JCTC Journal of Chemical Theory and Computation

MPI/OpenMP Hybrid Parallel Algorithm for Hartree−Fock Calculations

Kazuya Ishimura,* Kei Kuramoto, Yasuhiro Ikuta, and Shi-aki Hyodo

Materials Fundamental Research Division, Toyota Central Research & Development Laboratories, Inc., Nagakute, Aichi 480-1192, Japan

Received December 22, 2009

Abstract: A new message passing interface/open multiprocessing (MPI/OpenMP) hybrid parallel algorithm of the Hartree−Fock calculation is developed and implemented into the quantum chemistry program package GAMESS. In the algorithm, internode distribution is performed by MPI and intranode parallelization by OpenMP. It is applied to a TiO_2 cluster ($Ti_{35}O_{70}$, 6-31G, 1635 basis functions) and to insulin ($C_{257}H_{381}N_{65}O_{77}S_6^{2-}$, STO-3G, 2430 basis functions) using the Cray XT5 supercomputer (quad-core Opteron 2.3 GHz, 2048 CPU cores). The speed-ups of the whole calculations, including the initial guess generation, are 1238 and 745 using 2048 CPU cores for the TiO_2 cluster and insulin, respectively. Hartree−Fock calculations with hundreds or thousands of CPU cores are now practical.

1. Introduction

Quantum chemistry calculations have played important roles for analysis and prediction of chemical reactions, molecular spectra, and so on. Calculated molecular systems are becoming larger and larger, and large and complicated systems, such as nanomaterials, self-assembled materials, and biological materials, are now challenging targets. Such systems could introduce unique properties owing to the steric effects with bulky substituents or to a lot of noncovalent interactions, such as dispersions and hydrogen bonds, although the high computational cost is required. Because the size is one of the important factors, calculations of realistic systems are necessary. Several approaches have been proposed to make these calculations possible for large molecules. These approaches could be categorized into mostly two different ways, the reduction of computational costs and the acceleration of calculations. While most of the former are based on modeling by dividing large systems into several or many parts, the latter uses high-performance accelerators or a number of computers.

One of the approaches is the quantum mechanical/molecular mechanical (QM/MM) method[1,2] in which a system is divided into a small reaction center as the QM region and the other as the MM region. The ONIOM method[3,4] divides a molecular system into two or three layers which are treated with different methods and basis sets. These methods can reduce the computational costs by dividing a large molecular system and calculating the important part with a high-level method and the others with a low-level method. A different approach to divide a system into small fragments, called the fragment molecular orbital (FMO) method,[5,6] is also developed, which is especially suitable to proteins. The total computational cost can be reduced because the cost of each fragment is low. However, these methods include approximations or cut-offs, and their accuracies have been discussed by changing computational models or by the combination of computational methods.[7,8]

On the other hand, parallel calculations using central processing units (CPUs), graphics processing units (GPUs), and accelerators are now common because the performance improvement of single CPU cores almost stopped due to heat and power problems. GPUs[9−14] and accelerators[15] are applied to Hartree−Fock (HF), density functional theory (DFT), and quantum Monte Carlo (QMC) calculations. Many efforts have been made for effective GPU computation, for instance, the use of both single- and double-precision values to minimize numerical errors, and the sorting of bra- and ket-pairs for two-electron repulsion integrals (2-ERIs) to utilize the Schwartz screening[16] and to calculate ERIs of same type together.

We can now use hundreds or thousands of CPU cores of supercomputers and PC clusters, and the number of available

* Corresponding author. E-mail: e1502@mosk.tytlabs.co.jp.

10.1021/ct100083w
Published on Web 03/23/2010

CPU cores is increasing. Recent supercomputers have more than 100 000 CPU cores.[17] The parallel efficiency has to be raised for such computer systems by improving the load-balancing and parallelization ratio. Furthermore, large memory space is required for electron correlation calculations, such as perturbation and coupled cluster theories. If only the message passing interface (MPI), which is the most commonly used method, is applied to parallel calculations, the available memory space per process becomes small, because each CPU core allocates each array.

One of the solutions for these problems of CPU parallel calculations is global memory access models such as global arrays[18,19] and distributed data interface (DDI)[20] in which data of other nodes can be accessed through network communication. Another solution is to introduce open multiprocessing (OpenMP), which is a method of intranode parallelization. This supports dynamic load balancing and memory array sharing within a node. OpenMP parallelization has been applied to ab initio calculations[21−23] for multicore/multisocket shared-memory processors. Moreover, the combination of MPI and OpenMP can improve the parallel efficiency and available memory size, in which internode parallelization is performed by MPI and intranode parallelization by OpenMP. The MPI/OpenMP hybrid parallelization makes large molecular calculations using a high level of theory possible without errors caused by approximate models. The hybrid parallelization has been introduced into various calculations, QMC,[24] tight-binding,[25] four-atom QM,[26] and Car−Parrinello molecular dynamics.[27] A similar approach, the combination of Linda and OpenMP, has been implemented into the Gaussian program package.[28] Another approach, the combination of MPI and Pthreads,[29] has been employed to a four-index transformation of 2-ERIs for electron correlation methods in the massively parallel program suite MPQC.[30] Communication bottlenecks are avoided by creating computation and communication threads and overlapping them.

The distribution of the 2-ERI calculation is one of the most important steps to apply the hybrid parallelization to quantum chemistry calculations. In this study, the MPI/OpenMP hybrid parallelization technique is introduced to the HF calculation, which is the basic theory of quantum chemistry. For self-consistent field (SCF) calculations, efficient distributed data parallel algorithms[31,32] have been proposed for distributed memory platforms. Our target is to calculate nanosize molecules which consist of hundreds of atoms using hundreds or thousands of CPU cores. For such computer systems, network communication could be a critical bottleneck. Therefore, we employ replicated data parallelization to reduce and control communication. Furthermore, the initial guess calculation is also parallelized and accelerated to improve the parallel efficiency of the whole calculation. On the basis of the HF parallelization, algorithms of DFT and electron correlation theories will be developed for large molecules, and most calculations based on the QM method will become faster.

```
!$OMP parallel do schedule(dynamic,1) reduction(+:Fock)
  do M= nshell, 1, -1
    do N = 1, M
      MN = M(M- 1) + N
      Λstart = mod(MN + mpi_rank, nproc) + 1
      do Λ = Λstart, M, nproc
        do Σ = 1, Λ
          Schwartz screening
          calculate (μν|λσ) and add into Fock matrix
        end do
      end do
    end do
  end do
!$OMP end parallel do
  Call MPI_AllReduce(Fock)
```

Figure 1. MPI/OpenMP hybrid parallel algorithm for two-electron integral generation.

2. Algorithm

The HF calculation mainly consists of the initial guess, Fock matrix generation, and new MO coefficient generation from the Fock matrix. The most time-consuming step is the Fock matrix **F** generation from the 2-ERI $(\mu\nu|\lambda\sigma)$ and the density matrix **D**,

$$(\mu\nu|\lambda\sigma) = \int \frac{\chi_\mu(r_1)\chi_\nu(r_1)\chi_\lambda(r_2)\chi_\sigma(r_2)}{r_1 - r_2} dr_1 dr_2 \qquad (1)$$

$$\mathbf{F}_{\mu\nu} = \mathbf{H}_{\mu\nu} + \sum_{\lambda\sigma} \mathbf{D}_{\lambda\sigma}\{2(\mu\nu|\lambda\sigma) - (\mu\lambda|\nu\sigma)\} \qquad (2)$$

where χ and **H** denote the contracted basis function and the one-electron integral matrix, respectively.

A new parallel algorithm for the 2-ERI generation is shown in Figure 1. 2-ERIs are generated in the quadruple loop of basis shells, M, N, Λ, and Σ. Indices of the first loop are distributed in intranode by OpenMP and indices of the third loop are distributed in internode by MPI ranks. At the beginning of a calculation, MPI ranks are set to each process.

The OpenMP parallelization is performed at the outermost loop to reduce OpenMP overheads, such as thread generation and data copy. The dynamical distribution and the order of the loop index from a large to a small task are applied to obtain better load balancing. The Fock matrix is allocated by each thread before the quadruple loop and accumulated to the master thread after the loop. Other valuables in the loop are distinguished into shared and private ones. Basis functions, coordinates, molecular orbital, and density matrices are shared with all threads in a process, and blocks of 2-ERIs and intermediate valuables are not shared. Therefore, all large arrays except for the Fock matrix are shared in a node, and data synchronization of threads is not needed during the

quadruple loop. Shared valuables are set as common or module ones, and private valuables are set as subroutine arguments of the Fortran language.

The third loop is chosen for the parallelization by MPI ranks from the balance between the parallel granularity and the cost, because the granularity of divided tasks becomes small in an inner loop and the cost of distribution, such as counters, becomes large in an inner loop. To reduce the cost of distribution, an if clause is not used, and the counter is located in the second loop. After the calculation finishes in all processes, the Fock matrix is accumulated, and all processes have the full Fock matrix.

In the initial guess calculation, the extended Hückel (EH) calculation is performed, and the obtained orbitals are projected from the basis for the EH calculation to the basis for the HF calculation. The following formula[33] including many matrix−matrix multiplications is used to make parallelization easy and to reduce the number of floating operations, which is used in the parallel quantum solutions (PQS) program suite:[34]

$$C_1 = S_{11}^{-1} S_{12} C_2 \{ C_2^t S_{12}^t S_{11}^{-1} S_{12} C_2 \}^{-1/2} \quad (3)$$

where C_1 is the initial guess molecular orbital (MO) coefficient for the HF calculation, C_2 is the EH MO coefficient, S_{11} is the overlap integrals of the basis for the HF calculation, and S_{12} is the overlap integrals of the bases between the HF and the EH calculations.

The Fock matrix diagonalization is performed only at the first and last iterations. During HF iterations, an approximate second-order self-consistent field (SOSCF) method[35,36] is applied to reduce the number of matrix diagonalizations because linear algebra package (LAPACK) routines are parallelized only in intranode.

Intranode parallelization of matrix−matrix multiplications and diagonalization is achieved using thread-parallel basic linear algebra subprograms (BLAS) and LAPACK libraries. In the calculation of matrix−matrix multiplications, columns of the matrix on the right side are distributed to each process, and the result of each process is collected to the master process. The diagonalization calculation is performed only in the master process, and then obtained molecular energies and orbitals are distributed to all processes.

3. Results and Discussion

The algorithm was implemented into the quantum chemistry program package GAMESS[37] version April 2008. Benchmark calculations on Cray XT5 (Opteron 2.4 GHz Shanghai core, 512 KB L2-Cache and 6 MB shared L3-Cache, 8 CPU cores per node) 2048 CPU cores were performed using a TiO_2 cluster ($Ti_{35}O_{70}$, 6-31G, 1645 basis functions) as a complicated system including d electrons and insulin ($C_{257}H_{381}N_{65}O_{77}S_6^{2-}$ (PDB code: 1HIU),[38] STO-3G, 2430 basis functions) as a simple system in which only s and p basis functions are used. The computer has a PGI Fortran compiler-8.0.2, LIBSCI-10.3.3.5 as a BLAS and LAPACK library, and XT-mpt-3.1.2.1 based on MPICH2−1.0.6p1 as an MPI library. The divide and conquer method[39] is used for diagonalization calculations. The threshold of the density matrix convergence is set to be 1.0×10^{-4}. The numbers of SCF cycles for the TiO_2 cluster and insulin are 30 and 45, respectively.

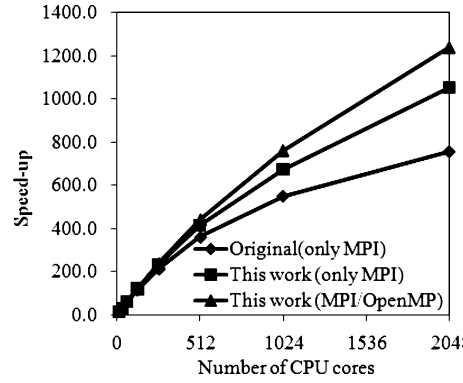

Figure 2. Speed-up of whole Hartree−Fock calculation for TiO_2 cluster.

Three kinds of parallel calculations were performed using the following programs: the original GAMESS program using only MPI, the developed programs using only MPI, and MPI for internode and OpenMP for intranode. Eight threads per process run were used in MPI/OpenMP hybrid calculations.

The speed-up of the whole HF calculations for the TiO_2 cluster is displayed in Figure 2. The calculations are performed using from 16 to 2048 CPU cores. The speed-up of 16 CPU cores is set to be 16, while the speed-up of 32 CPU cores is set to be 32 only for the original GAMESS of insulin. The speed-up of this work (only MPI) is improved in comparison with that of the original GAMESS (only MPI) because the initial guess calculation is simplified. Moreover, the speed-up of this work (MPI/OpenMP) is better than that of this work (only MPI) because the load balancing is improved by the MPI/OpenMP hybrid parallelization. It is surprising that the MPI/OpenMP calculation keeps the scaling performance even for 2048 CPU cores, though the parallel efficiency of the original GAMESS drops for 512 or 1024 CPU cores.

The MPI/OpenMP algorithm improves not only the speed-up but also the total computational (elapsed) time as shown in Table 1. The difference between the original GAMESS and this work (MPI/OpenMP) becomes large as the number of CPU cores increases. The MPI/OpenMP algorithm accelerates 1.6−3.7 times for 2048 CPU cores, while the computational times of the TiO_2 cluster for 16 CPU cores are almost the same. The MPI/OpenMP hybrid parallelization is achieved without increasing the computational cost.

Table 2 shows the computational time and speed-up of the Fock matrix generation for TiO_2 cluster and insulin. The speed-up of this work (MPI/OpenMP) is better than that of the original GAMESS, especially for over 512 CPU cores. For instance, the computational time of TiO_2 cluster is 174.8 s for 2048 CPU cores. Since the number of the SCF cycles is 30, the time per cycle is 5.8 s. This indicates that to make distributed tasks more equal by the hybrid parallelization is quite important to achieve extremely good speed-up.

The original GAMESS takes much more time than either of our two algorithms (only MPI, and MPI/OpenMP) for insulin.

Table 1. Computational Time (Second) and Speed-up (in Parentheses) of Whole Hartree-Fock Calculation

number of CPU cores		16	32	256	512	1024	2048
			TiO_2 Cluster				
original	only MPI	18 176.4	9223.1	1368.6	806.8	527.6	383.5
		(16.0)	(31.5)	(212.5)	(360.5)	(551.2)	(758.3)
this work	only MPI	18 045.6	9111.8	1241.2	695.2	428.7	273.7
		(16.0)	(31.7)	(232.6)	(415.3)	(673.5)	(1054.9)
this work	MPI/OpenMP	18 121.6	9052.4	1214.6	656.5	381.1	234.2
		(16.0)	(32.0)	(238.7)	(441.7)	(760.8)	(1238.0)
			Insulin				
original	only MPI		18 289.3	3629.9	2586.8	2054.1	1793.7
			(32.0)	(161.2)	(226.2)	(284.9)	(326.3)
this work	only MPI	21 988.8	11 157.5	1765.0	1073.4	721.7	540.9
		(16.0)	(31.5)	(199.3)	(327.8)	(487.5)	(650.4)
this work	MPI/OpenMP	22 377.1	11 337.6	1654.1	975.6	642.1	480.4
		(16.0)	(31.6)	(216.5)	(367.0)	(557.6)	(745.3)

Table 2. Computational Time (Second) and Speed-up (in Parentheses) of Fock Matrix Generation

number of CPU cores		16	32	256	512	1024	2048
			TiO_2 Cluster				
original	only MPI	17 881.8	8984.9	1175.2	614.0	334.0	188.6
		(16.0)	(31.8)	(243.5)	(466.0)	(856.6)	(1517.0)
this work	only MPI	17 953.5	9038.3	1175.2	627.9	360.0	203.1
		(16.0)	(31.8)	(244.4)	(457.5)	(797.9)	(1414.4)
this work	MPI/OpenMP	17 777.6	8903.9	1150.4	597.2	316.4	174.8
		(16.0)	(31.9)	(247.3)	(476.3)	(899.0)	(1627.2)
			Insulin				
original	only MPI		16 856.3	2491.9	1455.6	928.3	665.9
			(32.0)	(216.5)	(370.6)	(581.1)	(810.0)
this work	only MPI	21 454.4	10 827.8	1504.6	814.0	456.7	272.7
		(16.0)	(31.7)	(228.1)	(421.7)	(751.6)	(1258.8)
this work	MPI/OpenMP	20 664.9	10 392.8	1384.3	734.8	410.7	253.9
		(16.0)	(31.8)	(238.8)	(450.0)	(805.1)	(1302.2)

Table 3. Computational Time (Second) and Ratio (in Parentheses) of Initial Guess Calculation

number of CPU cores		16	32	256	512	1024	2048
			TiO_2 Cluster				
original	only MPI	163.7	151.6	142.0	141.2	141.7	141.9
		(0.9%)	(1.6%)	(10.4%)	(17.5%)	(26.9%)	(37.0%)
this work	only MPI	17.9	16.4	16.9	17.2	17.4	17.7
		(0.1%)	(0.2%)	(1.4%)	(2.5%)	(4.1%)	(6.5%)
this work	MPI/OpenMP	16.5	13.7	11.9	12.0	12.2	12.3
		(0.1%)	(0.2%)	(1.0%)	(1.8%)	(3.2%)	(5.3%)
			Insulin				
original	only MPI		695.6	667.8	671.6	674.0	667.8
			(3.8%)	(18.4%)	(26.0%)	(32.8%)	(37.2%)
this work	only MPI	74.8	72.8	73.2	73.7	74.2	74.6
		(0.3%)	(0.7%)	(4.1%)	(6.9%)	(10.3%)	(13.8%)
this work	MPI/OpenMP	68.6	59.1	52.8	52.6	53.0	53.3
		(0.3%)	(0.5%)	(3.2%)	(5.4%)	(8.3%)	(11.1%)

In the original, the if clause and the counter are used for the distribution of the 2-ERI calculation in the third loop, and all processes run until the if clause. A lot of 2-ERI are skipped by the Schwartz screening because of the tight basis set, STO-3G, and the computational cost for each integral is small. The cost for the if clause and the counter becomes relatively high, although that of the TiO_2 cluster is negligible because the computational cost of 2-ERI including *d* functions is large. Therefore, the developed algorithm can drastically reduce the computational time due to the simple distribution.

Table 3 summarizes the computational time and the ratio of the initial guess calculation for the TiO_2 cluster and the insulin. In the original, the initial guess calculation occupies about 37% of the total calculation time for 2048 CPU cores, though the ratio is less than 1% for 16 CPU cores. The ratio of this work (MPI/OpenMP) becomes about 5−11% even for 2048 CPU cores because of the simple orbital projection. The acceleration of the initial guess calculation contributes to the speed-up of the total time. The parallelization and acceleration of all steps are significant, and the initial guess calculation is not a bottleneck of parallel HF calculations for hundreds or thousands of CPU cores.

The computational time and the ratio of solving the Hartree−Fock equation are shown in Table 4, in which the diagonalization of the Fock matrix in the first and last iterations, the approximate SOSCF calculation in other

Table 4. Computational Time (Second) and Ratio (in Parentheses) of Solving the Hartree-Fock Equation

number of CPU cores		16	32	256	512	1024	2048
			TiO_2 Cluster				
original	only MPI	128.4 (0.7%)	84.4 (0.9%)	49.8 (3.6%)	49.6 (6.1%)	50.0 (9.5%)	51.1 (13.3%)
this work	only MPI	71.9 (0.4%)	55.2 (0.6%)	47.4 (3.8%)	48.3 (6.9%)	49.8 (11.6%)	51.4 (18.8%)
this work	MPI/OpenMP	325.4 (1.8%)	133.1 (1.5%)	51.0 (4.2%)	46.0 (7.0%)	51.1 (13.4%)	45.6 (19.5%)
			Insulin				
original	only MPI		732.3 (4.0%)	466.5 (12.9%)	456.4 (17.6%)	448.6 (21.8%)	456.6 (25.5%)
this work	only MPI	444.8 (2.0%)	246.4 (2.2%)	182.4 (10.3%)	181.6 (16.9%)	187.6 (26.0%)	191.1 (35.3%)
this work	MPI/OpenMP	1630.1 (7.3%)	875.7 (7.7%)	213.6 (12.9%)	185.4 (19.0%)	176.2 (27.4%)	171.1 (35.6%)

iterations, and the new density matrix generation are performed. The computational times of all algorithms are reduced as the number of CPU cores increases because the SOSCF calculation is originally parallelized by MPI. The difference of the computational time for insulin cluster is large for 2048 CPU cores. The dimension of the Fock and MO matrices is 1.5 times larger than that for the TiO_2 cluster, and cache misses easily occur in the density matrix generation because BLAS routines are not used in the original. The difference between only MPI and MPI/OpenMP of insulin is 20 s for 2048 CPU cores. This comes from the intranode parallelization of the Fock matrix diagonalization. This indicates that introducing internode parallelization of diagonalization is significant to reduce the computational time more.

4. Conclusion

We developed the new MPI/OpenMP hybrid parallel algorithm of the HF calculation and implemented it into the GAMESS program. The computational time and the speedup of the whole Hartree−Fock (HF) calculation are improved by introducing the MPI/OpenMP hybrid parallelization and the simple formula for the initial guess calculation. The basis sets used here are 6-31G and STO-3G. When a larger basis set is used, the hybrid parallelization effect is expected to be more important. The ratio of the initial guess calculation will decrease because the numbers of occupied orbitals and basis functions for the extended Hückel (EH) calculation are constant. The hybrid parallelization can reduce the amount of the memory use per node at the replicated data approach because all large matrices except for the Fock matrix are shared with all threads in a process.

For the Fock matrix generation, the better load balancing is obtained by the OpenMP dynamic distribution and the less MPI processes compared to that of the conventional MPI parallelization. The reduction of the computational time is also achieved by the distribution without the if clause. The ratio of the initial guess calculation becomes about 37% for 2048 CPU cores using the original GAMESS. The computational time and the ratio are drastically reduced by the use of the simple formula for the orbital projection and the parallelization of all steps. It is significant to apply basic linear algebra subprograms (BLAS) and linear algebra package (LAPACK) libraries of matrix multiplications and diagonalization for intranode parallelization and reduction of cache misses. The introduction of internode parallelization of diagonalization is necessary to accelerate more.

HF calculations of nanosize molecules without errors caused by modeling or approximations are now practical with hundreds or thousands of CPU cores. On the basis of the parallelization technique, algorithms for DFT and electron correlation calculations will be developed with high parallel efficiencies and large memory arrays, and most calculations based on the QM method will be accelerated.

References

(1) Warshel, A.; Karplus, M. *J. Am. Chem. Soc.* **1972**, *94*, 5612–5625.

(2) Warshel, A.; Levitt, M. *J. Mol. Biol.* **1976**, *103*, 227–249.

(3) Maseras, F.; Morokuma, K. *J. Comput. Chem.* **1995**, *16*, 1170–1179.

(4) Dapprich, S.; Komaromi, I.; Byun, K. S.; Morokuma, K.; Frisch, M. J. *J. Mol. Struct. (THEOCHEM)* **1999**, *461*, 1–21.

(5) Kitaura, K.; Ikeo, E.; Asada, T.; Nakano, T.; Uebayasi, M. *Chem. Phys. Lett.* **1999**, *313*, 701–706.

(6) Fedorov, D. G.; Kitaura, K. *J. Chem. Phys.* **2004**, *120*, 6832–6840.

(7) Chen, Z.; Nagase, S.; Hirsch, A.; Haddon, R. C.; Thiel, W.; Schleyer, P. v. R. *Angew. Chem., Int. Ed. Engl.* **2004**, *43*, 1552–1554.

(8) Fedorov, D. G.; Kitaura, K. *Chem. Phys. Lett.* **2004**, *389*, 129–134.

(9) Anderson, A. G.; Goddard, W. A., III; Schroder, P. *Comput. Phys. Commun.* **2007**, *177*, 298–306.

(10) Yasuda, K. *J. Chem. Theory Comput.* **2008**, *4*, 1230–1236.

(11) Yasuda, K. *J. Comput. Chem.* **2008**, *29*, 334–342.

(12) Vogt, L.; Olivares-Amaya, R.; Kermes, S.; Shao, Y.; Amador-Bedolla, C.; Aspuru-Guzik, A. *J. Phys. Chem. A* **2008**, *112*, 2049–2057.

(13) Ufimtsev, I. S.; Martinez, T. J. *J. Chem. Theory Comput.* **2008**, *4*, 222–231.

(14) Ufimtsev, I. S.; Martinez, T. J. *J. Chem. Theory Comput.* **2009**, *5*, 1004–1015.

(15) Brown, P.; Woods, C.; McIntosh-Smith, S.; Manby, F. R. *J. Chem. Theory Comput.* **2008**, *4*, 1620–1626.

(16) Whitten, J. L. *J. Chem. Phys.* **1973**, *58*, 4496–4501.

(17) TOP500 Supercomputing Sites; http://www.top500.org (accessed Mar 19, 2010).

(18) Nieplocha, J.; Harrison, R. J.; Littlefield, R. J. *Proceedings of the 1994 ACM/IEEE conference on Supercomputing,* **1994**, pp 340–349.

(19) Nieplocha, J.; Harrison, R. J.; Littlefield, R. J. *J. Supercomput.* **1996**, *10*, 197–220.

(20) Fletcher, G. D.; Schmidt, M. W.; Bode, B. M.; Gordon, M. S. *Comput. Phys. Commun.* **2000**, *128*, 190–200.

(21) Sosa, C. P.; Scalmani, G.; Gomperts, R.; Frisch, M. J. *Parallel Comput.* **2000**, *26*, 843–856.

(22) Woods, C. J.; Brown, P.; Manby, F. R. *J. Chem. Theory Comput.* **2009**, *5*, 1776–1784.

(23) Bolding, B.; Baldridge, K. *Comput. Phys. Commun.* **2000**, *128*, 55–66.

(24) Smith, L.; Kent, P. *Concurrency: Pract. Exper.* **2000**, *12*, 1121–1129.

(25) Shellman, S. D.; Lewis, J. P.; Glaesemann, K. R.; Sikorski, K.; Voth, G. A. *J. Comput. Phys.* **2003**, *188*, 1–15.

(26) Medvedev, D. M.; Goldfield, E. M.; Gray, S. K. *Comput. Phys. Commun.* **2005**, *166*, 94–108.

(27) Hutter, J.; Curioni, A. *Parallel Comput.* **2005**, *31*, 1–17.

(28) , Frisch, M. J.; Trucks, G. W.; Schlegel, H. B.; Scuseria, G. E.; Robb, M. A.; Cheeseman, J. R.; Scalmani, G.; Barone, V.; Mennucci, B.; Petersson, G. A.; Nakatsuji, H.; Caricato, M.; Li, X.; Hratchian, H. P.; Izmaylov, A. F.; Bloino, J.; Zheng, G.; Sonnenberg, J. L.; Hada, M.; Ehara, M.; Toyota, K.; Fukuda, R.; Hasegawa, J.; Ishida, M.; Nakajima, T.; Honda, Y.; Kitao, O.; Nakai, H.; Vreven, T.; Montgomery, Jr. J. A.; Peralta, J. E.; Ogliaro, F.; Bearpark, M.; Heyd, J. J.; Brothers, E.; Kudin, K. N.; Staroverov, V. N.; Kobayashi, R.; Normand, J.; Raghavachari, K.; Rendell, A.; Burant, J. C.; Iyengar, S. S.; Tomasi, J.; Cossi, M.; Rega, N.; Millam, J. M.; Klene, M.; Knox, J. E.; Cross, J. B.; Bakken, V.; Adamo, C.; Jaramillo, J.; Gomperts, R.; Stratmann, R. E.; Yazyev, O.; Austin, A. J.; Cammi, R.; Pomelli, C.; Ochterski, J. W.; Martin, R. L.; Morokuma, K.; Zakrzewski, V. G.; Voth, G. A.; Salvador, P.; Dannenberg, J. J.; Dapprich, S.; Daniels, A. D.; Farkas, O.; Foresman, J. B.; Ortiz, J. V.; Cioslowski, J.; Fox, D. J. *Gaussian 09*, Revision A.1; Gaussian, Inc.: Wallingford, CT, 2009.

(29) Nielsen, I. M. B.; Janssen, C. L. *Comput. Phys. Commun.* **2000**, *128*, 238–244.

(30) Janssen, C. L.; Nielsen, I. M. B.; Leininger, M. L.; Valeev, E. F.; Kenny, J. P.; Seidl, E. T. *The Massively Parallel Quantum Chemistry Program (MPQC)*; Sandia National Laboratories: Livemore, CA, 2008.

(31) Alexeev, Y.; Kendall, R. A.; Gordon, M. S. *Comput. Phys. Commun.* **2002**, *143*, 69–82.

(32) Takashima, H.; Yamada, S.; Obara, S.; Kitamura, K.; Inabata, S.; Miyakawa, N.; Tanabe, K.; Nagashima, U. *J. Comput. Chem.* **2002**, *23*, 1337–1346.

(33) Cremer, D.; Gauss, J. *J. Comput. Chem.* **1986**, *7*, 274–282.

(34) Baker, J.; Wolinski, K.; Malagoli, M.; Kinghorn, D.; Wolinski, P.; Magyarfalvi, G.; Saebo, S.; Janowski, T.; Pulay, P. *J. Comput. Chem.* **2009**, *30*, 317–335.

(35) Fischer, T. H.; Almlof, J. *J. Phys. Chem.* **1992**, *96*, 9768–9774.

(36) Chaban, G.; Schmidt, M. W.; Gordon, M. S. *Theor. Chem. Acc.* **1997**, *97*, 88–95.

(37) Schmidt, M. W.; Baldridge, K. K.; Boatz, J. A.; Elbert, S. T.; Gordon, M. S.; Jensen, J. H.; Koseki, S.; Matsunaga, N.; Nguyen, K. A.; Su, S.; Windus, T. L.; Dupuis, M.; Montgomery, J. A. *J. Comput. Chem.* **1993**, *14*, 1347–1363.

(38) Hua, Q. X.; Shoelson, S. E.; Kochoyan, M.; Weiss, M. A. *Nature* **1991**, *354*, 238–241.

(39) Cuppen, J. J. M. *Numer. Math.* **1981**, *36*, 177–195.

CT100083W