\gamma\delta|\gamma\delta)|^{1/2}, \tag{3}$$

where α, β, γ, δ are primitive basis functions in the present context. If ERI prescreening is performed at the level of primitive functions, care has to be taken that the resulting computational overhead does not become too large. For this reason the prescreening in our codes is done on pair entities $P(\alpha\beta)$ in a k^2 (two-index) setup loop over $(\alpha\beta)$ *before* the k^4 loop structure for ERI generation and transformation is entered. The result of the prescreening in the k^2 loop is a cut down of the range of the compound index $\zeta \doteq (\alpha\beta)$, and of the vectors containing the pair entities $P(\zeta)$, which are used later in the k^4 loop for the evaluation of the ERIs. Moreover, all the memory, allocated for the Cauchy–Schwarz integrals $(\alpha\beta|\alpha\beta)$ during the prescreening of the pair entities, can be released again before entering the k^4 loop structure. The actual prescreening criterion $U(\alpha, \beta, \gamma, \delta)$ for the

primitive ERI $(\alpha\beta|\gamma\delta)$ is computed as

$$U(\alpha, \beta, \gamma, \delta) = \max(c_{\alpha\mu}, \mu \in \mathrm{M}) \max(c_{\beta\nu}, \nu \in \mathrm{N}) |(\alpha\beta|\alpha\beta)|^{1/2} \times \max(c_{\gamma\lambda}, \forall\gamma, \forall\lambda) \max(c_{\delta\sigma}, \forall\delta, \forall\sigma) \max\{|(\gamma\delta|\gamma\delta)|^{1/2}, \forall\gamma, \forall\delta\}, \quad (4)$$

where $\mu, \nu, \lambda, \sigma$ denote contracted AOs here, M, N are shells, and $c_{\alpha\mu}, c_{\beta\nu}, c_{\gamma\lambda}, c_{\delta\sigma}$ are contraction coefficients. In Eq. (4), $\max(c_{\alpha\mu}, \mu \in \mathrm{M})$ stands for the maximum value of $c_{\alpha\mu}$ for a given primitive α over all contracted functions μ, belonging to shell M. On the other hand, $\max(c_{\gamma\lambda}, \forall\gamma, \forall\lambda)$ means the largest contraction coefficient over all primitives and contracted functions of all shells.

Besides the ERI prescreening scheme described above, there is an additional screening of SO integrals before the Q1 step of the transformation is performed. Near zero row-blocks over subranges of μ are removed from the corresponding SO 3L-subblock and the related MO coefficient matrix is reduced accordingly. This improves the efficiency of the Q1 step for extended systems with low symmetry, although is not crucial for the overall performance of the code.

3 The distributed parallel algorithm

The major goals that are to achieve for scalable parallel algorithms are to exploit the computational power as well as the aggregate memory, which both increase linearly with the number of processing nodes. This is particularly true for direct integral transformation algorithms, where there is a direct dependence of the execution time on the available memory, as discussed in the previous section. Exploiting both the combined processing and memory resources of a parallel computer for such algorithms implies, that *superlinear speedups* can be obtained, as shown below, and in Refs. [31, 36]. In the previous section it was demonstrated that the computationally dominant step in the calculation manifests in the evaluation of the AO ERIs (cf. Table 2). The Q1 step, due to its fifth power dependence on the system size, may become important for extended systems with many electrons. It is therefore important to distribute the evaluation of the AO ERIs and the Q1 step efficiently over all computational nodes. Furthermore, from Table 1 it is evident, that the memory requirements are dominated by the Q3 and Q4 integral blocks, which both depend quadratically and N (number of basis functions) or V (number of virtual orbitals), whereas the SO, Q1 and Q2 blocks (for the $[(\mu\leftrightarrow\nu), (\lambda\leftrightarrow\sigma)]_{\mathrm{seg(Q3)}}$ version) are only linear in these entities. It is thus essential to distribute the Q3 and Q4 blocks over the bulk memory of all the nodes, while the SO, Q1 and Q2 blocks can be kept in local memory of each individual node.

3.1 Data distribution and global arrays

Our distributed parallel implementation of integral direct MP2 derives from the $[(\mu\leftrightarrow\nu), (\lambda\leftrightarrow\sigma)]_{\mathrm{seg(Q3)}}$ variant of the sequential algorithm, which was discussed in the previous section (cf. Fig. 2), yet triangularity of the MO indices i and j is not exploited. This facilitates the distribution of the Q3 and Q4 blocks, although at the expense of a factor of two in the memory requirements of these entities. The SO, Q1, Q2 blocks and the MO coefficients are private to each computational node, whereas the Q3 and Q4 blocks are distributed evenly over all nodes, using the

concept of global arrays. We note here that the inclusion of the Coulomb integrals into this transformation scheme would inflict only minor additional requirements for local (non-distributed) memory, i.e. $\mathcal{O}(IOS_\Sigma S_A)$.

Due to molecular symmetry the Q3/Q4 blocks are decomposed into g^3 3L-subblocks, where g is the order of the molecular point group. Since the maximum number of these 3L-subblocks can become rather large (e.g. $8 \cdot 8 \cdot 8 = 512$ in the case of D_{2h} symmetry), distributing individual 3L-subblocks would impose substantial administrative overhead: $2g^3$ global arrays would have to be created and controlled. For this reason, all 3L-subblocks with the same leading dimension are collected into a single 1L-subblock, as described in the previous section, and the distribution of the Q3/Q4 data is done at the level of these 1L-subblocks, resulting in a total of $2g$ distinct global arrays. Individual 1L-subblocks are distributed evenly over all nodes with each node holding a column-block of the original 1L-subblock in local memory (cf. Fig. 4). Note that the boundaries of such a local patch of a global array usually do not coincide with the boundaries of the 3L-subblocks, which also form column-blocks within the corresponding 1L-subblock.

The underlying communication framework to access remote patches of global arrays is built on top of the Global Array (GA) toolkit of Nieplocha et al. [34, 35]. This toolkit provides a virtual "shared memory" programming model for two-dimensional arrays, yet non-uniform memory access is not hidden from the application program. *One-sided access* to remote patches without any need for cooperation between the computational threads on the data-requesting and the data-owning nodes is provided either by forking a shared-memory dataserver on each node, or by installing a data serving interrupt handler, as is the case on the IBM SP2, which allows for interrupt driven message passing. One-sided access of remote data eliminates unnecessary synchronization between nodes, implying enhanced performance relative to conventional message-passing schemes. A number of one-sided communication/operation primitives like "get", "put", "accumulate", "atomic read and increment", "gather" and "scatter" are provided with the toolkit.

In cases of high-symmetry and large basis sets, i.e. if there are centers with four- or eightfold degeneracy, it can happen, that for certain shells the size of the entire SO, Q1 or Q2 blocks, as defined in Table 1, exceeds the available local memory. In such cases, a first step is to perform the Q1 step immediately after the generation of the SO integrals (four shells fixed), rather than to postpone this step, until all SO integrals are generated for a fixed shell triplet. This cuts down the size of the SO block at the expense of vector length in the Q1 step. Note, that the Q1 block cannot be reduced in a similar way, i.e. by keeping three shells fixed rather than two, without abandoning $\mu \leftrightarrow \nu$ permutational symmetry. An alternative way to reduce the size of the SO, Q1 and Q2 blocks altogether, is to redefine those symmetry-unique shells K, which are *critical*: Functions belonging to such shells are *decontracted*, part of the evaluation of the ERIs is performed in the primitive basis, and the contraction matrices are absorbed into the MO coefficient matrix. For example, for a d-shell with three contracted functions the size of the shell S_d then reduces from 15 to 5, and the related SO, Q1 and Q2 blocks decrease by factors of 27 and 9, respectively. No redundant evaluation of primitive ERIs is imposed; however, the transfer equations [39] are now partly applied in the primitive, rather than the contracted basis, making that part of the ERI evaluation less efficient. Note, that, if at all, only few critical shells (usually those with highest angular momentum, belonging to few, critical centers) are decontracted, in order not to

blow up the number of basis functions N, and to keep ERI evaluation as efficient as possible.

3.2 Task distribution and load balancing

ERI generation and transformation up to the Q3 step are split up into individual *tasks*, which are distributed over the computational nodes and executed in parallel. A single task, indicated as shaded area in Fig. 2, corresponds to a single, symmetry-unique shell doublet Σ, $\Lambda \leqslant \Sigma$, and loops over all symmetry-unique shells N, M $\leqslant$ N. Hence, if the number of symmetry-unique shells Σ is denoted by n_Σ, then the number of individual tasks is equal to $n_\Sigma(n_\Sigma + 1)/2$, and grows quadratically with increasing size of the chemical system under study. For the rather modest calculation on phenanthren (cf. Table 2), the number of tasks amounts already to 351, which is sufficient for reasonable load balancing on a fair number of nodes. Note, that the use of $\mu\nu \leftrightarrow \lambda\sigma$ permutational symmetry would lead to a different number of symmetry-unique shell doublets N, M for each task, thus to completely inhomogenous task lengths (i.e. task flop counts).

According to Fig. 2 a single task comprises the generation of all ERIs with fixed Σ, Λ, and the subsequent transformation of three indices (two occupied, one virtual). The ERI generation, and the Q1 and Q2 transformation steps involve no interprocessor communications at all, since the whole SO, Q1, Q2 blocks and MO coefficient matrix are all local to each node. The Q3 step though requires write access to the global Q3 arrays: the actual, segmented transformation is carried out using local buffer space of size $\mathcal{O}(IVO)$, with a subsequent one-sided "accumulate" to the proper patches within the global Q3 arrays. All the memory, used before for ERI evaluation, SO and Q1 block, is reused again for this local Q3 buffer. Due to the one-sided character of the Q3 accumulate operation, all tasks are executed asynchronously and the first synchronization point is arrived not before all ERIs are generated and the Q3 step is completed. The communication costs in the Q3 step are formally $\mathcal{O}(n_\Sigma O^2 VN)$; however, substantial savings can be achieved, when locality of Q3 data is exploited (see below). Note that in difference to the algorithm of WHR [36] the Q3 step is performed here *before* the full set of Q2 integrals is generated, with the consequence that the Q2 integrals still can be kept in local memory. This is especially valuable in the context of super-CI-based MCSCF algorithms, where only a very small subset of MO integrals (three indices in the active space) is required [41]. For such an algorithm the actual communication costs of the transformation decrease to $\mathcal{O}(n_\Sigma A^3 N)$ with $A < O$.

Figure 5 schematizes the general structure of our distributed parallel MP2 implementation: in a preliminary step before the k^4 loop is entered, each node creates a *private task list* by enumeration of all symmetry-unique shell doublets $\Sigma, \Lambda \leqslant \Sigma$. This task list is sorted afterwards with respect to a task criterion, which depends (i) on the assessed *task length* (i.e. long tasks have priority, while short tasks are used for padding towards the end), and (ii) on the *locality* of each task. The individual task length is estimated by the number of primitive pairs that belong to the corresponding Σ, $\Lambda \leqslant \Sigma$ shell doublet. Since the prescreening of ERIs is based on primitive pair entities, and is already accomplished at that point (see above), the estimate of the task length is actually based on the *genuine number* of primitive pairs that remain *after the prescreening*. We note also, that the mechanism to decontract some critical shells, as outlined above, splits some large tasks into smaller pieces. As a side-effect, this improves also the load-balancing, since a

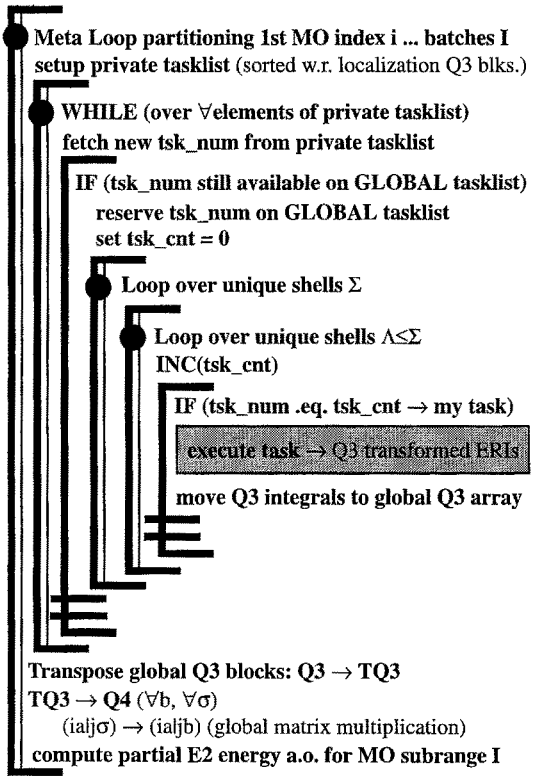

Fig. 5. Nested loop structure of the parallel MP2 algorithm. The outermost loop is segmenting the first MO index range in as large chunks as possible (which is primarily given by the available aggregate memory of the MPP system) and determines the number of passes through the AO integral list. The WHILE/IF construct trys to register the next task number of the private task list with the GLOBAL task list. After successful registration, the usual nested loop structure over symmetry-unique shells is entered, and the corresponding task is executed. The shaded box in the figure represents the shaded area of Fig. 2

larger number of small tasks for padding towards the end of the task list then is available.

The task locality, on the other hand, is defined as the ratio of the summed up sizes of the local Q3 patches, and the total Q3 patches, which will be accessed by a given task. The position of a given task in the list then reflects its priority for a given node. Task lists of different nodes usually look pretty different. Each node then works its way through its private task list, fetching the next task number (tsk_num) and tries to reserve it on the *global task list*. The global task list is held in a global array and accessed by the individual nodes using "atomic read and increment" operations. A given task is interpreted as already reserved by another node, if the corresponding read out is different from zero. After successful reservation of a task on the global task list the k^4 loop is entered and the corresponding task (shaded area in Fig. 5) is executed. When a node has finally reached the end of its private task list, it arrives at a synchronization point (barrier), which is necessary to ensure data consistency, before the Q4 step is performed. The mechanism of reserving individual tasks in a specific order on a global task list allows for dynamic load balancing in a "self-service" way, without any need for a special master or control process. Moreover, giving the tasks a priority relative to the locality of the data, which will be accessed, renders savings in interprocessor communications.

Before the Q4 step is performed, it is advantageous to *transpose* the individual Q3 3L-subblocks from (jai, σ) to (σ, jai), so that the last remaining SO index σ becomes the fastest. The succeeding Q4 step is then completely local, carried out as

local DGEMULs for each local slice of the individual Q3 3L-subblocks, without any interprocessor communications involved, at all. This transpose can either be performed on the fly immediately after the segmented Q3 step by replacing the blockwise "accumulate" operations of the local Q3 buffer to the global Q3 array by "scatter/accumulate" operations, or alternatively, as separate "scatter" operations after completion of all tasks, when the whole set of Q3 integrals is generated. The communication costs for the former are $\mathcal{O}(n_\Sigma O^2 V N)$ (which have to be spent anyway for the accumulate), while for the latter an extra expense $\mathcal{O}(O^2 V N)$ is imposed. On the other hand, "scatter" operations are at least twice as expensive as simple "get", "put" or "accumulate" operations, since two supplementary index arrays have to be transmitted in addition to the actual data. Furthermore, "scatter" operations imply also a rather expensive sorting of the data with respect to the corresponding target processor ID, in order to bundle data items targeting the same processor and so to avoid latency. We note here in passing that a regular sort algorithm as included in the GA tools (version 2.1), with the number of compares depending on the number of sort items n as $\mathcal{O}(n \log(n))$, is suboptimal for the sort problem in the "scatter" and "gather" operations. In the context of the present work this was replaced by a more efficient algorithm, based on exchanging items along corresponding permutational cycles, which scales linearly with a very small prefactor and outperforms the regular sort by a factor of 10–20 for 10^6 items.

Considering the costs of a "scatter" operation, it is clear that the second variant of the transpose minimizes the expenses for communication. Moreover, the locality of the blockwise "accumulate" after a segmented Q3 step is not destroyed.

A further alternative would be to proceed directly to the Q4 step, omitting the intermediate transpose. The Q4 step is non-local in this case, inflicting communication costs $\mathcal{O}(pO^2V^2)$, where p is the number of computational nodes sharing an individual Q3 3L-subblock (local presummation over the slowest σ SO index on each processor with subsequent blockwise "accumulate"). Yet, no expensive "scatters" are required. Moreover, if there is symmetry, the number of processors p sharing a common Q3 3L-subblock is usually only a small subset of the whole processor range, with most processors operating in parallel on *different* Q3 3L-subblocks. Hence, this third alternative may be most efficient for high-symmetry cases and/or a modest number of computational nodes. In the present work all three variants have been implemented and explored; the relative performance is discussed in the next section.

The one-sided character of either the "scatter" or "accumulate" operation ensure asynchronity of the computational threads for both the intermediate transpose and the Q4 step, yet two (one) further barriers, i.e. after the transpose and the Q4 step are needed to guarantee data consistency. However, only the first synchronization after completion of the private task list has significant effect on the parallel efficiency of the transformation, since it reflects the granularity of the individual tasks. After that, the individual computational threads are essentially synchronized anyway.

Among the remaining steps necessary to compute the MP2 energy according to Eq. (1) only the antisymmetrization of the MO ERIs needs some consideration, the rest is straightforward. The most compact way to accomplish the antisymmetrization is of the form

$$\mathcal{A}(ia|jb) = (jb|ia) - (ja|ib), \tag{5}$$

where j is the fastest index. The antisymmetrization then proceeds over common contiguous blocks of length O (i.e. index j). Due to the segmentation of the index i, it

is not possible to perform the antisymmetrization over a common virtual leading index, e.g. as $\mathscr{A}(ia|jb) = (bj|ai) - (bi|aj)$ with contiguous blocks of length V. For every Q4 3L-subblock, each node performs the antisymmetrization for its own local share of $(jb|ia)$ ERIs, requesting the related $(ja|ib)$ integrals from the corresponding global array. The communication costs for this step are $\mathcal{O}(O^2V^2)$, yet in form of $\mathcal{O}(OV^2)$ messages, since the j-vectors of the required $(ja|ib)$ ERIs are widely scattered over the global array. In order to reduce the number of messages, and hence latency, as many of the $(ja|ib)$ ERIs as possible are prefetched and stored in a local buffer, using a one-sided "gather" operation, which bundles requests that address the same node. The computational costs are of the same order as the communication costs for this step, which is not favourable. However, since antisymmetrization takes only a small fraction of the overall time (< 5%), the overall performance of the code is not severly hampered.

4 Results and discussion

In this section, preliminary timing results on the parallel performance of the algorithm are presented and discussed. As a test case we have chosen phenanthrene again, this time in a larger basis, i.e. 762 primitives, contracted to 412 basis functions. For molecular systems of this size, integral direct methods start to become the only possible route; hence, this test case may have some practical significance, although it is still small enough to be well suited for scalability measurements. The number of symmetry-unique shells for this system is 31, forming 496 individual tasks.

All calculations reported here were performed on an IBM SP2 with 48 nodes. The processors run at a clock speed of 66.7 MHz, resulting in a peak performance of 266 MFlop/s for each node. Communication bandwidth and latency are nominally 35 Mbyte/s and 40 μs, respectively. All calculations were run under AIX 4.1, with the GA tools linked directly to the native MPI library. The SCF wave function was generated using a replicated-data parallel SCF program (distributed generation of the Fock matrix) [27], built directly on top of MPI. One feature of this SCF code is that diagonalization of the Fock matrix is avoided, i.e. replaced by orbital rotations, connected to a second-order (BFGS) update scheme [42]. This reduces the sequential backbone of the algorithm considerably.

Table 3 compiles average wall clock times of the three different variants of the parallel algorithm discussed in the previous section, i.e. (a) with no transpose of the Q3 3L-subblocks; (b) with a separate transpose after the whole set of Q3 integrals is produced; and (c) with an immediate transpose on the fly, after each segmented Q3 step. The measured times are split up into the contributions of the individual steps of the calculation (i.e. ERI generation, Q1 step, etc.). The corresponding parallel speedups t_{16}/t_{32} and the total elapsed times t_{tot} are also included, for convenience. From these, it is evident that algorithm (b) shows the best performance. Note that the time spent for the transpose is included in $\bar{t}_{\text{Q4}}$. Actually, it is the major contribution to $\bar{t}_{\text{Q4}}$, since the real Q4 step is completely local, and takes only 7 ($16p$) and 4 ($32p$) seconds, the same as for algorithm (c).

For algorithm (a), t_{tot} is somewhat longer, and it also shows an inferior speedup. This is ascribed to the more expensive Q4 step, which involves communication costs, scaling linearly with the number of processing nodes p, as discussed in the previous section. Furthermore, there is also some imbalance in the load for the Q4 step, as can be seen from the non-zero timings for the second synchronization

Table 3. Wall clock times (in s) for the different parallel versions of the integral direct MP2 algorithm for a calculation on phenanthrene (762 primitives, 412 contracted functions, 94 correlated electrons)[a]

	No Q3 transpose			Q3 transpose after Q3 step			Q3 transpose during Q3 step		
	16*p*	32*p*	t_{16}/t_{32}	16*p*	32*p*	t_{16}/t_{32}	16*p*	32*p*	t_{16}/t_{32}
$\bar{t}_{\text{int}}$	1843	922	2.00	1844	922	2.00	1844	922	2.00
t_{Q1}	190	95	2.00	190	95	2.00	190	95	2.00
$\bar{t}_{\text{Q2}}$	30	15	2.00	30	15	2.00	30	15	2.00
$\bar{t}_{\text{Q3}}$	228	115	1.98	228	115	1.98	916	471	1.94
$\bar{t}_{\text{sync1}}$	32	43		26	31		76	74	
$\bar{t}_{\text{Q4}}$	183	111	1.64	56	41	1.37	7	4	1.61
$\bar{t}_{\text{sync2}}$	26	18		0	0		0	0	
$\bar{t}_{\text{asym}}$	65	52	1.27	42	24	1.77	42	24	1.77
t_{tot}	2685	1443	1.86	2510	1298	1.93	3287	1703	1.93

[a] ANO contracted as H(7*s*3*p*/3*s*2*p*), and C(10*s*6*p*3*d*/4*s*3*p*2*d*) [45] (real spherical representation)

point. On the other hand, the maximal memory requirements for (a) are somewhat smaller than for (b), since for (b), two distributed Q3 blocks have to be kept simultaneously in core (for the transpose), whereas for (a) and for (c), only one Q3 plus one Q4 block are needed (for the Q4 step). This is especially pronounced for minimal basis sets, where the Q3 block can be twice as large as the Q4 block.

For algorithm (c), considerably longer t_{tot} were measured, although a similar speedup was obtained as for algorithm (b). Comparing $\bar{t}_{\text{Q3}}$ for (a)–(c), it is evident, that the higher costs of the "scatter" operation (vs. blockwise "accumulate"), together with the loss in data locality, increase the costs of the Q3 step dramatically, i.e. by about a factor of four. An intrinsic inscalability of all the algorithms presented here manifests in $\bar{t}_{\text{sync1}}$, the time spent idle at the first synchronization point. $\bar{t}_{\text{sync1}}$ reflects the "granularity" of the tasks, hence the load imbalance in the task assignment. The ratio $\bar{t}_{\text{sync1}}/t_{\text{tot}}$ naturally grows with increasing p, and obviously, there is an upper limit for a given problem size where a further increase of p is no longer meaningful. Similar load-balancing problems were also observed by Wong et al. [36] for their algorithm. For small, highly symmetric systems with large basis sets, this problem becomes more serious, since there are fewer tasks of larger size in these cases. It might be advantageous then to decontract some shells, as described in the previous section, in order to split some of the largest tasks into smaller pieces. This will improve the load balance, yet at the expense of efficiency in the evaluation of the ERIs.

In order to illustrate the cumulative effect of exploiting both the increased aggregate memory and compute power that are obtained by incrementing the number of computational nodes, a series of calculations on the phenanthrene molecule was performed, varying the number of nodes between 2 and 48. Algorithm (b), i.e. with a separate transpose after completion of the Q3 step was used. Table 4 compiles the measured wall clock times, the number of passes through the integral list, and the resulting speedup factors relative to two nodes. A single-node calculation still is possible, although it would take 16 passes and an estimated elapsed time of 130 h. Due to this excessively large elapsed time this calculation was not performed. However, assuming the same speedup from 1 to 2 nodes, as was

Table 4. Wall clock times (in s) and speedup factors relative to two nodes, measured for the parallel algorithm with the Q3 transpose *after* completion of the Q3 step (algorithm (b)). Calculations on phenanthrene (762 primitives, 412 contracted functions, 94 correlated electrons) were performed using 2–48 nodes, each with 12 MWords (double precision) of memory. For basis set specification see Table 3

Nodes	T_{elapsed}	# Passes	Speedup
2	122238	8	1.00
4	32045	4	3.81
8	8730	2	14.00
16	2510	1	48.69
32	1315	1	92.97
39	1109	1	110.25
48	884	1	138.21

observed from 2 to 4 nodes (3.81), the speedup on 48 nodes relative to a single node is estimated to 526.8. Using 16 nodes and more, only a single integral pass is required. Hence, beyond this limit, the algorithm scales linearly at best. The parallel efficiency on 48 vs. 16 nodes is 95%. Thus, even in the linear regime, our implementation still shows good performance. However, for a system of moderate size (ca. 400 basis functions), there are probably no practical reasons to go beyond 48 nodes, since the elapsed time is already less than 15 min. The situation may be different though, if the algorithm is used in the context of an MCSCF program, where an integral transformation is required for each iteration.

In Fig. 6 the observed speedup factors from Table 4 are plotted vs. the number of nodes, and compared to a linear speedup curve. On 48 nodes, the observed speedup is 5.8 times larger than linear speedup (with a reference of two nodes). Table 4 and Fig. 6 both demonstrate clearly, that speedup factors far beyond linear speedup (i.e. *superlinear speedup*) can be obtained, if both the aggregate memory and the compute power of an MPP system are exploited efficiently.

5 Conclusions

In this paper, we described a scalable, distributed-data parallel algorithm for integral direct four-index transformation. Our implementation was used in the context of an MP2 program, but is easily adapted for use in e.g. an MCSCF program. Molecular symmetry up to the D_{2h} point group is exploited. Calculations in excess of 1000 basis functions should be possible within reasonable elapsed times. The largest calculations performed so far comprised 1160 primitives contracted to 640 basis functions, and 74 correlated electrons. On 48 SP2 nodes, this calculation took 12281 s wall clock, and three integral passes [43].

Formally, the algorithm scales as $\mathcal{O}(ON^4)$, where O and N denote the number occupied (non-frozen) MOs, and the number of basis functions, respectively. Nevertheless, in practice, the generation of the ERIs dominates the calculation, thus the large prefactor of the flop count $\mathcal{O}(N^4)$ for ERI evaluation outweighs the $\mathcal{O}(ON^4)$ dependence of the first transformation step. Hence, improvements in the integral evaluation will have significant effects on the overall-performance of

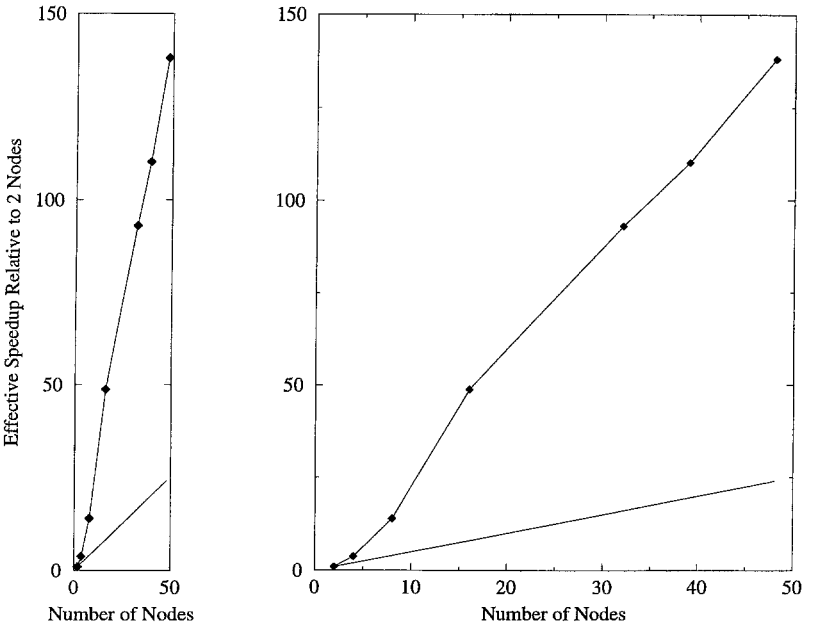

Fig. 6. Observed and linear speedup relative to two nodes, in the range of 2 and 48 SP2 nodes. The filled diamonds correspond to the timings, compiled in Table 4. The second graph displays the same data, but with a more appropriate scaling of the ordinate than the first one, which has the conventional scaling and yields the linear speedup line at 45°

the implementation. The formal communication costs are $\mathcal{O}(n_\Sigma O^2 V N)$, where n_Σ and V denote the number of symmetry-unique shells, and the number of virtual MOs, respectively. However, exploitation of data locality renders substantial savings in communication time. The communication framework is built on top of "Global Arrays", which simplifies the distribution of data considerably. Moreover, *one-sided access* to remote data by use of interrupt-driven message passing, as implemented in "Global Arrays", means a significant performance advantage, compared to conventional message passing. Some care has to be taken if global "scatter" and "gather" operations are used, since these are considerably more expensive than blockwise "get", "accumulate" and "put" operations.

Our algorithm exhibits high parallel efficiency. For a molecular system with 412 basis functions, speedup factors far beyond linear speedup were observed. Thus, the parallel execution of the calculation renders not only shorter wall clock times but also considerable savings in CPU time as compared to single-node execution. This superlinear speedup is a result of efficient use of both the compute power and the aggregate memory of the MPP system, where the latter reduces the number of necessary passes through the AO integral list. In the linear regime, i.e. when a single pass is sufficient, our implementation still exhibits good (near linear) performance. Other quantum chemical methods are likely to have a similar potential for distributed-data parallel algorithms, since the available memory, as in the case of direct MP2, is in many cases at least partly connected to the efficiency of the algorithm.

Acknowledgements. The authors would like to thank Dr. Alistair Rendell and co-authors for a preliminary version of their manuscript. This study was supported by a grant from the Swedish Natural Science Research Council (NFR), and by IBM Sweden under a joint study contract. Granted computer time from the Parallel Computer Center (PDC) at the Royal Institute of Technology (KTH), Stockholm, is gratefully acknowledged.

References

1. Price SL, Harrison RJ, Guest MF (1989) J Comput Chem 10:552
2. Scuseria GE (1991) Chem Phys Lett 176:423
3. Cioslowski J (1991) Chem Phys Lett 181:68
4. Häser M, Almlöf J, Scuseria GE (1991) Chem Phys Lett 181:497
5. Scuseria GE (1995) Chem Phys Lett 243:193
6. Furlani TR, King HF (1995) Proc quantum mechanical simulation methods for studying biological systems. Centre De Physique Des Houches, Les Houches, France
7. Foster IT, Tilson JL, Wagner AF, Shepard RL, Harrison RJ, Kendall RA, Littlefield RJ (1996) J Comput Chem 17:109
8. Harrison RJ, Guest MF, Kendall RA, Bernholdt DE, Wong AT, Stave M, Anchell JL, Hess AC, Littlefield RJ, Fann GL, Nieplocha J, Thomas GS, Elwood D (1996) J Comput Chem 17:124
9. Koch H, Christiansen O, Kobayashi R, Jørgensen P, Helgaker T (1994) Chem Phys Lett 228:233
10. Klopper W, Noga J (1995) J Chem Phys 103:6127
11. Koch H, Sánchez de Merás A, Helgaker T, Christiansen O (1996) J Chem Phys 104:4157
12. Almlöf J (1995) In: Yarkony DR (ed.), Modern electronic structure theory, Advanced Series in Physical Chemistry, Vol. 2. World Scientific, Singapore, p 110
13. Almlöf Jr J, Faegri K, Korsell K (1982) J Comput Chem 3:385
14. Taylor PR (1987) Int J Quantum Chem 31:521
15. Frisch M, Ragazos IN, Robb MA, Schlegel HB (1992) Chem Phys Lett 189:524
16. Sæbø S, Almlöf J (1989) Chem Phys Lett 154:83
17. Head-Gordon M, Pople JA, Frisch MJ (1988) Chem Phys Lett 153:503
18. Frisch MJ, Head-Gordon M, Pople JA (1990) Chem Phys Lett 166:275
19. MPI: A Message-Passing Interface standard. (1994) MPI University of Tennessee
20. Dupuis M, Watts JD (1987) Theor Chim Acta 71:91
21. Guest MF, Sherwood P, van Lenthe J (1993) Theor Chim Acta 84:423
22. Brode S, Horn H, Ehrig M, Moldrup D, Rice J, Ahlrichs R (1993) J Comput Chem 14:1142
23. Feyereisen MW, Kendall RA (1993) Theor Chim Acta 84:289
24. Feyereisen MW, Kendall RA, Nichols J, Dame D, Golab JT (1993) J Comput Chem 14:818
25. Lüthi HP, Almlöf J (1993) Theor Chim Acta 84:443
26. Lüthi HP, Mertz JE, Feyereisen MW, Almlöf J (1992) J Comput Chem 13:160
27. Schütz M, unpublished.
28. Clementi E, Corongiu G, Detrich J, Chin S, Domingo L (1984) Int J Quantum Chem: Quantum Chem Symp 18:601
29. Furlani TR, King HF (1995) J Comput Chem 16:91
30. Harrison RJ, Shepard R (1994) Ann Rev Phys Chem 45:623
31. Márquez AM, Dupuis M (1995) J Comput Chem 16:395
32. Bernholdt DE, Harrison RJ (1995) J Chem Phys 102:9582
33. Schütz M, Fülscher MP, Lindh R, An integral-direct, distributed parallel CASSCF algorithm (to be published).
34. Nieplocha J, Harrsion RJ, Littlefield RJ (1994) Global Arrays: A portable "shared memory" programming model for distributed memory computers. IEEE, New York, p 330
35. Bernholdt DE, Aprà E, Früchtl HA, Guest MF, Harrison RJ, Kendall RA, Kutteh RA, Long X, Nicholas JB, Nichols JA, Taylor HL, Wong AT, Fann GL, Littlefield RJ, Nieplocha J (1995) Int J Quantum Chem: Quantum Chem Symp 29:475
36. Wong AT, Harrison RJ, Rendell AP (1996) Theor Chim Acta 93:317
37. Werner HJ, Meyer W (1980) J Chem Phys 73:2342

38. Taylor PR (1992) In: Roos BO (ed.), Lecture Notes in Quantum Chemistry, European Summer School in Quantum Chemistry, Lecture Notes in Chemistry, Vol. 58. Springer, Berlin, p 89
39. Lindh R, Ryu U, Liu B (1991) J Chem Phys 95:5889
40. Almlöf J, Taylor PR (1987) J Chem Phys 86:4070
41. Roos BO (1980) Int J Quantum Chem: Quantum Chem Symp 14:175
42. Fischer TH, Almlöf J (1992) J Chem Phys 96:9768
43. Schütz M, Lindh R (to be published).
44. Widmark P-O, Persson BJ, Roos BO (1991) Theor Chim Acta 79:419
45. Pierloot K, Dumez B, Widmark P-O, Roos BO (1995) Theor Chim Acta 90:87