A PVM Implementation of the MCHF Atomic Structure Package

Andreas Stathopoulos, Anders Ynnerman, Charlotte F. Fischer

Computer Science Department,
Vanderbilt University,
Nashville, TN 37235

Abstract

The MCHF package is a suite of programs that provide the atomic data required by many science and engineering disciplines. Previous implementations of the MCHF on parallel computers provided means for meeting the ever increasing computational demands posed by these calculations. However, the resulting codes had strict limitations on speed, solvable problem size and communication bottlenecks.

In this paper, a PVM (Parallel Virtual Machine) implementation of the MCHF is considered on a cluster of high-end workstations. Some of the limitations are thus overcome because of the high utilization of resources (CPU, prime memory and disk space). The development of efficient routines for global operations and of a user-friendly interface exploits the special characteristics of PVM-network programming. Moreover, a restructuring of the methods provides new codes that do not bear the above limitations and that exhibit significantly better speed-ups. Besides the algorithmic improvements, this paper presents a flexible code that can be used equally well on workstations and on the IBM SP2 multiprocessor to solve problems of one order of magnitude larger than any previous attempts, and thus facilitate new research in various scientific fields.

Introduction

Atomic data is needed by many areas of science and engineering. Energy level structures, autoionization rates, cross-sections for excitation, ionization, charge exchange, and recombination all enter into model calculations for controlled thermonuclear fusion (Hansen, 1990). 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 similar 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 transferable to metals, surfaces, and other polyatomic systems.

The above data can often be predicted only through computation. The Multiconfiguration Hartree-Fock (MCHF) method (Fischer, 1991b) is one of the most powerful approaches for atomic structure calculations. The MCHF package consists of a number of interrelated programs that yield the required atomic data. Several of the techniques that the MCHF method uses can be naturally extended to similar problems in other fields, such as quantum chemistry and solid state physics. Improvements on this method can contribute to those fields as well.

Accurate atomic data requires large and time-consuming computations that pose huge demands on computer resources. Supercomputers are often the only means for handling such computations. The advent of vector, parallel and massively parallel computers has provided a powerful tool towards facing these demands and a promise for tremendous advances in the future. In earlier work, the MCHF package has been ported to the Cray (Fischer et al., 1988), a shared memory parallel-vector computer, and to iPSC/860 (Bentley and Fischer, 1992; Fischer et al., 1994) a distributed memory parallel-vector computer. Both implementations have provided valuable insight on the parallelization of the codes but they also have their shortcomings. The Cray resources make it possible to tackle large problems, but the speed-ups over SUN workstations are not significant because of the non-vectorizable nature of many programs of the package. Considering also the high cost of Cray user time, this solution has proved far from optimal. On the other hand, the parallel nature of many parts of the package suggests the use of a multiprocessor. The hypercube implementation of the MCHF shows better scalability, at least for some of the programs. However, new problems have to be faced on a distributed memory environment: duplication of data, redundancy of work, static load balancing and high synchronization/communication costs. Despite the techniques developed to improve the package's efficiency, the memory remains the limiting factor. On the iPSC/860, with 8 MBytes per node, the computation of very accurate and large atomic cases is infeasible. In addition to the above shortcomings, parallel and vector computers are not always accessible to the average researcher.

The Parallel Virtual Machine (PVM) (Geist et al., 1993; Sunderam, 1990) is a tool that integrates networked resources, such as workstations, parallel computers and supercomputers, into a distributed computational environment. Because of its flexibility and high utilization of existing resources, PVM has become very popular in computer and computational sciences. In this paper a PVM implementation of the MCHF package is described on a cluster of IBM/RS6000 workstations. The first objective is to study the strategies that are required for an efficient implementation of the MCHF on a network of computers, and more generally on a multiprocessor. The second objective is to identify those important kernels that a parallel implementation is based on, and provide efficient, general purpose utility routines. Finally, the third objective is to provide a working and user-friendly PVM package that facilitates new atomic data computations that previously were not allowed because of storage and execution time restrictions.

The paper is organized as follows: Section 1 states the physical and the corresponding mathematical problem to be solved by MCHF, and describes the computational methods that MCHF is based on. Section 2 outlines the special characteristics of network parallel programming and how these affect the decisions for communication/synchronization algorithms. Section 3 gives a brief overview of the previous parallelization attempts and identifies the PVM improvements. In section 4 timings and results from various atomic cases are provided so that the efficiency of the approach is evaluated. The paper concludes with final remarks in section 5.

Results and Discussion

1 The Physical 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 focuses 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 $i^{\rm th}$ electron from the nucleus, and r_{ij} is the distance between electron i and electron j. This equation is derived under the assumption of a point-nucleus of infinite mass. The operator $\mathcal{H}$ has both a discrete and continuous spectrum. 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 (Fischer, 1977) and the MCDF method as implemented in GRASP^2 (F.A. Parpia, I.P. Grant and C.F. Fischer in preparation) based on a relativistic formalism including also some 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.

1.1 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

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

where $\{\mathbf{r}_j\} = \{r_1, \theta_1, \phi_1, \sigma_1, \cdots, r_N, \theta_N, \phi_N, \sigma_N\}$. 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,

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

where the spherical harmonics, Y_{lm_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

$$E = \langle \Psi_{\gamma LS} \mid \mathcal{H} \mid \Psi_{\gamma LS} \rangle, \tag{5}$$

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

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

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

$$\begin{aligned} E &= \sum_{ij} c_i c_j H_{ij} && (7) \\ &= \sum_{ij} c_i c_j \left(\sum_{stuv;k} A^{ij}_{stuv;k} R^k(s,t;u,v) - \frac{1}{2} \sum_{qw} C^{ij}_{qw} L_{qw} \right), && (8) \end{aligned}$$

where the R^k are Slater integrals,

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

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

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

The abbreviation, $P_i = P_{n_i l_i}$, has been used here. The $A^{ij}_{stuv;k}$ and C^{ij}_{qw} are called angular coefficients and can be computed using Racah algebra (Fano and Racah, 1959). 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 (Fischer, 1986). Thus their evaluation is non-trivial.

The $M \times M$ symmetric matrix with elements

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

is called the interaction matrix. Applying the variational condition (Fischer, 1977) 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

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

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 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".

In the MCHF atomic structure package, an interactive program GENCL (Fischer and Liu, 1991) is used to generate configuration state lists using some simple rules. Then the NONH program (Hibbert and Fischer, 1991) 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 (Fischer, 1991a) program determines selected eigenvalues and eigenvectors. By optimizing the radial functions for a particular state, much better accuracy can be obtained: the MCHF program (Fischer, 1991b) 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.

$$\langle \text{property} \rangle = \langle \Psi_{i'} | \text{OP} | \Psi_i \rangle, \tag{13}$$

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 (13) we then get the result

$$\langle \text{property} \rangle = \sum_{i,i'} c_i c_{i'} \, \langle \Phi(\gamma_i LS) | \text{OP} | \Phi(\gamma_{i'} LS) \rangle. \tag{14}$$

The computation of the matrix elements $\langle \Phi(\gamma_i LS) | \text{OP} | \Phi(\gamma_{i'} LS) \rangle$ and the summations over the CSF's involved 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 (Fischer et al., 1993) evaluates the specific mass shift and HYPERFINE (Jönsson et al., 1993) evaluates the hyperfine structure parameters, given the wave function expansions and the radial functions.

2 The PVM Choice

Conceptually, PVM consists of distributed support software that executes on participating UNIX hosts on a network, allowing them to interconnect and cooperate in a parallel computing environment. A daemon (*pvmd*) is started on each of these hosts and programs communicate between hosts through the local daemons. This distributed communication control eliminates the overhead of a master control process, and facilitates functionalities as point-to-point data transfers and dynamic process groups. In addition the software is written in a low level language, thus dispensing with the overheads that other systems have (eg., P4, Linda, etc). PVM provides a suite of user interface primitives to control the parallel environment that may be incorporated into existing procedural languages. Therefore, porting a single CPU program to PVM should present no additional difficulties from porting it to a distributed memory multiprocessor such as the CM-5 or iPSC/860.

There are many more attractive reasons for using PVM other than its flexibility. It offers an inexpensive platform for developing and running parallel applications. Existing workstations or even mini and supercomputers can be linked together on existing networks (Ethernet, FDDI, or even Internet) and challenge the performance of many massively parallel computers. The support of heterogeneous machines offers a better utilization of resources and it facilitates a partitioning of special tasks according to the specialized hardware. Also, the support of the popular programming languages C and Fortran enables the use of highly optimized compilers, debuggers, editors and monitoring tools that are not always available on parallel machines. Most important, however, is the integration of disjoint memories and disk space from the participating machines. It is not uncommon that today's workstations have main memories in the range 64-256 MBytes. In this case an accompanying disk of 1-2 GByte is also reasonable. Eight 128-MByte workstations working under PVM have more memory than a 128-node iPSC/860 and a considerable amount of disk space. Considering also the data duplication required on 128 processors, the solvable problem size under PVM is much larger, for a significantly lower cost.

Porting MCHF to PVM is suggested by many of the above features as well as by its previous parallelization on the iPSC/860. Specifically, an earlier Cray implementation has demonstrated poor vector performance and does not account for using such an expensive machine. Subsequent iPSC/860 implementations have yielded better speed-ups, but because of the limited memory size on the nodes (8-16 MBytes), solution of large problems has been infeasible. Solving large problems more accurately, faster, and without the high costs for current multiprocessors machines are the motives behind the PVM implementation.

However, PVM is not without disadvantages. First it is public domain! Although this is usually cited as an advantage, it implies lack of a full working guarantee. Caution should be taken whenever an application is ported to PVM. A possible problem with heterogeneous computing is the incompatibility between mathematical libraries. However, most of the known manufacturers adhere to the standards. The major problem in distributed memory multiprocessors is the high communication and synchronization costs. Various high performance networks have been adopted to alleviate the problem; the hypercube interconnection for hypercubes, the fat-tree interconnection for the CM-5, various mesh interconnects for CM-2, Paragon and others. In the case of PVM, the network is simply an Ethernet Bus or FDDI connections. For generality and compatibility reasons, the low performance communication protocols have been retained in PVM. As a result, PVM communication is slow, with high latencies, and no specific topology with nearest neighbor links. In increased network traffic, communication becomes the bottleneck of the application. In addition to network problems, current PVM version (3.2) performs excessive buffering of messages during message passing, further slowing communications. Finally, PVM does not support asynchronous send/receive operations and collective operators are not efficiently implemented.

2.1 Implications to code development

The above discussion of PVM implies that porting an application to PVM is not merely a translation of a previous parallel code. The characteristics of MCHF that affect the PVM-parallel performance are studied in the next section. In this section the effect of network programming on basic global communication primitives is discussed.

Parallel scientific applications are built around some basic communication utilities that involve all or groups of processors: global synchronization, global element-wise summation or computation of the minimum/maximum values of an array, global concatenation (collection) of a distributed array etc. A complete list of such utilities developed for PVM is shown in Table 1. This section studies the global sum (GSUM) as a representative of the reduction operations (i.e., operations that combine values from different processors to give one result-value), and also the global collection operation (GCOL). Since these routines are called numerous times during execution, their efficient implementation is vital to the performance of the algorithm.

Table 1: Global communication operations implemented for PVM. Except for the global collections all routines apply element-wise to arrays.

GCOL	Global concatenation	GCOL_SPC	Global special collection
GSUM	Global summation	SWAP	2-node info swapping
GLOW	Global minimum	GMAX	Global maximum
GAND	Global AND	GOR	Global OR

Algorithms for multiprocessors

In massively parallel processors the fan-in algorithm is usually the choice for implementing reduction operations, and the fan-out for broadcast or collection operations. They are both logarithmic in the number of processors and linear in the size of the arrays involved. Assuming that the number of processors is a power of two ($P = 2^d$), the fan-in/out algorithms are based on interchanging information in the d dimensions successively. After the d steps all processors have the final information. In the fan-in algorithm the communicated array is always of the same size (N). For the fan-out the size of the array doubles in each step, starting from $N/2^d$. If the network has a hypercube topology or it supports direct communication links between nearest neighbors, processors can execute in parallel, or partly in parallel, in each of the d above steps. However, for bus based networks as the Ethernet, all these messages have to serialize. Efficient PVM routines should take this into account, minimizing the number of messages and the amount of traffic incurred.

Algorithms for PVM

Let $P = 2^d$ the number of PVM processors and N the size of an array. The fan-in algorithm that performs a GSUM on the array, sends dP messages and communicates a total of dPN numbers. The amount of traffic can be reduced by the following simple algorithm: Node 1 sends the array to node 2, node 2 adds its contribution and sends it to node 3, and so on until node P has added its own contribution. At that time the result can be broadcasted to nodes 1 to $P-1$. The number of messages is now $2(P-1)$, having communicated $2(P-1)N$ numbers. Moreover, the last message is a broadcast which can be usually implemented as a fast, single message on a bus-network. This does not apply, however, to the current release 3.2 of PVM, because broadcast messages pass through the local daemons and they do not exploit the point-to-point communication facility (Douglas et al., 1993; Geist et al., 1993). It is thus more efficient to explicitly send the message to each processor. Even in this case, the network traffic is smaller and the algorithm should perform better than the fan-in. Figures 1(a) and 1(b) demonstrate this for small and large size arrays respectively.

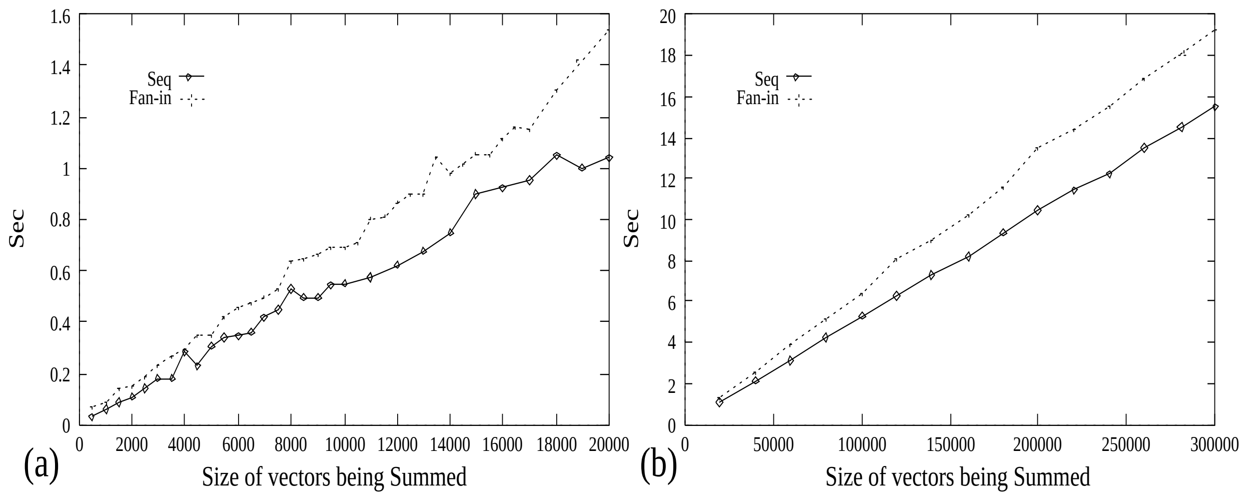

Figure 1: Execution times for the fan-in and sequential algorithms, for small (a) and large (b) vector sizes. Times represent averages from 100 executions.

For the GCOL operation the fan-out algorithm is optimal with respect to the number and size of messages sent, when these are serialized. It can easily be shown that the number of messages is dP and the total size of communication is $(P-1)N$ numbers. Any algorithm under PVM which required at least one broadcast of the final concatenated array, would involve communication of more than $(P-1)N$ numbers. Despite this optimality, another PVM problem related with the fan-in/out approach may cause inefficiencies for small sizes. In each step of the fan-in/out algorithm, half of the processors attempt to send a message simultaneously and a lot of conflicts occur on the Ethernet. Because of the high latencies of PVM sends, these repeated sending attempts cause overheads, especially with small size arrays. To alleviate this problem without increasing the communication volume, an algorithm similar to the above for GSUM has been implemented. Each node receives the local parts of the array (approximate size N/P) from lower numbered nodes, it then broadcasts its local part to all nodes, and waits to receive the local parts from higher numbered nodes. Again broadcasting takes place in the aforementioned way. It can be seen that the number of messages has increased slightly to $(P-1)P$ but the communicated volume is the same: $(P-1)N$. However, the ordered communication offers smaller overheads than the fan-out algorithm which is clear from Figure 2(a). Otherwise, the two algorithms scale similarly and for large sizes they are identical (Figure 2(b)).

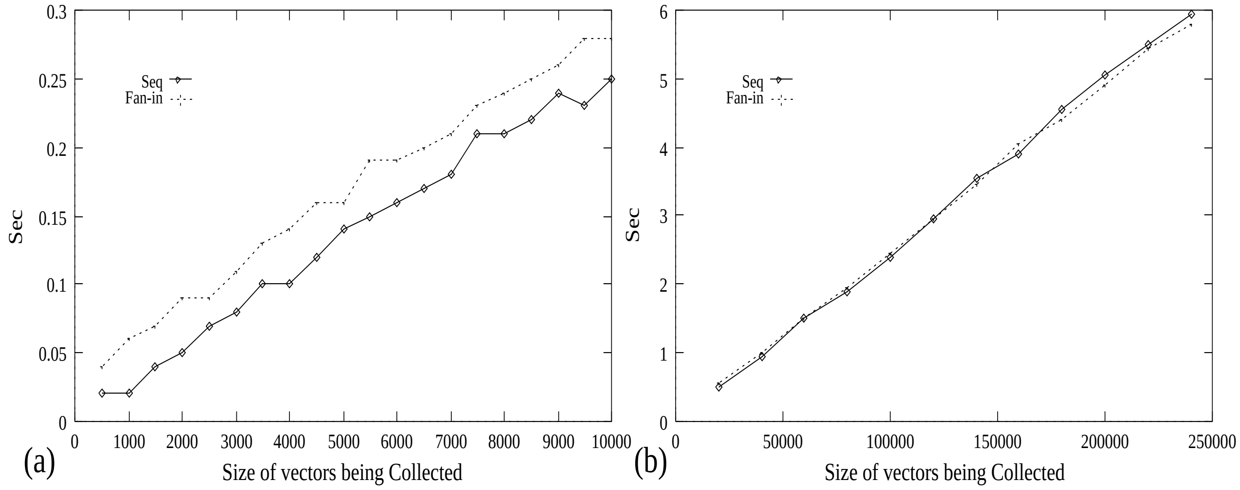

Figure 2: Execution times for the fan-out and sequential algorithms, for small (a) and large (b) vector sizes. Times represent averages from 100 executions.

Finally, it should be mentioned that in general the number of processors is not a power of two, in which case the benefits from using the above algorithms over the fan-in/out can be larger. The above strategies have been followed for the rest of the global operations appearing in the next section.

3 Implementing MCHF on PVM

3.1 The parallel profile of the package

The MCHF atomic structure package consists of a series of programs that communicate information through files. The basic two programs are the NONH and MCHF, while the rest (CI, HFS, ISO and BRCI) share similar structure and algorithms. The NONH (Hibbert and Fischer, 1991) program computes the static data needed for the self-consistent-field (SCF) iteration. The MCHF (Fischer, 1991b) program performs the SCF iteration and in each cycle an eigenvalue problem involving the Hamiltonian is solved for the required lowest eigenvalues (energy states). The radial functions are updated using the solution of the coupled system of integro-differential equations. The BRCI is a straightforward extension to the NONH and CI programs, that computes the additional Breit-Pauli interaction on each of the matrix elements. However, it is usually called when a full optimization of the radial functions cannot be afforded, because of the size of the problem.

As it has been mentioned in the previous implementation (Fischer et al., 1994), the computation of the angular coefficients and the determination of associated integrals is logical in nature and does not vectorize well. However, the calculations for each matrix element are independent and thus easily distributed over many processors. The distribution by columns has been followed both in (Fischer et al., 1994) and in the current PVM implementation. With this distribution option, processors can compute the angular parts of each element in the local columns completely in parallel in the NONH program. The MCHF program is iterative in nature and its parallelism does not share the same coarse grain granularity with NONH. Its phases however are easily parallelizable. In the first phase the values of the integrals are calculated in a distributed manner, the coefficients are read from disk and the local parts of the Hamiltonian are computed on each node. At the end of this phase a large eigenvalue problem is solved using a parallel method. The method used in the PVM implementation differs from that in (Fischer et al., 1994). In the second phase the integral coefficients are updated in parallel and after a global summation they are used for setting up and solving the integro-differential equations. Most of the solution process takes place in parallel with some intermediate synchronization points. Therefore, the parallelization profile of NONH and MCHF is similar to the one in (Fischer et al., 1994). NONH entails a fork of tasks initially and a join only at the end. MCHF consists of several forks and joins in each iteration.

3.2 Problems with previous implementations

The code in (Fischer et al., 1994) provided partial solutions to the parallelization inefficiencies of a previous effort in (Bentley and Fischer, 1992). The source of inefficiencies in (Bentley and Fischer, 1992) can be traced in the following two problems:

1. The storage of the full matrix despite its symmetry, and the use of a full matrix method to solve the eigenvalue problem.
2. The choice of distribution strategy and data structure for computing the values of integrals.

In an attempt to solve the first problem, the full matrix was replaced by a half matrix, storing only the lower triangular part (Fischer et al., 1994). The Rayleigh Quotient Iteration was then applied to find the required eigenstates. Since in general the Hamiltonian is not positive definite, a parallel implementation of the Doolittle factorization was used, such that $A = LDL^t$, where L is a triangular matrix, D a diagonal matrix, and only half of A is stored (Shen, 1991).

The distribution of columns to processors is performed in an interleaved (or wrap-around) way. With this static scheme the local columns tend to span uniformly the whole matrix, providing good load balancing. The data structure used in the calculations consists of two coupled lists. The first is the list of integrals generated for the local columns of the node. Each integral has a pointer pointing to the group of coefficients associated with that integral. The second is the local list of coefficients together with the pointer that points to the matrix element that this coefficient contributes to. This information is built in NONH once, and it is used in each iteration of MCHF. Therefore, all necessary computations of integral values in MCHF are performed locally on each node and independently from other nodes. The problem stems from the fact that the many integrals often appear in more than one nodes. Thus, in the eigenvalue phase, many nodes are computing the same integral, though using it only in the local columns. Similarly, in the differential equation phase, many nodes may be computing the same functions contributing to their local $G_i(r)$. To dispense with this time-consuming redundancy, NONH was modified in (Fischer et al., 1994) to merge-sort all the local integral lists before exiting, so that a duplicate-free global list is available on each node. When later the integral values are needed in MCHF, each node computes the values of only a small part of integrals and the array of values is globally collected (GCOL).

The above two changes improved the MCHF performance significantly. However, NONH's parallel profile and execution speed was impaired by a long communication and sorting step at the end. There are also other problems that the implementation in (Fischer et al., 1994) did not address, a list of which appears below:

(a) The symmetric matrix still has to be factored. The cubic computational order of this task makes the method prohibitively expensive for very large problems.

(b) As the number of nodes increases, static load balancing proves more and more ineffective for distributing the work equally among nodes. This is especially the case when each node has very few columns.

(c) The problem sizes solvable by the iPSC implementation are limited because of the restricted main memory on each node, as well as the the static memory allocation and the absence of virtual memory that the operating system imposes.

(d) Finally, the sorting phase at the end of NONH not only slows down the program, but its memory demands cause an application bottleneck by limiting the size of coefficient lists that can be processed.

The PVM implementation successfully resolves the above problems leading to a fast and robust code.

3.3 PVM enhancements

An efficient implementation should take into account both program and hardware characteristics. The current locally available hardware configuration is a cluster of four IBM/RS6000-370 workstations connected on an Ethernet network. Each computer has 128 MBytes main memory and nearly 2 GBytes of disk. A number of Sun SPARCstations are also available but have not been integrated into the system. This is mainly because of the increase in communication costs between heterogeneous machines and the small computational contribution. Currently, we are looking for ways to efficiently embed these machines in the system.

The Davidson algorithm

One of the major changes in the PVM implementation is the use of the Davidson sparse-matrix, iterative solver for the eigenvalue problem (Davidson, 1975). There are many advantages of such a solver: First, the matrix is not factored or modified. Second, the matrix is only needed implicitly through a matrix-vector multiplication. Third, the zero elements of the matrix may not be stored. As a result this method is much faster than the method used in (Fischer et al., 1994). In addition to the above, the Davidson code developed in (Stathopoulos and Fischer, 1994) possesses many additional features, such as targeting specific eigenpairs, using a block method and other powerful techniques.

The Davidson algorithm is similar to the Lanczos algorithm, in that it relies on matrix-vector multiplications to build a Krylov space from which the required eigenpairs are approximated. The difference is that the Davidson algorithm preconditions the new basis vector in each iteration (Stathopoulos and Fischer, 1994; Morgan and Scott, 1986; Umar and Fischer, 1989). Therefore, the simple three-term recurrence of Lanczos is lost and a full orthogonalization of the new vector must be performed. In return for this increase in computation, the Davidson method considerably reduces the number of iterations depending on the quality of the preconditioner. Since in atomic structure calculations the diagonal preconditioning (scaling) has been shown to be very effective (Stathopoulos and Fischer, 1994), and because of its excellent parallel properties it is adopted in the PVM implementation. A parallel implementation of the Davidson algorithm for the hypercube has been developed in (Stathopoulos and Fischer, 1991), and its parallel properties have been studied in (Stathopoulos and Fischer, 1993). The matrix-vector multiplication is computed from the half part of the sparse matrix. After each node has computed its contributions to the resulting vector, the vector is globally summed over the nodes. Since this incurs a communication step in each Davidson iteration, reducing the number of iterations is one of the very desirable features of the Davidson over the Lanczos method.

Load balancing

Providing a solution to problem (b), i.e., the inefficiencies caused by static load balancing, is not as easy as for the eigenvalue problem (a). Three are the main reasons:

- Employing dynamic load balancing involves several additional communication and synchronization points.
- On a network of workstations, there are two sources for load imbalances: machine heterogeneity and various loads from other users. The first is predictable but the second is not and thus any static load balancing scheme will bear this problem.
- The data structure used in MCHF is produced by NONH. A load balanced NONH does not necessarily imply a good load balancing for MCHF, since the load of each system might change between the runs.

The NONH may benefit from dynamic load balancing because of its parallel nature with no synchronization points. The same is not the case with the eigenvalue phase of MCHF since the matrix distribution has been determined earlier. Dynamic load balancing can only be applied to the solution of the integro-differential equation. However, because of the much shorter duration of this phase than NONH, the additional synchronizations would probably offset any benefits from better load distribution.

Dynamic load balancing is still a topic of research, so the current PVM implementation provides two static load balancing alternatives. One is the previously used interleaved column scheme (Fischer et al., 1994), and the other is a block scheme, where processors are assigned a number of contingent columns. The reason for the inefficiencies of the interleaved scheme in (Fischer et al., 1994) has been traced to the relatively large number of processors (8-64), and to the small sizes of the matrices. In the PVM code this is not a concern since a small number of workstations with large amounts of memory are linked together. In the experiments and timings appearing in the next section, the interleaved scheme has given almost perfect load balancing for both NONH and MCHF. The only concern for this scheme is the aforementioned second reason: heterogeneity and variations in system loads. This is where the block scheme offers some improvements. On input the user provides the relative speeds of the machines that participate in the calculation. Based on these data and assuming that the nonzero elements are evenly distributed between distant matrix locations, a routine determines the number of columns that each node computes. The same distribution must inevitably be followed by MCHF. It should be noted that the relative speeds can model heterogeneity or different loads.

Facilitating large problems

Problems (c) and (d) of the previous implementation are intimately related since the sorting phase places restrictions on the solvable problem size. The transition from the iPSC/860 to workstations has increased the availability of resources. In the PVM implementation this increase is exploited in order to overcome problem (c) of the previous codes. Specifically, dynamic memory allocation is used for meeting the memory needs of the phases of each program. For example, in MCHF the eigenvalue phase requires the matrix and its indices to be in memory while in the later differential equation phase the allocated matrix memory is freed to be used for other storage. In addition to the large main memories of the current hardware configuration, the virtual memory of the systems can also be exploited. A maximum 256 MBytes of virtual memory has been assigned to each processor. In this way many memory-demanding parts of the code do not have to be rewritten to use a disk-version, which often is not a trivial or even possible task. As a result of the above, very large and accurate calculations can be obtained from the new implementation. The sizes of the test problems in the next section are more than tenfold the sizes solved previously on the iPSC/860 (Stathopoulos et al., 1994).

In a first attempt to alleviate problem (d) for the NONH sorting phase, the code has been optimized and with the help of dynamic memory allocation the memory requirements have been reduced by one fourth. Despite this fact and the large available memories, some of the very large problems still encountered memory limitations. In this first attempt to make a working package, a disk version of the merge-sort routine has been written that does not require all the integrals to be in memory. Instead, sorting and merging proceed by reading blocks of integrals. The work is still distributed over the nodes but execution time deteriorates significantly. With this solution the goal of a working package has been achieved, but the concern over the efficiency requires rethinking of the algorithm and data structure. This is addressed below.

Restructuring improvements

During the NONH phase of the computation the lists of angular coefficients and corresponding radial integrals associated with the non-zero matrix elements of the Hamiltonian matrix are generated. As mentioned in section 3.2, the same integral can occur in many different places of the coefficient list. To avoid the repeated evaluation of the same integrals from many nodes in MCHF, the data is merge-sorted according to integrals at the end of the NONH run. After this merge-sort phase all nodes have a duplicate-free global list of integrals. Problems (c) and (d), i.e., speed and solvable problem size restrictions, are associated with this approach.

To overcome this merge-sort "barrier" a new approach is adopted. Instead of creating the associated integrals for each coefficient sequentially, a list of all possible integrals is generated. This is done using simple selection rules, based only on the list of all electrons that are occupied in at least one of the configurations (CFG's). Computationally, this step is not very expensive since only the list is generated and not the values. It turns out that for large cases, more than 1000 CFG's, most of the possible integrals are actually used. Even so, a logical index array that indicates whether an integral is used in the construction of the local matrix has been included. If pointers are assigned to the corresponding integrals as the local coefficients are generated, no sorting is needed. In the MCHF phase all the integrals are evaluated in a distributed manner as before. The difference is that the global logical index array produced in the NONH phase is used to determine if an integral is to be evaluated or not.

An additional benefit of this approach is that coefficient data appears in the order that it is used during the generation of the matrix elements. This simplifies the file handling and additional memory can be saved. Moreover, the matrix can be generated one column at a time. This enables disk matrix-storage and thus the facilitation of extremely large cases, even when physical memory does not allow it.

The above outlined scheme is implemented in the codes prefixed with a "U". The UNONH has very good parallel characteristics which is clear from the next section. Also, execution time is usually better than the old codes since the overhead introduced by having a predefined integral list accessed through a lookup procedure is compensated by the lack of sorting and decreased communication needs.

For the UMCHF code there is a communication overhead introduced by the somewhat longer list of integrals. In this list, even though unused integrals are not calculated, their zero values are communicated. Future implementations will dispose of this overhead in the expense of gathering and scattering of integrals. The book-keeping of the non-zero matrix elements is also somewhat more involved. Therefore, the total runtime for the UMCHF code should be slightly longer than the corresponding times for the MCHF code. However, the new codes are more flexible and robust and can handle arbitrarily large cases.

The UBRCI code has benefited the most from the above data structure since previously integrals were computed in an inefficient way. The changes result in not only better parallelizability but in much shorter execution time as well. Since UBRCI is used for cases so large that UMCHF cannot be used efficiently, the benefits of the new data structure are significant.

Other enhancements and tools

Further optimizations have been possible to some pieces of the code. In the differential equation phase the work for the computation of the Y_i and G_i functions can be distributed over the nodes, requiring global summation at the end. In the hypercube implementation the global summation is performed twice, once for the values of each function. In the PVM code all work requiring the Y_i and G_i functions is postponed until both sums are at hand. Thus, only one global sum is needed, reducing synchronization and overheads. Since the above routines are time consuming and they are called frequently, this modification improved the speed-up and the execution time of this phase.

Similar enhancements have been extended to the other programs in the package, namely CI, ISO, HFS, because they share a form similar to NONH. BRCI, not present in (Fischer et al., 1994), has also been implemented in PVM. The efficient parallelization of this code is important, since it is used for very large cases. The initial phase is similar to NONH while in the second phase the parallel Davidson algorithm is used for solving the eigenvalue problem. To provide fault tolerance to the long runs incurred by BRCI, a restarting mechanism has been embedded that uses the data files that each node stores on its local disk.

Finally, besides the global operations in Table 1, various tools and utility routines have been developed that provide a user-friendly interface to the user and some "performance debugging" capabilities to the developer. A user should only provide the list of machines to be used and their expected speeds, and invoke a simple X-interface to PVM on one of these machines. Temporary files on local disks, paths, distribution, and output files are handled by the utilities. After the completion of the run a cleaning utility frees the temporary disk space. Minor code modifications have also allowed the production of trace-files for monitoring performance of parallel processes. Tools, that depict this information in a way similar to PARAGRAPH (Heath and Etheridge, 1991) have been of considerable aid to the code improvements. In Figure 3 the system load and the communication pauses of each process for one MCHF iteration are shown.

4 Results and Timings

Experiments with the above codes have been performed on two and four nodes of the aforementioned cluster. Briefly, the experiments show that the initial PVM implementation behaves in a similar way to the hypercube one, as a result of the special care taken for transporting it. They also show that the bottlenecks of the first implementation are successfully faced with the new data structure.

Four test cases have been used in the experiments. They are all based on the calculation of the hyperfine structure in the ground-state of the neutral Na atom. The first list contains all double and single excitations that can be performed when the single electron orbitals are limited to have quantum numbers with $n \leq 11$ and $l \leq 5$. The second list is obtained with $n \leq 12$ and $l \leq 5$. The third and fourth lists also contain some selected triple excitations and the number of CFG's is dramatically increased. These cases constitute a worst case scenario of many possible test cases in three ways. First, they have a large number of integrals associated with a comparatively limited number of configurations. Thus, all the codes are expected to spend a significant amount of time communicating,

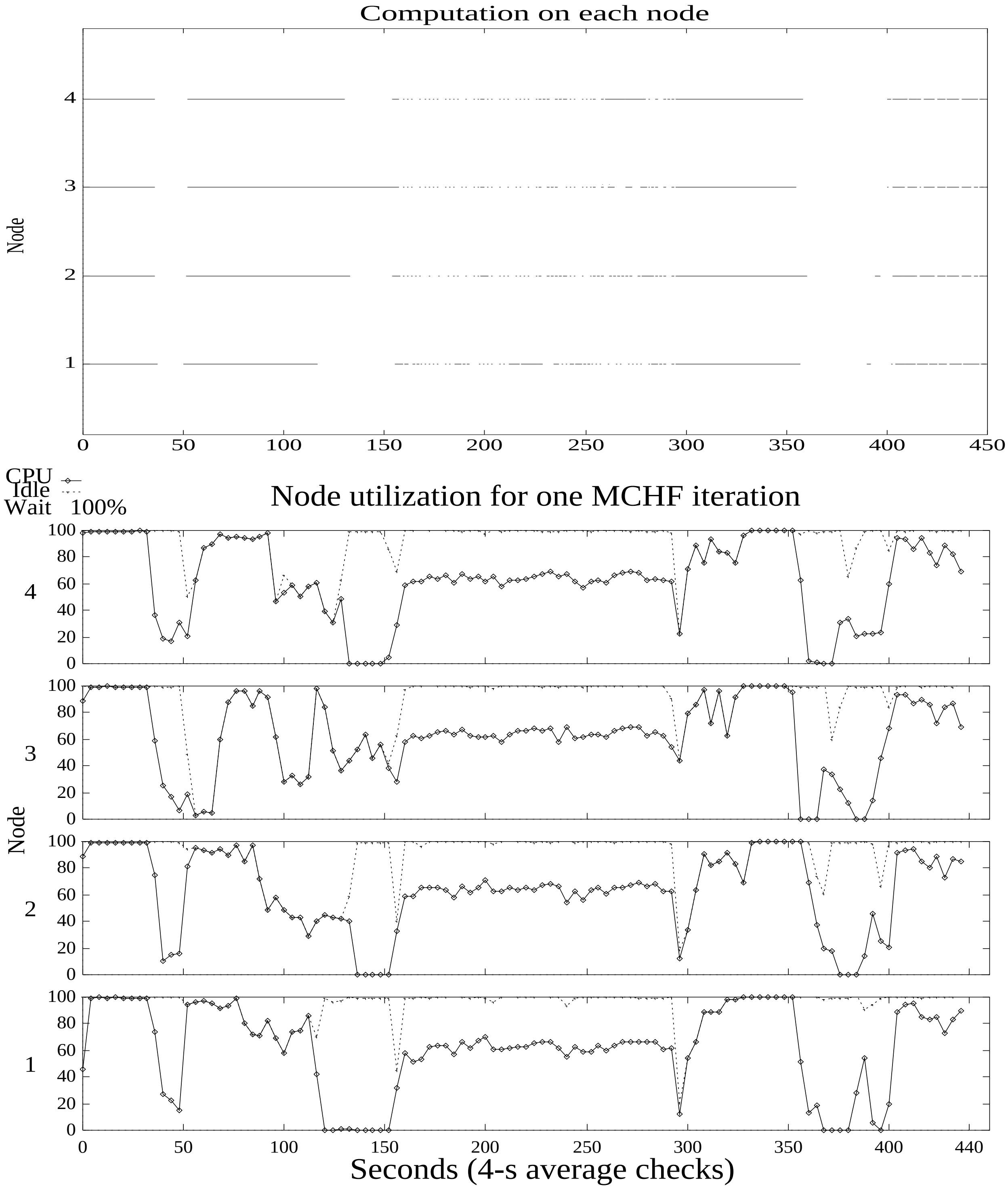

Figure 3: Top: communication (blanks) and computation (lines) intervals for each processor, for one MCHF iteration. Bottom: accumulative percentages of time distribution on each processor, to CPU, Idle and Wait time, for the same MCHF iteration.

calculating and book-keeping integrals. Second, the eigenvalue problem that is solved in each SCF iteration is not well conditioned and thus the Davidson algorithm takes more than 100 steps to converge. This involves a corresponding number of communication steps in each SCF iteration. Third, the memory requirements of these cases can be met by the main memories of even a single machine in the cluster. Thus, speed-ups emanating from reduced page-swapping, by distributing the virtual memory needs, are not present in the following timings. Different cases or use of clusters with smaller memories demonstrate better efficiencies.

Table 2: Initial PVM implementation. Time (T), Speedup (S) and Efficiency (E) for 1, 2 and 4 nodes.

Benchmark results for NONH

CFG's	1 Node	2 Nodes			4 Nodes		
	T	T	S	E	T	S	E
7158	7490	3832	1.95	0.977	2272	3.29	0.824
8960	11731	6160	1.90	0.952	4038	2.90	0.726
10965	17274	9057	1.91	0.954	6515	2.65	0.663
16091	29459	14989	1.97	0.983	8430	3.49	0.874

Benchmark results for MCHF

CFG's	1 Node	2 Nodes			4 Nodes		
	T	T	S	E	T	S	E
7158	2499	1325	1.87	0.943	1051	2.38	0.594
8960	3862	2367	1.63	0.816	1786	2.16	0.541
10965	5756	3692	1.56	0.780	2495	2.31	0.577
16091	5429	3336	1.63	0.814	2109	2.57	0.644

Benchmark results for BRCI

CFG's	1 Node	2 Nodes			4 Nodes		
	T	T	S	E	T	S	E
7158	16177	12860	1.26	0.629	7861	2.06	0.514
8960	32913	27281	1.21	0.603	16290	2.02	0.505
10965	65674	56396	1.16	0.582	36760	1.79	0.447

Tables 2 and 3 show results from the initial and restructured PVM implementations respectively. For each of the four cases, time, speed-up and efficiency for one, two and four nodes are reported. The best available uniprocessor code is used for this benchmarking. Timings are given only for NONH, two MCHF iterations and BRCI. The rest of the suite programs (ISO, CI, HFS) have very short execution times (see also (Fischer et al., 1994)) and their parallelization serves more for maintaining format compatibility.

In Table 2 the initial PVM implementation effort is depicted. The speed-ups and efficiencies are similar to the ones reported for the hypercube version (Fischer et al., 1994). The effect of the much slower network is counterbalanced by the larger problems. This enables also a very good load balancing stemming from the interleaved scheme. This table provides a verification of the bottlenecks present in NONH and BRCI despite the inherent parallelism of the methods. In NONH parallelism is impaired by the communication-intensive merging phase, reducing the efficiency on four processors to 66%. Since BRCI was not implemented in the hypercube version, the initial data structure is proved inefficient for this code. Parallel efficiencies hardly reach 50% for four nodes, even for long execution times. What is more alarming is that increasing problem size does not improve the efficiency because of duplication of integrals. The iterative nature of MCHF which forces a medium grain parallel method accounts for the relatively low efficiencies on table 2. Despite these bottlenecks and inefficiencies, the size of the problems solved is more than eight times larger than those in (Fischer et al., 1994).

Table 3: PVM implementation of the restructured ("U") codes. Time (T), Speedup (S) and Efficiency (E) for 1, 2 and 4 nodes.

Benchmark results for UNONH

CFG's	1 Node	2 Nodes			4 Nodes		
	T	T	S	E	T	S	E
7158	8459	3646	2.32	1.160	1855	4.56	1.140
8960	11232	5690	1.97	0.985	2909	3.86	0.965
10965	16864	8579	1.97	0.985	4363	3.86	0.966
16091	28677	14471	1.98	0.990	7293	3.93	0.983

Benchmark results for UMCHF

CFG's	1 Node	2 Nodes			4 Nodes		
	T	T	S	E	T	S	E
7158	3083	1486	2.07	1.035	997	3.09	0.773
8960	4707	2308	2.04	1.020	1526	3.08	0.771
10965	6782	4066	1.67	0.835	2338	2.90	0.725
16091	5908	3501	1.69	0.845	2068	2.86	0.714

Benchmark results for UBRCI

CFG's	1 Node	2 Nodes			4 Nodes		
	T	T	S	E	T	S	E
7158	7502	3853	1.95	0.974	2067	3.63	0.907
8960	11779	6015	1.96	0.979	3200	3.68	0.920
10965	18014	9062	1.99	0.994	4877	3.69	0.923
16091	29709	14966	1.99	0.993	7787	3.82	0.954

In Table 3 the significant improvements of the restructuring are evident. After the disposal of the sorting and merging phase, UNONH exhibits an ideal speed-up for both two and four nodes and for all cases. The uniprocessor code has also been sped up slightly.

Table 4: PVM restructured ("U") codes on the IBM SP2 multiprocessor. Results for 4, 8 and 14 nodes. Speedup and efficiency are based on the 4-node time.

Benchmark results for UNONH

CFG's	4 Nodes	8 Nodes			14 Nodes		
	T	T	S	E	T	S	E
7158	1180	616	1.92	0.958	380	3.11	0.887
8960	1830	950	1.93	0.963	569	3.22	0.919
10965	2765	1406	1.97	0.983	849	3.26	0.931
16091	4638	2355	1.97	0.985	1389	3.34	0.954

Benchmark results for UMCHF

CFG's	4 Nodes	8 Nodes			14 Nodes		
	T	T	S	E	T	S	E
7158	538	345	1.56	0.780	355	1.52	0.434
8960	846	519	1.63	0.815	504	1.68	0.480
10965	1317	775	1.70	0.850	719	1.83	0.523
16091	1091	643	1.70	0.848	640	1.70	0.487

Benchmark results for UBRCI

CFG's	4 Nodes	8 Nodes			14 Nodes		
	T	T	S	E	T	S	E
7158	1304	736	1.77	0.886	503	2.59	0.741
8960	2028	1148	1.77	0.883	775	2.62	0.748
10965	3043	1691	1.80	0.900	1136	2.68	0.765
16091	4875	2584	1.89	0.943	1634	2.98	0.852

The UMCHF uniprocessor timings have increased about 10-20% from the initial ones, as predicted in the previous section, but the parallel efficiency of the code has improved by more than 15% for most cases. As a result, the UMCHF code is faster than the MCHF on four nodes and looks more promising for larger clusters. The benefits of the new data structure are more pronounced in the UBRCI case. The uniprocessor code is almost four times faster than the BRCI and it also exhibits an excellent parallel efficiency. Considering the execution times of cases for which BRCI is used, the new codes contribute enormous time savings.

An advantage of the developed PVM codes is that they are readily transportable to the IBM SP2 multiprocessor. To show the scalability of the new codes, the above cases have also been tested on the IBM SP2, with 16 Thin Power 2 RS6000 processors, using the High Performance Switch interconnection. The results shown in Table 4 reveal that even for these relatively small cases, good scalability persists even for 14 nodes for the UNONH and UBRCI. The iterative nature of UMCHF results in smaller efficiencies, but only for 14 nodes. The UMCHF and UBRCI speedup curves for the smallest case on the SP2 and on the local

4-node cluster are shown in Figures 4 and 5 correspondingly. These timings show that the SP2 multiprocessor can efficiently be used for extremely large atomic data calculations. In one of our experiments, a 68000 CFG's BRCI problem was solved in less than 2 hours using 12 processors.

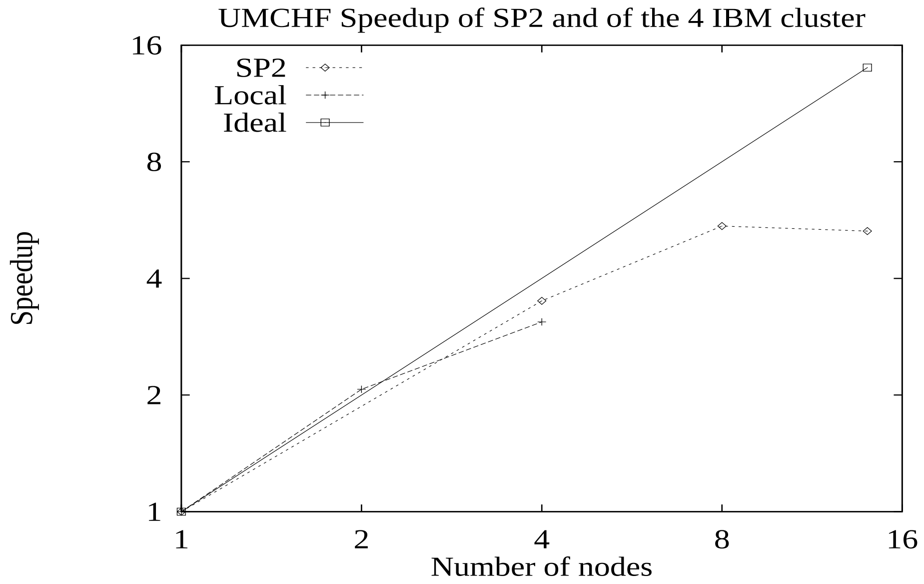

Figure 4: UMCHF speedup curves for the smallest case on the SP2 and on the local 4-node cluster

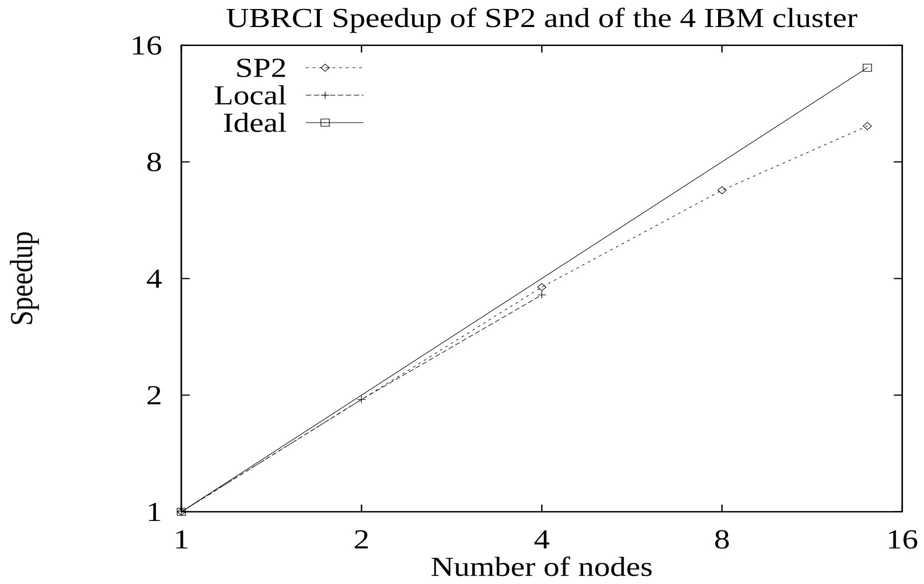

Figure 5: UBRCI speedup curves for the smallest case on the SP2 and on the local 4-node cluster

Finally, it should be mentioned that tables 2 and 3 do not show the effect of the memory barriers' removal from the "U" codes. It is now possible to run cases with even 100000 configuration states, provided that the disk space is sufficient. In such cases more efficient use of virtual memory would lead to superlinear speed-ups. This is obvious even from the initial codes when run on 64 MBytes of memory (see table 5 and also (Stathopoulos et al., 1994)).

Table 5: Superlinear speed-ups for the initial codes when run in smaller memory configurations

NONH

Case	1 Node	4 Nodes	Speed up
7158	9169	2622	3.50
8960	23350	5745	4.06
10965	15725	5544	2.84

MCHF

Case	1 Node	4 Nodes	Speed up
7158	9388	4876	1.93
8960	21541	2910	7.04
10965	11894	2011	5.91

Benchmark results for 64 MBytes memory

5 Conclusions

Accurate calculation of atomic data requires large and time-consuming computations. The MCHF package is a suite of codes designed for this purpose. A PVM implementation is based on a previous hypercube code. The special characteristics of network programming and PVM are taken into account so that the PVM code retains the parallel behavior of the hypercube one. For this purpose, utility routines for global reduction and collection operations have been written and a user-friendly interface has been developed.

The data structure used in this program is the source of the time inefficiencies observed in both the hypercube and the PVM version. In addition, this data structure puts severe limits on the size of the problems that can be solved. A restructuring of the methods has provided new codes that do not bear these limitations and that exhibit significantly better speed-up for the MCHF code and almost ideal speed-ups for the BRCI and NONH.

Besides these algorithmic improvements, this paper has also presented a flexible code, that can be used in workstation clusters to solve problems of one order of magnitude larger than any previous attempts, and thus facilitate new research in various scientific fields. Moreover, by using the code on the IBM SP2 multiprocessor with 8, or more processors, new breakthrough calculations are being performed.

Acknowledgments

This research was supported by the National Science Foundation Grant No. ACS-9005687. Computer facilities were provided by a grant from the Division of Chemical Sciences, Office of Basic Energy Research, U.S. Department of Energy and through a Joint Study Agreement with IBM, Kingston, N.Y.

References

Bentley, M. and Fischer, C. F. (1992). Hypercube conversion of serial codes for atomic structure calculations. *Parallel Computing*, 18:1023.

Davidson, E. R. (1975). The iterative calculation of a few of the lowest eigenvalues and corresponding eigenvectors of large real-symmetric matrices. *J. Comput. Phys.*, 17:87.

Douglas, C. C., Mattson, T. G., and Schultz, M. H. (1993). Parallel programming systems for workstation clusters. Technical Report YALEU/DCS/TR-975, Yale University.

Ewing, R. E., Sharpley, R. C., Mitchum, D., O'Leary, P., and Sockacki, J. S. (1993). Distributed computation of wave propagation models using PVM. In *Proceedings of Supercomputing '93 Conference*, page 22. ACM Press, Portland Kaufmann.

Fano, U. and Racah, G. (1959). *Irreducible Tensorial Sets.* Academic Press.

Fischer, C. F. (1977). *The Hartree-Fock Method for Atoms: A numerical approach.* J. Wiley & Sons, New York.

Fischer, C. F. (1986). Self-consistent-field (SCF) and multiconfiguration (MC) hartree-fock (HF) methods in atomic calculations: Numerical integration approaches. *Comput. Phys. Repts.*, 3:273.

Fischer, C. F. (1991a). A configuration interaction program. *Comput. Phys. Commun.*, 64:473.

Fischer, C. F. (1991b). The MCHF atomic structure package. *Comput. Phys. Commun.*, 64:369.

Fischer, C. F. and Liu, B. (1991). A program to generate configuration-state lists. *Comput. Phys. Commun.*, 64:406.

Fischer, C. F., Scott, N. S., and Yoo, J. (1988). Multitasking the calculation of angular integrals on the CRAY-2 and CRAY X-MP. *Parallel Computing*, 8:385.

Fischer, C. F., Smentek-Mielczarek, L., Vaeck, N., and Miecznik, G. (1993). A program to compute isotope shifts in atomic spectra. *Comput. Phys. Commun.*, 74:415.

Fischer, C. F., Tong, M., Bentley, M., Shen, Z., and Ravimohan, C. (1994). A distributed-memory implementation of the mchf atomic structure package. *The Journal of Supercomputing*, 8:117.

Geist, A., Beguelin, A., Dongarra, J., Jiang, W., Manchek, R., and Sunderam, V. (1993). PVM 3 users's guide and reference manual. Technical Report TM-12187, ORNL.

Hansen, J. E. (1990). Atomic spectra and oscillator strengths for astrophysics and fusion research. In *Proceedings of the Third International Colloquium*. North-Holland, Amsterdam.

Heath, M. T. and Etheridge, J. A. (1991). Visualizing the performance of parallel programs. *IEEE Software*, 8:29.

Hibbert, A. and Fischer, C. F. (1991). A program for computing angular integrals of the non-relativistic hamiltonian with non-orthogonal orbitals. *Comput. Phys. Commun.*, 64:417.

Jönsson, P., Wahlström, C.-G., and Fischer, C. F. (1993). A program for computing magnetic dipole and electric quadrupole hyperfine constants from mchf wavefunctions. *Comput. Phys. Commun.*, 74:399.

Morgan, R. B. and Scott, D. S. (1986). Generalizations of davidson's method for computing eigenvalues of sparse symmetric matrices. *SIAM J. Sci. Stat. Comput.*, 7:817.

Shen, Z. (1991). Symmetric, non-positive definite matrix decomposition on a hypercube multiprocessor. Master's thesis, Dept. of Comp. Sci., Vanderbilt University.

Stathopoulos, A. and Fischer, C. F. (1991). A hypercube implementation of davidson's algorithm for the large, sparse, symmetric eigenvalue problem. In *Intel Supercomputer Users' Group, 1991 Annual Users' Conference*, page 343.

Stathopoulos, A. and Fischer, C. F. (1993). Reducing synchronization on the parallel davidson method for the large, sparse, eigenvalue problem. In *Proceedings of Supercomputing '93 Conference*, page 172. ACM Press, Portland Kaufmann.

Stathopoulos, A. and Fischer, C. F. (1994). A davidson program for finding a few selected extreme eigenpairs of a large, sparse, real, symmetric matrix. *Comput. Phys. Commun.*, 79:268.

Stathopoulos, A., Ynnerman, A., and Fischer, C. F. (1994). PVM implementations of atomic structure programs. In *PVM Users' Group 1994 Meeting*, Oak Ridge, TN.

Sunderam, V. S. (1990). A framework for parallel distributed computing. *Concurrency: Practice and Experience*, 2:315.

Umar, V. M. and Fischer, C. F. (1989). Multitasking the davidson algorithm for the large, sparse, eigenvalue problem. *The International Journal of Supercomputer Applications*, 3:28.