Sparse Matrix Multiplications for Linear Scaling Electronic Structure Calculations in an Atom-Centered Basis Set Using Multiatom Blocks

CHANDRA SARAVANAN,[1,2] YIHAN SHAO,[1] ROI BAER,[3] PHILIP N. ROSS,[2] MARTIN HEAD–GORDON[1]

[1]*Department of Chemistry, University of California, Berkeley, California 94720*
[2]*Materials Sciences Division, Lawrence Berkeley National Laboratory, Berkeley, California 94720*
[3]*Department of Physical Chemistry, The Hebrew University, Jerusalem 91904, Israel*

Received 21 June 2002; Accepted 3 October 2002

Abstract: A sparse matrix multiplication scheme with multiatom blocks is reported, a tool that can be very useful for developing linear-scaling methods with atom-centered basis functions. Compared to conventional element-by-element sparse matrix multiplication schemes, *efficiency is gained* by the use of the highly optimized basic linear algebra subroutines (BLAS). However, some *sparsity is lost* in the multiatom blocking scheme because these matrix blocks will in general contain negligible elements. As a result, an optimal block size that minimizes the CPU time by balancing these two effects is recovered. In calculations on linear alkanes, polyglycines, estane polymers, and water clusters the optimal block size is found to be between 40 and 100 basis functions, where about 55–75% of the machine peak performance was achieved on an IBM RS6000 workstation. In these calculations, the blocked sparse matrix multiplications can be 10 times faster than a standard element-by-element sparse matrix package.

© 2003 Wiley Periodicals, Inc. J Comput Chem 24: 618–622, 2003

Key words: linear scaling; electronic structure calculations; multiatom blocks

Introduction

The last decade has seen a surge of computational methods that address the scaling issues in self-consistent field (SCF) electronic structure methods such as Hartree–Fock (HF) and density functional theory (DFT). With atom-centered basis sets, SCF calculations involve two computationally expensive steps: (1) the evaluation of an effective one-electron hamiltonian (Fock operator) for a given electron density, and (2) the updating of a Slater determinant wave function for a given Fock operator.

Effective $O(N)$ algorithms now exist for evaluation of all components of the Fock operator: Coulomb, exact exchange, and exchange-correlation potential. For the Coulomb matrix, these include the continuous fast multipole method (CFMM)[1–5] and tree-code approaches.[6] The ONX and LinK algorithms are conditionally linear scaling approaches for forming the Hartree–Fock exact exchange matrix.[7–10] Linear scaling quadrature methods are used for formation of the exchange-correlation potential.[11]

Traditionally, the Slater determinant wave function is updated by first diagonalizing the Fock matrix and then using the eigenvectors as new molecular orbitals. The diagonalization formally scales cubically with the system size. Clearly, with the advent of fast Fock matrix build algorithms, the diagonalization step can become the bottleneck in HF or DFT calculations on large systems.[12–14] To avoid this cubic-scaling step, alternative methods have been proposed.[12] They usually involve (1) solving directly for the one-particle density matrix (1PDM) in a direct-minimization procedure (such as Li–Nunes–Vanderbilt scheme,[15–21] curvy-step approach,[22–24] a quadratically convergent approach[25]), (2) an iterative procedure,[26,27] or (3) a Chebyshev polynomial expansion method.[28,29]

All these methods involve only matrix manipulations—multiplications, additions, traces, etc. In a local basis, such as the atom-centered Gaussian functions commonly used in electronic

Correspondence to: M. Head–Gordon; e-mail: mhg@cchem.berkeley.edu

Contract/grant sponsor: NSF; contract/grant number: CHE-9981997 (to M.H.G.)

Contract/grant sponsor: U.S. Binational Science Foundation; contract/grant number: 1998108 (to R.B.)

Contract/grant sponsor: Assistant Secretary or Energy and Renewable Energy, Office of Advanced Automotive Technology of the Department of Energy; contract/grant number: DE-AC03-76SF0098 (to P.N.R.)

structure calculations,[30] all required matrices (operator matrices, and the 1PDM) become sparse as system size grows. Asymptotically the number of significant elements in the matrices grows linearly with the system size. Therefore, all matrix manipulations can be carried out in $O(N)$ cost, thereby paving the way for an overall linear-scaling behavior in updating the wave function.

In this article, we explore efficient ways to carry out such matrix multiplications. Other matrix manipulations can be done in a similar way, and are far less expensive than multiplications. One standard way to carry out sparse matrix multiplications (e.g., SPARSKIT[31]) is to represent the matrices in a compressed sparse row (CSR)[32,33] format and to multiply one pair of matrix elements at a time. This is a good approach (indeed, a necessary approach) if the data is distributed randomly in a structureless sparse matrix. However, it is suboptimal compared to approaches that exploit the structure present in the sparse matrices because of the particular physical problem being solved. Such structure can increase the efficiency of matrix multiplies by lowering the overhead of indexing the sparse array.

In SCF calculations, the sparse matrices indeed have structure. For example, in insulating systems the off-diagonal elements of the 1PDM $\rho(r, r')$ in the position representation decay exponentially with distance i.e., $\rho(r, r') \approx \exp(-\gamma|r - r'|)$, where γ is related to the band gap of the system (see ref. 12 for a review). As a result, the density matrix is local in an atom-centered basis representation. In this basis, other matrices such as the Fock matrix and the overlap matrix are even more local[20,34] because of the basis function locality. To make use of the locality of these matrices, one can group basis functions with common or close-by atom centers together into blocks. A large matrix then becomes a set of small submatrices, each corresponding to two blocks of basis functions. One then needs to keep track of submatrices rather than individual matrix elements.

The first effort in this direction for electronic structure calculations was reported by Challacombe,[21,35] who organized a blocking scheme at the level of grouping together all functions on a common atom. With this scheme, the cost of bookkeeping is reduced, but the submatrices are still small. For example, for a molecule containing first-row elements (Li–Ne) and hydrogen, the submatrices are 5×5, 5×1, or 1×1 with an STO-3G basis, and 15×15, 15×2, or 2×2 with a 6-31G* basis.[36] For these small submatrices, it is not very efficient to use the widely used vendor-supplied level-3 basic linear algebra subroutines (BLAS), which are optimized for larger matrices. Challacombe employed alternate linear algebra routines from PHiPAC[37] optimized for small blocks. ATLAS[38] subroutines can also be used instead of PHiPAC. Our benchmark calculations show that ATLAS performs marginally better than BLAS (on IBM RS6000 375 MHz) for small block sizes. For example, for multiplications involving 15×15 matrices (carbon in a 6-31G* basis) ATLAS achieves 26% of machine peak performance while BLAS achieves 20% of peak performance. Although ATLAS offers some promise for small blocks, its performance is very similar to that of BLAS for large blocks. For example, for multiplications involving 57×57 matrices (3 CH_2 groups in a 6-31G* basis) both BLAS and ATLAS achieve 57% of the machine peak performance.

In this article, we explore the extent to which further improvements are possible by considering generalized blocking schemes, where all functions from several atoms are grouped together to define a block. The resulting blocks will be larger, which permits greater efficiency to be obtained on the dense block–block matrix multiplies. However, some sparsity is lost compared to using smaller blocks. In this article, we will assess the efficiency of using many-atom blocks. The rest of the article is arranged as follows. In the next section, we outline our algorithm for dividing a molecular system into blocks, and describe our procedure to carry out multiplications with such multiatom blocks. Then we present our preliminary results on linear alkanes, estane polymers, polyglycines, and water clusters. Finally, we provide some concluding remarks.

Algorithm

There are many conceivable ways to partition a molecular system. For example, one can solve a saleman's problem to find a shortest route covering every atom once, and then divide the system along the route. We can also define other target functions and optimize those functions to get a partitioning. The simplest way, however, is to follow the partitioning scheme in the continuous fast multipole method (CFMM).[2,39] The molecule is encapsulated into a rectangular box. For systems with dimensions much larger than a prespecified cutoff distance R_{cut} (discussed later in this section), we divide the parent boxes along the lengthy dimensions using a binary scheme, where each parent box is divided into two child boxes. Once the box dimensions are less than or equal to R_{cut}, each box is divided into multiple small boxes. The division of the boxes stops when the average number of basis functions in a box (excluding the empty boxes) reaches a target number of basis functions. Although this scheme is straightforward and general, it is only optimal for systems that are spatially homogeneous (i.e., have very small density fluctuations). Otherwise, the number of basis functions per block may fluctuate substantially about the optimal average. More elaborate schemes to define blocks of uniform size can certainly be envisaged (nonuniform box sizes, etc.), but are beyond our present scope.

In the context of SCF calculations, one can partition the system well before all the SCF cycles, which amounts to a reordering of the atoms (and thus basis functions) in the molecule. Equivalently, we can keep the ordering of the atoms as in the input and carry out two permutations in every SCF cycle: the permutation of the Fock matrix into the desired blocked structure at the end of a Fock build and the back permutation of 1PDM. Both permutations can be done with an $O(N)$ effort with a very small prefactor. In this work, we follow the latter scheme.

Once the boxes and thus the basis function blocks are made, we can save a sparse matrix in terms of its submatrices. The elements in the submatrices can be stored continuously in memory to avoid big memory strides and associated cache misses when a submatrix is being used. If a system is divided into N_{box} blocks, then there are $N_{\text{box}} * N_{\text{box}}$ submatrices in total, of which many can be neglected. As far as the total memory requirement is concerned, one can have an $O(N^2)$ storage if one allocates space for all the submatrices including the negligible ones. $O(N)$ memory storage is also possible, but it requires the knowledge of the number of nonnegligible submatrices per row (or column). In other words, we need to know a cutoff radius. In SCF calculations, it is easy to find the cutoff radius for Fock matrix. The cutoff radius for density matrix is

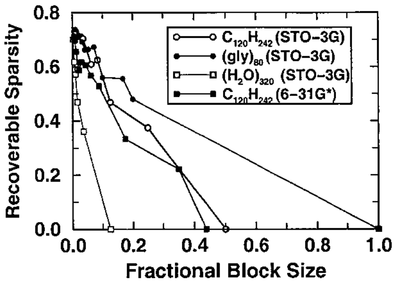

Figure 1. Recoverable sparsity as function of fractional block sizes. Fractional block size is defined as the ratio of the block size (in number of basis functions) to the total number of basis functions. This is shown for a linear alkane ($C_{120}H_{242}$, STO-3G basis), a glycine oligomer (80 gly units, STO-3G basis), a water cluster (320 water molecules, STO-3G basis), and linear alkane ($C_{120}H_{242}$) in a larger basis (6-31G*).

slightly larger.[20,34] We found it safe to set R_{cut}, the cutoff radius for all matrices, to be two or three times that of the Fock matrix. This corresponds to a fixed format (FF) scheme[20] with a large cutoff radius.

There are many ways to store a blocked sparse matrix and do the bookkeeping. For a given submatrix one needs easy access to important information such as the indices of the multiatom blocks involved, the size of the submatrix, and a pointer to its address. But such choices do not affect the overall performance for the multiatom blocking scheme because the "real" work is done by DGEMM, the level-3 BLAS subroutine for multiplying matrices. We adopt the blocked compressed sparse row (B-CSR) $O(N)$ storage scheme,[21,32,33,35] in which a double precision number is used to keep track of the largest (absolute) element in submatrices of different sizes. During multiplication, submatrices A_{ik} and B_{kj} (within the radius R_{cut}) are multiplied to obtain a contribution to submatrix C_{ij}, where i, j, and k represent multiatom blocks. The block C_{ij} is obtained only if C_{ij} is within the cutoff radius, i.e., the shortest distance between atoms in the ith block and those in the jth block is less than or equal to R_{cut}. Next, we compute the product of the largest (absolute) elements from the two submatrices. DGEMM is called if the product is larger than $thresh \times 10^{-2}$, where *thresh* is a prespecified cutoff.

Results

In this section we present results of our multiatom block matrix multiplication scheme. Test calculations are performed by multiplying converged Fock matrices, **F**, and the corresponding density matrices, **P**, from a Hartree–Fock calculation.

In Figure 1 we plot *recoverable sparsity* (s_r, which is defined as the fraction of negligible submatrices) of matrix **F** × **P** against the fractional average block size (defined as the ratio of the average block size to total number of basis functions). We show this for $C_{120}H_{242}$ (STO-3G, 842 basis functions), polyglycine with 80 gly units (STO-3G basis, 1842 basis functions), water cluster with 320 water molecules (STO-3G basis, 2240 basis functions), and $C_{120}H_{242}$ (6-31G* basis, 2284 basis functions). This figure clearly shows that as one uses larger atom blocks, one recovers less sparsity. In the limit of the whole system being a big block, all sparsity is lost. Although this is generally true, in Figure 1 we observe that at some block sizes the sparsity actually increases with increase in block size. This intriguing bumpy behavior of curves in Figure 1 can be easily understood as follows. For example, in a linear alkane suppose that R is the range (in the number of CH_2 units) that a particular CH_2 group can "see" in one direction, and U is the average number of CH_2 units per block. Then the total number of nonnegligible blocks (N_B) in a row is given by

$$N_B = 2 \times (\text{int})\text{ceil}(R/U) + 1, \tag{3.1}$$

where, ceil(R/U) is the smallest integer greater than or equal to R/U. The fraction of nonnegligible blocks is N_B/N_{tot}, where N_{tot} is the total number of blocks per row. The recoverable sparsity is then given by

$$s_r = 1 - N_B/N_{tot}. \tag{3.2}$$

Note that for a given system, the basis set plays a role only through R. For $C_{120}H_{240}$, we find $R \cong 16$ with a STO-3G basis and a $thresh = 10^{-6}$. If U is increased from 7 (49 basis functions) to 8 (56 basis functions), N_B "jumps" from 7 to 5, N_{tot} "jumps" from 18 to 15 causing an increase in s_r from 0.61 to 0.67.

In Figure 2 we plot the fraction of nonnegligible blocks ($1 - s_r$) (with a fixed block size of approximately 50 basis functions) against the fraction of nonnegligible elements ($1 - s_e$, where s_e is the actual sparsity) for linear alkanes in a STO-3G basis. Because the actual sparsity is the maximum sparsity that one can recover, $1 - s_r$ is always larger than $1 - s_e$, as can clearly be seen from Figure 2. The actual ratio of ($1 - s_r$) to ($1 - s_e$) is found to be around 2 for all linear alkanes, suggesting that in every submatrix about half of the elements are negligible.

As block size increases, the submatrices become larger. Because BLAS is better optimized for larger matrices, one would expect to perform more floating point operations per second (FLOPs/s). This is confirmed from Figure 3, which shows FLOPs/sec achieved on an IBM RS/6000 375-MHz workstation capable of performing up to four FLOPs per clock cycle. From Figure 3 it is clear that we achieve 70–80% of machine peak performance

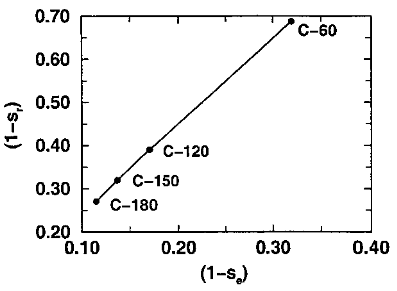

Figure 2. Fraction of nonnegligible blocks ($1 - s_r$) vs. fraction of nonnegligible elements ($1 - s_e$) for a block size of approximately 50 basis functions for linear alkanes $C_{60}H_{122}$, $C_{120}H_{242}$, $C_{150}H_{302}$, and $C_{180}H_{362}$ in a STO-3G basis.

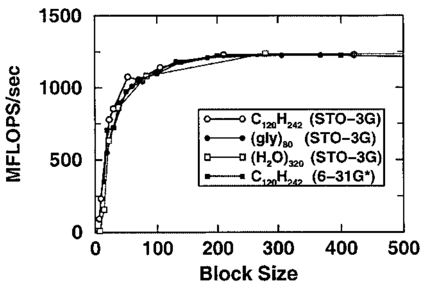

Figure 3. Efficiency of BLAS subroutines (in the Engineering Scientific Subroutine Library (ESSL) on IBM RS/6000) as a function of block size. It is clear that for block sizes above 75 basis functions, we obtain up to 70–80% of machine peak performance (1500 MFLOPs per second).

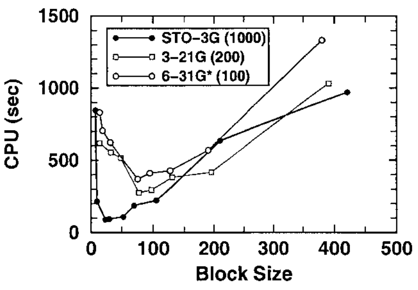

Figure 5. Showing basis-set dependence of optimal block size. The optimal block size increases going from the mimimal basis to any other basis. The figure also shows a similar optimal block sizes for 3-21G and 6-31G* basis sets.

(1500 MFLOPs per second) with a block size of over 75 basis functions.

From Figure 1 we realize that small blocks are needed to recover more sparsity. However, Figure 3 shows that large blocks are needed to approach peak efficiency of BLAS (i.e., more FLOPs per second). We, therefore, expect to see an optimal block size that balances these two effects. This is shown in Figure 4, where the CPU time is plotted as a function of block size. The numbers in the parentheses are the number of repeated **F** × **P** multiplications. The figure suggests an optimal block size of between 40–100 basis functions for all test systems, where we achieved about 55–75% (825–1125 MFLOPs/s) of the machine's peak performance.

In Figure 4 we also compare the performance of SPARSKIT[31] with our blocked multiplication. Benchmark calculations are performed on linear alkane $C_{120}H_{242}$ in a STO-3G basis. We note that SPARSKIT is around 10 times slower than our multiplication scheme. Note that part of this speedup is because element-by-element sparsity schemes must evaluate all contributions that are implied by the two input matrices, because testing the magnitude of each contribution is prohibitively expensive. By contrast, with blocking one can threshold based on the maximum elements in each block–block product with virtually no overhead. This is an additional advantage of using the blocking approach.

It is interesting to examine the basis set effect on the optimal block size. Figure 5 shows that the optimal block size increases with the size of basis set when we go from a minimal basis to any larger basis set. However, the optimal block size does not change a lot among the larger basis sets (3-21G and 6-31G*). This is explained as follows. The CPU time t for a single sparse matrix multiplication can be written as

$$t = \frac{\text{number of FLOPs}}{\text{number of FLOPs per second}}. \qquad (3.3)$$

The denominator (i.e., the number of FLOPs per second) does not depend on the size of the basis set (see Fig. 3) for a given block size. On the other hand, the numerator (total number of FLOPs per matrix multiplication) strongly depends on the size of the basis set. If N_b is the total number of basis functions, the numerator is approximately given as:

$$\text{number of FLOPs} = \alpha \cdot N_b^3 \cdot (1 - s_r)^2, \qquad (3.4)$$

where $\alpha = 1$ in a let-it-grow (LIG) scheme,[20] and $\alpha < 1$ for a sparse matrix multiplication performed *only* with a distance cutoff. The basis set dependence of the number of FLOPs can be seen in Figure 6, where we plot the number of FLOPs for $C_{120}H_{242}$ in the STO-3G (842 basis functions), 3-21G (1564 basis functions), and

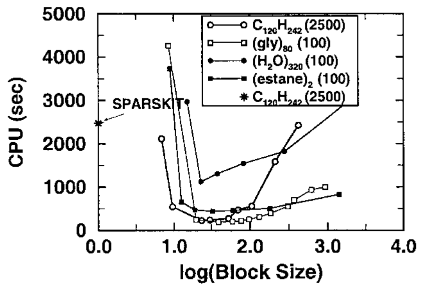

Figure 4. CPU time (in seconds) as a function of the block size showing an optimal block size of between 40–100 basis functions. The numbers in parentheses represent the number of **F** × **P** multiplications. We note that SPARSKIT takes 2470 s for 2500 **F** × **P** multiplications (for $C_{120}H_{242}$ in a STO-3G basis) while our blocked multiplication (with average block size of 30) takes only 230 s.

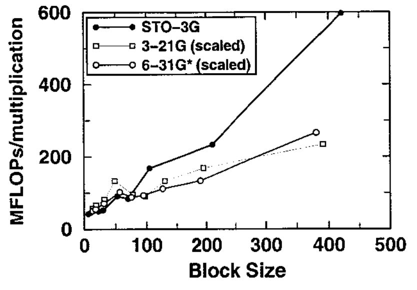

Figure 6. Showing the number of FLOPs per sparse multiplication as a function of block size for $C_{120}H_{242}$ at STO-3G (842 basis functions), 3-21G (1564 basis functions), and 6-31G* (2284 basis functions) level. The number of FLOPs for 3-21G and 6-31G* are scaled by $(842/1564)^3$ and $(842/2284)^3$, respectively.

Table 1. Showing the Scaling for the Multiatom Block Matrix Multiplication Scheme.

Alkane	Average block size	CPU seconds
$C_{60}H_{122}$	52.75	110
$C_{120}H_{242}$	52.62	276
$C_{150}H_{302}$	52.60	360
$C_{180}H_{362}$	52.58	449

The CPU time represents time for 2500 $\mathbf{F} \times \mathbf{P}$ multiplications.

6-31G* (2284 basis functions) basis sets. The curves for 3-21G and 6-31G* basis set are scaled by $(842/1564)^3$ and $(842/2284)^3$, respectively, for the purpose of comparison. As larger basis sets assign more basis functions to a CH_2 unit, the FLOP count grows slower with respect to block size for 3-21G and 6-31G* basis sets. Moreover, as discussed previously, the time per FLOP is independent of basis set size (see Fig. 3). These two factors result in a larger optimal block size [i.e., minimum CPU time in eq. (3.3)] for larger basis sets as shown in Figure 5.

In Table 1, we show the overall scaling for our block matrix multiplication scheme for linear alkanes in STO-3G basis. Clearly, the CPU time grows almost linearly with the system size between $C_{120}H_{242}$ and $C_{180}H_{362}$.

Concluding Remarks

We have discussed a multiatom blocked sparse matrix multiplication scheme, a tool useful for developing linear-scaling methods based on atom-centered orbitals. In this scheme large nonzero submatrices are obtained by forming many-atom blocks. We show that although the fraction of negligible submatrices (recoverable sparsity) is lower than the actual elemental sparsity, we benefit from the use of highly optimized level-3 basic linear algebra subroutines for large matrix sizes such that total time is reduced.

By performing matrix multiplications on sample systems (linear alkanes, polyglycines, water cluster, and estane globules) with various basis sets, the optimal block size is found to range between 40–100 basis functions. In this range we obtain 55–75% of the machine's peak performance. We also show that our scheme can be 10 times faster than SPARSKIT, a package based on element-by-element sparse matrix operations. Our calculations indicate that the optimal block size increases from minimal basis set to any larger basis sets. We also show that the optimal block sizes are similar for larger basis sets. In the future, we will report the use of this multiplication tool in our linear-scaling density functional theory calculations.[40]

Acknowledgments

The authors thank E. Schwegler and M. Challacombe for valuable discussions. M.H.G. is on appointment as a Miller Research Professor in Miller Institute for Basic Research in Science.

References

1. White, C. A.; Johnson, B. G.; Gill, P. M. W.; Head–Gordon, M. Chem Phys Lett 1994, 230, 8.
2. White, C. A.; Johnson, B. G.; Gill, P. M. W.; Head–Gordon, M. Chem Phys Lett 1996, 253, 268.
3. White, C. A.; Head–Gordon, M. J Chem Phys 1996, 104, 2620.
4. Shao, Y.; Head–Gordon, M. Chem Phys Lett 2000, 323, 425.
5. Strain, M. C.; Scuseria, G. E.; Frisch, M. J. Science 1996, 271, 51.
6. Challacombe, M.; Schwegler, E. J Chem Phys 1999, 106, 5526.
7. Burant, J. C.; Scuseria, G. E.; Frisch, M. J. J Chem Phys 1996, 105, 8969.
8. Schwegler, E.; Challacombe, M.; Head–Gordon, M. J Chem Phys 1997, 106, 9708.
9. Ochsenfeld, C.; White, C. A.; Head–Gordon, M. J Chem Phys 1998, 109, 1663.
10. Schwegler, E.; Challacombe, M. J Chem Phys 1999, 111, 6223.
11. Stratmann, R. E.; Scuseria, G. E.; Frish, M. J. Chem Phys Lett, 1996, 257, 213.
12. Goedecker, S. Rev Mod Phys 1999, 71, 1085.
13. Scuseria, G. E. J Phys Chem A 1999, 103, 4782.
14. Shao, Y.; White, C. A.; Head–Gordon, M. J Chem Phys 2001, 114, 6572.
15. Li, X. P.; Nunes, W.; Vanderbilt, D. Phys Rev B 1993, 47, 10891.
16. Nunes, R. W.; Vanderbilt, D. Phys Rev B 1994, 50, 17611.
17. Daw, M. S. Phys Rev B 1993, 47, 10895.
18. Xu, C. H.; Scuseria, G. E. Chem Phys Lett 1996, 262, 219.
19. Daniels, A. D.; Millam, J. M.; Scuseria, G. E. J Chem Phys 1997, 107, 425.
20. Millam, J. M.; Scuseria, G. E. J Chem Phys 1997, 106, 5569.
21. Challacombe, M. J Chem Phys 1999, 110, 2332.
22. Helgaker, T.; Larsen, H.; Olsen, J.; Jørgensen, P. Chem Phys Lett 2000, 327, 397.
23. Larsen, H.; Olsen, J.; Jørgensen, P.; Helgaker, T. J Chem Phys 2001, 115, 9685.
24. Head–Gordon, M.; Shao, Y.; Saravanan, C.; White, C. A. Mol Phys 2003, 101, 37.
25. Ochsenfeld, C.; Head–Gordon, M. Chem Phys Lett 1997, 270, 399.
26. Palser, A. H. R.; Manolopoulos, D. E. Phys Rev B 1998, 58, 12704.
27. Bowler, D. R.; Gillan, M. J. Comp Phys Commun 1999, 120, 95.
28. Goedecker, S. J Comp Phys 1995, 118, 261.
29. Baer, R.; Head–Gordon, M. J Chem Phys 1997, 107, 10003.
30. Davidson, E. R.; Feller, D. Chem Rev 1986, 86, 681.
31. Saad, Y. SPARSKIT, Version 2.0, Department of Computer Science and Engineering, University of Minnesota, http://www.cs.umn.edu/Research/arpa/SPARSKIT/sparskit.html, 1999.
32. Duff, I. S.; Erisman, A. M.; Reid, J. K. Direct Methods for Sparse Matrices; Oxford University Press: London, 1986.
33. Saad, Y. Iterative Methods for Sparse Linear Systems; PWS: Boston, MA, 1996.
34. Maslen, P. E.; Ochsenfeld, C.; White, C. A.; Lee, M. S.; Head–Gordon, M. J Phys Chem A 1998, 103, 2215.
35. Challacombe, M. Comp Phys Commun 2000, 128, 93.
36. Hehre, W.; Radom, L.; Pople, J. A.; von R. Schleyer, P. Ab Initio Molecular Orbital Theory; Wiley: New York, 1986.
37. Bilmes, J.; Asanovic, K.; Vuduc, R.; Iyer, S.; Demmel, J.; Chin, C. W.; Lam, D. PHiPAC, Version 1.0, International Computer Science Institute, University of California, Berkeley, http://www.icsi.berkeley.edu/bilmes/phipac/.
38. Whaley, R. C.; Petitet, A. Dongarra, J. J. Parallel Comput 2001, 27, 3.
39. White, C. A.; Head–Gordon, M. Chem Phys Lett 1996, 257, 647.
40. Shao, Y.; Saravanan, C.; Head–Gordon, M.; White, C. A. J Chem Phys, in press.