Massively Parallel Self-Consistent-Field Calculations

Jeffrey L. Tilson
Mathematics and Computer Science Division
Argonne National Laboratory
Argonne, IL 60439

October 29, 1994

Abstract

The advent of supercomputers with many computational nodes each with its own independent memory makes possible extremely fast computations. Our work, as part of the U.S. High Performance Computing and Communications Program (HPCCP), is focused on the development of electronic structure techniques for the solution of Grand Challenge-size molecules containing hundreds of atoms. Our efforts have resulted in a fully scalable Direct-SCF program that is portable and efficient. This code, named NWCHEM, is built around a distributed-data model. This distributed data is managed by a software package called Global Arrays developed within the HPCCP. We present performance results for Direct-SCF calculations of interest to the consortium.

1 Introduction

Advances in theoretical chemistry over the past two decades have consistently improved the ability of electronic structure calculations to accurately predict from first principles the structure, spectra, and energetics of molecules and radicals. Such predictions permit theoretical determinations of both thermochemistry and kinetics, fundamental information for all chemical processes. As might be expected, more accurate and computationally intensive methods are restricted to smaller molecular systems. But even for the simpler ab initio electronic structure techniques, there is a frequently encountered limit to the size of molecules that can be feasibly studied. Many practical problems in chemistry today requires information on molecules too large for conventional electronic structure codes to feasibly handle. The fate of chlorofluorocarbon alternates in the atmosphere, the properties of ligand substituents in polymerization catalysis, and the mechanism of enzymatic destruction of toxins are all examples of current academic and industrial research areas where important electronic structure applications are frequently too large to be feasibly done.

Improvements in theoretical chemistry have frequently exploited advances in computer hardware. At present, such an advance is occurring through the use of massively parallel processors (MPP), high speed networds of hundreds to thousands of computers. The economies of scale make such hardware the least expensive for assembling large scale computational resources, precisely the kind of resources necessary for electronic structure applications for large molecular system such as discussed above. In recognition of this direction in computer architecture, theoretical chemists and computer scientists at Argonne National Laboratory, Pacific Northwest Laboratory, three major oil companies, and two major chemical companies have formed a collaboration to adapt electronic structure methods to the MPP architecture for the purpose of applications to large molecular systems. This collaboration has operated through the Department of Energy under the auspices of the High Performance Computing and Communications Initiative. This paper reports the results of one particular MPP adaptation, that of the self-consistent field (SCF) [1, 2] electronic structure method. The results to date suggest that efficient coding for MPP technology can qualitatively change the size of the molecule that can be treated by the SCF method.

Efficient coding for the MPP architecture is not straightforward, because an MPP computer is fundamentally different from a vector supercomputer. A typical MPP computer consists of a collection of processors each with its own memory and each connected to a high performance network. When designing algorithms for these computers, important issues include avoiding replicated computation (*computational efficiency*), distributing data structures so as to avoid wasting memory (*data distribution*), distributing computation to processors so as to avoid idle time when one processor is busy and others are not (*load balance*), and minimizing time spent sending and receiving messages (*communication efficiency*). A metric that integrates these different criteria is *scalability*: the extent to which an algorithm is able to solve larger problems as the number of processors is increased.

The complex architecture of MPP computers makes intuitive notions of performance unreliable. Hence, a sound methodology when developing parallel algorithms is to begin by examining algorithmic alternatives at a theoretical level. Only after scalability has been established should effort be devoted to implementations on parallel computers. In this paper, we apply this methodology to the SCF method. In addition to being important in its own right, the SCF method is the starting point for many other more rigorous methods. The SCF approach is also typical of other more sophisticated methods in its use of large data structures and irregular data access patterns. Because of its importance, others have developed various parallel MPP SCF codes (see [3] and references therein). However, the code reported here is (in our opinion) the most scalable SCF code currently available.

2 SCF Wavefunctions

The SCF wavefunction is constructed from an antisymmetrized product of single particle functions. This is the molecular orbital (MO) approximation. These MOs represent the motion of an individual particle (electron) within the field of all the remaining electrons and a static (clamped) configuration of nuclei. This wavefunction form and the

approximate Hamiltonian yield the Hartree-Fock energy and wavefunction.

Solution of the SCF problem has been shown useful in the determination of the nuclear geometry. Typical values differ from experiment by 0.1-0.2 angs. This makes the SCF equation a useful way to examine molecular geometries. Another quantity of interest is the total energy and the orbital energies. The SCF gives reasonable total energies for many molecules at their equilibrium geometry. It cannot, however, describe the important structural correlations in molecules. This limitation, for example, prevents an accurate description of bond breaking and forming. The individual orbital energies may be used for a qualitative analysis of the electronic spectra. A very important aspect of the SCF equations is as a starting point for more accurate higher order methods. These methods almost exclusively correct the set MOs, making the SCF technique an integral part of them.

2.1 Formalism

The total energy of a molecular system constructed from a set (ϕ_i) of occupied orthonormal MOs is

$$Energy \quad = \quad \sum_{i,j}^{occupied} h_{ij} D_{ij} + \frac{1}{2} \sum_{i,j,k,l}^{occupied} g_{ijkl} d_{ijkl}. \tag{1}$$

The terms h_{ij} and g_{ijkl} are one and two electron integrals, respectively and are independent of the precise form of the wavefunction.

$$h_{ij} \; = \int d\tau_1 \phi_i(1) \hat{h} \phi_j(1) \quad = \quad < \phi_i(1) | \hat{h} | \phi_i(1) >$$

$$g_{ijkl} \; = \int d\tau_1 d\tau_2 \phi_i(1) \phi_j(1) \frac{1}{r_{12}} \phi_k(2) \phi_l(2) \; = < \phi_i(1) \phi_j(1) | \frac{1}{r_{12}} | \phi_k(2) \phi_l(2) >$$

$\hat{h}$ denotes the one-particle operator and r_{12} the interparticle distance. The form of the wavefunction influences the structure of the one- and two-particle density matrices, D_{ij} and d_{ijkl}. Generally, the one-particle density matrix is simple to generate becoming a delta function for the canonical SCF, $D_{ij} = \delta_{ij}$. The two-particle density matrix typically requires a substantial amount of effort for more accurate, highly correlated electronic structure methods (MCSCF, MRCI, full-CI, etc.) and can be a substantial computational process. This matrix, however, takes on a particularly simple structure for SCF wavefunctions becoming sums of products of the one-particle densities. The simplicity of the SCF two-particle density matrix shifts the computational burden onto the generation of the integrals themselves.

The SCF total energy may be simplified by substituting into Eqn. 1 the nonzero values for the D matrix.

$$Energy \quad = \quad \sum_{i}^{occupied} h_{ii} + \frac{1}{2} \sum_{i,j}^{occupied} g_{iijj} - g_{ijji} \tag{2}$$

This equation satisfies the requirements necessary for application of the variation principle. In essence, the best MOs will result in the lowest (best) SCF energy. Hence, one can find the best MOs by minimizing Eqn. 2 subject to orbital orthonormality and energy constraints. The details of this derivation are widely available, and so only the results are presented here. This minimization results in the total energy expression

$$Energy \quad = \quad \frac{1}{2}\sum_i (h_{ii} + F_{ii}) \tag{3}$$

where F_{ij} is the MO *Fock* matrix.

$$F_{ij} \quad = \quad h_{ij} + \sum_k^{occupied} g_{ijkk} - g_{ikkj} \tag{4}$$

The F matrix is constructed from the MO integrals and so depends on the final solution, requiring an iterative solution of the problem.

This energy has been solved exactly by using numerical techniques but only for very small systems. This exact solution for the SCF wavefunction is called the *Hartree-Fock* solution. Modern implementations of the SCF procedure parameterize the orbitals by using a finite set of *basis functions*, χ_μ, with expansion coefficients, C. The basis functions are linearly independent functions with a metric $S_{\mu\nu}$ selected to simplify the calculation of the two-particle integrals, g_{ijkl}.

$$\phi_i \quad = \quad \sum_\mu \chi_\mu C_{\mu i} \tag{5}$$

$$S_{\mu\nu} \quad = \quad \int d\tau \chi_\mu \chi_\nu$$

The (closed-shell) MO density matrix when transformed to the atomic orbital (AO) basis and with electron spin integrated out becomes

$$D^{AO}_{\mu\nu} \quad = \quad 2\sum_{j=1}^{occupied} C_{\mu j} C_{\nu j} \tag{6}$$

$$D \quad = \quad 2CC^t \tag{7}$$

Substitution of Eqns. 5 and 7 into Eqn. 4 results in the canonical AO Fock matrix.

$$F^{AO}_{\mu\nu} \quad = \quad h_{\mu\nu} + \frac{1}{2}\sum_{\lambda\rho} \left(2g^{AO}_{\mu\nu\lambda\rho} - g^{AO}_{\mu\lambda\nu\rho}\right) D^{AO}_{\lambda\rho} \tag{8}$$

$$\tag{9}$$

where the integrals are now over the AO functions.

The optimized MOs are determined by finding the optimal coefficients, C, that satisfy Roothaans [1] nonorthogonal matrix eigenvalue problem

$$FC \quad = \quad SC\epsilon. \tag{10}$$

The eigenvalues, ϵ, can be interpreted as the set of individual electron energies. In the limit of a complete basis, the true Hartree-Fock limit is attained.

Equations 2–10 give us a prescription for solving the SCF problem.

1. Select a basis set, χ_μ.
2. Select an initial coefficient matrix, C, and generate the current density matrix, D^{AO}.
3. Construct the matrix F using the current D^{AO} and generating the AO integrals.
4. Solve the generalized eigenvalue problem of Eqn. 10 to obtain the new orbitals.
5. Check the new orbitals for self-consistency. If they have not converged, construct a new D^{AO} matrix, and repeat.

Once the converged orbitals are found, Eqn. 1 is solved, and the SCF calculation is finished.

2.2 Algorithm

Efficient, scalable SCF software requires a detailed understanding of the SCF algorithm. The principal operations in the SCF procedure are two primary steps that are iterated until a self-consistent solution of Eqn. 10 is obtained. These steps are the generation of the AO integrals to construct the AO Fock matrix and the diagonalization step to construct the new coefficients.

The construction of the AO Fock matrix, even when the integrals are available, requires many more operations than the subsequent diagonalization. The two-electron integrals depend on four indices. These indices sample the space of AO basis functions; therefore, the number of integrals grows as $\mathcal{O}(\frac{N_{basis}^4}{8})$, becoming huge for even small problems. As an example, a small hydrocarbon might require 100 basis functions for an adequate representation of the electron field. This requirement results in $\mathcal{O}(10^8)$ bytes of memory to store all the integrals. This exorbitantly high storage forces the algorithm either to off-load these integrals to disk or re-calculate then as needed. The integrals are constructed from localized basis functions that introduces a considerable amount of sparsity. This sparsity and the very high CPU/IO capabilities for most computers greatly favors a recomputation strategy. This type of SCF algorithm is denoted the *direct-SCF* method and is the method selected for our work. The integrals can differ in computational effort by $\mathcal{O}(10^2)$ arithmetic operations. In a sequential environment, this is of little consequence, but is an important issue in a parallel environment. The integrals are independent and may be grouped (blocked) into nearly any convenient manner. Finally, the symmetry properties of the two electron integrals and the AO F and D matrices results in a given integral, $I = g_{ijkl}^{AO}$ contributing to at most six elements of the AO F matrix and requiring at most

six elements of the AO D matrix. A generic AO F construction algorithm is displayed in Fig. 1.

The second primary step is the diagonalization. Once the Fock matrix is constructed we must solve Eqn. 10 to obtain the optimum orbitals. The operation count for a diagonalization is typically $\mathcal{O}(N_{basis}^3)$ and is insignificant relative to the AO Fock construction on a sequential computer. The diagonalization step takes on a much greater importance in a parallel environment, often becoming the computational bottleneck.

3 Parallel SCF

In this section we describe our fully scalable SCF program named *NWCHEM*. A description of NWCHEM is available in a recent review [3] on parallel SCF programs and algorithms. Here we summarize the important points of our scalable distributed-data SCF algorithm.

A fully scalable, parallel direct-SCF algorithm must parallelize both the AO F construction and diagonalization steps. For Grand Challenge-size problems this parallelism must address not only greatly reducing the time for solution but also efficiently managing the aggregate memory of the computer. For example, the number of integrals required for a problem of size $N_{basis} = 100$ is on the order $\mathcal{O}(10^7)$. If each integral requires on average 1000 arithmetic operations, and the chosen CPU executes at 40 Mflops (millions of floating-point instructions per second), the total time to generate one integral is 25μsec. The total time to compute the integrals becomes 250 sec. A calculation on Decane with $N_{basis} = 250$ would require approximately 3.4 hours.

The parallel algorithm must also address the memory requirements of persistent matrix data required for the calculation. The solution of a typical SCF problem requires the storage of $\mathcal{O}(10)$ persistent matrices each of size $(N_{basis} \times N_{basis})$ elements. SCF solutions of molecular problems useful particularly to industry require matrices of dimension $N_{basis} = \mathcal{O}(10^2 - 10^4)$ double-precision words. A typical SCF calculation would then require local memory capacity of nearly $\mathcal{O}(10^9)$ bytes.

3.1 Replicated Data Model

Several mature programs are now widely available on parallel architectures. The focus of these initial efforts was to use parallel computers to greatly decrease the turnaround time of a calculation. This was accomplished (most often) by using the direct-SCF technique and parallelizing the integral generation step. The density and Fock matrix data are *replicated* on all computational nodes. Batches of integrals are then collected into a computational task that is allocated to a waiting node. These integrals are contracted with the locally available D matrix to create a partial F matrix. This technique constructs a F matrix efficiently. The algorithms, however, are not inherently scalable, since memory storage is limited to that available on a single node. Furthermore, these algorithms use a sequential diagonalization routine; hence the partial F matrices residing on each node must be summed together onto one node. This one node then solves Eqn. 10 for the new orbitals. These orbitals are then replicated back onto all nodes. This approach achieves very good speedup on large numbers of nodes and is fairly straightforward to implement

in existing programs. This technique, unfortunately, shifts the computational bottleneck from the highly parallel integral generation step to the diagonalization and is limited by the amount of memory on one node.

3.2 Distributed-Data Model

Several models of scalable Fock matrix construction algorithms have been previously analyzed [4]. The resulting program has been thoroughly discussed in [5]. We summarize the important parallel details here.

To develop a fully scalable parallel SCF program requires efficiently distributing matrix data throughout the aggregate memory of the parallel computer. This process eliminates the memory restrictions of the replicated-data model algorithm. This distribution, however, forces the program to communicate data (send messages) between nodes. Our communications are performed with a new library of software functions that emulate a shared-memory model using the primitive massage-passing capabilities of the MPP.

We first partition all AO matrix data, D, F, S, etc. into atomic blocks. These blocks are submatrices with indices that span all basis functions for a given atomic center. These blocks are then arbitrarily allocated to the different nodes on the computer. We also generate integrals in atomic blocks (each of the four indices span all basis functions for the given atom) and dynamically allocate blocks to a node with a shared counter. When a node is instructed to generate an integral block, a check on sparsity is performed; then, appropriate blocks of the D matrix are fetched and resulting F matrix blocks are updated.

The simplicity of this algorithm is complicated by the varying data requirements for different integral blocks. Our algorithm performs these communications with a library of routines called Global Arrays. These Global Arrays support a lightweight one-sided communications model, thereby greatly simplifying development of our scalable program [6, 7].

The scalable construction of the F matrix requires that the integrals be allocated dynamically and that nonlocal data requirements be satisfied without unduly synchronizing the computational progress. These requirements are difficult to satisfy by using a traditional point-to-point communications scheme. The integral blocks very greatly in their computational effort; and, equally important, for a given integral block the actual amount of F and D data required depends upon the indices. Dynamic data caching increases the difficulty of data management.

In developing software for a typical message-passing environment, data is transferred to a remote node by explicitly having SEND and RECEIVE calls made by the participating nodes. A program written this way essentially blocks the progress of the calculation until both nodes have satisfied their respective communication operation. This approach places an effective synchronization step into the program. Asynchronous point-to-point communications and double buffering can lessen the impact of such a scheme. This artificial synchronization is not related to the algorithm at all. For many kinds of calculations a natural synchronization step exists and so is of no consequence.

The Global Arrays library eliminates this explicit synchronization. It allows the programmer to simply insert into a code a "request" for data. No companion "send" need be made. This local request activates a mechanism that finds the data, interrupts the work on the node holding the data, and commands the node to send the data. The interrupted

node then resumes with its work. If the data are local to the requesting node, no messages are sent. The overhead associated with this type of communications is higher than a primitive message-passing function but is not inhibiting. The much greater integral load balance obtained in this way greatly compensates for the slightly higher communication cost. The Global Arrays are capable of several one-sided kinds of communications (read, send, accumulate, etc.) and also support all traditional point-to-point communications. The library is currently portable to several different parallel architectures. The simplicity of using Global Arrays to write distributed-data applications does not obviate the need for algorithm modeling. The applications engineer still must consider the memory and network characteristics of the target computer for efficient implementation.

Once the F matrix is constructed, the optimum orbitals must be generated. We have the capability to perform the generalized eigenvalue analysis in parallel. The scalability, however, is much worse than construction of the F matrix because of the nature of the diagonalization algorithm. [8, 9] This fact led us to investigate and include alternative schemes as suggested by Shepard [10] . These techniques are all second-order convergence techniques that try to find the minimum SCF energy within the space of parameters, C. A recent paper [11] compares various techniques for direct-SCF calculations. Shepard and Tilson are exploring the use of a simultaneous vector expansion method for overlapping computational effort. These second-order techniques can greatly accelerate the time for solution for some kinds of problems. They are strongly dependent, however, on the initial guess of C and so do not always exhibit quadratic convergence. These importance of these techniques is in their exposing highly scalable AO F constructions to the optimization scheme.

4 Benchmarks

A set of molecular problems that represent the interests of our consortium has been assembled. The problems include simple alkanes and transition-metal containing species. The largest alkane problem, $C_{20}H_{44}$, represents the interaction of two decane molecules. The three other presented benchmarks are

- $(C_5H_5)Co(NO)(CH_3)$ designated *cobalt*
- $((Cp)_2(CH_2))TiCl_2$ designated *titanium*
- $2,2^{'}-di(trifluoromethyl)biphenyl$ named *biphenyl*

We note that all total energies have been verified by independent calculations. The speedup is a measure of the efficiency with which parallelism has been implemented. If a program executes in time $T(1)$ on a single node and in time $T(P)$ on P nodes the speed up (SU) becomes

$$SU(P) \quad = \quad \frac{T(1)}{T(P)}.$$

If the parallel program is perfectly parallelized, $T(P) = \frac{T(1)}{P}$ and $SU(P)$ becomes simply P. A percent SU may be calculated as $\frac{SU(P)}{P} \times 100$.

Table 1 lists the time to construct the Fock matrix on the IBM SP1 and Intel Touchstone DELTA computers as a function of the number of nodes. Analytical performance models predict that speedup will approach $90-95$ percent of ideal for very large problems. This is observed in Table 1, where we observe a speedup of 98 percent for the biphenyl benchmark on the DELTA computer.

The IBM SP1 results appear degraded relative to the DELTA. Detailed analysis of the SP1 behavior on smaller problems (see butane results) indicates that the overhead associated with creating the parallel environment markedly degrades performance on two SP1 nodes. As the number of SP1 nodes increases, the observed speedup relative to one node is actually greater than ideal. This situation suggests that favorable overlap of operations occurs less frequently on two nodes. We find that the *slope* of the speedup curves for calculations on the SP1 closely parallels the ideal line. The speedup for the larger benchmarks is derived by assuming ideal speedup for the calculation with the fewest number of nodes.

These benchmarks were also analyzed by using an available replicated data model program on the CRAY C90 computer. This program is implemented as a shared-memory model and is expected to be well vectorized. In this model the communications overhead is very cheap, since relatively few messages are sent. We expect a speedup on the 16-node C90 of close to 15. The time per AO F construction for these C90 tests are collected in Table 2. We also find that generally a C90 node is observed to be $15-20$ times faster than a node on the Intel DELTA and $3-5$ times faster than a node on the IBM SP1. We find that calculations on the DELTA and SP1 can be made to run faster than on the C90 by application of enough computational nodes.

The scalability of NWCHEM is found to be quite good for large molecular problems. The somewhat lessened performance for the smaller benchmarks is not an issue, since this MPP software is designed for the solution of *massive* problems that are not currently possible. In particular, the SU begins to decrease when the number of processors (P) approaches $\mathcal{O}(N_{atoms}^2)$. Clearly, for large problems ($N_{atoms} = 1000$) high performance is expected on all available MPPs. This high performance stems primarily from the use of integral and data blocking and the asynchronous communications made possible by Global Arrays.

The total times for solution are presented in Table 3. These calculations were performed on the IBM SP1 computer. The wavefunctions were optimized with a second-order convergence scheme, where the number of iterations are called *macro* iterations. Each macro iteration corresponds to an SCF iteration and generally requires several AO F matrix construction steps for the optimization. All energies are converged to 10^{-12} atomic units (au). The large number of macro iterations for the *titanium* benchmark reflects the *expected* convergence difficulties of a transition metal containing species.

5 Conclusion

The HPCCP consortium has developed a fully scalable and efficient direct-SCF program called NWCHEM. We have validated the program and demonstrated high performance on two currently available MPP computers. This work was accomplished by using a

few molecular benchmarks of interest to the consortium. Comparisons with a fully functional replicated-data direct-SCF code on the CRAY C90 indicate that MPP performance can surpass that of traditional vector supercomputers when using appropriately designed scalable software.

This work has focused on efficient use of MPP CPUs and high-speed networking. Future efforts must address utilization of all MPP resources, especially I/O. In the direct-SCF, the recomputation of integrals eliminates the potentially massive storage of integrals while decreasing the total number of arithmetic operations. This fortuitous behavior is not necessarily applicable to other electronic structure algorithms nor to algorithms in general. Our consortium is now beginning to address the issues of parallel I/O and its applications to remote data storage.

The work of the consortium is also not limited to direct-SCF. Several parallel projects are currently in place, including MP2, SCF gradients, and MCSCF. These techniques allows us to fully optimize SCF geometries and determine corrections to the SCF wavefunction.

Acknowledgments

This work was performed under the auspices of the High Performance Computing and Communications Program of the Office of Scientific Computing, U.S. Department of Energy under contract W-31-109-Eng-38 with the University of Chicago which operates the Argonne National Laboratory.

This research was performed in part using the Intel Touchstone Delta System operated by Caltech on behalf of the Concurrent Supercomputing Consortium. Access to this facility was provided by Argonne National Laboratory.

The author gratefully acknowledges use of the Argonne High-Performance Computing Research Facility. The HPCRF is funded principally by the U.S. Department of Energy Office of Scientific Computing.

The author thanks A. F. Wagner and M. Minkoff for helpful discussions and assistance.

References

[1] Roothaan, C., *Reviews of Modern Physics*, 23, 69, 1951.

[2] Almlöf, J., Faegri, K., and Korsell, K., *J. Comp. Chem*, 3, 385, 1982.

[3] Harrison, R. J., and Shepard, R., *Annual Review of Physical Chemistry*, to appear 1994.

[4] Foster, I., T., Tilson, J., L., Shepard, R. L., Wagner, A. F., Harrison, R. J., Kendall, R. A., Littlefield, R. L. *submitted J. Comp. Chem.*, 1994.

[5] Harrison, R. J., Guest, M. F., Kendall, R. A., Bernholdt, D. E., Wong, A. T., Stave, M., Anchell, J., Hess, A. C., Littlefield, R. L., Fann, G. L., Nieplocha, J., Thomas, G. S., Elwood, D., Tilson, J., Shepard, R. L., Wagner, A. F., Foster, I., T., Lusk, E., and Stevens, R. *submitted J. Comp. Chem.*, 1994.

[6] Harrison, R., *Theo. Chim. Acta.*, 84, 363, 1993.

[7] Nieplocha, J., Harrison, R., J., Littlefield, R., J. *For submission to Supercomputing 1994*, 1994

[8] Littlefield, R., and Maschhoff, K., *Theor. Chim. Acta*, 84, 457, 1993.

[9] Kendall, R., A., Harrison, R., J., Littlefield, R., J., Guest, M., F. *Reviews in Computational Chemistry*, VCH Publishers, Inc., New York, 1994.

[10] Shepard, R., *Theor. Chim. Acta*, 84, 343, 1993.

[11] Wong, A. T. and Harrison, R., J. *submitted J. Comp. Chem.*,1994.

AO Fock Construction
DO i = 1, N
   DO j = 1, i
      IF (i,j pair survive screening) THEN
         DO k = 1, i
IF (k.EQ.i) lhi = j
IF (k.NE.i) lhi = k
               DO l = 1, lhi
   IF (k,l pair survive screening) THEN
   EVALUATE $I = g_{ijkl}$
   F_{ij} = F_{ij} + $D_{kl}I$
   F_{kl} = F_{kl} + $D_{ij}I$
   F_{ik} = F_{ik} - $\frac{1}{2}D_{jl}I$
   F_{il} = F_{il} - $\frac{1}{2}D_{jk}I$
   F_{jl} = F_{jl} - $\frac{1}{2}D_{ik}I$
   F_{jk} = F_{jk} - $\frac{1}{2}D_{il}I$
               ENDDO
         ENDDO
      ENDIF
   ENDDO
ENDDO

Figure 1: Basic logic for Fock matrix construction

Table 1: Speedup characteristics of NWCHEM on the Intel Touchstone Delta and IBM SP1 computers. All times are for one AO Fock matrix construction. Speedup times for a given molecular species are relative to the measure timed on the fewest number of nodes. ND = Not Done

		DELTA			*IBM SP1*		
Molecule	*NBF*	*Time per AO F*	*Number of Nodes*	*Speed-up*	*Time per AO F*	*Number of Nodes*	*Speed-up*
C_4H_{10}	110	1140.51	1	1	218.43	1	1
	110	580.63	2	1.96	134.77	2	1.62
	110	293.65	4	3.88	63.2	4	3.45
	110	74.08	16	15.39	15.13	16	14.44
	110	37.31	32	30.57	ND	ND	ND
$C_{20}H_{44}$	520	1287.79	32	32	1060.39	8	8
	520	860.90	48	47.8	539.25	16	15.7
	520	647.73	64	63.6	272.04	32	31.2
	520	333.11	128	123.7	184.19	48	46.1
	520	188.51	256	218.6	141.11	64	60.1
cobalt	114	891.93	2	2	330.18	1	1
	114	453.05	4	3.9	247.75	2	1.33
	114	233.16	8	7.6	106.98	4	3.1
	114	121.47	16	14.7	50.33	8	6.6
	114	63.84	32	27.9	ND	ND	ND
biphenyl	324	2291.61	16	16	846.48	8	8
	324	1148.13	32	31.9	429.33	16	15.8
	324	575.86	64	63.7	260.85	32	25.96
	324	290.59	128	126.2	131.73	64	51.4
titanium	147	3008.09	4	4	713.26	4	4
	147	773.48	16	15.6	364.51	8	7.8
	147	400.13	32	30.1	186.64	16	15.3
	147	212.82	64	56.5	95.69	32	29.8
	147	119.3	128	100.8	50.55	64	56.4

Table 2: Time for an AO F matrix construction on the Cray C90 using a commonly available ab-initio package

Molecule	*Number of Nodes*	*Time Per AO F*
C_4H_{10}	1	78.24
C_4H_{10}	4	19.59
$C_{20}H_{44}$	1	2490.23
$C_{20}H_{44}$	2	1204.26
cobalt	1	64.36
titanium	1	382.47
biphenyl	8	474.5

Table 3: Time to solution. Initial estimates from atomic densities. Energy converged to 10^{-12}au

	Time To Solution IBM SP1				
Molecular Species	Number Atoms	Number Basis Ftns	Number Nodes	Number Iterations	Total Time *seconds*
CH_4	5	35	16	4	9.80
C_4H_{10}	14	110	16	4	328.35
C_8H_{18}	26	210	32	5	1342.10
titanium	24	174	128	14	3761.35
biphenyl	28	324	64	7	4539.57
biphenyl	28	324	128	7	2382.35