Dynamic Load Balancing of Atomic Structure Programs on a PVM Cluster

Andreas Stathopoulos * and Anders Ynnerman *

Abstract

The MCHF package is a suite of programs that enable the calculation of atomic data required by many science and engineering disciplines. As a means of meeting its high computational demands, the package has previously been implemented in PVM. The codes have been used on a dedicated cluster of workstations with a static load balancing scheme. However, the cluster needs to be shared with other users, and different architecture workstations need to be embedded. In this paper, modifications of two well-known dynamic load balancing schemes are implemented and tested. The resulting codes exhibit perfect load balancing for a variety of system loads, facilitating the solution of large problems and the efficient utilization of current resources.

1 Introduction

Atomic data is needed by many areas of science and engineering. Model calculations for controlled thermonuclear fusion, geophysical studies of the atmosphere, laser research as well as astrophysics are such examples. 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 calculations for which the Multiconfiguration Hartree-Fock (MCHF) method [4] is one of the most powerful approaches. The MCHF package consists of a number of interrelated programs that yield the required atomic data.

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. In an earlier work, the MCHF package was ported to the Cray parallel-vector computer [5], and to iPSC/860 hypercube [1], but both implementations have their shortcomings. In [10], a Parallel Virtual Machine (PVM) implementation of the MCHF is described on a cluster of IBM/RS6000 workstations. PVM [6, 11] 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. The PVM implementation effectively resolves the limitations posed by both the hardware and software of previous efforts by exploiting the distributed resources and by a modification to the data structure. The resulting codes have already been used for large, break-through atomic structure calculations.

The PVM implementation employs a static load balancing scheme. Each processor is assigned a fixed number of columns of a Hamiltonian matrix, and throughout the package it works only on those local columns, communicating when necessary. Static load balancing on a homogeneous cluster of workstations can be very efficient in some situations. Homogeneity, however, is a very

*Computer Science Department, Box 1679-B, Vanderbilt University, Nashville, TN 37235. E-mail: andreas@vuse.vanderbilt.edu, ynnerman@vuse.vanderbilt.edu

restricting requirement, since the power of PVM computing relies in the integration of available (and possibly heterogeneous) resources. Static load balancing on heterogeneous machines can be detrimental to the application's efficiency because the slowest machine becomes the execution bottleneck. Heterogeneity can also be observed in homogeneous clusters when the individual machines have different loads. Since these loads may vary in time, a dynamic load balancing scheme is needed. The purpose of this paper is to provide a proper dynamic load balancing scheme for some of the time demanding programs of the MCHF package. Modifications of two well known schemes are presented and compared. As a result, the cluster of IBMs can be used effectively both as a dedicated and a general use system. Further, a large number of existing SUN workstations can be incorporated to the system for even larger and more accurate calculations.

The rest of the paper is organized as follows: Section 2 states the physical and the mathematical problem to be solved by MCHF. Section 3 discusses the reasons for the choice of static load balancing scheme in the previous PVM implementation. Section 4 describes the two dynamic load balancing strategies followed in this paper. Section 5 presents the timing results and conclusions.

2 The physical problem, methods and environment

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.) and the quantum mechanical formalism (non-relativistic, Dirac-Coulomb, Dirac-Breit, etc.). Here, we focus on atomic systems and the non-relativistic Schrödinger equation for which the Hamiltonian (in atomic units) is

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

Z is the nuclear charge of the atom with M electrons, and r_i, r_{ij} are the distances of the i_{th} electron from the nucleus and electron j respectively. Schrödinger's equation for atoms is among the simplest equations for many-electron systems. 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 and more accurate Hamiltonians need to be used.

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, Ψ_γ, for a state labeled γ is written as an expansion of N antisymmetrized configuration state functions (CSF), $\Phi(\gamma_i)$. Then $\Psi_\gamma = \sum_{i=1}^{N} c_i \Phi(\gamma_i;\ \{\mathbf{r}_j\})$, where $\{\mathbf{r}_j\}$ is a vector consisting of the spherical coordinates in three dimensional space, and the spin-space coordinate for all electrons, and γ represents any quantum numbers that are needed for complete specification of the state.

The $N \times N$ symmetric matrix $H = (\langle \Phi(\gamma_i) \mid \mathcal{H} \mid \Phi(\gamma_j)\rangle)_{ij}$, is called the interaction matrix. Applying the variational condition and requiring that the energy be stationary with respect to solution perturbations yields the matrix eigenvalue problem $(H - E)c = 0$. Thus the total energy is an eigenvalue of H, and the expansion coefficients of the wave function form the corresponding eigenvector. The CSF's, needed for computing H, are determined iteratively by the multi-configuration self-consistent field method. Using estimates of CSF's, expansion coefficients are obtained. Then, in turn, CSF's are updated so as to leave the energy stationary and a new interaction matrix is computed. This process is iterated until "self-consistency".

In the MCHF atomic structure package, the NONH program generates the angular coefficients and associated list of integrals that define the energy expression and the interaction matrix. If the radial parts of the CSF's are already determined, a BRCI program determines selected eigenvalues and eigenvectors. The problem of optimizing the CSF's for a particular state, is solved by the MCHF program which computes both the CSF's and the mixing coefficients.

3 Static load balancing

In the previous PVM implementation there is an option between two static load balancing schemes:

a. Interleaved (wrap-around) partitioning of the columns. Processor $me = 0, \ldots, p$ computes all columns i that satisfy $me = \mathrm{mod}(i, p)$, where p is the number of processors.
b. Partitioning of the matrix into p blocks of contingent columns.

Option a. is efficient if the matrix size is large, the non-zero elements are uniformly spread over the matrix, and the processors have the same computing power. In MCHF calculations the matrices have the above characteristics and the few homogeneous processors in the cluster have been used as a dedicated system. The tests performed in [10] have demonstrated excellent load balancing using the interleaved scheme. However, the system cannot be time or space shared with other users and available workstations cannot be efficiently incorporated in it.

Option b. is an intermediate solution to the above problems. Its efficiency depends strongly on the uniform spread of the non-zero elements and the invariance of the processor speed ratios. Provided these ratios, a routine computes the appropriate column blocks to be assigned to the processors so that each node has roughly the same number of elements from the lower triangular matrix. This strategy improves the load balancing over the interleaved scheme on heterogeneous clusters, but is not robust since it may perform poorly on load-free homogeneous clusters.

Although NONH and BRCI may benefit from dynamic load balancing because of their parallel nature, this is not employed in [10] for two reasons. First, the data structure used in MCHF is produced by NONH. A load balanced NONH does not necessarily imply the same for MCHF, since system load might change between runs. Second, additional communication and/or synchronization overheads would be involved. In the following section we address these problems.

4 Dynamic load balancing

In this paper the dynamic load balancing of only the BRCI program is considered. Many reasons suggest this first choice: First, BRCI is used for cases so large that MCHF cannot be used efficiently, and therefore a good load balancing is imperative. Second, the results are readily extendible to NONH which has an almost identical program structure. Third, the MCHF, as an inner-outer iterative procedure, has numerous synchronization points per iteration that diminish the effects of good load balancing. Therefore its performance is not expected to vary significantly with the different distributions provided by a dynamically balanced NONH.

The BRCI structure consists of two phases. The first, long phase is similar to a NONH pass with a loop going through the columns of the symmetric Hamiltonian matrix, creating all the non-zero elements below and on the diagonal. In the second phase an eigenvalue problem is solved using the Davidson iterative method [2, 9]. Columns should be distributed according to each processor's speed. It is common that the average load observed on the machines during the creation of the matrix carries on to the eigenvalue phase. Even if this is not true, the short duration of this phase and the above considerations related to MCHF make any imbalances insignificant.

The concern about increased communication overheads from dynamic load balancing of a loop (or *self-scheduling*) is valid only when the tasks to be scheduled have very short execution times. For the case of BRCI, each of the thousands of columns often takes several seconds to compute. A few milliseconds (cf [3]) for scheduling computation are almost negligible. To overcome synchronizations caused by distributed self-scheduling algorithms, the master-worker paradigm is adopted, where the master is a process residing on one of the processors. Many strategies exist in the literature for the master to allocate tasks to the workers. Single iteration scheduling, chunk scheduling, guided self-scheduling, factoring and trapezoid self-scheduling are the most popular ones [8, 7, 12]. In the following, two of these schemes are modified for scheduling columns from a symmetric matrix (i.e., only the lower triangular half).

4.1 Trapezoid Self-Scheduling

In chunk scheduling, the amount of total work is divided into a number of equal size chunks and the master assigns workers with the next current work chunk, as they become idle. However, this strategy requires a fine tuning of the chunk size, since a size that is too large or too small may result in poor load balancing or increased scheduling overhead correspondingly. The Trapezoid Self-Scheduling (TSS) [12] starts with a large chunk size c_0 and in each scheduling step it decreases it by s, until a lower threshold c_m is reached. If $s = (c_0 - c_m)/(m-1)$, the total number of steps is $m = \lceil 2N/(c_0 + c_m) \rceil$, where N is the number of iterations. The large chunks in the beginning reduce the total number of scheduling steps while the small chunks near the end smooth out any load imbalances between processes.

The first method tested for the BRCI is a hybrid of the above ones. The master schedules a number of columns onto the processes. However, the column size starts from N and reduces to 1 at the right end of the lower triangular matrix. If a constant chunk of columns is assigned at each step (chunk scheduling on columns), the scheme is equivalent to TSS applied on the non-zero elements of the lower triangular matrix. The implicit assumption which holds in MCHF is that non-zero elements are evenly spread over the matrix.

The size of the chunk should be chosen in a way so that the work at the end of the iterations is large enough not to cause communication bottlenecks. We propose a different choice of chunk size. The user defines the maximum allowable load imbalance δ, i.e., the difference in execution times between processors. An initial chunk size of 30 columns is assigned to each processor after the completion of which estimations of processor speeds are obtained. Using these estimates the master decides on c, the chunk size to be used, so that the last c columns are executed in about δ seconds. Since the non-zero elements involved are $c(c+1)/2$, c is the positive solution of:

$$c^2 + c - 2\delta \, \frac{30 \times \text{Average Column size}}{\text{Time for 30 columns}} = 0. \tag{3}$$

As a result, the number of scheduling steps is N/c. The parameter δ would be an upper bound on the load imbalance if the load of the processors did not change during execution. However δ, as an overestimation, is a good approximate bound to the imbalance.

4.2 Factoring

The TSS method may involve a large number of communication steps since the size of assigned work decreases only linearly. The Guided Self-Scheduling method (GSS) [8] attempts to solve this problem by exploiting the fact that only the last few steps smooth the imbalances effectively. GSS assigns $1/p$ of the remaining iterations to the next idle processor. However, GSS allocates too many iterations in the first few steps/processors and therefore it may yield poor load balancing (cf [7]). Factoring provides an alternative method since it still decreases the allocated iteration space exponentially, but not at every scheduling step. At each factoring step $1/f = 1/2$ of the remaining iterations are split into p equal chunks and these are scheduled from the master.

The second scheduling method tested for the BRCI is a variant of the factoring method with two generalizations. First, the factor f can be any number $f > 1$. In this case, it can be easily verified that for the k_{th} factoring step there are p allocations to idle processors, each of $\frac{N}{fp}\ (1 - \frac{1}{f})^{k-1}$ iterations. Notice that with this generalization, GSS is equivalent to factoring with $f = p$, and all p "inner" steps assigned to one processor. Second, as in TSS, our factoring method schedules according to the non-zero elements of the lower triangular matrix. The columns participating in the next factoring step should contain $1/f$ of the remaining non-zero elements. Again, the implicit assumption is that non-zero elements are evenly spread. If N_{old} is the size of the first column that has not been assigned yet, the number of columns c satisfying the above is:

$$c = \frac{1}{2} + N_{old} - \sqrt{(N_{old}^2 + N_{old})(1 - \frac{1}{f}) + \frac{1}{4}} \qquad \text{and} \qquad N_{old} \leftarrow N_{old} - c. \tag{4}$$

After c is computed, the master assigns c/p columns in each of p consecutive scheduling steps. Similarly to TSS, the chunk size c should not be permitted to become too small to avoid communication overheads. In the current factoring strategy the smallest size c is determined by equation (3) by specifying the maximum allowable load imbalance δ. In this way an approximate favorable bound on the imbalance of the method is ensured.

Assuming that no processor is assigned less than c_m columns, except probably at the last step, the number of scheduling steps for this factoring scheme can be shown to be:

$$M_{fact} \quad = \quad \left\lceil p\left(\frac{\log N - \log(pfc_m)}{\log f - \log(f-1)} + f\right)\right\rceil \tag{5}$$

This number is considerably less than N/c, the number of steps in TSS. Consequently, in a PVM network environment, factoring should result in communication benefits with minimal load imbalances since they can be adjusted through the parameter δ.

5 Results and Discussion

The above two schemes have been implemented and tested with the BRCI program. An implementation to the NONH is also straight-forward. To test the efficiency and robustness of the schemes, comparisons with the interleaved load balancing are provided in both the presence and the absence of external machine loads. In the tables' notation, a load noted as "2-2-2-1" signifies that three of the machines have an additional job executing (total of 2), while one machine executes only the PVM job. Total execution time and maximum time imbalance between processors are reported in seconds. The test case is a small to medium size calculation of the ground-state of the neutral Na atom. The list contains all double and single excitations when the single electron orbitals are limited to quantum numbers with $n \leq 11$ and $l \leq 5$. The size of the matrix is 7158.

In Table 1, a load "2-2-1-1" is assumed, in order to study the behavior of the schemes under various values of δ. TSS performs best when an approximate load imbalance of $\delta \geq 30$ secs is used. Requiring smaller imbalances results in a large number of scheduling steps which causes overhead. Factoring with $f = 2$ performs best when $\delta = 0.5$ secs since it assigns large chunks in the beginning and thus it can afford small chunk scheduling at the end. Factoring with $f = 4$ displays a similar character but with $\delta = 5$ secs, since the initial chunks are smaller and thus more steps are required. For large δ both factoring variants demonstrate the same behavior. The claim that δ is a good approximate bound for the imbalances is verified by all observed cases.

In Table 2, the three schemes have been considered with their optimal values of δ, and compared with interleaved load balancing across four different machine loads. As expected, interleaved partitioning performs well in the absence of external load. Otherwise, the slowest machine becomes the execution bottleneck. TSS improves the times considerably but it involves some overhead. Factoring with both $f = 2$ and $f = 4$ outperforms TSS for almost all loads. The gains are more pronounced in a heavily loaded cluster, i.e., when most of the machines are time-shared by two or more jobs. Factoring with $f = 4$ is better than $f = 2$ for a heavily loaded system, because the larger the processor imbalances, the more scheduling steps are required. The situation is reversed for a lightly loaded system. The dynamic load balancing schemes tested prevent processors from idling, as it is evident from the time ratios between different loads. Besides their efficiency, these schemes are robust since they reduce the execution time in the absence of external load.

With these dynamically balanced programs, exploitation of heterogeneous resources is feasible and new, break-through calculations can be performed.

References

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

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

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

[4] C. F. Fischer. The MCHF atomic structure package. *Comput. Phys. Commun.*, 64:369, 1991.

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

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

[7] S. F. Hummel, E. Schonberg, and L. E. Flynn. Factoring: A practical and robust method for scheduling parallel loops. In *Proceedings of Supercomputing '91 Conference*, page 610. IEEE Press, Los Alamitos, California, 1991.

[8] C. Polychronopoulos and D. Kuck. Guided self-scheduling: A practical scheduling scheme for parallel supercomputers. *IEEE Transactions on Computers*, 36:1425, 1987.

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

[10] A. Stathopoulos, A. Ynnerman, and C. F. Fischer. A pvm implementation of the mchf atomic structure package. *The International Journal of Supercomputer Applications and High Performance Computing*, submitted.

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

[12] T. H. Tzen and L. M. Ni. A practical scheduling scheme for parallel computers. *IEEE Transactions on parallel and distributed systems*, 4:87, 1993.

Table 1: Time and imbalance for TSS and Factoring on four processors with load "2-2-1-1", and for different values of δ and f. Table values are in seconds.

	TSS		Factoring f=2		Factoring f=4	
δ	Time	Imbal	Time	Imbal	Time	Imbal
0.5	2733	0.4	2677	0.8	2779	0.9
5	2740	1.8	2678	0.9	2687	1.1
30	2679	26.8	2730	6.9	2736	16.4
90	2700	14.5	2736	10.2	2734	3.2

Table 2: Time and imbalance for Interleaved, TSS (δ=30), Factoring (f=2, δ=0.5 secs) and (f=4, δ=5 secs) under different machine loads. Loads are characterized by the number of jobs on each processor. Table values are in seconds.

	Interleaved		TSS δ=30		Factoring f=2, δ=0.5		Factoring f=4, δ=5	
Load	Time	Imbal	Time	Imbal	Time	Imbal	Time	Imbal
1-1-1-1	2067	16	2067	3.6	2028	0.5	2048	1.4
2-2-1-1	4016	1932	2679	26.8	2677	0.8	2687	1.1
2-2-2-1	3998	1938	3392	3.6	3262	0.3	3204	3.0
3-2-2-1	6083	3974	3643	22.7	3495	0.5	3418	2.7