Parallel computing in Quantum Chemistry - message passing and beyond for a general ab initio program system

Hans Lischka and Holger Dachsel
Institut für Theoretische Chemie und Strahlenchemie, University of Vienna, Austria
Ron Shepard
Argonne National Laboratory, Argonne, Illinois
and
Robert J. Harrison
Pacific Northwest Laboratory, Richland, Washington

1. Introduction

One of the most prominent aims in Computational Chemistry is the modeling of chemical reactions and the prediction of molecular properties. Quantum chemical methods are used for the calculation of molecular structures, spectra, reaction energy profiles and many other interesting quantities. Nowadays, the accuracy of the theoretical calculations can compete to an increasing extent with the experimental one. Therefore, theoretical methods have become a very useful tool for the solution of many realistic chemical questions. The just described capabilities are not only of pure academic interest. Quantum chemical methods are also well established in the research laboratories of the chemical industry. There, they are used successfully for many routine applications. However, what is much more important, they provide a detailed source of information which helps in a better understanding of chemical processes - a knowledge which is crucial for a directed development of new classes of chemical compounds and materials. All of the computational methods used are extremely time consuming and rely heavily on the availablity of sufficient computer power. Parallel computing is the only way to open new dimensions in the field of the computer simulation of molecules.

A great variety of quantum chemical methods exist ranging from the standard Hartree-Fock theory to sophisticated electron correlation approaches. From a computational point of view all these methods require rather lengthy and complicated program codes (ten thousands to several hundreds of thousands of lines) and have to handle a large amount of data to be stored on external devices. In the simplest case, the Hartree-Fock (SCF) method, "direct" algorithms have eliminated the I/O and storage bottleneck and have opened the way to parallel implementations. For post-Hartree-Fock methods the situation is much more complicated as will be demonstrated below. Therefore, most of the previous attempts in parallelizing quantum chemical ab initio programs concentrated on SCF methods.

Starting with the pioneering work by Clementi and coworkers on "loosely coupled array of processors (LCAP)" [1] several investigations on the parallelization of SCF programs have been reported [2-11]. In addition, electron correlation methods based on Møller-Plesset Second Order Perturbation Theory [12], Coupled-Cluster theory [13,14] and full CI [15,16] have been considered as well. For reviews on the use massively parallel computers in Quantum Chemistry see e.g. [17,18].

DISCLAIMER

DISCLAIMER

Portions of this document may be illegible in electronic image products. Images are produced from the best available original document.

Our investigations presented here are a continuation of our previous work [19] on the the parallelization of the COLUMBUS program system [20,21]. The COLUMBUS program is based on the multireference single- and double-excitation configuration interaction (MRSDCI) approach. is very well portable and runs on a large variety of computers including numerous Unix-based workstations. VAX/VMS minicomputers, IBM mainframes and Cray supercomputers.

2. Quantum Chemical Methods

In the configuration interaction method (see e.g. [22]) the Ritz variation principle is used to solve the molecular Schrödinger equation

$$H\Psi = E\Psi \tag{1}$$

where H is the Hamiltonian operator describing the molecular system, Ψ is a many-electron wave function and E is the molecular energy. Expansion of Ψ into a linear combination of configuration state functions (CSFs)

$$\Psi = \sum_i c_i \Psi_i . \tag{2}$$

and application of the variation principle leads to the following matrix eigenvalue problem

$$\mathbf{Hc} = \tilde{E}\mathbf{c} . \tag{3}$$

$\mathbf{H}$ is the matrix represention of the Hamiltonian H in the basis of the CSFs, the vector $\mathbf{c}$ collects the coefficients c_i of Eq. (1) and $\tilde{E}$ is an approximation to the exact energy E. The many-electron functions Ψ_i are constructed from one-elctron functions (molecular orbitals. MOs) Φ_j according to the Pauli principle. The MOs in turn are expanded into a fixed basis χ_k (atomic orbitals. AOs) as

$$\Phi_j = \sum_k d_{kj}\chi_k . \tag{4}$$

The actual choice of CSFs and basis sets has a long tradition in Quantum Chemistry and need not be discussed here. In MRCI wave functions. the dimension of the matrix eigenvalue problem in Eq. (3) can easily reach hundreds of thousands or several millions. Because of the properties of the Hamiltonian operator the Hamiltonian matrix $\mathbf{H}$ is sparse. Without going into details. the following main steps have to be executed: calculation of one- and two-electron integrals in the atomic orbital (AO) basis, calculation of molecular orbitals (MOs) by means of a SCF or MCSCF procedure. transformation of the AO integrals to the MO basis and solution of the aforementioned eigenvalue problem. The last step is the most complicated and in many cases also by far the most time consuming one. For this reason we concentrated on it even though it is clear that. finally, all the aforementioned computational steps have to be parallelized.

In most cases one is only interested in a few of the lowest eigenvalues and eigenvectors in Eq. (3). They are usually obtained by a subspace expansion into a set of trial vectors according to Davidson [23]. In this method the eigenvector $\mathbf{c}$ is approximated by a vector $\mathbf{u}$ which is expanded into a linear combination of correction vectors $\mathbf{v}_i$

$$\mathbf{u} = \sum_{i=1}^{N} \alpha_i \mathbf{v}_i \tag{5}$$

N is the dimension of the subspace and the expansion coefficients α_i are determined from the small eigenvalue problem $\overline{\mathbf{H}}\alpha = \overline{E}\alpha$ where $\overline{H}_{ij} = \mathbf{v}_i^{\dagger}\mathbf{H}\mathbf{v}_j$. From the residuum $\mathbf{r} = (\mathbf{H} - \overline{E})\mathbf{u}$ a new expansion vector $\mathbf{v}_{N+1}$ is computed (for details see [23]) and an improved approximation $\mathbf{u}$ according to (5) is determined. This iteration scheme is completed until a certain convergence limit is reached. The most time consuming step is the calculation of the matrix-vector product

$$\mathbf{w}_i = \mathbf{H}\mathbf{v}_i. \qquad (6)$$

Because of the aforementioned sparsity of the matrix $\mathbf{H}$ the Hamiltonian matrix times vector product in Eq. (6) can be split into a series of dense matrix operations in such a way that $\mathbf{H}$ is never constructed explicitly (direct CI). The dense matrices are, however, of the dimension of the MO basis only (i.e. up to a few hundred at most). We use dense-matrix product kernels (e.g. BLAS(3) routines [24]) which have proven to be very efficient on vector and scalar pipelined computers. In this way efficiency and portability are achieved to a very large degree. The computational step shown in Eq. (6) is not only important in terms of CPU time but also because large amounts of data (two-electron integrals) have to be processed and because the whole logic of the computation of the respective contributions to $\mathbf{H}$ (formula tape) has to be done here.

3. Parallel algorithm

3.1. General considerations

Our strategy for parallelization was strongly guided by portability considerations. We wanted to have a program system which should work efficiently on shared memory and on distributed memory machines including workstation clusters as well. From the aforementioned small size of the dense matrices an attempt to parallelize at this matrix level was not very attractive. After analyzing various choices we decided for coarse grain parallelization at the topmost level in our program [19]. In order to do this, $\mathbf{v}$ and $\mathbf{w}$ are split into segments. The multiplication of the symmetric matrix $\mathbf{H}$ times the vector $\mathbf{v}$ (Eq. (6)) was originally written in our sequential program as loops over segments pairs in the following way:

```
DO SEG1 = 1 , NSEG
    READ v_SEG1 , w_SEG1
    DO SEG2 = 1 , SEG1
        READ v_SEG2 , w_SEG2
        UPDATE w_SEG1 , w_SEG2     ! Contributions from H_SEG1,SEG2
                                   and H_SEG2,SEG1 Hamiltonian blocks
        WRITE w_SEG2
    ENDDO
WRITE w_SEG1
ENDDO
```

The actual work - not shown in this scheme - is done in the routine UPDATE. Also not shown is the handling of the case SEG1 = SEG2 and various other special cases. In the parallel program the same loop structure as in the sequential case is used. To each process work for updating one segment pair is passed at a time and load

balancing is used to distribute the work evenly over all processes. The advantage of this scheme is that the routine UPDATE which comprises most of the total program code remains completely intact. Thus, we can still use the optimized dense matrix multiplication routines and it will be straightforward to make use of any further improvements which will be made in the sequential program. For this scheme message passing is adequate and straightforward to implement. The actual implementation is performed via the portable programming toolkit TCGMSG developed by one of us (RJH) [25]. TCGMSG supplies a set of Fortran and C callable library routines by which message passing can be introduced into the application program code. A Single Program Multiple Data approach is used.

3.2. First implementations

In our first implementation [19] only the sparse matrix vector vector product of Eq. (6) was parallelized. The vector v was kept on disk as one single file shared by all processes. Local copies of the update vector w were held for each process and were added up via a global sum operation after completion of the loop over all segment pairs. The Davidson step was not parallelized at all. This version was installed on a variety of parallel computers like the Alliant FX/2800, the CRAY Y-MP, the Convex C2 and the Intel iPSC/860. The program worked very well on the Alliant, CRAY and Convex. E.g., on a dedicated 8 processor CRAY Y-MP roughly one GFLOP per wall clock second was achieved. On the iPSC/860 a serious degradation in performance was observed already with 7 processors due to the slow data transfer rate to and from disk.

In any case, we could show that our overall approach of defining tasks by segmenting the v and w vectors was successful. Thus, in the next step efforts were made to reduce the I/O while changing the basic outline of the program as little as possible. This goal was achieved by introducing two features: a) the concept of a local virtual disk and b) by developing a data compression scheme for the v and w vectors. The virtual disk gave us a flexible tool to store files in the local central memory of each processor. By means of the data compression the size of the v and w vectors were reduced by factors of four to five. Obviously, best results could be achieved when all data could be kept in core.

This second program version was tested on the Intel Touchstone Delta and and the IBM SP1. Now, the performance was very good for up to about 32 processors. Almost 100% efficiency could be achieved for the H·v step. However, we still had the bottleneck to distribute the v vector for each iteration to all processors and to perform a global sum for w at the end of each iteration. Also, the Davidson iteration was still not performed in parallel. From our experience gained so far it was clear that we needed more flexibility in the data organization. For larger calculations we could not keep identical copies of all files in the memory of each processor as would have been necessary for an optimal calculation. What we needed was the possibility to distribute the contents of a file globally over the memory of all processors and to allow all compute processes asynchronous access to these data. Again, as it has already been stressed above, portability was a crucial requirement for such software tools.

3.3. Global arrays

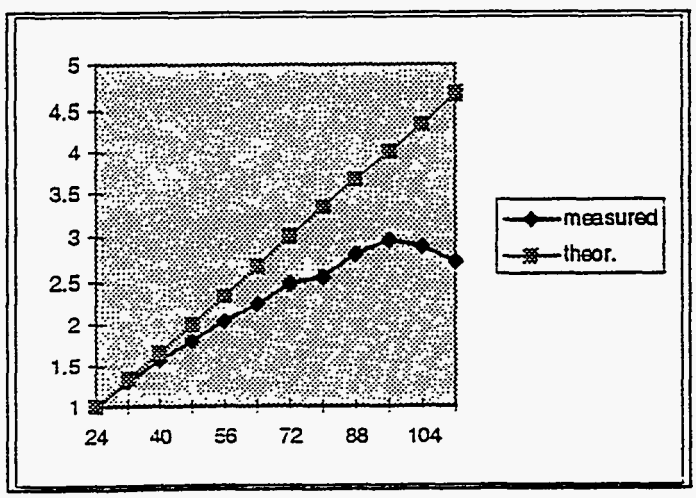
The global-array tools [26] which we were using can be characterized in the following way: these tools support one-sided access to data structures (here limited to one- and two-dimensional arrays) in the spirit of shared memory. With some effort this can be done portably, and in return for this investment we gain a much easier programming environment, speeding code development and improving extensibility and maintainability. We also gain a significant performance enhancement from increased asynchrony of execution of processes. The tools efficiently support both task and data parallelism.

By means of the global-array tools all major files (v and w, two-electron integrals) were now stored as a single copy. In particular, reading of the v vector and accumulating the contributions to the w vector could be done asynchronously by the different processes as needed. No overall distribution step at the beginning of an iteration and no collection of results at the end of each iteration is necessary anymore. It was also straightforward to parallelize the Davidson procedure.

3.4. Benchmark calculations

At present, the program works on the Intel Touchstone Delta and on the IBM SP1. Benchmark tests (see below) were taken on the Delta, those on the SP1 will follow shortly. As test example we used a C_{2v}-*pVTZ* calculation on the CH_3 molecule as given in full detail in Ref. [19]. The dimension of the CI expansion is 624 334 CSFs. The number of segments was held constant at 24 giving 481 segment pairs (including subdivisions of certain pair types) in total. Calculations with up to 112 processors were performed.

In Fig. 1 for one typical iteration the speedup with the number of processors is shown and compared to the theoretical value.



The results are very satisfactory up to about 64 processors and then start to deteriorate. Beginning with 96 processors there is no increase in the speedup anymore. The main reason for this retardation is the fact that the load balancing

mechanism is not so efficient anymory because only a few segment pairs are available for each processor. Moreover. also other events which were not relevant before now increase in relative importance.

4. Conclusions and outlook

We regard it as a great success that we can run the CI section of the COLUMBUS program system - which incorporates all the complexity of the MRSDCI method - efficiently on a distributed memory system like the Delta. Investigations on the SP1 machine are in progress. Our next main step will be the introduction of a "double direct" approach which avoids the storage and sorting of the 3- and 4-external two-electron integrals. A sequential. AO driven code is already available [27]. We are confident that with this new features included we will be able to run our program efficiently on several hundred processors.

Acknowledgments

This work was performed under the auspices of the Austrian "Fonds zur Förderung der wissenschaftlichen Forschung", project nr. P9032 and the High Performance Computing and Communication Program of the Office of Scientific Computing, U.S. Department of Energy. The calculations on the Intel Touchstone Delta were performed at the CCSF at Caltech. those on the IBM SP1 at the ACRF of the Argonne National Laboratory. We are grateful for the competent support of our work by these computer centers.

References

1 E. Clementi. in *Modern Techniques in Computational Chemistry*, chap. 1. E. Clementi. Ed.. Escom Science Publishers. 1990: D. Folsom. in *Modern Techniques in Computational Chemistry*, chap. 27. E. Clementi, Ed., Escom Science Publishers. 1990

2 M. Dupuis and J. D. Watts. Theor. Chim. Acta **71** (1987) 91

3 R. J. Harrison and R. A. Kendall. Theor. Chim. Acta **79** (1991) 337

4 H. P. Lüthi. J. E. Mertz. M. W. Feyereisen and J. E. Almlöf. J. Comp. Chem. 13 (1992) 160

5 S. Kindermann. E. Michel and P. Otto. J. Comp. Chem. 13 (1992) 414

6 M. W. Feyereisen. R. A. Kendall. J. Nichols, D. Dame and J. T. Golab. J. Comp. Chem. 14 (1993) 818

7 S. Brode. H. Horn. M. Ehrig, D. Moldrup, J. E. Rice and R. Ahlrichs, J. Comp. Chem. **14** (1993) 1142

8 M. W. Schmidt. K. K. Baldridge, J. A. Boatz. S. T. Elbert. M. S. Gordon. J. H. Jensen. S. Koseki. N. Matsunaga, K. A. Nguyen, S. Su. T. L. Windus. M. Dupuis and J. A. Montgomery. Jr., J. Comp. Chem. **14** (1993) 1347

9 M. E. Colvin. C. L. Janssen. R. A. Whiteside and C. H. Tong, Theor. Chim. Acta **84** (1993) 301

10 L. G. M. Petterson and T. Faxen. Theor. Chim. Acta **85** (1993) 345

11 A. Burkhardt. U. Wedig and H. G. v. Schnering. Theor. Chim. Acta **86** (1993) 497

12 J. D. Watts and M. Dupuis. J. Comp. Chemistry 9 (1988) 158