

A Distributed Memory Implementation of the MCHF Atomic Structure Package

Charlotte Froese Fischer, Ming Tong,
Murry Bentley*, Zuchang Shen, and C. Ravimohan
Vanderbilt University, Box 1679B
Nashville, TN 37235 USA

Abstract

The MCHF atomic structure package consists of a series of programs that predict a range of atomic properties and communicate information through files. Several of these have now been modified for the distributed-memory environment. On the Intel iPSC/860, the restricted amount of memory and the lack of virtual memory required a redesign of the data organization with large arrays residing on disk. Data structures also needed to be modified. To a large extent, data could be distributed among the nodes, but crucial to the performance of the MCHF program was global information needed for an even distribution of the workload. This paper outlines the computational problems that need to be solved in an atomic structure calculation and describes the strategies used to distribute both the data and the workload on a distributed-memory system. Performance data is provided for some benchmark calculations on the Intel iPSC/860.

*Present Address: Van der Waals-Zeeman Laboratorium, Valckenierstraat 65, NL-1018 XE, Amsterdam, The Netherlands

1 Introduction

Atomic data is needed in many areas of scientific endeavor. Energy level structures, autoionization rates, cross-sections for excitation, ionization, charge exchange, and recombination all enter into model calculations for controlled thermonuclear fusion. In geophysical studies of the atmosphere, the emission features of atomic oxygen are related to the abundance of oxygen in the thermosphere. Astrophysics has a longstanding need for large amounts of data. With the Hubble Space Telescope reporting new data with increased precision, more accurate theoretical calculations are needed for interpretation and analysis. Some of the more abundant elements of the solar system, such as iron, are so complex that their accurate study has not been feasible to date. Laser research also requires energy level and transition data for the development of improved laser techniques. Because atoms are the building blocks of molecules and solids, knowledge gained in the study of atoms is transferrable to metals, surfaces, and other polyatomic systems.

The needed data can often be predicted only through computation. With massively parallel computers, tremendous advances are possible in the near future.

2 The Nature of the Problem

The state of a many-electron system is described by a wave function Ψ that is the solution of a partial differential equation (called the wave equation),

$$(\mathcal{H} - E)\Psi = 0, \tag{1}$$

where \mathcal{H} is the Hamiltonian operator for the system and E the total energy. The operator \mathcal{H} depends on the system (atomic, molecular, solid-state, etc.) as well as the quantum mechanical formalism (non-relativistic, Dirac-Coulomb, or Dirac-Breit, etc.). The present paper will focus on atomic systems and the non-relativistic Schrödinger equation for which the Hamiltonian (in atomic units) is

$$\mathcal{H} = -\frac{1}{2} \sum_{i=1}^N \left(\nabla_i^2 + \frac{2Z}{r_i} \right) + \sum'_{ij} \frac{1}{r_{ij}}. \tag{2}$$

Here Z is the nuclear charge of the atom with N electrons, r_i is the distance of the i^{th} electron from the nucleus, and r_{ij} is the distance between electron i and electron j . This equation was derived under the assumption of a point-nucleus of infinite mass. The operator \mathcal{H} has both a discrete and continuous spectrum: for the former $\Psi(\mathbf{r}_1, \mathbf{r}_2, \dots, \mathbf{r}_N)$ has a probability interpretation

and consequently must be square integrable. For the latter, this restriction does not apply. Schrödinger's equation does not involve the electron spin, but when spin-functions are introduced, the physically meaningful solutions are antisymmetric in the interchange of all the coordinates of any two electrons.

Schrödinger's equation for atoms is among the simplest equations for many-electron systems. The computational schemes for its solution have many features in common with other formulations and the ideas developed here could be applicable to them as well. In particular, there is a close similarity between the MCHF method [1] that will be described here and the MCDF method as implemented in GRASP² [2], based on a relativistic formalism including also QED corrections.

The prediction of atomic properties is a challenging interaction between computational techniques and theoretical physics. As the many-body problem is solved to a higher level of accuracy more physical effects need to be included as well. An obvious example is the inclusion of the effect of the finite volume and finite mass of the nucleus. But more accurate Hamiltonians need to be used as well, the exact forms of which are the subject of current physics research.

3 Computational Methods

Because of the high dimensionality of the wave equation, approximate methods must be used. A very successful model has been the configuration interaction model in which the wave function, $\Psi_{\gamma LS}$, for a state labeled γLS is written as an expansion of M antisymmetrized configuration state functions (CSF), $\Phi(\gamma_i LS)$, each one being an eigenfunction of the total angular momentum L and total spin S . Then

$$\Psi_{\gamma LS}(\{X_j\}) = \sum_{i=1}^M c_i \Phi(\gamma_i LS; \{\mathbf{r}_j\}), \quad (3)$$

where $\{\mathbf{r}_j\} = \{r_1, \theta_1, \phi_1, \sigma_1, \dots, r_N, \theta_N, \phi_N, \sigma_N\}$. The r_j, θ_j, ϕ_j are spherical coordinates in three dimensional space, σ_j is the spin-space coordinate for electron j , and γ represents any quantum numbers other than LS that are needed for complete specification of the state. Each CSF, in turn, is a linear combination of terms of the form,

$$\prod_{j=1}^N \frac{1}{r_j} P_{n_j l_j}(r_j) Y_{l_j m_{l_j}}(\theta_j, \phi_j) \chi_{m_{s_j}}(\sigma_j), \quad (4)$$

where the spherical harmonics, $Y_{l m_l}$, and spinors, χ_{m_s} , are known. The combination satisfies the antisymmetry requirement and represents the coupling of orbital and spin momenta, for which an appropriate algebra is well known. The set $\{n_j l_j\}_{j=1}^N$ of quantum numbers as well as the

coupling is specified by γ_i . The radial functions, $P_{nl}(r)$, may be known functions or may need to be determined.

By Eq. (1), the total energy of the atom is given by

$$E = \langle \Psi_{\gamma LS} | \mathcal{H} | \Psi_{\gamma LS} \rangle, \quad (5)$$

assuming $\langle \Psi_{\gamma LS} | \Psi_{\gamma LS} \rangle = 1$. Using Eq. (3) and the multipole expansion for $1/r_{ij}$,

$$\frac{1}{r_{ij}} = \sum_k \frac{r_{<}^k}{r_{>}^{k+1}} P^k(\cos \theta), \quad (6)$$

where $r_{<}$, $r_{>}$ are the lesser and greater of r_i and r_j , respectively, and $P^k(\cos \theta)$ is a Legendre polynomial in $\cos \theta$ where θ is the angle between \mathbf{r}_i and \mathbf{r}_j , the energy may be expressed as

$$E = \sum_{ij} c_i c_j H_{ij} \quad (7)$$

$$= \sum_{ij} c_i c_j \left(\sum_{stuv;k} A_{stuv;k}^{ij} R^k(s, t; u, v) - \frac{1}{2} \sum_{qw} C_{qw}^{ij} L_{qw} \right), \quad (8)$$

where the R^k are Slater integrals,

$$R^k(s, t; u, v) = \int_0^\infty \int_0^\infty dr dr' P_s(r) P_t(r') \frac{r_{<}^k}{r_{>}^{k+1}} P_u(r) P_v(r'), \quad (9)$$

and the L_{qw} are the one-body integrals,

$$L_{qw} = \int_0^\infty dr P_q(r) \left[\frac{d^2}{dr^2} + \frac{2Z}{r} - \frac{l(l+1)}{r^2} \right] P_w(r). \quad (10)$$

The abbreviation, $P_i = P_{n_i l_i}$, has been used here. The $A_{stuv;k}^{ij}$ and C_{qw}^{ij} are called angular coefficients and can be computed using Racah algebra [3]. Because of the cusp in the integrand of Eq. (9), Slater integrals are often evaluated by first solving a pair of first-order differential equations followed by a one-dimensional integral [4]. Thus their evaluation is non-trivial.

The $M \times M$ symmetric matrix with elements

$$H_{ij} = \langle \Phi(\gamma_i LS) | \mathcal{H} | \Phi(\gamma_j LS) \rangle, \quad (11)$$

is called the interaction matrix. Applying the variational condition [1] to Eq. (7) and requiring that the energy be stationary with respect to perturbations in the solution (i.e. $\partial E / \partial c_i = 0$ for all i) leads to the matrix eigenvalue problem

$$(H - E)c = 0, \text{ where } H = (H_{ij}). \quad (12)$$

Thus the total energy is an eigenvalue of the interaction matrix, and the expansion coefficients of the wave function form the corresponding eigenvector. Of course, in order to compute the interaction matrix, the radial functions need to be known.

The MCHF method requires that the energy be stationary not only with respect to variations in the expansion coefficients, but also with respect to variations in the radial functions. This condition leads to a system of coupled integro-differential equations of the form

$$\frac{d^2 P_{nl}(r)}{dr^2} = \left[\frac{l(l+1)}{r^2} - \frac{2}{r}[Z - Y_{nl}(r)] + \varepsilon_{nl,nl} \right] P_{nl}(r) + G_{nl}(r), \quad (13)$$

called the MCHF equation for P_{nl} , the solution being subject to the boundary conditions, $P_{nl}(0) = 0$ and $\lim_{r \rightarrow \infty} P_{nl}(r) = 0$. In this equation,

$$Y_i(r) = r \sum_{jk} a_{ijk} \int_0^\infty dr' P_j(r') \frac{r^k}{r^{k+1}} P_j(r'), \quad (14)$$

$$G_i(r) = \sum_{j \neq i; i' j' k} b_{ii' j j' k} P_j(r) \int_0^\infty dr' P_{i'}(r') \frac{r^k}{r^{k+1}} P_{j'}(r') + \sum_{j \neq i} [\varepsilon_{ij} P_j(r) + c_{ij} L P_j], \quad (15)$$

and l and Z are fixed constants. The constants, $\varepsilon_{nl,nl}$ (diagonal parameter) and $\varepsilon_{nl,n'l}$ (off-diagonal parameters) are related to Lagrange multipliers assuring orthonormality of the radial functions, and L is the operator in square brackets in Eq. 10. The a_{ijk} , $b_{ii' j j' k}$, and c_{ij} coefficients are simple multiples of the angular coefficients of the $R^k(i, j; i, j)$, $R^k(i, j; i', j')$, and L_{ij} integrals, respectively. These coefficients depend on the expansion coefficients so the eigenvalue problem and the system of non-linear, integro-differential equations are coupled. (A more detailed derivation may be found in reference [1]).

The MCHF problem is solved iteratively by what is often called the multi-configuration self-consistent field (MC-SCF) method. Using estimates of radial functions, expansion coefficients are determined. Then, in turn, radial functions are updated so as to leave the energy stationary: a new interaction matrix is computed and an improved expansion vector obtained. This process is iterated until results are “self-consistent”.

One feature of large atomic structure problems is the fact that the same Slater integral may occur in many different interaction matrix elements. Since the evaluation of an integral is not trivial, it is desirable to arrange the data structure describing the energy expression in such a fashion that each integral is evaluated only once. At the same time, when the function $G_i(r)$ is required, the list of integrals must be scanned for integrals involving the radial function P_i ; if one is found, the associated contribution to the energy should readily be computed. The data

structure in MCHF that meets both these needs is a list of integrals, each integral having a pointer to a set of coefficients along with a pair of indices specifying the location in the matrix where the integral makes a contribution. This data structure corresponds to a rearranged form of Eq. (8), namely

$$E = \sum_{stuv;k} R^k(s, t; u, v) \left(\sum_{ij} A_{stuv;k}^{ij} c_i c_j \right) - \frac{1}{2} \sum_{qw} L_{qw} \left(\sum_{ij} C_{qw}^{ij} c_i c_j \right) \quad (16)$$

where $A_{stuv;k}^{ij}$ or C_{qw}^{ij} are zero in matrix elements where the integral does not appear. Alternatively, the sum on i, j may be restricted to matrix elements containing contributions from the particular integral.

In the MCHF atomic structure package, an interactive program GENCL [5] is used to generate configuration state lists using some simple rules. Then the NONH program [6] generates the angular coefficients and associated list of integrals that define the energy expression and the interaction matrix. If the radial functions are already determined, a CI [7] program determines selected eigenvalues and eigenvectors. By optimizing the radial functions for a particular state, much better accuracy can be obtained: the MCHF88 program [8] solves the optimization problem, computing both the radial functions and the mixing coefficients.

Once the wave function has been determined, other atomic properties can be predicted as expectation values of appropriate operators, i.e.

$$\langle \text{property} \rangle = \langle \Psi_{i'} | \text{OP} | \Psi_i \rangle, \quad (17)$$

where OP is the operator associated with the property and Ψ_i and $\Psi_{i'}$ are wave functions for the initial and final state, respectively. In some cases, as for the energy, $\Psi_i \equiv \Psi_{i'}$.

Substituting (3) into (17) we then get the result

$$\langle \text{property} \rangle = \sum_{i,i'} c_i c_{i'} \langle \Phi(\gamma_i LS) | \text{OP} | \Phi(\gamma_{i'} LS) \rangle. \quad (18)$$

The operator matrix element may be expanded further as

$$\langle \Phi(\gamma_i LS) | \text{OP} | \Phi(\gamma_{i'} LS) \rangle = \sum_{j,j',k} \text{Coeff}(k, a_j, a_{j'})_{ii'} \text{RI}^k(a_j, a_{j'}), \quad (19)$$

where a_j and $a_{j'}$ refer to sets of one or more electrons (or radial functions) of the initial and final states respectively, RI is a ‘‘radial integral’’, and k refers to any additional parameters that may be present. Substituting Eq. (19) into Eq. (18) we get

$$\langle \text{property} \rangle = \sum_{i,i'} c_i c_{i'} \sum_{j,j',k} \text{Coeff}(k, a_j, a_{j'}) \text{RI}^k(a_j, a_{j'}) \quad (20)$$

$$= \sum_{a_j, a_{j'}, k} \text{RI}^k(a_j, a_{j'}) \sum_{ii'} c_i c_{i'} \text{Coeff}(k, a_j, a_{j'})_{ii'}, \quad (21)$$

where the sum on i, i' is over all pairs with a non-zero coefficient of the radial integral. This sum is the “contribution factor” $\text{CF}(a_j, a_{j'}, k)$ for the radial integral to the property under consideration. Introducing this abbreviation into Eq. (21) we get

$$\langle \text{property} \rangle = \sum_{a_j, a_{j'}, k} \text{RI}^k(a_j, a_{j'}) \text{CF}(a_j, a_{j'}, k) \quad (22)$$

Clearly, Eq. (16) for the energy expression can also be rewritten as a sum over integrals in a similar manner.

Thus, the computation of the matrix elements $\langle \Phi(\gamma_i LS) | \text{OP} | \Phi(\gamma_{i'} LS) \rangle$ and the summations of Eq. (21) are fundamental to the prediction of atomic properties. Such matrices are symmetric so only the lower (or upper) portion needs to be computed. Two programs for operator evaluation have been developed: `ISOTOPE` [9] evaluates the specific mass shift and `HYPERFINE` [10] evaluates the hyperfine structure parameters, given the wave function expansions and the radial functions.

4 Distribution Strategies

A large part of an atomic property computation is the calculation of the angular coefficients for an operator, for example the Hamiltonian. With vector-coupled configuration state functions, this calculation to a large extent is logical in nature (present FORTRAN codes contain many IF statements) and DO loops in the program are relatively short. Consequently, the CPU speed-up in going from a SUN Sparcstation to a Cray 2 may be as low as a factor 2. On the other hand, since the calculations for each matrix element are independent, the calculation can easily be distributed over many processors with excellent efficiency. A commonly used strategy for matrices is to distribute the columns over the nodes in an interleaved fashion. This strategy has been used in all the results reported here. Only the lower portion was computed, leading to some bias in the workload.

The matrices associated with an operator are frequently sparse, and selection rules apply that may quickly determine that a given entry is zero after only a simple test. Thus the time required for the evaluation of matrix elements may vary greatly. A dynamic scheduling strategy could lead to improvement in performance, as was the case in multitasking studies on the Cray [11]. However, in a distributed environment, the static distribution is easier to implement and reduces communication in programs such as `MCHF` where the matrix eigenvalue problem needs to be solved

during each iteration. One of the objectives of this work is to evaluate the effectiveness of the static distribution strategy.

5 Factors Affecting Performance

Our first implementation for the Intel iPSC/2 included only the `NONH` and `MCHF` program [12]. In the former, each node produced data for the energy expression associated with the lower portion of *my_columns* and results were stored on disk, one file for each node. Then `MCHF` performed its calculations with columns and data distributed in the same fashion.

1. The Eigenvalue Phase

A single particular eigenvalue and eigenvector of the matrix was needed and was computed by an iterative method requiring the solution of a system of equations. This was a full-matrix algorithm, requiring that the computed lower half of the matrix be distributed among the nodes to fill out the upper portions of their columns. A column-wrapped algorithm was used for solving the system of linear equations.

2. The Differential Equation Phase

Since the most CPU-intensive part is the calculation of the functions $G_i(r)$, the radial functions were updated on each node (avoiding the communication of numerous items of data) but with each node contributing towards the calculation of the needed functions. Effectively, the functions were decomposed into parts related to the distributed data so that

$$Y_i(r) = \sum_{nodes} Y_i^{my_columns}(r) \quad (23)$$

$$G_i(r) = \sum_{nodes} G_i^{my_columns}(r) \quad (24)$$

Thus each node computes the contributions to these functions for the integrals and coefficients associated with *my_columns* and a global sum produces the required function.

This implementation assumed that the radial functions and all the data associated with the energy data structure could be stored in memory.

Table 1 shows timing and efficiency results for a relatively small problem with $M = 410$. However, with the severe memory constraint (4 Mbytes per node) the smallest system on which the problem could be solved was an eight-node system. Thus the definition of efficiency is $e = 8t_8/(pt_p)$, where p is the number of processors or nodes and t_p is the execution time for p

processors. The timings here do not include the time required to write or read information from disk, which was a severe bottleneck on the iPSC/2.

The range of times for NONH is an indication of the load imbalance. For a relatively small number of nodes (each node performing calculations for 51—52 columns) the work is in reasonable balance, but as the number of nodes increases to the point where each node is responsible for only 6—7 columns, the imbalance has reached almost a factor of two. We conclude that, as long as the number of columns per node is fairly large (50 or more), the work distribution is satisfactory. In some applications, NONH supports an option in which the calculations for some columns are to be omitted and only the diagonal computed. Clearly, in this case, only “full” columns should be included.

The execution times for MCHF clearly demonstrate the inefficiency of this implementation. In spite of the fact that a large portion of the calculation is associated with the matrix eigenvalue phase, which should have a relatively high degree of parallelism, this is not evident in the total times. Degradation in performance can be traced to a number of factors:

1. The full-matrix method for finding an isolated eigenvalue and eigenvector introduced too much communication. Each column of the matrix needs to have portions distributed to *all nodes* when a lower-triangular matrix is extended to a full form.
2. The integrals appearing in the energy data structure may be present in many nodes. Thus, in the eigenvalue phase, many nodes are computing the same integral, though using it only in *my_columns*. Similarly, in the differential equation phase, many nodes may be computing the same functions contributing to $G_i(r)$, for example, but again multiplying the function only by the coefficients arising from *my_columns*.

Since MCHF is an iterative computation, referencing the entire energy data structure many times during an iteration, changes should be incorporated into NONH.

6 Improving Performance

The second implementation was for a four-node iPSC/860 with 8 Mbytes/node that attempted to address the shortcomings of the implementation described above. Even though the memory/node ratio was larger by a factor 2, memory imposed severe constraints for large calculations. The angular coefficients and the associated indices are by far the largest arrays, and these are now computed and stored on disk by NONH with only sections read one at a time by MCHF.

Gains in performance come from an analysis of MCHF. Efficiency will be improved if the evaluation of the integrals can be distributed in the eigenvalue phase, and the contribution to the $Y_i(r)$ and $G_i(r)$ functions in the differential equation phase. This distribution requires that each node have access to a global list of integrals (without duplicates). Further analysis shows that in the eigenvalue phase, nodes need access to the values of these integrals, whereas at the differential equation phase, they need the contribution factor (CF), the coefficient of an integrals contribution to the energy defined as the sum on i, i' in Eq. (21). Thus the VALUE array and the CF array may share the same memory. The computation of both can now be distributed but will require a global sum. The revised energy data structure on each node is a global list of integrals, each integral having a pointer to data associated with *my_columns*. The data for the latter is on disk and is scanned only twice per iteration – once for computing the VALUE array needed for setting up the interaction matrix and once for computing the CF array. The NONH program was modified to incorporate this modification. After each node computes its data structure, the list of integrals is merged, duplicates removed, and pointers updated. This process requires $\log_2 p$ merges and introduces some communication overhead.

The efficiency of MCHF was improved further by replacing the full-matrix method used in the eigenvalue phase by an upper-triangular matrix method.

In the standard version of MCHF, the method used for obtaining a single eigenvalue and eigenvector is based on an iterative algorithm requiring the solution of systems of equations:

- From an estimate of the eigenvector, use the Rayleigh Quotient to estimate the eigenvalue
- Assuming $v_1 = 1$, solve equations $2, \dots, M$ of $(H - E)v = 0$ for v_2, \dots, v_M
- Let $c = v/\|v\|$

This method requires that the user define the problem in such a way that the first configuration state in the wave function expansion has a large coefficient (c_1). The method has been used for many years without encountering numerical instability and has the advantage that excited states may be studied without knowledge of the lower states (eigenvalues).

The above iterative method, requires the solution of a system of equations, say $Ax = b$, where A is a submatrix of $(H - E)$. Since the matrix A cannot be guaranteed in all cases to be positive definite, a Doolittle decomposition was used such that $A = LDL^t$, where L is a triangular matrix and D a diagonal matrix. In order to maximize numerical stability, the order of factorization was reversed, starting from the (M, M) position rather than the usual $(1,1)$ position.

Two routines, `SYMFA-F` and `SYMFA-B`, are implementations of the Doolittle method for forward and backward elimination, respectively, assuming the matrix is stored as a packed triangular matrix [13]. The performance of this implementation is shown in Table 2. Figure 1 compares the time for factorization of the full-matrix algorithm `PGEFA` and `SYMFA-F` (forward elimination) for matrices of different sizes on a 4-node iPSC/860. With column-wrapped interleaving, the parallelism during the solve phase is reduced, but the execution time is also much smaller. Figure 2 compares the full-matrix methods (`PGEFA` and `PGESL`) with similar routines using only the upper triangular matrix (backward elimination) on the same matrices as those of Figure 1. For the symmetric case, two solve routines are compared `-SYMSL-B1` is a straight-forward implementation whereas the `SYMSL-B2` solve routine, uses the parallel algorithm for the solution of triangular systems of equations proposed by Li and Coleman, [14] shown to be competitive when M/p is large. The latter was also used in the full-matrix implementation `PGESL`. Comparing Figure 2 with Figure 1, it is immediately evident that most of the CPU time is spent in factorization, as expected, but Figure 3 shows that the speed-up for a matrix of size 800 is affected by the algorithm used for the solve phase.

The introduction of the symmetric form not only cuts the memory required for storing the matrix, it also reduces the total computation time by a factor of two. Figure 4 compares the ratio of the time required for a full- versus symmetric-matrix factorization and eigenvalue solver for matrices of different sizes. For large matrices the ratio approaches two. Because factorization destroys the matrix, the latter was stored on disk and so some I/O is involved in the eigenvalue problem, introducing some degradation as compared with factorization alone.

As wave function expansions get large, the interaction matrices also get more sparse, though only in “first-order” calculations, when some blocks of interactions are omitted, have matrices been encountered in which the non-zero elements account for less than 1% of the total. In a new version of the MCHF program, the sparse-matrix Davidson algorithm [15], implemented for a distributed-memory system `DVDSON` [16], has been installed. This algorithm relies mainly on matrix-vector multiplication for finding a few of the lower eigenvalues. It also has the ability to target specific eigenvalues and eigenvectors that would be convenient for MCHF calculations of excited states. No detailed comparison has been made of the two eigenvalue solvers but the sparse-matrix implementation tends to be faster for large matrices: for a case with $M = 862$, the sparse version was 13% faster.

The structure of the MCHF program has also been simplified. For large cases it was found

that the rotation analysis [4], which can drastically reduce the number of MC-SCF iterations when only a few radial functions are present, is no longer effective. The time required for the rotation analysis depends on the number of orthogonality constraints. If all radial functions up to a given n are present, the number of pairs is $\sum_{i=2}^n i(i-1)/2 = n(n^2-1)/6$. Thus rotation analysis quickly becomes a large part of the calculation. In a large, systematic study of the energy of the Li ground state, it was found that the total time required to reach a converged result was less without rotation analysis, even though the rate of convergence was somewhat slower. There also were indications that the calculations were more stable. For these reasons, the rotation analysis was eliminated.

Table 3 provides timing data, the speed-up, and the efficiency for a Li case as a function of the number of processors executed on the iPSC/860. The size of the expansion is $M = 862$. The times reported are the maximum run times in seconds, which now include I/O. Because all these calculations involve synchronization (NONH requires synchronization for the merge), the method of obtaining the timings obscures any load imbalance present. The calculations for eight nodes were performed at the *Advanced Research Computing Facility* at Argonne National Laboratory.

The case considered in Table 3 is more than twice as large as the one considered in Table 1, yet a benchmark for a single node was possible whereas the earlier version required at least eight nodes. This is due to the larger memory on the iPSC/860 and to the availability of the concurrent file system, which makes this a much more powerful machine. Increasing the number of nodes by a factor of eight is accompanied by a slight loss of efficiency for NONH (0.62 vs 0.76 previously) at the same time the efficiency of MCHF has improved significantly (0.66 vs 0.22 previously). Since the execution time of the latter is at least seven times that of the former, the overall benefits are evident.

Since performance usually improves with the size of the problem, similar timing data is presented in Table 4 for a case in which $M = 1943$. These calculations were performed on an iPSC/860 with 16 Mbytes/node. The minimum size system is now a two-node system. The efficiency of the eight-node calculation relative to the two-node system is again 0.66.

It is interesting to observe that in both cases, the efficiency for a large MCHF calculation in which all radial functions are varied, tracks the efficiency of NONH very closely, in fact, almost exactly.

Table 3 also includes some timing studies for the CI program in which the radial functions are read from disk, the interaction matrix is generated, and the Davidson algorithm is called for

finding the lowest eigenvalue and eigenvector. The calculation is relatively short, involving a lot of I/O.

Both tables also compare the performance of two programs that evaluate atomic properties. The first, ISOTOPE, [9] evaluates the specific mass shift using the energy data structure determined by NONH. The second, HYPERFINE, [10] determines the hyperfine structure parameters. Unlike ISOTOPE, the matrix elements are evaluated at the same time as summations are performed. Thus, in Eq. (20), the summation on i is distributed with each node evaluating all the integrals associated with its data. For this particular property, the radial integrals are one-dimensional integrals; in fact, for the Li ground state, the integral reduces to a constant. For this reason, timing information is also given for the Li $2p^2P$ case where a number of one-dimensional integrals need to be evaluated.

7 Conclusion

The distributed-memory implementation of the MCHF atomic structure package has shown that a fairly simple underlying model can yield acceptable performance compared with a Cray 2. Memory is a critical resource. So far our implementation assumes all radial functions and the complete list of integrals are present on every node. It is anticipated, that as our problem size grows, some of this information may need to migrate to disk as well. The integral list is a candidate since it is scanned sequentially. In quantum chemical applications, the interaction matrix is stored on disk, whereas in our implementation it is in memory. However, since it is distributed over the nodes, a memory constraint can be solved through the use of more nodes.

The Davidson algorithm for the eigenvalue problem does not require column (or row) wrapped interleaving. In the NONH program memory requirements could be minimized by partitioning the matrix into rectangular blocks. Since the matrix may also be sparse, the partitioning strategy should avoid load imbalance. Ideally, the performance of the matrix-vector multiplication operation at the core of Davidsons algorithm should also be maintained. Thus many inter-related factors need to be considered.

By far the longest computations are the MCHF calculations. Performance here probably could be improved by updating a number of radial functions simultaneously, possibly one per node. In scanning the list of integrals, the node would check whether a given integral contributes to the current set of radial functions. Then each node would update one radial function. However, this

strategy would require that the results be communicated to all nodes, adding communication overhead.

8 Acknowledgments

This research has been supported by a grant from the US Department of Energy, Office of Basic Energy Sciences (C.F. Fischer and M. Bentley) and a National Science Foundation Grant No. ACS-9005687 (M. Tong, Z. Shen, and C. Ravimohan).

References

- [1] C. Froese Fischer, *The Hartree-Fock Method for Atoms: A numerical approach*, (J. Wiley & Sons, New York, 1977).
- [2] F.A. Parpia, I.P. Grant, and C.F. Fischer (in preparation).
- [3] U. Fano and G. Racah, *Irreducible Tensorial Sets*, (Academic Press, 1959).
- [4] C. Froese Fischer, *Comput. Phys. Reports*, **3**, 273 (1986).
- [5] C. Froese Fischer and B. Liu, *Comp. Phys. Commun.* **64**, 406 (1991).
- [6] A. Hibbert and C. Froese Fischer, *Comp. Phys. Commun.* **64**, 417 (1991).
- [7] C. Froese Fischer, *Comput. Phys. Comm.* **64**, 473 (1991).
- [8] C. Froese Fisher, *Comp. Phys. Comm.* **64**, 431 (1991).
- [9] C.Froese Fischer, L. Smentek-Mielczarek, N. Vaeck and G. Miecznik, *Comput. Phys. Commun.* , (accepted).
- [10] P.Jönsson, Claes-Göran Wahlström and C. Froese Fischer, *Comput. Phys. Commun.* , (accepted).
- [11] C. Froese Fischer, N.S. Scott, and J. Yoo, *Parallel Computing*, **8** 385 (1988).
- [12] M. Bentley and C. Froese Fischer, *Hypercube conversion of serial codes for atomic structure calculations*, Technical Report DOE/ER/13867-9, Vanderbilt University, Box 1679B, Nashville TN, 37235.
- [13] Zuchang Shen, *Symmetric, Non-Positive Definite Matrix Decomposition on a Hypercube Multiprocessor*, Thesis, Department of Computer Science, Vanderbilt University, Nashville, TN 37235, 1991.
- [14] G. Li and T.F. Coleman, *Siam Sci. Stat. Comp.* **19**, 485 (1988).
- [15] E.R. Davidson, *J. Comput. Phys.* **17**, 87, (1975).
- [16] A. Stathopoulos and C. Froese Fischer, *Comput. Phys. Commun.* (in preparation)

Figure 1: Execution times for full-matrix (PGEFA) and symmetric matrix (SYMFA-F) column-wrapped implementations on an iPSC/860 with four nodes for a range of matrices.

Figure 2: Execution times for full-matrix (PGESL) and symmetric matrix (SYMSL-B) column-wrapped implementations on an iPSC/860 with four nodes for a range of matrices. Unlike the other routines SYMSL-B1 does not include the Li and Coleman parallelization algorithm.

Figure 3: Speed-ups for the full-matrix and symmetric matrix algorithms on an iPSC/860 for a matrix of size $M = 600$.

Figure 4: A comparison of the algorithmic speed-ups for factorization and for the determination of a single eigenvalue and associated eigenvector. EIG1 used PGEFA and PGESL whereas EIG2 used SYMFA-B and SYMSL-B2.

Table 1: iPSC/2 timing information (in seconds) for NONH and MCHF for different numbers of nodes, p , and a problem of size $M = 410$. Efficiency is reported using largest t_p .

p	t_p Range		Efficiency	
	NONH	MCHF	NONH	MCHF
8	209.2—219.8	6782.2—6782.7	1.00	1.00
16	102.2—116.0	5663.3—5663.8	0.94	0.60
32	46.2—65.5	4542.3—4545.8	0.84	0.37
64	19.4—36.2	3810.9—3816.5	0.76	0.22

Table 2: Speed-up and efficiency of SYMFA for a random matrix of size $M = 600$. The execution time on an iPSC/860 is given in seconds.

p	maximum t_p	Speed-up	Efficiency
1	6.577	1.00	1.00
2	3.762	1.75	0.88
4	2.388	2.75	0.69

Table 3: Run time in seconds on an iPSC/860, speed-up, and efficiency of various programs for different number of processors, p . The case arises in an MCHF calculation for Li $2s^2S$, with an active set, $n = 6$, leading to $M = 862$ CSF.

p	max t_p	speed-up	efficiency
<i>Program NONH</i>			
1	651.39	1.00	1.00
2	333.14	1.96	0.98
4	185.94	3.50	0.88
8	129.27	5.04	0.62
<i>Program MCHF, varying 6 radial functions</i>			
1	760.09	1.00	1.00
2	418.83	1.81	0.91
4	263.43	2.89	0.72
8	219.04	3.47	0.43
<i>Program MCHF, varying 21 radial functions</i>			
1	4231.78	1.00	1.00
2	2213.09	1.91	0.96
4	1230.52	3.44	0.86
8	801.55	5.28	0.66
<i>Program CI</i>			
1	61.25	1.00	1.00
2	57.16	1.07	0.54
4	55.48	1.10	0.28
<i>Program ISOTOPE</i>			
1	93.54	1.00	1.00
2	51.60	1.81	0.91
4	31.87	2.94	0.74
8	22.53	4.15	0.52
<i>Program HYPERFINE</i>			
1	149.03	1.00	1.00
2	92.48	1.61	0.86
4	66.54	2.24	0.56
8	56.73	2.63	0.33

Table 4: Run time in seconds on an iPSC/860, speed-up, and efficiency of various programs for different number of processors, p . The case arises in an MCHF calculation for Li $2s^2S$, with an active set, $n = 7$, leading to $M = 1943$ CSF.

p	max t_p	speed-up	efficiency
<i>Program NONH</i>			
1	N/A		
2	1433.66	1.00	1.00
4	797.99	1.80	0.90
8	551.27	2.60	.65
<i>Program MCHF, varying 28 radial functions</i>			
1	N/A		
2	2565.34	1.00	1.00
4	1445.46	1.77	0.89
8	977.93	2.62	0.66
<i>Program ISOTOPE</i>			
1	N/A		
2	158.03	1.00	1.00
4	96.43	1.64	0.82
8	70.49	2.26	0.56
<i>Program HYPERFINE</i>			
1	610.74	1.00	1.00
2	347.67	1.67	0.83
4	216.96	2.82	0.70
8	159.77	3.82	0.48

Table 5: Run time in seconds on an iPSC/860, speed-up, and efficiency of the HYPERFINE program for different number of processors, p . The case arises in an MCHF calculation for Li $2p\ ^2P$, with an active set, $n = 6$, leading to $M = 1880$ CSF.

p	max t_p	speed-up	efficiency
1	991.02	1.00	1.00
2	539.02	1.84	0.92
4	307.75	3.22	0.81
8	205.51	4.82	0.60