Theor Chem Acc (2003) 110: 156–164
DOI 10.1007/s00214-003-0469-8

Theoretical
Chemistry Accounts

Semidirect parallel self-consistent field: the load balancing problem in the input/output intensive self-consistent field iterations

Roland Lindh[1], **Jesper Wisborg Krogh**[1], **Martin Schütz**[2], **Kimihiko Hirao**[3]

[1] Department of Theoretical Chemistry, Chemical Center, P.O.Box 124, 221 00 Lund, Sweden
[2] Institut für Theoretische Chemie, Universität Stuttgart Pfaffenwaldring 55, 70569 Stuttgart, Germany
[3] Department of Applied Chemistry, School of Engineering, The University of Tokyo, Hongo, Bunkyo-ku, Tokyo 113-8656, Japan

Received: 2 June 2002 / Accepted: 3 September 2002 / Published online: 6 October 2003

Abstract. The full capacity of contemporary parallel computers can, in the context of iterative ab initio procedures like, for example, self-consistent field (SCF) and multiconfigurational SCF, only be utilized if the disk and input/output (I/O) capacity are fully exploited before the implementation turns to an integral direct strategy. In a recent report on parallel semidirect SCF http://www.tc.cornell.edu/er/media/1996/collabrate.html, http://www.fp.mcs.anl.gd/grand-challenges/chem/non-direct/index.html it was demonstrated that super-linear speedups are achievable for algorithms that exploit scalable parallel I/O. In the I/O-intensive SCF iterations of this implementation a static load balancing, however, was employed, dictated by the initial iteration in which integral evaluation dominates the central processing unit activity and thus determines the load balancing. In the present paper we present the first implementation in which load balancing is achieved throughout the whole SCF procedure, i.e. also in subsequent iterations. The improved scalability of our new algorithm is demonstrated in some test calculations, for example, for 63-node calculation a speedup of 104 was observed in the computation of the two-electron integral contribution to the Fock matrix.

Keywords: Parallel – Input/Output – Semidirect – integral direct

1 Introduction

The availability of efficient parallel ab initio algorithms has been limited by the absence of high-level parallel utility libraries which address the specific requirements of computational chemistry. Standard message passing libraries, like MPI, PMV, or Linda, are too low-level and not really suitable to support the complex problems of memory disk management of ab initio electronic structure algorithms. Additionally, the concepts of distributed versus. shared memory implementations have complicated the options for program developers in their search for an algorithm suitable for distributed and shared memory architectures.

The computational chemistry community has been aware of these shortcomings in industry standards for some time and started to develop a suitable platform for parallel program development on its own. The Global Array (GA) toolkit, developed by Pacific North National Laboratory (PNNL) and Argonne National Laboratory (ANL) in collaboration with others, is the result of such efforts [1,2]. The GAs solve the problem related to, for example shared and distributed memory architecture such that the application programmer can employ the very same algorithm and computer implementation on both architectures. To reduce the time invested in parallel program developments the use of portable utilities is a necessity. The availability of the GA toolkit has been instrumental to subsequent progress in developments of parallel electronic structure codes as manifested in the NWChem project [3], the superlinear scaling MP2 implementation [4] and parallel implementation efforts in program packages like Columbus [5], MOLPRO [6], and MOLCAS [7], to mention a few.

Another project pushed by PNNL and ANL is the development of high-level utilities for parallel input/output (I/O) [8]. The ChemIO library, which address needs specific to ab initio electronic structure methods, is a high-level software interface to disk resident arrays, exclusive file access and shared files. This new utility offers new possibilities for program development which so far were closed to those with limited knowledge on low-level features of different parallel architectures.

Implementations of parallel electronic structure codes have until recently avoided the use of secondary storage devices. This has been a manifestation of both lack of suitable tools for transparent implementation on various

Contribution to the Björn Roos Honorary Issue

Correspondence to: Roland Lindh
e-mail: roland.lindh@teokem.lu.se

hardware platforms and poor I/O characteristics. For Hartree–Fock self-consistent field (SCF) and Kohn–Sham density functional theory (KS-DFT), two of the most commonly used electronic structure methods in computational chemistry, this implied the use of integral direct algorithms [9, 10] for parallel implementations. In the conventional SCF/KS-DFT programs the two-electron repulsion integrals are computed once, stored on disk, and later retrieved from disk in each iteration for contraction with the new density matrix in the calculation of the new Fock matrix. In integral direct approaches, on the other hand, the two-electron repulsion integrals are computed on-the-fly as they are needed in each iteration. The computational burden of the latter hence appears to be considerably higher. However, by virtue of efficient prescreening techniques which exploit the sparsity of the one-particle density, the average Central Processing Unit (CPU) time per iteration of a direct SCF/KS-DFT calculation is smaller than that of evaluating all two-electron repulsion integrals (performed once) in a conventional calculation. This is particularly true in the context of incremental Fock matrix construction [11]. On the other hand, since a typical SCF/KS-DFT calculation usually converges in about a dozen iterations the breakeven point of a conventional single-node calculation and a parallel integral direct calculation occurs for 2-4 nodes depending on the case (provided that it is possible to store all integrals on disk).

This situation is especially annoying for the middle range of calculations, where it is still possible to perform the conventional computation on a high-end work station much faster than on a small standard parallel computer (lower than 4–8 nodes). However, a recent preliminary report on a parallel semidirect SCF implementation (a hybrid of conventional and integral direct SCF [11]) implementation demonstrated that this no longer need be the case [12]. Using the ChemIO utilities scientists at the ANL implemented the semidirect approach for a hardware configuration with scalable I/O [13]. In the initial iteration each node writes all or a subset of the two-electron repulsion integrals which it computes to a local disk. In subsequent iterations these integrals are retrieved from the local disk as they are needed. While the distribution of the integrals to process on each node can be modified to achieve almost perfect parallelization, in the first iteration the same distribution has to be fixed in all subsequent iterations: a particular node processes the very same set of integrals that it computed initially, which now is available on its local disk. In subsequent iterations the computational work is therefore much less balanced. Nevertheless, with this limitation in load balancing of the I/O-intensive SCF iterations superlinear scaling was observed. Hence, the loss of a proper load balancing in subsequent iterations is more than offset by the fact that large fractions of the two-electron repulsion integral distribution only need to be computed once.

The present paper describes a natural extension of the parallel semidirect SCF implementation in which efficient load balancing is achieved throughout the whole iterative procedure. This is achieved by employing the idle time of the nodes in subsequent iterations to compute and store integral sets on the local disk which later on can be processed if the node otherwise would become idle. This report is divided into two sections, Implementation strategy which describes the characteristics of our parallel distributed I/O semidirect SCF implementation, and Performance assessments and discussions, where the performance of the implementation is discussed.

2 Implementation strategy

The implementation described here is an modification of our integral direct SCF code in the MOLCAS program package version 5.2. The integral evaluation is based on the original Rys–Gauss codes as implemented in the integral code SEWARD [14]. The SCF code is implemented with differential densities and a quasi–Newton update scheme [15, 16, 17]. Densities are desymmetrized and contracted with the atomic orbital (AO) integrals to form unique intermediate AO Fock matrices. These matrices are transformed on-the-fly to the symmetry adapted orbital (SO) basis. All symmetry treatment is handled via the double coset decomposition procedure [18, 19] already implemented in MOLCAS for symmetry treatment of SO integrals, gradients [20], and force constants[21]. The parallelization of the SCF/KS–DFT code discussed in the present text only applies to the Fock matrix construction.

2.1 Parallel integral direct SCF

The parallel implementation of the integral direct SCF/KS-DFT is straight forward [22] in the context of a replicated data implementation (each node has a private copy of the density and Fock matrices). A batch of two-electron integrals is computed and traced with the one-particle density in the construction of the Fock matrix. At the start of each SCF iteration the density matrix is broadcast to all participating nodes and at the end of the iteration the final Fock matrix is constructed as a global reduction of the partial Fock matrix as residing on each node. In between, the independent tasks of integral computation and contraction with the density matrix for a particular shell quadruplet are distributed dynamically on the nodes to maintain load balancing. In our case, load balancing is achieved as follows:

1. A globally uniform list (each node has a private copy), a so-called task list, is constructed in which the individual calculations of integrals over shell quadruplets are grouped together in tasks. The number of such tasks should typically be significantly larger than the number of nodes employed in the calculations. Furthermore, we vary the number of shell quadruplets in the tasks such that the first ones are the largest and the last ones are of a rather modest size. A large task includes several shell quadruplets (ranging from just a few to several thousands depending on

the size of the calculation), while a small task might only contain an individual shell quadruplet. The process of defining the number and sizes of the tasks is a dynamic process which depends on the number of CPUs and shell quadruplets.

2. A list unique to each node, a so-called private priority list, is constructed describing the order in which a specific node or processor should execute all tasks. Such priority lists were introduced in the context of our scalable parallel MP2 program [6] in order to exploit data locality (i.e., minimizing communications) of individual tasks. Typically the list is such that tasks with a large number of shell quadruplets to process appear before smaller tasks.
3. A global reservation list, a distributed vector, is initiated. This list is accessible to all nodes and individual nodes make reservations of tasks on this list via the read-and-increment utility of the GA toolkit. Any node which reads the value of the initiated list is allowed to process that particular task. Since the private priority lists are different on each node (for the original parallel MP2 program this was a natural consequence of exploiting data locality), congestion during task reservation can be avoided to a large extent.

As the work of an iteration is proceeding the nodes move down the task list processing smaller and smaller tasks. By varying the number of tasks and their relative size almost perfect parallelization in the Fock matrix construction step of the SCF/KS–DFT implementation is achieved. Once a new iteration is started the global reservation list is reinitiated and the nodes will again start to try to make reservations, and there is no restriction that the a particular task is processed by a particular node.

2.2 Parallel semi-direct SCF

The semidirect parallel SCF implementation is a simple extension of the procedure just described, augmenting the procedure of handling the private priority list. During the first iteration the implementation is similar to the parallel integral direct SCF/KS-DFT with two exceptions:

1. The integrals are stored on the local disk of the node.
2. Those tasks which were executed by an individual node are now booked in the augmented private priority list of that particular node.

2.2.1 Static load balancing

In the case of static load balancing the subsequent iterations are commenced such that each node only processes those tasks which were booked in the augmented private priority list [13]; hence, no reservation procedure is needed for these iterations. The computational burden of specific tasks, however, is different to that during the first iteration (some integrals are now read from disk rather than being evaluated, prescreening will also

Fig. 1. The C_{24} fullerene

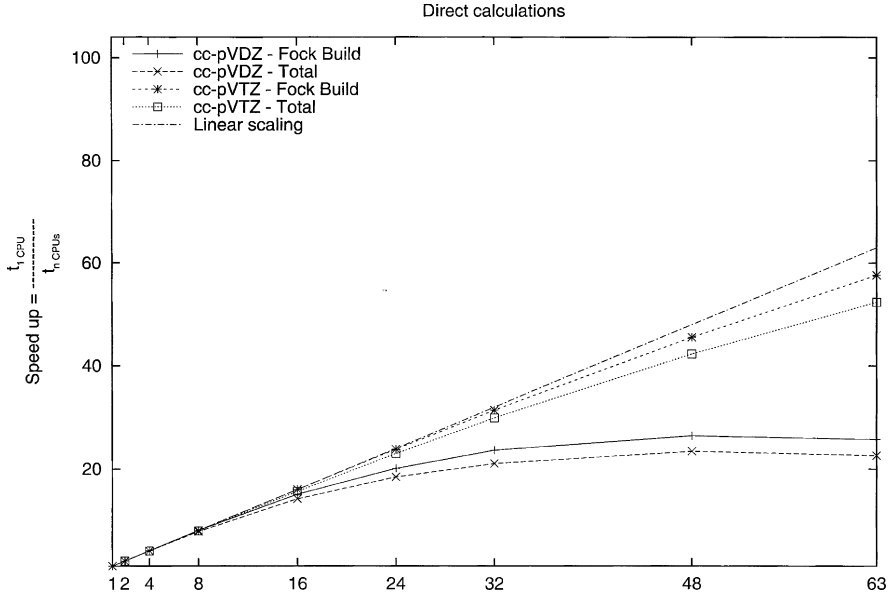

Fig. 2. The speedup for the direct self-consistent field (SCF) calculations on the C_{24} "Fock build" are the speedups for two-electron integrals contribution to the Fock matrix, and "Total" are the speedups for the total SCF time

affect the computational load). Hence, ideal load balancing cannot be expected for such an approach, and the imbalance becomes even more severe when the number of nodes is increased (for the same chemical system).

2.2.2 Dynamic load balancing

In order to remedy this situation, i.e., to achieve a well-balanced work distribution also during the I/O intensive subsequent iterations, we suggest that an individual node, after processing all tasks for which it had integrals on disk, should reserve a task anyway and process it. This of course implies that two-electron repulsion integrals are recomputed in the spirit of an integral direct approach, even though they might be stored already on some other (nonlocal) disk. However, since this happens late in the iterative work the size of the task is small and the associated CPU time in relative terms is small as well. The alternative would be to leave the node idle. Furthermore, since the computed integrals are appended to the local disk the computational expense can be amortized over several iterations if the tasks on subsequent iterations are processed by the same node. This is not an unlikely event. For the node which previously executed the task the local integral file is now truncated. If this node, for some reason, later on becomes idle while there are still tasks to be reserved and executed it will preform as just mentioned previously, i.e., reserve a not yet executed task and store the recomputed integrals on its local disk. The procedure as we described here will allow tasks to migrate to idle nodes and after a few iterations a steady-state or near steady-state will appear as a manifestation by a near perfect load balancing in the I/O dominated iterations. We have experimented with

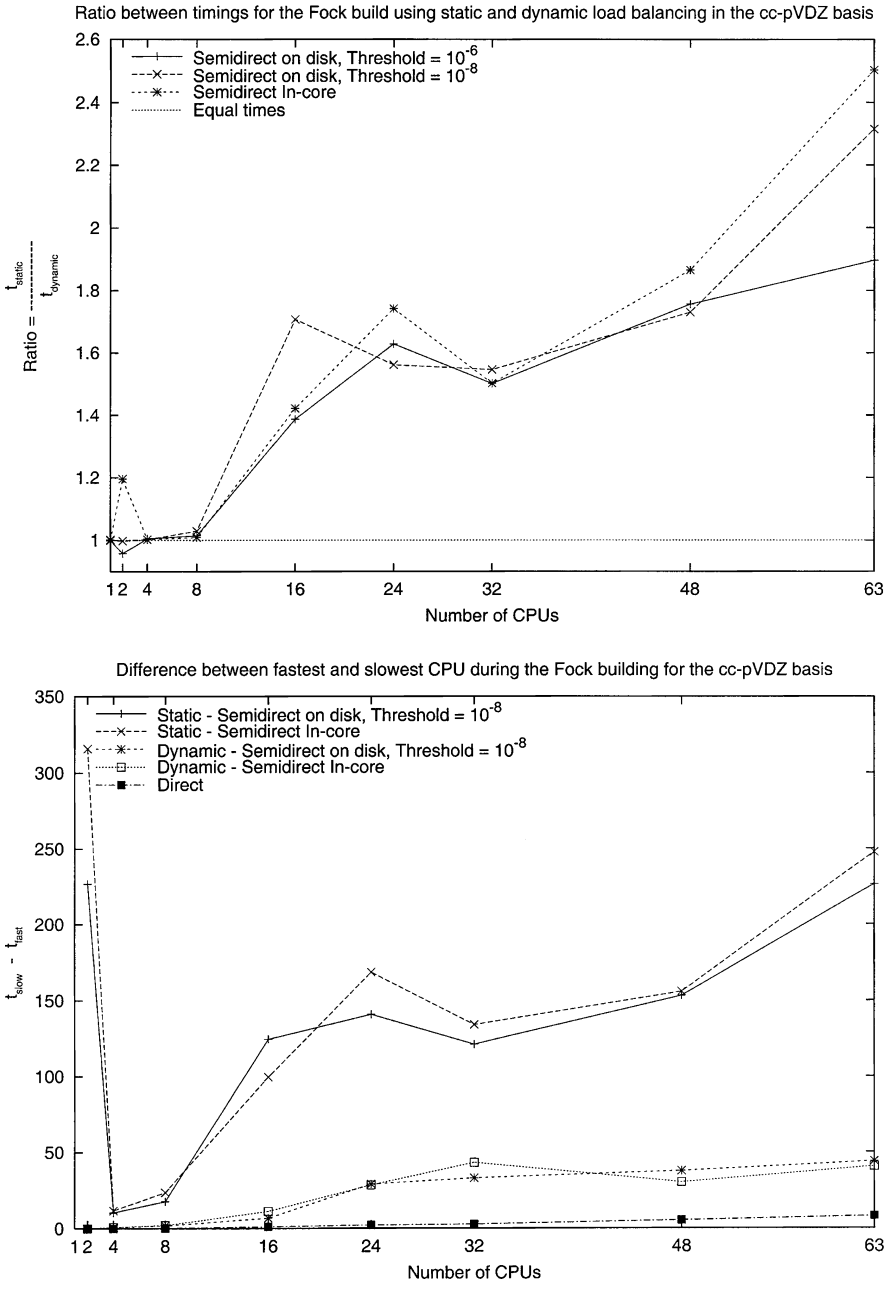

Fig. 3. **a** A comparison between the relative timings for the Fock build using the static and dynamic load balancing and **b** the idle time during the formation of the two-electron contribution to the Fock matrix as an function of the number of nodes employed in a calculation on the C_{24} molecule using a cc-pVDZ basis

various approaches in selecting the tasks which can be "stolen" by an idle node. One could either let the node take a batch from the start of the remaining task list, i.e. try to take a task which is as large as possible, or take one from the very end. The major difference between the two approaches is that the former will cause a loss of several tasks in the truncation of the local integral disk of the "losing" node, whereas the latter approach will only truncate the local integral disk from the very end of the file. It is our experience that the optimum is the latter.

2.2.3 Disk-based versus in-core semidirect SCF

The implementation described in this paper is best suited for a parallel computer with scalable I/O. Standard shared memory parallel installations are not typically configured with scalable I/O and thus are not well suited to efficiently run the current algorithm. However, this problem can to some extent be avoided by offering a simple in-core version of the disk-bound algorithm. Here this is simply achieved by increasing the size of the I/O buffers and inhibiting I/O requests. It is shown later that the in-core implementation offers benefits which can also be utilized on distributed parallel machines.

3 Performance assessment and discussions

In this section we assess and compare the performance of

1. The fully-direct SCF method.

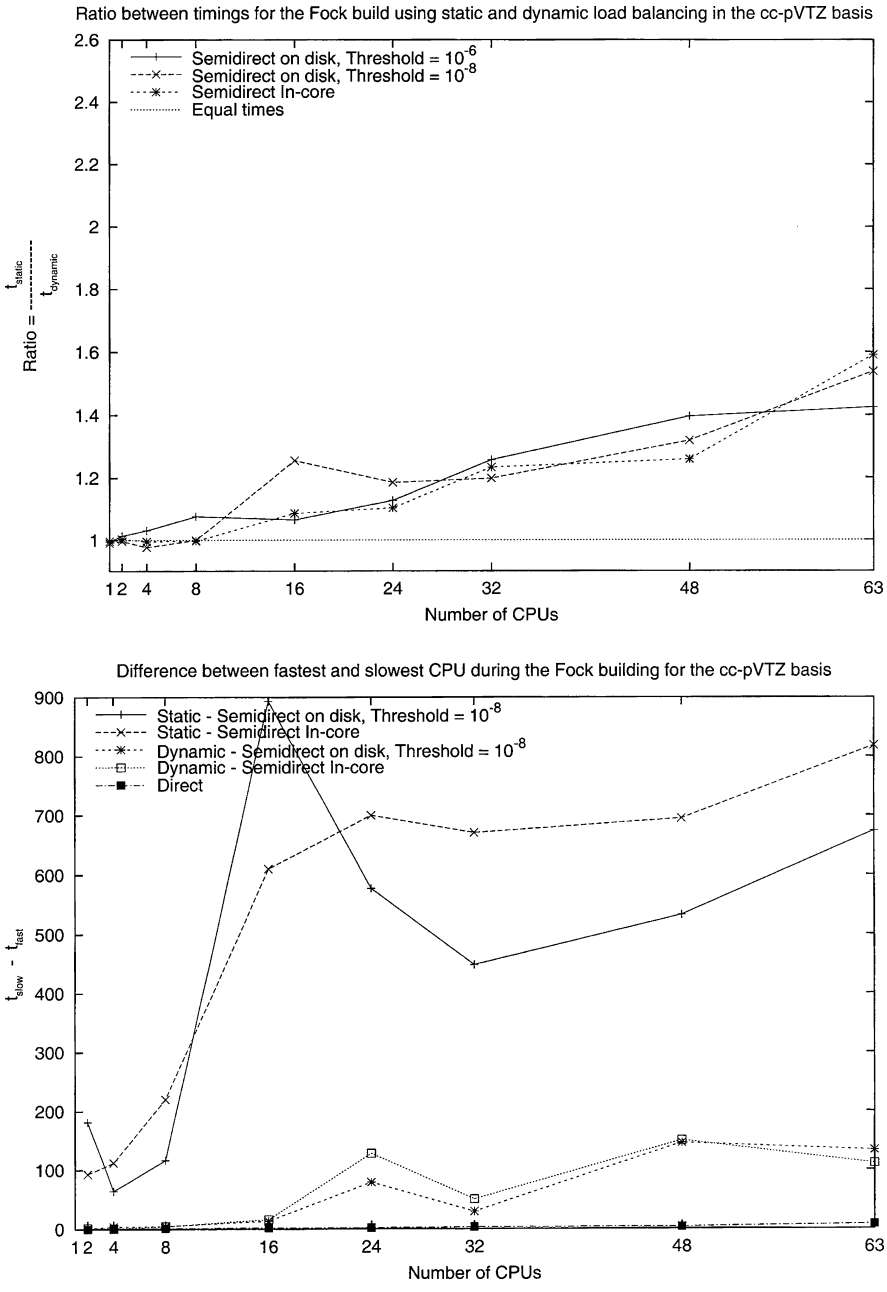

Fig. 4. **a** A comparison between the relative timings for the Fock build using the static and dynamic load balancing and **b** the idle time during the formation of the two-electron contribution to the Fock matrix as an function of the number of nodes employed in a calculation on the C_{24} molecule using a cc-pVTZ basis

Table 1. The speedup for building the Fock matrix for semidirect calculations as a function of the number of nodes. Numbers are tabulated for two disk-bound cases with a maximum of 2000 MB integrals written to each local disk and an input/output (**I/O**) buffer of 512 kB in combination with a threshold for storing two-electron integrals of *A* 1.0×10^{-6} and *B* 1.0×10^{-8}, and *C* for one in-core case with a memory for storing integrals set to 900 MB per node. All numbers refer to calculations with the cc-pVDZ basis

Nodes	Static			Dynamic		
	A	B	C	A	B	C
1	1.0[a]	1.0[b]	1.0[c]	1.0[d]	1.0[e]	1.0[f]
2	2.1	2.0	2.1	2.0	2.0	2.5
4	5.8	5.9	7.5	5.8	5.9	7.5
8	10.7	10.9	13.9	10.9	11.2	14.0
16	13.8	11.6	16.7	19.1	19.8	23.8
24	13.7	14.0	15.5	22.3	21.8	27.0
32	17.3	17.1	20.1	26.0	26.4	30.2
48	16.7	17.3	19.8	29.3	29.9	36.9
63	13.8	13.4	14.7	26.3	31.0	36.9

[a]Elapse time 3605 s
[b]Elapse time 3586 s
[c]Elapse time 4135 s
[d]Elapse time 3610 s
[e]Elapse time 3586 s
[f]Elapse time 4125 s

Table 2. The speedup for building the Fock matrix for semidirect calculations as a function of the number of nodes. Numbers are tabulated for two disk-bound cases with a maximum of 2000 MB integrals written to each local disk and an **I/O** buffer of 512 kB in combination with a threshold for storing two-electron integrals of *A* 1.0×10^{-6} and *B* 1.0×10^{-8}, and *C* for one in-core case with a memory for storing integrals set to 900 MB per node. All numbers refer to calculations with the cc-pVTZ basis

Nodes	Static			Dynamic		
	A	B	C	A	B	C
1	1.0[a]	1.0[b]	1.0[c]	1.0[d]	1.0[e]	1.0[f]
2	2.0	2.1	2.0	2.0	2.0	2.0
4	4.2	4.4	4.1	4.4	4.3	4.1
8	8.9	9.7	8.5	9.5	9.7	8.4
16	22.9	20.2	16.3	24.4	25.3	17.7
24	29.3	30.2	24.4	33.1	35.8	26.9
32	37.5	40.1	31.9	47.2	48.0	39.3
48	61.1	62.6	48.1	85.3	82.6	60.6
63	69.4	67.5	58.7	98.9	103.9	93.4

[a]Elapse time 84703 s
[b]Elapse time 84420 s
[c]Elapse time 86430 s
[d]Elapse time 85114 s
[e]Elapse time 85227 s
[f]Elapse time 86695 s

2. The disk-bound semidirect SCF method without and with dynamic load balancing.
3. The in-core version of the semidirect SCF method.

The performance assessments are based on the elapsed times for the construction of the two-electron part of the Fock matrix (Fock build). All test calculations were restricted to calculations on the C_{24} fullerene molecule (Fig. 1) using the cc-pVDZ and cc-pVTZ basis sets of Dunning [23] and using no symmetry. The test calculations were performed on a 64-processor Beowulf cluster sited at the University of Lund. Each node is equipped with a 1.6 GHz AMD XP 1900+ CPU mounted on an Asus A7V266-E motherboard, local 40 GB Western Digital UDMA/100 disks, and 1 Gb DDR SDRAM main memory [16]. In all calculations the local disc storage was limited to 2000 MB per node.

The measured speedups for the fully direct SCF are plotted in Fig. 2. The graph shows the expected scaling (slightly worse than linear) with the number of nodes. For example, for the cc-pVTZ basis we observe a speedup of 57.8 for 63 nodes (86403s and 1498s for 1 and 63 nodes, respectively), i.e., an efficiency of 99.86% per node.[1] The efficiency decreases as expected on going to the smaller basis set, for which we observe a speedup of 26.1 on 63 nodes (5072s and 194s for 1 and 63 nodes, respectively), or an efficiency of 98.61% per node. The performance documented here demonstrates that the computation of the two-electron integrals dominates the SCF time and that the fully direct parallel implementation as described in the previous section is efficient.

The performance of the semidirect SCF with the static and the dynamic load balancing is compared in Figs. 3 and 4. What we expect to see here is that the potential imbalance of the I/O-intensive iterations eventually destroys the performance of the static load balancing as the number of nodes is increased. Two different sets of timings corresponding to a threshold of storing two-electron integrals to disk of 1.0×10^{-6} and 1.0×10^{-8} are used, respectively. That is, the second set of calculations should have more two-electron integrals on disk, which should extend the superlinear scaling range compared to the set with the higher threshold. The data demonstrate in a vivid way the ultimate breakdown of the static load balancing approach as the parallelization is pushed further by increasing the number of nodes used in the calculation. This occurs, of course, earlier for the smaller basis set, for which we note a relative performance enhancement for the dynamic approach compared to the static approach of a factor of about 2.5 using 63 nodes. The relative performance improvement of the larger basis set is 1.6 at 63 nodes. The ratio is expected to increase with an increased number of nodes. It is also noted that the integral threshold makes a difference. Naively is seems that the more integrals which are written to disk the better. There is, however, a limit at which further reduction in the integral threshold only induces nonproductive I/O on low-value integrals which the direct approach avoids by the prescreening technique. From Figs. 3 and 4 it is evident that while the idle time in the static approach increases worse than linearly with the number of nodes, the dynamic approach demonstrates an almost constant idle time.

In passing we mention our experience with different approaches to what tasks should be "stolen" in the dynamic approach. We suggest "stealing" tasks from the end of the task list, i.e., to take over tasks as small

[1] Efficiency per node is defined as $\left(\frac{t_1}{t_n n}\right)^{\frac{1}{n}}$.

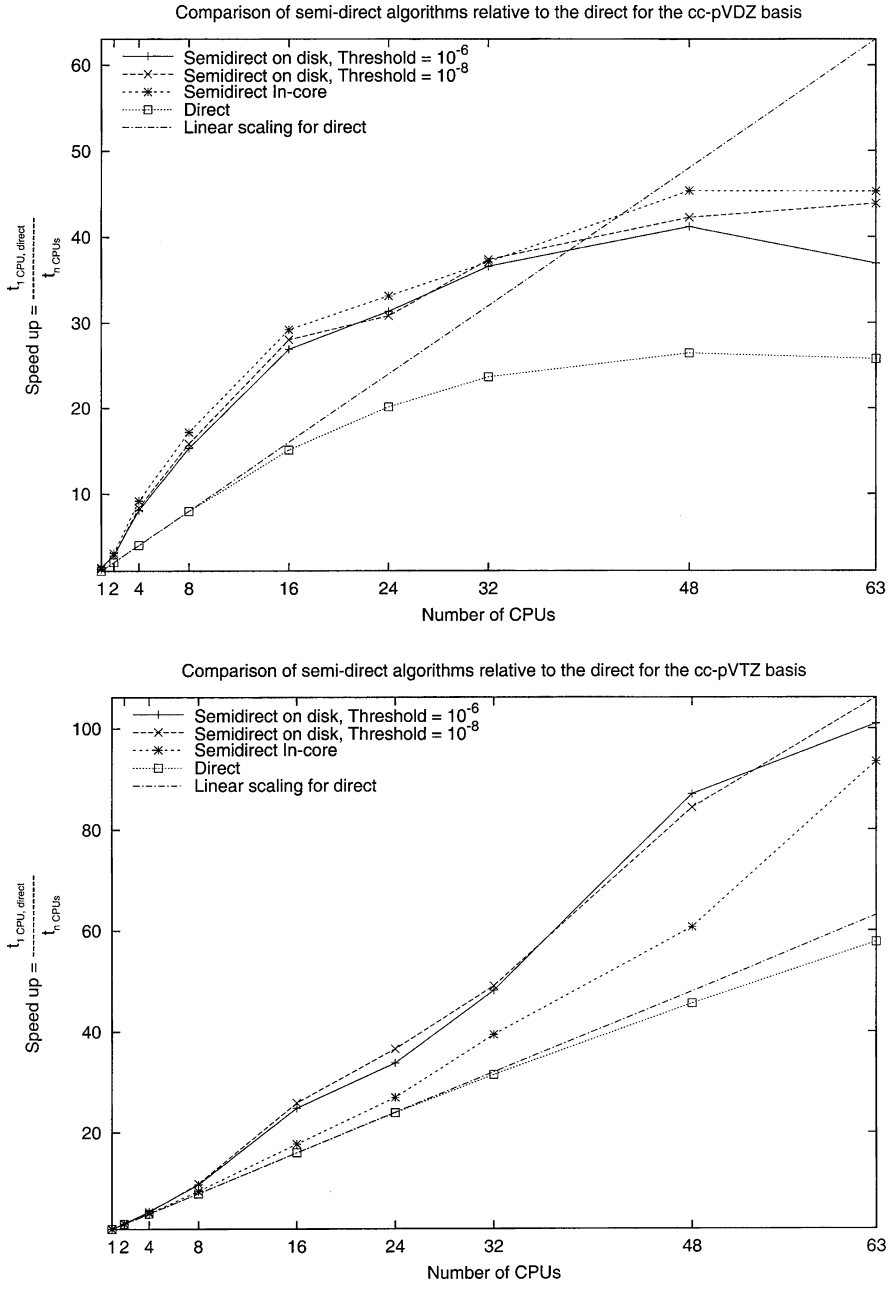

Fig. 5. Relative speedup versus number of nodes for two semidirect SCF calculations with different integral thresholds for writing to disk, a semidirect in-core SCF calculation and a direct SCF calculation on the C_{24} molecule using **a** a cc-pVDZ basis and **b** a cc-pVTZ basis

as possible. This reduces the idle time of the nodes by up to 1 order of magnitude. The other extreme would be to take a task as large as possible. Our experience with the latter was that the performance difference between the static and dynamic load balancing was minimal. A possible reason for this is that the first approach truncates the local integral file of the node losing the task from the end, minimizing the number of lost tasks compared to the second approach in which the local integral file is truncated several tasks away from the end of the file.

The performance of the two on-disk and the in-core semidirect SCF are compared in Tables 1 and 2 and Fig 5, the latter two using the dynamic load balancing approach. First we note that the superlinear scaling for the semidirect SCF with the smaller cc-pVDZ basis set breaks down in a way similar to that of the fully direct SCF. No significant difference is noted between the on-disk or in-core semi-direct approaches. The breakdown is explained by the fact that the size of the problem in relation to the number of nodes is simply too small to generate a set of tasks which are large enough to allow for efficient load balancing. This is clearly not the case yet for the calculations with the larger basis set. In particular we note excellent superlinear scaling over the whole range of nodes, a speedup of 103.9 on 63 nodes corresponding to a parallel efficiency of 100.8% per node. We also note that for both basis sets that the in-core algorithm ultimately breaks through and becomes the fastest and most efficient approach. For the smaller basis set the in-core approach is definitely most appropriate, while for the larger basis set the limitations in integral storage (disk-based 2000 MB, while in-core 900 MB) makes a

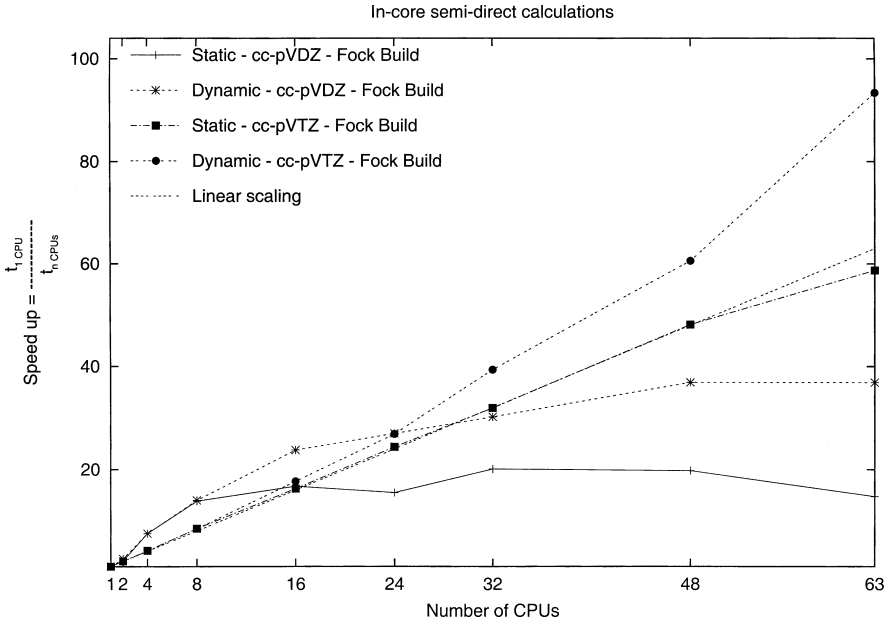

Fig. 6. Relative speedup versus number of nodes for semidirect in-core parallel SCF for the static and dynamic load balancing as experienced in calculations on the C_{24} molecule using a cc-pVDZ and a cc-pVTZ basis

difference. Nevertheless, following the trend displayed in Fig. 5 it is evident that the in-core version will also here ultimately become the most efficient approach as the number of nodes is increased further. Furthermore, we note that the semidirect approach is always more efficient than the fully direct approach. In our implementation only the Fock build is presently parallelized. According to Amdahl's law the speedup of the total time deteriorates substantially beyond a certain number of nodes. For example, for semi-direct disk-bound calculations using the cc-pVTZ basis and an integral threshold of 1.0×10^{-8} speedups of 103.9 and 75.4 were measured on 63 nodes for the Fock build and the total SCF calculation, respectively. Nevertheless, in spite of Amdahl's law also for the total times of the SCF calculations the measured speedup is still superlinear.

Finally we compare the in-core semidirect SCF implementation implementation with static and dynamic load balancing (Tables 1 and 2, and Fig. 6). The data show that for the smaller basis set improvement of the parallelization requires the modification of other parts of the SCF algorithm than the Fock build (to overcome Amdahl's law). For the larger basis set, however, the test calculations underline the importance of the in-core implementation but indicate that distributed nodes with large memory capacity are required to bring the in-core approach clearly above linear scaling.

4 Summary

A parallel semidirect SCF implementation with scalable I/O potential has been presented. In particular the issue of achieving load balancing on a massive parallel processing system during the I/O-intensive SCF iterations was addressed. A strategy in which some integral batches are recomputed during these iterations was suggested to improve the load balance and to reduce the overall wall clock for the application. The performance of the implementation was investigated by test calculations on the C_{24} fullerene. These tests show superlinear scaling for semidirect parallel SCF calculations with the two-electron integrals stored on disk and in-core semi-direct parallel SCF calculations. The latter is particularly useful in the context of shared memory parallel machines without scalable I/O. In comparison to earlier semidirect SCF algorithms with static load balancing during subsequent iteration steps a considerable improvement in the performance was observed by virtue of the new approach focusing on dynamic load balancing in all iterations.

Acknowledgement. We thank J. Nieplocha for valuable help and making the toolkit (including ChemIO) available to us. R.L. acknowledges the Intelligent Modeling Laboratory and the University of Tokyo for financial support during his stay in Japan.

References

1. Nieplocha J, Harrison RJ, Littlefield RJ (1994) Proc Supercomput 340
2. Nieplocha J, Harrison RJ, Littlefield RJ (1996) J Supercomp 10: 197
3. Bernholdt DE, Apra E, Fruchtl HA, Guest MF, Harrison RJ, Kendall RA, Kutteh RA, Long X, Nicholas JB, Nichols JA, Taylor HL, Fann GI, Littlefield RJ, Nieplocha J (1995) Int J Quantum Chem Symp 29: 475
4. Schütz M, Lindh R (1997)Theor Chem Acc 95: 13
5. Dachsel H, Lischka H, Shepard R, Neiplocha J, Harrison RJ (1997) J Chem Phys 18: 430
6. Werner H.-J Knowles PJ MOLPRO (a package of ab initio programs) with contributions from Amos RD, Bernhardsson A, Berning A, Celani P, Cooper DL, Deegan MJO, Dobbyn AJ, Eckert F, Hampel C, Hetzer G, Korona T, Lindh R, Lloyd AW, McNicholas SJ, Manby FR, Meyer W, Mura ME, Nicklass A, Palmieri P, Pitzer R, Rauhut G, Schütz M, Stoll H, Stone AJ, Tarroni R, Thorsteinsson T
7. Andersson K, Barysz M, Bernhardsson A, Blomberg MRA, Carissan Y, Cooper DL, Fleig T, Fülscher MP, Gagliardi L, de Graaf C, Hess BA, Karlström G, Lindh R, Malmqvist P-Å,

Neogrády P, Olsen J, Roos BO, SChimmelpfenning B, Schütz M, Seijo L, Serrano-Andrés L, Siegbahn PEM, Stålring J, Thorsteinsson T, Veryazoy V, Wierzbowska M, Widmark PO (2001) MOLCAS version 5. 2. University of Lund, Lund, Sweden
8. Nieplocha J, Foster I, Kendall RA (1998) Int J Supercomp App High Perf Comput 12: 345
9. Almlöf J, Faegri K, Korsell K (1982) J Comput Chem 16: 1291
10. Almlöf J, Taylor PR (1984) In: Dykstra CE (Ed.) Advanced theories and computational approaches to the electronic structures of molecules. Kluwer, Dordrecht, p 107
11. Häser M, Ahlrichs R (1989) J Comput Chem 10: 104
12. Cornell Theory Center Press Release. http://www.tc.cornell.edu/er/media/1996/collaborate.html
13. Dritz K, Minkoff M, Shepard R, Tilson JL, Wagner AF http://www-fp.mcs.anl.gov/grand-challenges/chem/nondirect/index.html
14. Lindh R, Ryu U, Liu B (1991) J Chem Phys 95: 5889
15. Shepard R (1993) Theor Chim Acta 84: 343
16. Wong AT, Harrison RJ (1995) J Comput Chem 16: 1291
17. Fischer TH, Almlöf J (1992) J Phys Chem 96: 9768
18. Davidson ER (1975) J Chem Phys 62: 400
19. Taylor PR (1992) In: Roos BO (Ed.) Lecture notes in quantum chemistry – European summer school in quantum chemistry. Springer, Berlin Heidelberg New York, p 89
20. Lindh R (1993) Theor Chim Acta 85: 423
21. Bernhardsson A, Lindh R, Olsen J, Fülscher M (1999) mol Phys 96: 617
22. Kendall RA, Harrison RJ (1991) Theor Chim Acta 79: 337
23. Dunning TH Jr (1989) J Chem Phys 90: 1007
24. LUNARC at Lund University http://www.lunarc.lu.se