JCTC Journal of Chemical Theory and Computation

Local Møller−Plesset Perturbation Theory: A Massively Parallel Algorithm

Ida M. B. Nielsen* and Curtis L. Janssen

Sandia National Laboratories, P.O. Box 969, Livermore, California 94551

Received June 1, 2006

Abstract: A massively parallel algorithm is presented for computation of energies with local second-order Møller−Plesset (LMP2) perturbation theory. Both the storage requirement and the computational time scale linearly with the molecular size. The parallel algorithm is designed to be scalable, employing a distributed data scheme for the two-electron integrals, avoiding communication bottlenecks, and distributing tasks in all computationally significant steps. A sparse data representation and a set of generalized contraction routines have been developed to allow efficient massively parallel implementation using distributed sparse multidimensional arrays. High parallel efficiency of the algorithm is demonstrated for applications employing up to 100 processors.

1. Introduction

The high-degree polynomial scaling of conventional molecular orbital-based correlated electronic structure methods is an obstacle to their application to mainstream chemical problems. Even for one of the computationally least expensive correlated methods, second-order Møller−Plesset (MP2) perturbation theory, the computational cost formally scales as $O(n^5)$, where n is the size of the molecular system.

The computational scaling may be improved, however, by formulating the equations in a basis of localized orbitals instead of the canonical molecular orbitals, which are by nature delocalized. This idea has been utilized by Pulay and Saebø in their work on the so-called local correlation method.[1−3] Using localized orbitals, Pulay and Saebø proposed restricting the excitation space for a given occupied orbital to a domain consisting of virtual orbitals that are all spatially close to the occupied orbital out of which the excitation occurs. By reducing the number of configurations to be included in the description of the wave function, this restriction of the correlation space may be utilized to attain reduced computational scaling of the correlation procedure.

Many applications of the local correlation idea, using various choices for the localized orbitals, have been presented for Møller−Plesset perturbation theory. Pulay and co-workers have developed local formulations for second- through fourth-order Møller−Plesset perturbation theory[1,2,4] using an orbital-invariant formulation employing localized orbitals and requiring an iterative solution of the amplitude equations, and low-order scaling implementations of the local MP2 method have been achieved.[5,6] Scuseria and Ayala[7] used a Laplace transform method[8] to obtain an atomic orbital-based linear scaling MP2 algorithm, and Friesner and co-workers[9,10] have implemented a pseudospectral localized MP2 method using the local correlation idea of Pulay and Saebø. Maslen and Head-Gordon[11] have developed a local MP2 method using nonorthogonal orbitals and an atoms-in-molecules local ansatz, and this method has been further developed to include a diatomics- and a triatomics-in-molecules approach.[12,13] Notably, Schütz and co-workers[14,15] employed the Pulay−Saebø local correlation scheme, using localized molecular orbitals for the occupied space and atomic orbitals, projected against the occupied space, for the virtual space, and achieved an algorithm that formally scales linearly with molecular size for local second-order Møller−Plesset theory (LMP2).

The achievement of low-order or linearly scaling implementations of MP2 methods has widened the range of molecules to which they can be applied, but, nonetheless, the application of reduced-scaling correlation methods to large molecules places heavy demands on computing resources. The scope of reduced-scaling correlation methods may be further extended, however, by developing algorithms tailored to application to massively parallel computers. In

* Corresponding author e-mail: ibniels@ca.sandia.gov.

10.1021/ct600188k CCC: $37.00
Published on Web 11/22/2006

this work, we focus our attention on the local MP2 method LMP2, using the local correlation idea of Pulay and Saebø in the context of MP2 theory, to develop a linear scaling massively parallel LMP2 algorithm. We are not aware of any parallel implementations of the LMP2 method reported in the literature, although a massively parallel algorithm for doing conventional (O(n^5)) MP2 using the orbital-invariant formulation (on which the LMP2 method is based) has been reported.[16] Additionally, the pseudospectral localized MP2 method has been parallelized previously, and the parallel performance has been illustrated for runs employing up to 16 processors.[17]

2. Theory

We will employ the local correlation scheme developed by Pulay and Saebø,[1−3] using an occupied orbital space represented by molecular orbitals localized by the Pipek−Mezey procedure[18] and an unoccupied space consisting of atomic orbitals that have been projected against the occupied orbital space to ensure orthogonality of the occupied and virtual spaces. The localized occupied orbitals are expressed as linear combinations of atomic orbitals using the coefficient matrix **L**

$$|i\rangle = \sum_{\mu} |\mu\rangle L_{\mu i} \tag{1}$$

with indices i and μ representing a localized molecular orbital and an atomic orbital, respectively. The projected atomic orbitals are obtained from the original atomic orbitals using the projection matrix **P** with elements $P_{\mu r}$

$$|r\rangle = \sum_{\mu} |\mu\rangle P_{\mu r} \tag{2}$$

$$\mathbf{P} = \frac{1}{2}\mathbf{D}_{\mathrm{v}}\mathbf{S} \tag{3}$$

where r represents a projected atomic orbital, **S** is the overlap matrix in the unprojected atomic orbital basis, and $\mathbf{D}_{\mathrm{v}}$ denotes the closed-shell Hartree−Fock density matrix constructed from the virtual orbitals. By forming **P** this way, rather than using $\mathbf{P} = \mathbf{I} - (1/2)\mathbf{DS}$ as has been done previously,[6,14,19] we avoid difficulties that may arise if the occupied density matrix **D** is constructed from a truncated set of atomic orbitals (when some orbitals have been eliminated in the Hartree−Fock procedure to avoid near linear dependencies), whereas **I** represents the full AO space. We note that, while the unoccupied space is not orthogonal, the occupied and unoccupied spaces are mutually orthogonal, and the localized occupied orbitals are orthogonal among themselves.

In the following, the indices i, j, k denote localized occupied molecular orbitals, and the indices r, s represent projected atomic orbitals. For each localized occupied orbital i, a domain [i] is created, comprising all projected atomic orbitals associated with a certain subset of the atoms. This subset is determined by including the atoms whose atomic orbitals contribute the most to the Mulliken population of i, until a spanning criterion, measuring how well the included atomic orbitals span orbital i, is satisfied. The procedure is described in detail elsewhere.[19,20] Our implementation uses a default threshold of 0.02 for the residual error associated with the spanning criterion,[19,20] but a different (typically smaller) threshold can be specified in the input, if desired. Using an orbital-invariant formulation of MP2 theory, the MP2 correlation energy can be expressed as[1]

$$E_{\mathrm{MP2}}^{\mathrm{corr}} = \sum_{ij} \mathrm{Tr}[\mathbf{K}_{ij}(2\mathbf{T}_{ji} - \mathbf{T}_{ij})] = \sum_{ijrs} K_{ij}^{rs}(2T_{ij}^{rs} - T_{ij}^{sr}) \tag{4}$$

where $\mathbf{K}_{ij}$ is a matrix whose elements K_{ij}^{rs} are the two-electron integrals ($ri|sj$), and $\mathbf{T}_{ij}$ is the matrix of double substitution amplitudes T_{ij}^{rs}. The double substitution amplitudes are determined from the set of linear equations

$$\mathbf{R}_{ij} = \mathbf{K}_{ij} + \mathbf{F}\mathbf{T}_{ij}\mathbf{S} + \mathbf{S}\mathbf{T}_{ij}\mathbf{F} - \mathbf{S}\sum_{k}[F_{ik}\mathbf{T}_{kj} + F_{kj}\mathbf{T}_{ik}]\mathbf{S} = 0 \tag{5}$$

which must be solved iteratively. $\mathbf{R}_{ij}$ represents the residual matrix with elements R_{ij}^{rs}, **F** and **S** are the virtual−virtual blocks of the Fock matrix and the overlap matrix, respectively, in the projected atomic orbital basis, and F_{ik} is an element of the Fock matrix in the basis of localized occupied orbitals.

In the iterative solution of eq 5, the double substitution amplitudes are updated using a procedure previously described for the local coupled-cluster method.[19] This procedure involves transforming the residual matrix $\mathbf{R}_{ij}$ to an orthogonal basis, computing an update to the double substitution amplitudes from the transformed residual elements

$$\Delta T_{ij}^{rs} = \frac{-R_{ij}^{rs}}{\epsilon_r + \epsilon_s - F_{ii} - F_{jj}} \tag{6}$$

and transforming the computed amplitude update back to the original projected atomic orbital basis. The transformation to an orthogonal basis is performed for each ij pair individually using a transformation matrix obtained by diagonalization of the Fock matrix in the subspace spanned by the projected atomic orbitals in the [ij] pair domain, and ϵ_r, ϵ_s in eq 6 denote diagonal elements of the Fock matrix in this basis.

The local MP2 method uses the orbital-invariant expressions from above and reduces the computational scaling of the MP2 procedure as follows. The excitation space is restricted by allowing double excitations out of occupied orbitals into only the virtual orbitals in the corresponding orbital domains. Thus, for an occupied orbital pair ij, double excitations are allowed only into the pair domain [ij], which is formed by combining the individual orbital domains [i] and [j]. To further reduce the number of double substitution amplitudes, excitations are allowed out of only a subset of the ij pairs, namely those pairs for which the minimum distance R_{ij} between any two atoms in the domains [i] and [j] is less than a certain threshold value. Using these restrictions on the included double substitutions, the number of double substitution amplitudes and integrals K_{ij}^{rs} will grow only linearly with the size of the molecule, and the computation of the LMP2 correlation energy (from eq 4), thus, scales linearly with the molecular size. To achieve a linearly scaling LMP2 procedure, the integral transformation generating the K_{ij}^{rs} integrals and the iterative solution of eq

Localize occupied molecular orbitals
Create domains
Compute screening quantities required for integral transformation
Perform integral transformation generating $(ri|sj)$

Begin LMP2 iterations
  Compute LMP2 residual
  Compute $\Delta\mathbf{T}$
  DIIS extrapolation of $\mathbf{T}$
  Compute $E_{\mathrm{MP2}}^{\mathrm{corr}}$, $\Delta E_{\mathrm{MP2}}^{\mathrm{corr}}$
  Check for convergence on $\Delta E_{\mathrm{MP2}}^{\mathrm{corr}}$ and $\Delta\mathbf{T}$
End LMP2 iterations

Figure 1. Outline of the local MP2 procedure.

5 must also scale linearly, however, and this requires application of integral prescreening throughout the transformation and utilization of the sparsity of the overlap and Fock matrices. The employed screening procedures will be explained in more detail below.

3. A Scalar LMP2 Algorithm

In this section we describe the scalar implementation of the LMP2 method from which our parallel algorithm has been developed. An outline of the steps involved in the computation of the LMP2 energy is given in Figure 1. Initially, the occupied molecular orbitals are localized, the domains are created, and the various screening quantities that must be employed to achieve linear scaling are computed. The transformation matrices required in the iterative procedure to transform the residual to an orthogonal basis (cf. section 2) are also computed at this point and stored for the remainder of the calculation. The two-electron integral transformation is then performed to generate the transformed integrals required for the LMP2 procedure. The LMP2 integral transformation is similar to that of a conventional MP2 computation but uses different transformation matrices and a different screening protocol. The screening employed in the integral transformation is outlined in Appendix A. The effect of varying the various thresholds used in the screening required to achieve linear scaling in the local MP2 procedure has been investigated in some detail by Schütz et al.[14] and by Saebø and Pulay.[6] In the present work, we have used the thresholds given in Appendix A.

When the two-electron integral transformation is complete, the iterative procedure is entered. In each iteration, the residual is first computed from eq 5, and an update, $\Delta\mathbf{T}$, for the double substitution amplitudes is then computed from eq 6. Using $\Delta\mathbf{T}$ and DIIS extrapolation, if desired, the double substitution amplitudes $\mathbf{T}$ are then updated, and a new value for the energy $E_{\mathrm{corr}}^{\mathrm{MP2}}$ is computed. Finally, a check for convergence on both $\Delta\mathbf{T}$ and $\Delta E_{\mathrm{corr}}^{\mathrm{MP2}}$ is performed.

In our implementation, all four-index quantities, i.e., the two-electron integrals K_{ij}^{rs} (as well as the half- and three-quarter-transformed integrals), the double substitution amplitudes T_{ij}^{rs}, and the residual elements R_{ij}^{rs} are stored as multidimensional sparse arrays. In previous implementations, these quantities have been stored as matrices, e.g., storing matrices $\mathbf{K}_{ij}$ with elements K_{ij}^{rs}.[6,14] We have implemented a parallel sparse data representation to handle distributed sparse multidimensional arrays and also a set of generalized contraction routines to perform multiplications of such arrays. Using this data representation and the associated contraction routines greatly simplifies programming, eliminating the need for explicit index manipulation, allowing contractions to be performed in a single line of code, and making parallelization more straightforward. The parallel sparse multidimensional arrays and contraction routines are discussed in more detail in the next section and in Appendix B.

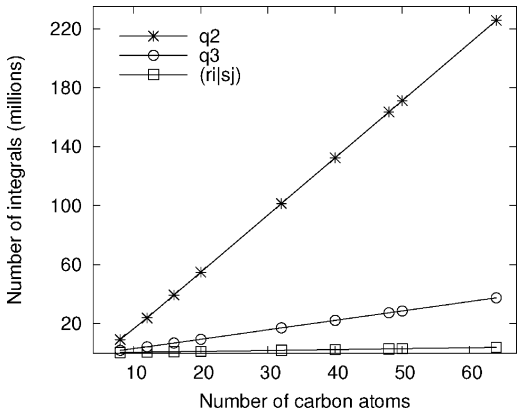

Figure 2. The size of the integral arrays required for the LMP2 procedure for linear alkanes C_nH_{2n+2} using the cc-pVDZ basis; q2 and q3 represent the half-transformed and three-quarter-transformed integral arrays, respectively.

For the last two terms in eq 5, involving left and right multiplication with the overlap matrix, the multiplication can be carried out either inside or outside of the summation over *k*. Schütz et al.[14] found that performing the multiplication inside of the summation was faster because the multiplication could be carried out with only the small local dimensions of the overlap matrix. Saebø and Pulay,[6] however, found multiplication outside of the summation to be the faster alternative in their local MP2 program. We have chosen to perform the multiplications with the overlap matrix outside of the summation because this makes possible a straightforward parallel implementation of this step using our parallel sparse data representation for **S**, **F**, and **T** (vide infra).

To illustrate the performance of the scalar LMP2 algorithm, computations were performed for a series of linear alkanes C_nH_{2n+2} using the cc-pVDZ basis set,[21] freezing the core orbitals, and using a cutoff of 8.0 bohr for the distance threshold R_{ij} (cf. section 2) for including *ij* pairs. For the computations reported here, convergence of the energy to 10^{-7} hartrees required 10 iterations. We note that a dynamic update scheme has been used previously,[5,6,14] updating the double substitution amplitudes as soon as the corresponding residual elements have been computed. This procedure speeds up convergence but would entail interprocessor communication (and result in nondeterministic convergence) when parallelized. We have elected to implement a parallel computation of the residual that does not require interprocessor communication (vide infra), updating all double substitution amplitudes at once and using the DIIS procedure[22] to accelerate convergence. The scaling behavior of the algorithm is shown in Figures 2 and 3. The number of transformed two-electron integrals, which, once generated, must be kept for the duration of the LMP2 computation, scales linearly with the molecular size. Likewise, the size of the intermediate arrays (partially transformed two-electron

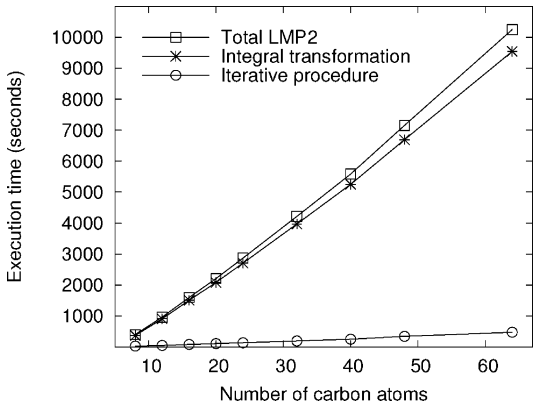

Figure 3. Computational times (wall times) for the LMP2 procedure obtained for linear alkanes C_nH_{2n+2} using the cc-pVDZ basis. The computations were performed on an AMD 2.4 GHz Opteron processor.

integrals), which are required during the integral transformation only, increases linearly with the molecular size. The number of double substitution amplitudes T_{ij}^{rs} is the same as the number of transformed integrals ($ri|sj$).

The computational time (wall time) required for the integral transformation, the iterative procedure, and the full LMP2 procedure is illustrated in Figure 3. Both the integral transformation and the iterative procedure scale linearly with the molecular size. The scaling obtained for the entire LMP2 procedure is slightly above linear because the timings include the time required to localize the orbitals, and this step, although fast, scales as n^3.

4. A Massively Parallel LMP2 Algorithm

Using the linear scaling LMP2 implementation described in the previous section, we have developed a massively parallel LMP2 algorithm. The parallel LMP2 program has been implemented as part of the massively parallel quantum chemistry program MPQC.[23] We have chosen to design a parallel LMP2 algorithm that uses only standard MPI collective communication to ensure portability of the code to architectures for which one-sided message passing is not available. To create a scalable algorithm, all large data arrays have been distributed, all computationally significant steps have been parallelized, and a communication scheme that does not create bottlenecks has been used. Both data and computational tasks are distributed over two occupied indices, allowing efficient utilization of a large number of processors.

The computationally dominant steps in the LMP2 procedure are the two-electron integral transformation and the computation of the LMP2 residual in the iterative procedure. The computation of the correlation energy (eq 4) and the update of the amplitudes (eq 6) are fast steps that are parallelized by letting each node process only the terms corresponding to the locally held *ij* pairs. We will describe the parallel implementation of the two-electron integral transformation and the computation of the residual in detail below.

4.1. Parallel Two-Electron Integral Transformation. Two-electron integrals of the type ($ri|sj$) with two occupied

Loop over all M, N shells $|M \geq N$
  If (my M, N pair):
  Loop over all R, S shells
    Allocate AO integral block $(\underline{M}R|\underline{N}S)$
  end R,S loop
  Compute $(\underline{M}R|\underline{N}S)$ for current M, N pair and all R, S
  Loop over R, S blocks of $(\underline{M}R|\underline{N}S)$
    Loop over j
      Allocate $(\underline{M}R|\underline{N}j)$ block
    end j loop
  end R, S loop
  $(\underline{M}R|\underline{N}j) = (\underline{M}R|\underline{N}S) * L(S, j)$
  Loop over R, j blocks of $(\underline{M}R|\underline{N}j)$
    Loop over $i|ij$ valid pair
      Allocate $(\underline{M}i|\underline{N}j)$ block
    end i loop
  end R, j loop
  $(\underline{M}i|\underline{N}j) = (\underline{M}R|\underline{N}j) * L(R, i)$
end $\underline{M}, \underline{N}$ loop

Redistribute half-transformed integrals:
$(\underline{M}i|\underline{N}j)|M \geq N \rightarrow (M\underline{i}|N\underline{j})|i \geq j$

Loop over blocks $M, N, \underline{i}, \underline{j}$ of $(M\underline{i}|N\underline{j})$
  Loop over $s|s \in [ij]$
    Allocate $(M\underline{i}|s\underline{j})$ block
  end s loop
end $M, N, \underline{i}, \underline{j}$
$(M\underline{i}|s\underline{j}) = (M\underline{i}|N\underline{j}) * P(N, s)$

Loop over blocks $M, \underline{i}, s, \underline{j}$ of $(M\underline{i}|s\underline{j})$
  Loop over $r|r \in [ij]$
    Allocate $(r\underline{i}|s\underline{j})$ block
  end r loop
end $M, \underline{i}, s, \underline{j}$ loop
$(r\underline{i}|s\underline{j}) = (M\underline{i}|s\underline{j}) * P(M, r)$

Figure 4. Outline of the parallel integral transformation. Indices *M*, *R*, *N*, *S* represent shells of atomic orbitals. The transformation matrices **L** and **P** are defined in the text. Distributed indices are underlined. The steps in which the integrals ($\underline{M}R|\underline{N}S$), ($\underline{M}R|\underline{N}j$), ($\underline{M}i|\underline{N}j$), ($M\underline{i}|s\underline{j}$), and ($r\underline{i}|s\underline{j}$) are allocated are preceded by integral screening as outlined in Appendix A.

and two unoccupied indices are required for the LMP2 computation. The fully transformed integrals ($ri|sj$) can be kept in aggregate memory without introducing serious memory bottlenecks, but some of the partially transformed integral arrays are too large to fit in memory, and these arrays are therefore generated in a direct fashion that requires storage of only a small subsection of the arrays at any given time. To facilitate evenly distributed loads, the integrals are distributed over two indices, and in the iterative procedure, each node will process only the locally held integrals. The parallelization scheme employed here is similar to that used by Wong et al.[24] in a canonical parallel integral transformation.

The parallel integral transformation is outlined in Figure 4. The screening steps are not shown but are performed as described in Appendix A. The transformation involves computation of the two-electron integrals in the atomic orbital (AO) basis and four subsequent quarter transformations, each transforming one index. The transformation matrices are the matrix **P** for the unoccupied indices and the matrix **L** for the occupied indices. The computation of the AO integrals and the first two quarter transformations are performed within a loop over pairs of shells *M* and *N* ($M \geq N$), and these steps are parallelized by distributing the *M*, *N* pairs across nodes. For each *M*, *N* shell pair, the AO integral array

(*MR*|*NS*) is computed for all contributing *R*, *S*, and this array is then transformed to generate the quarter-transformed integrals (*MR*|*Nj*) for the current *M*, *N* pair. The (*MR*|*Nj*) array, subsequently, is transformed to generate the half-transformed integrals (*Mi*|*Nj*). The number of AO integrals and quarter-transformed integrals can become quite large, even though the size of the arrays scales linearly with the molecular size. A memory bottleneck is avoided by only generating these arrays for one *M*, *N* shell pair at a time before they are transformed and discarded. The half-transformed integral array is much smaller, and this array, and, subsequently, the three-quarter and fully transformed integral arrays, can be kept in memory. The first half-transformation generates the integrals (*Mi*|*Nj*) distributed over *M*, *N*, and no interprocessor communication is required.

After completion of the first half-transformation, a redistribution of the half-transformed integrals is performed, replacing the distribution over *M*, *N* shell pairs with a distribution over *i*,*j* pairs ($i \geq j$). This is the only communication step required in the entire integral transformation, and after the redistribution of integrals, each node independently completes the integral transformation generating the fully transformed integrals (*ri*|*sj*). The work in the second half-transformation is distributed by letting each node process only the *i*,*j* pairs held locally, and upon completion of the integral transformation, each node holds the (*ri*|*sj*) array for a subset of the included *i*,*j* pairs.

We note that the parallel integral transformation exploits the $M \geq N$ symmetry in the computation of the AO integrals and in the first two quarter transformations. Thus, a 4-fold redundancy is required in the AO integral computation. In the last two quarter transformations, the $i \geq j$ symmetry is utilized, producing fully transformed integrals (*ri*|*sj*) for $i \geq j$, and the included *i*,*j* pairs are distributed across nodes.

4.2. Parallel Computation of the LMP2 Residual. The computation of the LMP2 residual accounts for the majority of the time spent in the iterative procedure. We have implemented two parallel schemes for computation of the residual, employing replicated and distributed double substitution amplitudes, respectively. By replicating the double substitution amplitudes, load balance can more easily be achieved in the iterative procedure, and the amplitudes can be replicated without introducing memory bottlenecks. Replication of the amplitudes, however, necessitates a global summation step involving all the elements of $\Delta\mathbf{T}$ in the iterative procedure, which will reduce parallel performance as the number of processors grows large. An alternative parallel implementation of the residual computation employing distributed double substitution amplitudes has therefore also been investigated.

In the parallel implementation employing replicated double substitution amplitudes, the residual is computed from eq 5 by letting each node compute the residual arrays $\mathbf{R}_{ij}$ for the subset of the *ij* pairs for which the (*ri*|*sj*) integrals are held locally. The first term on the right-hand side of eq 5 is a simple update of the residual with the locally held (*ri*|*sj*) integrals. The second and third terms are contractions of blocks of double substitution amplitudes with virtual−virtual blocks of the Fock and overlap matrices. These terms are computed by looping over the local *ij* pairs on each node and, for the current *ij* block of **T**, performing the contraction as two subsequent matrix multiplications. The last two terms in eq 5 involve multiplication of *k*, *j* and *i*, *k* blocks of the double substitution amplitudes by the occupied−occupied Fock matrix elements F_{ik} and F_{kj}, respectively, forming the term $\sum_k[F_{ik}\mathbf{T}_{kj} + F_{kj}\mathbf{T}_{ik}]$, followed by matrix multiplication with the overlap matrix on the left and right. Again, only the *i*, *j* values corresponding to the locally held *ij* pairs are processed on each node, and because the double substitution amplitudes are replicated, the required $\mathbf{T}_{kj}$ and $\mathbf{T}_{ik}$ elements are readily available. The computation of the residual requires no interprocessor communication, and the distribution of work over two indices makes it possible to achieve load balance even for a large number of processors, provided that the number of processors is significantly smaller than the number of included *ij* pairs. A global communication step, accumulating the individual contributions to $\Delta\mathbf{T}$ (eq 6) computed on each node, is required, however, and as demonstrated below, this step impacts the parallel scaling of the iterative procedure when using a large number of processors.

To avoid the global accumulation step for the double substitution amplitudes, we have also implemented a version of the iterative procedure in which the double substitution amplitudes are distributed. Two different distributions of **T** are then required, namely distribution over *ij* pairs and distribution over just one occupied index. The former distribution is needed for parallel efficiency in the computation of the correlation energy and for the $\mathbf{FT}_{ij}\mathbf{S}$ and $\mathbf{ST}_{ij}\mathbf{F}$ terms in eq 5, and the latter distribution is required for computation of the last two terms on the right-hand side of eq 5. The only communication step required is now a redistribution of **T** (from *ij* pairs to just one occupied index), but this step is scalable, and the overall communication time can therefore be reduced. In this scheme, however, load imbalance becomes a problem as the number of processors increases, because the last two terms in eq 5 are now parallelized by distribution of only one occupied index.

To avoid both communication bottlenecks and load imbalance in the iterative procedure, a one-sided asynchronous message passing scheme could be implemented, and we have explored the use of such schemes in our previous work on massively parallel computation of MP2 energies and gradients.[25,26] Using asynchronous message passing in the iterative LMP2 procedure would allow distribution of the double substitution amplitudes (eliminating the need for global communication) and distribution of work over two indices (achieving load balance) by letting each node request amplitudes from other nodes when needed.

5. Parallel Performance

Computations reported in this section were performed on a Linux cluster with dual-processor 3.6 GHz Xeon nodes connected via an InfiniBand network with a full fat-tree topology.

The parallel performance of the algorithm is illustrated in Figure 5. The speedup curves shown were obtained for a linear alkane, $C_{32}H_{66}$, using the correlation-consistent basis

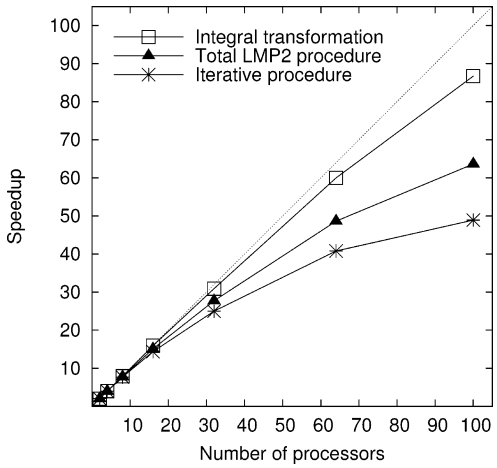

Figure 5. Parallel speedups for the LMP2 procedure obtained for the linear alkane $C_{32}H_{66}$ using the cc-pVDZ basis.

set cc-pVDZ[21] (778 basis functions). Computations were carried out using a number of processors ranging from 1 to 100, employing only one processor per node. The speedups shown were computed relative to the computational time required on one processor. Converging the energy to 10^{-7} hartrees required 10 iterations. As explained above, we have implemented two different versions of the iterative procedure, employing replicated and distributed double substitution amplitudes, respectively. For the chosen test case, load imbalance caused the algorithm using distributed amplitudes to be the slower of the two for all measured processor counts, and the results presented below were obtained using the algorithm employing replicated amplitudes.

The dominant computational steps are the two-electron integral transformation and the iterative procedure, which take up 92% and 7% of the computational time, respectively, when running on one node. As the number of nodes increases, these percentages change somewhat because a few remaining serial steps take up an increasing fraction of the computational time and because the speedups are higher for the integral transformation than for the iterative procedure.

As shown in Figure 5, the parallel efficiency obtained in the integral transformation remains high as the number of processors increases, and a speedup of 87 is obtained using 100 processors. In the integral transformation, work is distributed over two indices, viz., shell pairs in the first half-transformation and pairs of occupied orbitals in the second half-transformation, and provided that the number of processors is significantly smaller than the number of such pairs, high parallel efficiency can be obtained. The only communication step required in the integral transformation is the redistribution of the half-transformed integrals. This step is scalable because the total number of integrals to be communicated by each processor is inversely proportional to the number of processors, and it takes a negligible amount of time, less than 2% of the time required for the integral transformation in all cases.

For the iterative procedure, high parallel efficiency is obtained up to around 32 processors after which the performance starts to degrade, and the efficiency is about 50% when running on 100 processors. As mentioned, the iterative procedure is fast compared to the integral transformation for basis sets that would typically be used in an MP2 computation, so the somewhat lower parallel performance of the iterative procedure does not have a serious impact on the overall parallel performance (vide infra). The performance degradation observed for the iterative procedure is due mainly to a communication step required in each iteration to collect partial contributions to the updated double substitution amplitudes from all nodes.

The speedups obtained for the total LMP2 procedure are also illustrated in Figure 5. The efficiency is high up to about 64 processors (obtaining a speedup of 49 on 64 processors) and degrades as the number of processors increases. The performance degradation observed for the overall speedup is caused mainly by the presence of some small computational steps that have not yet been parallelized and to a lesser extent by the degrading performance in the iterative procedure. The steps that have not been parallelized include the localization of the occupied orbitals, the computation of the domains, and the computation of some of the screening quantities required to attain linear scaling with the size of the molecule.

To further illustrate applications of the LMP2 code, we have computed LMP2 and conventional MP2 energies for a couple of globular molecules, namely two isomers of the boron nitride $C_{14}H_{20}BNO_3$. The two isomers, illustrated in Figure 6, are both candidates for the global minimum for this molecule. The cc-pVDZ, cc-pVTZ, and cc-pVQZ basis set were employed,[21] and the results are shown in Table 1. In all cases, the relative energies computed with the LMP2 method are close to the conventional MP2 energies, with somewhat better agreement for the larger sets. The fraction of the MP2 correlation energy recovered at the LMP2 level increases with basis set improvement, ranging from about 97% at the cc-pVDZ level to 99% at the cc-pVQZ level.

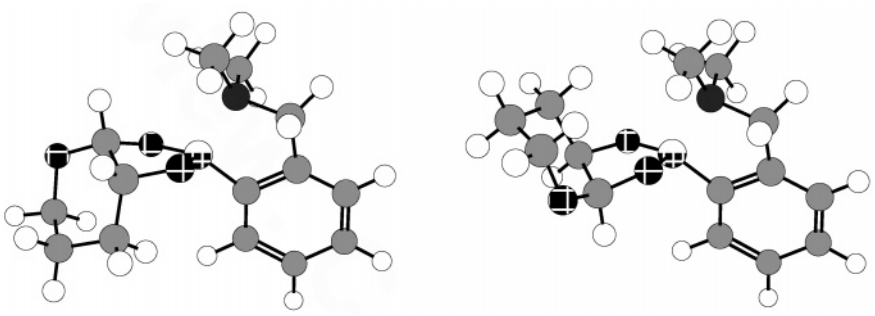

Figure 6. Two isomers of the boron nitride $C_{14}H_{20}BNO_3$ (isomer 1 on the left, isomer 2 on the right).

Table 1. LMP2 and Conventional MP2 Energies for Two Isomers of $C_{14}H_{20}BNO_3$[a]

	LMP2		MP2	
	isomer 1	isomer 2	isomer 1	isomer 2
		cc-pVDZ		
E_{corr}	−2.642280	−2.640938	−2.716272	−2.715842
E_{total}	−848.308810	−848.303178	−848.382802	−848.378083
ΔE	3.53		2.96	
		cc-pVTZ		
E_{corr}	−3.278991	−3.277799	−3.333433	−3.333024
E_{total}	−849.168225	−849.163091	−849.222666	−849.218315
ΔE	3.22		2.73	
		cc-pVQZ		
E_{corr}	−3.516106	−3.515809	−3.549913	−3.549537
E_{total}	−849.458891	−849.454743	−849.492698	−849.488470
ΔE	2.60		2.65	

[a] Relative energies ($\Delta E = E_{total}$[isomer 2] − E_{total}[isomer 1]) in kcal mol^{-1}, other energies in hartrees; employing a domain completeness threshold of 0.01 and an R_{ij} distance threshold of 15.0 bohr (cf. section 2) in the LMP2 procedure.

Finally, to show computational times required for some larger LMP2 computations, we have also computed LMP2 energies with larger basis sets for a couple of the linear alkanes, viz. $C_{24}H_{50}$ using the aug-cc-pVTZ basis[21,27] (2254 basis functions) and $C_{50}H_{102}$ employing the cc-pVTZ set (2928 basis functions). Using 100 nodes, the total wall times required for the LMP2 computation (excluding the Hartree−Fock part) for $C_{24}H_{50}$ ($E_{LMP2} = -942.508278$ hartrees) and $C_{50}H_{102}$ ($E_{LMP2} = -1962.104490$ hartrees) were 102 and 14 min, respectively, using a value of 8.0 bohr for the distance threshold R_{ij} for including ij pairs. The time required for the basis set with 2254 functions is significantly longer than for the set with 2928 functions because the former is an augmented set for which larger domains are required and for which the screening in the integral transformation is less effective.

6. Concluding Remarks

We have demonstrated the feasibility of highly efficient massively parallel computation of local correlation energies by developing and implementing a massively parallel LMP2 algorithm. Our implementation of the LMP2 method scales linearly with the molecular size in the number of two-electron integrals and double substitution amplitudes as well as in the computational time. Using a parallel sparse data representation and a set of generalized contraction routines developed to handle distributed sparse multidimensional arrays, a massively parallel implementation of the LMP2 algorithm has been achieved. The parallel implementation employs only standard MPI collective communication to increase portability of the code, and the communication scheme is designed to prevent serious bottlenecks as the number of processors increases. To avoid memory bottlenecks, the first half of the integral transformation is direct, and the second half of the transformation employs fully distributed data arrays. Load balance has been achieved by distributing all computationally significant tasks over two indices, allowing utilization of a large number of processors. The parallel performance of the algorithm has been illustrated, and high parallel efficiency has been demonstrated for computations performed using up to 100 processors.

Appendix A. Screening of Two-Electron Integrals

To achieve linear scaling in the integral transformation, it is necessary to employ integral screening for the computation of the AO integrals and in each of the four subsequent transformation steps. Each of the five steps in the integral transformation in which integrals are allocated (cf. Figure 4) is preceded by one or more screening steps, so that the integral will be allocated only if it is greater than a certain threshold. In our notation, M, R, N, and S are shells of atomic orbitals, ν and σ represent atomic orbitals, i and j denote localized occupied molecular orbitals, and r and s are projected atomic orbitals. The following quantities are employed in the screening procedure:

$$T_{NS}^{\text{Schwarz}} = \text{Max}_{\nu,\sigma}|(\nu\sigma|\nu\sigma)|^{0.5},\ \nu \in N, \sigma \in S \quad \text{(A-1)}$$

$$T_{N}^{\text{Schwarz}} = \text{Max}_{S}(T_{NS}^{\text{Schwarz}}),\ \text{all } S \quad \text{(A-2)}$$

$$P_{Ns}^{\text{max}} = \text{Max}_{\nu}(|P_{\nu s}|),\ \nu \in N \quad \text{(A-3)}$$

$$P_{N}^{\text{max}} = \text{Max}_{s}(P_{Ns}^{\text{max}}),\ \text{all } s \quad \text{(A-4)}$$

$$(PP)_{MN}^{\text{max}} = \text{Max}_{r,s}(P_{Mr}^{\text{max}}\ P_{Ns}^{\text{max}}),\ rs \in [ij]\ \text{(all valid } ij \text{ pairs)} \quad \text{(A-5)}$$

$$L_{Nj}^{\text{max}} = \text{Max}_{\nu}(|L_{\nu j}|),\ \nu \in N \quad \text{(A-6)}$$

$$L_{N}^{\text{max}}(j) = \text{Max}_{i}(L_{Ni}^{\text{max}}),\ \text{all } i \mid ij \text{ valid pair} \quad \text{(A-7)}$$

$$\text{L}_{N}^{\text{max}} = \text{Max}_{j}(L_{Nj}^{\text{max}}),\ \text{all } j \quad \text{(A-8)}$$

$$L^{\text{max}} = \text{Max}_{N}(L_{N}^{\text{max}}),\ \text{all } N \quad \text{(A-9)}$$

$$(LL)_{MN}^{\text{max}} = \text{Max}_{i,j}(\text{L}_{Mi}^{\text{max}}\ \text{L}_{Nj}^{\text{max}}),\ \text{all } i,j \mid ij \text{ valid pair} \quad \text{(A-10)}$$

$$(LL)_{M}^{\text{max}} = \text{Max}_{N}((LL)_{MN}^{\text{max}}),\ \text{all } N \quad \text{(A-11)}$$

$$(LL)^{\text{max}} = \text{Max}_{M}((LL)_{M}^{\text{max}}),\ \text{all } M \quad \text{(A-12)}$$

Using the loop structure detailed in Figure 4, the screening protocol is implemented as follows. Initially, after entering the M, N loops, a preliminary screening is performed

$$T_{M}^{\text{Schwarz}} * T_{N}^{\text{Schwarz}} * (PP)_{MN}^{\text{max}} * (LL)^{\text{max}} \geq \epsilon \quad \text{(A-13)}$$

A loop over shells R is then entered, using an ordering of the R shells determined by sorting T_{MR}^{Schwarz} (for the current M and all R) in descending order. The screening

$$T_{N}^{\text{Schwarz}} * T_{MR}^{\text{Schwarz}} * (PP)_{MN}^{\text{max}} * (LL)^{\text{max}} \geq \epsilon \quad \text{(A-14)}$$

is then carried out, and, if not passed, the algorithm breaks out of the R loop, skipping all remaining R values. Otherwise, for the current R, another screening is performed

$$T_N^{\text{Schwarz}} * T_{MR}^{\text{Schwarz}} * (PP)_{MN}^{\max} * (LL)_R^{\max} \geq \epsilon \quad \text{(A-15)}$$

and, if passed, the S loop is entered. The S shells are sorted according to descending size of T_{NS}^{Schwarz}. Inside the S loop, a screening

$$T_{NS}^{\text{Schwarz}} * T_{MR}^{\text{Schwarz}} * (PP)_{MN}^{\max} * (LL)_R^{\max} \geq \epsilon \quad \text{(A-16)}$$

is first done to test whether to break out of the S loop. If passed, another screening step

$$T_{NS}^{\text{Schwarz}} * T_{MR}^{\text{Schwarz}} * (PP)_{MN}^{\max} * (LL)_{RS}^{\max} \geq \epsilon \quad \text{(A-17)}$$

determines whether to allocate the AO integral block $(MR|NS)$.

Allocation of the $(MR|Nj)$ integrals entails a loop over j indices that have been ordered according to decreasing $L_{Sj}^{\max}$. The screening step

$$|(MR|NS)|^{\max} * (PP)_{MN}^{\max} * L_{Sj}^{\max} * L_R{}^{\max} \geq \epsilon \quad \text{(A-18)}$$

determines whether to break out of the j loop, and, if continuing within the j loop, another screening

$$|(MR|NS)|^{\max} * L_{Sj}^{\max} * (PP)_{MN}^{\max} * L_R^{\max}(j) \geq \epsilon \quad \text{(A-19)}$$

determines whether to allocate the $(MR|Nj)$ block. The integral bound $|(MR|NS)|^{\max}$ represents the maximum absolute value of the AO integral for all atomic orbitals in the shell quartet M, R, N, S. In the second, third, and fourth quarter transformations, the following screening steps are performed before allocating the $(Mi|Nj)$, $(Mi|sj)$, and $(ri|sj)$ integrals, respectively:

$$|(MR|Nj)|^{\max} * L_{Ri}^{\max} * (PP)_{MN}^{\max} \geq \epsilon \quad \text{(A-20)}$$

$$|(Mi|Nj)|^{\max} * P_{Ns}^{\max} * P_M^{\max} \geq \epsilon \quad \text{(A-21)}$$

$$|(Mi|sj)|^{\max} * P_{Mr}^{\max} \geq \epsilon \quad \text{(A-22)}$$

The bound $|(MR|Nj)|^{\max}$ represents the maximum absolute value of a quarter-transformed integral for a fixed occupied index j and for all atomic orbitals in the shells M, R, and N, and the remaining integral bounds are defined analogously. In the present work, a value of 10^{-8} was used for all ϵ thresholds in the screening procedure.

Appendix B. Parallel Sparse Data Representation

Efficient massively parallel implementation of local correlation methods requires a means of handling distributed sparse matrices and multidimensional arrays. The sparsity of matrices and multidimensional arrays, e.g., the overlap matrix, the double substitution amplitudes T_{ij}^{rs}, and the two-electron integrals $(ri|sj)$, must be exploited to achieve linear scaling, and at least some of the multidimensional arrays, including the partially transformed two-electron integrals, must be distributed across nodes to avoid memory and communication bottlenecks. We have therefore implemented a parallel sparse data representation and an associated set of generalized contraction routines to perform contractions of distributed sparse matrices and multidimensional arrays. The most important features of these routines are discussed here, and a more thorough discussion of the implementation can be found elsewhere.[28]

Our parallel sparse data representation has been designed to permit rapid prototyping, ease of use, and good performance. The primary type is the C++ generic class Array. The primary template parameter for the Array class is an integer specifying the dimensionality the array, and of the Array relies on a variety of auxiliary classes that describe blocks, distribution schemes, index ranges, etc. An Array stores the block descriptors for contributing blocks and the data for each block. The block sizes for virtual indices are the number of basis functions on the atom associated with the block, and for occupied indices the block size is one. This permits the majority of blocks to be large enough to make their manipulation efficient while also enabling sufficiently fine-grained parallel distribution of the work. An optimized dense matrix multiplication routine is used for multiplication of the individual blocks. The Array class contains functions for manipulating the blocks, including copying arrays, summation of arrays, performing a binary product (contraction) of arrays to produce a third array, and doing a contraction that results in a scalar. Parallelization is supported by member functions that convert replicated to distributed arrays, distributed to replicated arrays, and distributed arrays to a different distribution scheme. Helper classes are also employed to permit the use of the implicit summation convention in the C++ source code, greatly increasing the readability of the program.

Upon declaration of an Array, its dimensionality must be specified, and the array must be initialized to get the proper ranges for its indices in each dimension. Additionally, the initialization of the array may specify a bound such that only blocks containing elements greater than the bound will be stored. For instance, the four-dimensional array Ints holding the transformed integrals $(ri|sj)$ may be declared and initialized as

```
Array<4> Ints;

Ints.init(vir,occ,vir,occ,bound);
```

An array may be allocated by allocating each of its blocks individually or by allocating the entire array at one time. For the Ints array, a single block may be allocated as

```
Ints.allocate_block(blockinfo);
```

where blockinfo specifies which block is to be allocated. Alternatively, an entire array may be allocated in one setting, e.g., from another, already allocated, array using the |= allocation operator

```
Array<4> T;

T.init(vir,occ,vir,occ,bound);

T(r,i,s,j) |= Ints(r,i,s,j);
```

As an example of application of our contraction routines, consider computation of the fully transformed two-electron integrals $(ri|sj)$, which are kept in the Ints array. These

integrals are obtained in the fourth quarter transformation, and this transformation can be performed in parallel in a single line of code in the program

$$\text{Ints(r,i,s,j)} = \text{q3(mu,i,s,j)*P(mu,r);}$$

The q3 and Ints arrays are both distributed across nodes, and the contraction will be done locally on each node using only the local elements of the arrays. Thus, once the data distribution has been determined for the arrays involved, the distribution of work is automatically achieved. Finally, to illustrate a contraction of arrays producing a scalar, consider the computation of the correlation energy (cf. eq 4) from the integrals and the double substitution amplitudes, stored in the arrys Ints and T, respectively. The Ints array (and T, if desired) is distributed, and the parallelization is handled by the contraction routine, which, again, works only on the locally stored elements. The parallel computation of the correlation energy, then, is handled by the following three lines of code

$$e_{corr} = 2 * \text{Ints(r,i,s,j)} * \text{T(r,i,s,j);}$$

$$e_{corr} - = \text{Ints(r,i,s,j)} * \text{T(s,i,r,j);}$$

$$\text{msg} \rightarrow \text{sum}(e_{corr});$$

The summation after the two contractions is required because each node computes only a partial contribution to the correlation energy.

References

(1) Pulay, P.; Saebø, S. *Theor. Chim. Acta* **1986**, *69*, 357−368.

(2) Saebø, S.; Pulay, P. *J. Chem. Phys.* **1987**, *86*, 914−922.

(3) Saebø, S.; Pulay, P. *Ann. Rev. Phys. Chem.* **1993**, *44*, 213−236.

(4) Saebo, S.; Pulay, P. *J. Chem. Phys.* **1988**, *88*, 1884−1890.

(5) Rauhut, G.; Pulay, P.; Werner, H.-J. *J. Comput. Chem.* **1998**, *19*, 1241−1254.

(6) Saebø, S.; Pulay, P. *J. Chem. Phys.* **2001**, *115*, 3975−3983.

(7) Ayala, P. Y.; Scuseria, G. E. *J. Chem. Phys.* **1999**, *110*, 3660−3671.

(8) Häser, M.; Almlöf, J. *J. Chem. Phys.* **1992**, *96*, 489−494.

(9) Murphy, R. B.; Beachy, M. D.; Friesner, R. A.; Ringnalda, M. N. *J. Chem. Phys.* **1995**, *103*, 1481−1490.

(10) Friesner, R. A.; Murphy, R. B.; Beachy, M. D.; Ringnalda, M. N.; Pollard, W. T.; Dunietz, B. D.; Cao, Y. *J. Phys. Chem. A* **1999**, *103*, 1913−1928.

(11) Maslen, P. E.; Head-Gordon, M. *Chem. Phys. Lett.* **1998**, *283*, 102−108.

(12) Maslen, P. E.; Head-Gordon, M. *J. Chem. Phys.* **1998**, *109*, 7093−7099.

(13) Lee, M. S.; Maslen, P. E.; Head-Gordon, M. *J. Chem. Phys.* **2000**, *112*, 3592−3601.

(14) Schütz, M.; Hetzer, G.; Werner, H.-J. *J. Chem. Phys.* **1999**, *111*, 5691−5705.

(15) Hetzer, G.; Schütz, M.; Stoll, H.; Werner, H.-J. *J. Chem. Phys.* **2000**, *113*, 9443−9455.

(16) Bernholdt, D. E.; Harrison, R. J. *J. Chem. Phys.* **1995**, *102*, 9582−9589.

(17) Beachy, M. D.; Chasman, D.; Friesner, R. A.; Murphy, R. B. *J. Comput. Chem.* **1998**, *19*, 1030−1038.

(18) Pipek, J.; Mezey, P. G. *J. Chem. Phys.* **1989**, *90*, 4916−4926.

(19) Hampel, C.; Werner, H.-J. *J. Chem. Phys.* **1996**, *104*, 6286−6297.

(20) Boughton, J. W.; Pulay, P. *J. Comput. Chem.* **1993**, *14*, 736−740.

(21) Dunning, T. H., Jr. *J. Chem. Phys.* **1989**, *90*, 1007−1023.

(22) Pulay, P. *J. Comput. Chem.* **1982**, *3*, 556−560.

(23) Janssen, C. L.; Nielsen, I. B.; Leininger, M. L.; Valeev, E. F.; Seidl, E. T. *The Massively Parallel Quantum Chemistry Program (MPQC), Version 2.3.0-alpha*; Sandia National Laboratories: Livermore, CA, U.S.A., 2005. http://www.mpqc.org (accessed October 4, 2006).

(24) Wong, A. T.; Harrison, R. J.; Rendell, A. P. *Theor. Chim. Acta* **1996**, *93*, 317−331.

(25) Nielsen, I. M. B. *Chem. Phys. Lett.* **1996**, *255*, 210−216.

(26) Janssen, C. L.; Nielsen, I. M. B.; Colvin, M. E. In *Encyclopedia of Computational Chemistry*; Schleyer, P. v. R., Ed.; Wiley: Chichester, 1998; pp 1990−2000.

(27) Kendall, R. A.; Dunning, T. H., Jr.; Harrison, R. J. *J. Chem. Phys.* **1992**, *96*, 6796−6806.

(28) Nielsen, I.; Janssen, C.; Leininger, M. *Scalable Fault Tolerant Algorithms for Linear-Scaling Coupled-Cluster Electronic Structure Methods*; Sandia Report, SAND2004-5462; Sandia National Laboratories: Livermore, CA, U.S.A., 2004.

CT600188K