Theor Chim Acta (1993) 86: 497–510

Theoretica Chimica Acta

SCF calculations on MIMD type parallel computers

A. Burkhardt, U. Wedig, and H. G. v. Schnering
Max-Planck-Institut für Festkörperforschung, Heisenbergstr. 1, D-70569 Stuttgart, Germany

Received May 11, 1992/Accepted April 27, 1993

Summary. One of the key methods in quantum chemistry, the Hartree–Fock SCF method, is performing poorly on typical vector supercomputers. A significant acceleration of calculations of this type requires the development and implementation of a parallel SCF algorithm. In this paper various parallelization strategies are discussed comparing local and global communication management as well as sequential and distributed Fock-matrix updates. Programs based on these algorithms are bench marked on transputer networks and two IBM MIMD prototypes. The portability of the code is demonstrated with the portation of the initial Helios version to other operating systems like Parallel VM/SP and PARIX. Based on the PVM libraries, a platform-independent version has been developed for heterogeneous workstation clusters as well as for massively parallel computers.

Key words: Parallel – MIMD – SCF – Massively parallel computers

1 Introduction

The rapid development of more and more powerful computers is a prerequisite for the successful application of quantum theory on various chemical systems. Therefore, it is not surprising that vector supercomputers dominated this area in the last decade. During this period, however, the peak performance of vector supercomputers was only tripled, whereas microprocessor performance grew by several orders of magnitude. This stagnation in the supercomputer area led to the concept of parallel computers consisting of several processors. Based on this idea a variety of different architectures (shared memory, distributed memory) and concepts (SIMD, MIMD) for parallel computers have been proposed. The best suited concept for large scale numerical calculations is still under discussion. The trend, however, goes towards MIMD-type parallel computers with distributed memory, since only this architecture offers sufficient scalability. The transputer chips developed by INMOS combine a RISC processor with floating point unit and fast communication hardware [1]. Therefore transputer networks may act as prototypes for MIMD type parallel computers.

One of the key methods in quantum chemistry, the Hartree–Fock SCF method, is performing poorly on typical vector supercomputers. A significant acceleration of calculations of this type requires the parallelization of the SCF algorithm and the implementation on a powerful parallel computer.

2 Parallelization strategies

The Hartree Fock approximation describes an electron in the field of the nuclei and an averaged field of all other electrons. The so-called Fock-matrix contains all these interactions. The Fock-matrix is constructed from the one-electron-integrals h and the two-electron-integrals g in conjuction with the one particle density matrix P:

$$F_{pq} = h_{pq} + \sum_{r}^{n}\sum_{s}^{n} P_{rs}(2g_{qrps} - g_{qrsp}).$$

The most time-consuming step in this process is the calculation of the two-electron integrals and the associated Fock-matrix update. Once $\boldsymbol{F}$ is constructed, the generalized eigenvalue problem:

$$\boldsymbol{FC} = e\boldsymbol{SC}.$$

must be solved, where $\boldsymbol{S}$ denotes the overlap matrix. With the new eigenvectors $\boldsymbol{C}$ a new density matrix is constructed and the process is repeated until the differences between two iterations are sufficiently small. In the conventional or indirect SCF algorithm the two-electron-integrals are computed once and stored on mass storage devices. Therefore, available disk space and I/O bandwith are limiting factors for conventional SCF calculations. These limitations are overcome in the direct SCF algorithm [2, 3], where the two-electron-integrals are recalculated each iteration. The price, however, is a significantly increased computational effort.

The parallelization of the two-electron-integral evaluation is the critical step towards a parallel direct SCF-program, since this is the most time consuming step for the direct SCF process. The calculation of different integrals is independent and therefore distributable to several processors. The use of integral batches as smallest job item allows the efficient use of intermediate results and turns out to be advantageous over the use of single integrals.

The computational effort for the calculation of one integral batch depends strongly on the l-values of the associated basis functions. This wide granularity range makes a predetermined distribution of tasks difficult. The farming concept introduced by Hey promises a better load balance under these circumstances [4, 5]. Farming is a synonym for dynamic load balancing based on a random distribution of tasks using the same algorithm but different input data (SPMD – Single Program Multiple Data). One client process generates the job items and distributes them randomly to the server tasks. The application of the farming concept is restricted to those cases where the number of data sets significantly exceeds the number of nodes. As outlined below, this constraint will become important for relatively small calculations and large processor numbers.

The dynamic farming strategy for the parallel integral evaluation avoids the problems other groups report with deterministic distribution approaches [6, 7, 8]. There are, however, still several choices for communication management:

- *Global communication management*: One distinct process (Load Balancer) distributes the jobs to all server processes. This process receives the results of all server processes and schedules new jobs to the idle processes.
- *Local communication management*: Each server task decides whether to process a given job or whether to sent it to another server task.

Another decision to be made concerns the processing of the calculated integrals:

- *Sequential Fock-matrix update*: All calculated integrals are returned to the client task. The client task evaluates the Fock matrix using the integrals and the density matrix.

- *Distributed Fock-matrix update*: Each server task receives the actual density matrix and builds its own partial Fock-matrix from the calculated integrals.

In the next section we will describe the implementation and performance of a direct SCF program on a transputer network and another MIMD computer. All four possible combinations of the communication strategies outlined above have been studied, using two benchmark molecules. The first benchmark is the calculation of the two-electron integrals and the duration of one complete SCF iteration for trans-formic acid using a DZP basis set (58 basis functions). For powerful networks and scalar computers P_4S_3, again with a DZP basis set (168 basis functions) or even a DZ2P basis set (210 basis functions), is a more adequate benchmark.

3 The Helios implementation

At the beginning of the project (fall 1988), the INMOS Transputer Development System (TDS) was the one and only development environment available for transputer networks [9, 10]. All parallel processes had to be programmed in OCCAM [11]. However, the inclusion of FORTRAN subroutines was possible. Under TDS the programmer is responsible for the communication and the distribution of the tasks. Operating system functions like access to external mass storage, keyboard or screen are not available on the server nodes. As a consequence, any program running under TDS has the complete control of the arithmetic and communication facilities and the interprocessor communication is handled very efficiently. Starting from existing FORTRAN integral routines [12] we developed a distributed direct SCF program using OCCAM as programming language. Cooper and Hiller reported recently about a similar approach using the MEIKO computing surface [13]. In addition to the farming concept we used a sequential Fock-matrix update and a local communication management based on bidirectional pipelines. Computational details and the performance of this program version have been described in [14]. Although the interprocessor communication was heavy, this DSCF program performed surprisingly well. We measured a worst-case speedup of 7.1 for a complete SCF iteration using eight processors. This corresponds to an overall efficiency of 90 percent. Despite these good results, the program development under TDS turned out to be a dead end. The language OCCAM was not accepted as a standard for parallel programming. The majority of programmers in the scientific community continued to develop programs in FORTRAN. Since the integration of new developments is a crucial point for scientific applications, we looked for a way to develop parallel programs exclusively or at least mainly in FORTRAN.

The distributed operating system Helios [15] promised the program development with conventional languages. Under Helios, any sequential part of a parallel program is called a 'task'. The parallelism of the program as a whole is not defined within the tasks, but with a meta-language. This 'Component Distribution Language' (CDL) is derived from the pipelining familiar to UNIX users. CDL,

however, offers not only constructors for unidirectional pipelining, but also bidirectional pipelining and even a constructor for a complete farm. The routines for the inter-task communication and process synchronization can be written in C, where the required extensions are part of library. Programs written this way are hardware independent, since HELIOS is mapping the virtual streams to the actual network topology. The Helios command form a subset of UNIX and the libraries follow the POSIX standard. Our first experiences with Helios are described in [16].

3.1 Global communication management and sequential Fock-matrix update

Since farming was supported by CDL, the portation of the TDS-based program was supposed to be easy. CDL farms, however, use a global communication management. One process, called 'Load Balancer', handles all communication between client and server processes. Furthermore, all server processes are directly connected to the Load Balancer. Since the Load Balancer shipped with the Helios versions 1.1 and 1.15 did not meet the specifications in the documentation, we were forced to develop a new Load Balancer (lb). The internal structure of the lb task is shown in Fig. 1. All processes shown are executed quasiparallely on one node. The *Read–Client* process is storing data items from the client process into the input buffer. One of the *Send-to-Server* processes associated with each server reads the item and transfers it to a server task. The results – in our case the calculated integrals – are read by the associated *Receive-from-Server* process and stored into the output buffer. Finally, the *Write–Client* process transfers the contents of the output buffer back to the client task. To ensure the validity of the buffer contents, the access to both buffers is controlled with semaphores.

The OCCAM-DSCF program described in [14] communicates single integrals and batch-index quadruples. Using the same approach under Helios leads to disaster. A network of 14 processors is not significantly faster than a single

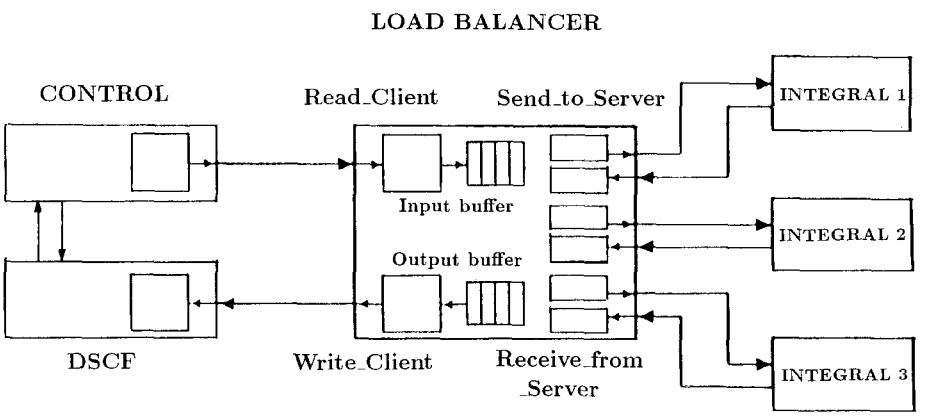

Fig. 1. Helios DSCF-farm with, for example, 3 servers. The load balancer task consists of several processes running concurrently on one network node. The function of each process is described in the text

processor. The reason for the failure of this fine granular communication is the presence of the operating system. The Helios kernel is situated between application and hardware. This additional software layer causes significantly longer startup times for the interprocessor communication. In this case, coarse granular communication like the transmission of a few large data items is favorable. As a consequence, the next program version transmitted packets of batch indices and packets of integrals. Using an optimized packet size for batch and integral packets the speedup of the calculation of the two-electron-integrals reached ten using 14 server processors. The speedup changes with varying packet size are instructive. Rising the number of integrals in the packets returned to the client increases the speedup, since the influence of the startup time is minimized. Rising the number of batch index quadruples in the packets results in a sharp rise of the speedup for up to five quadruples. Any further increase results in a slow decline of the overall speedup. At first glance this behaviour is surprising, since the use of larger packets minimizes the interprocessor communication. Using larger job packets, however, decreases the absolute number of jobs. Therefore the overall load balance of the farm is reduced, since – as mentioned above – in efficient farming applications, the number of jobs must significantly exceed the number of server tasks.

Even with optimized packet sizes, the average processor utilization of this DSCF version is only about 70% for 14 nodes. This fact leads to the conclusions that global communication management is a bottleneck for any network with relatively slow interprocessor communication. As a consequence we implemented an alternate farming mechanism under Helios based on bidirectional pipelines, described in the following section. During the portation of the program to the IBM PPCS, however, the global communication management is retested in a more suitable hard- and software environment.

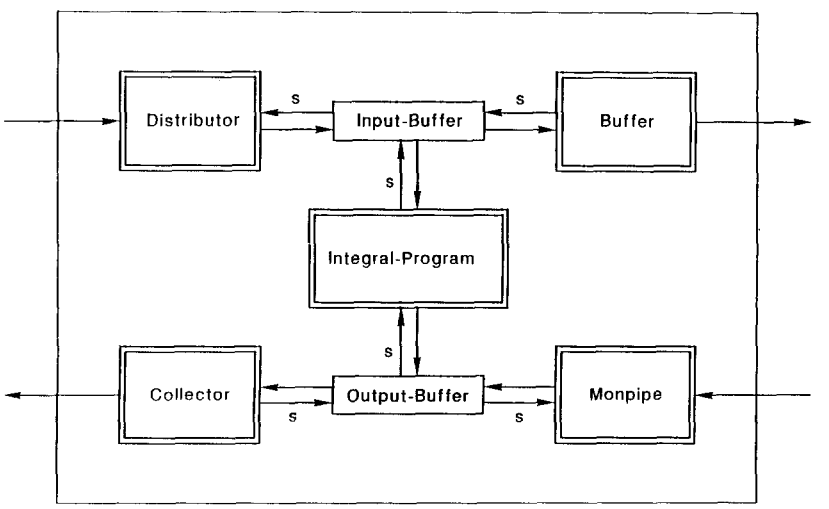

Fig. 2. One server task in the Helios pipeline-DSCF-farm: All processes shown are executed quasiparallely on each server node. The interaction of the processes is described in the text

3.2 Local communication management and sequential Fock-Matrix update

A farming approach based on bidirectional pipelines should distribute the communication management more evenly within the network. Therefore, a set of augmenting subroutines for the local communication management has been developed. Figure 2 shows all processes executed quasiparallely within one server task. The basic idea is similar to the OCCAM program version. Data exchange between the processes, however, is implemented as access to shared local memory, the so-called input and output buffers. The buffer access is, as in the case of the *lb*-version, guarded by semaphores. An index packet from the client task or the previous server task is read by the *Distributor* process and written into the input buffer. Whether the packet is processed by this server task or sent to the next task in the pipeline depends on the next read request to the input buffer. If the request is issued by the *Integral Program*, then the integrals of the batch are calculated by this server task. If, however, the *Buffer* process issues the request first, then the packet is sent further down the pipeline. Any integral packets from the next server processes in the pipeline are read by the *Monpipe* process and placed into the output buffer.

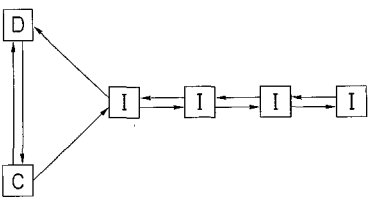

Fig. 3. A DSCF-farm based on a bidirectional pipeline. The C-task generates the index-tuples, the D-task performs the scalar program parts, the I-tasks are calculating the integral batches

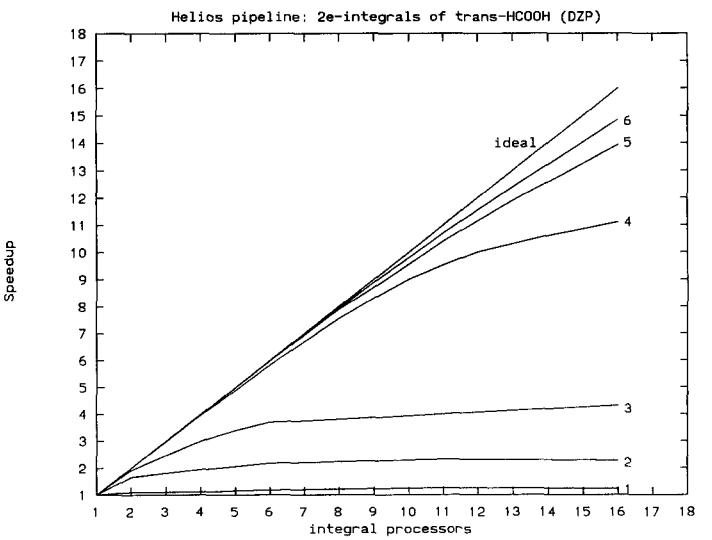

Fig. 4. Optimization of the communication for the Helios pipeline version. With increasing integral packet and medium index packet size the throughput increases dramatically (details and explanations are given in the text)

The integrals calculated locally are also written into this buffer. The *Collector* process sends the collected integrals in the output buffer towards the client task. Figure 3 shows the general scheme of the resulting distributed DSCF program, where the **C** and **D** processes together form the client task and the integral calculating server tasks are denoted by **I**. Each of the server tasks consists of the five concurrent processes described above.

In the following we will investigate the performance of this approach. For the same reasons as in the *lb*-version, the total throughput of the program during the HCOOH benchmark remains almost constant, if (as shown in Fig. 4, curve 1) single index quadruples and integrals are transmitted. Assembling larger index packets results in a slight performance increase (Fig. 4, curve 2). Doing the same with the calculated integrals, the largest part of the inter-task communication, increases the speedup significantly. As in the *lb*-version, the speedup increases with increasing packet size (Fig. 4, curves 3, 4 and 5). Using the optimal packet sizes, the average processor utilization for the calculation of the two-electron integrals of HCOOH reaches 87% (speedup of 14) with 16 server nodes. The average

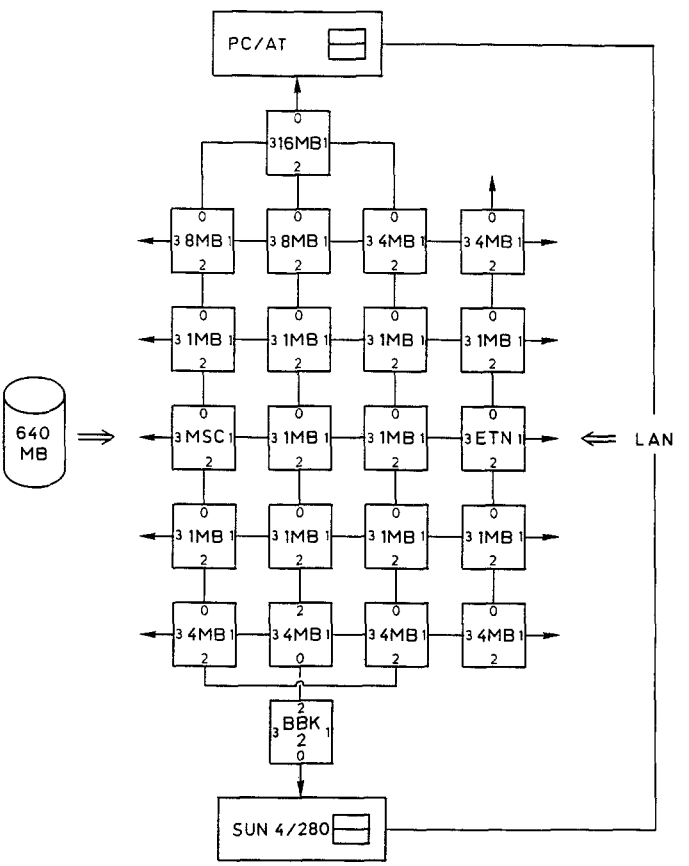

Fig. 5. The topology of the 22-transputer-network at the MPI-FKF. The nodes form a torus, disturbed by the connections to the host machines

communication distance between client and server tasks is shortened, if several pipelines are used. We tested program versions with 16 server tasks aligned in up to five pipelines and found additional enhancement to the speedup (14.8, Fig. 4, curve 6) and the average processor utilization (92.5%).

However, on a network of 22 T800 transputers (topology shown in Fig. 5), we observed a distinct deviation from quasilinear speedup for more than 16 processors. To ensure the scalability of the program, we ran the HCOOH benchmark on a PARSYTEC Supercluster with variable topology consisting of 64 T805 transputers. All tested program versions, independent of the number of pipelines, showed no throughput increase above 16 to 20 processors. Since this saturation remains almost unchanged by variations of the communication management, the algorithm itself had to be reanalyzed for bottlenecks. The update of the Fock-matrix, which to date had been performed sequentially by the client task, turned out to be that bottleneck. If the effort to generate the Fock-matrix elements resulting from one integral is, e.g., five percent of the effort to calculate the integral, the number of efficiently usable servers is limited to twenty. Above this limit the integral calculation is faster than the Fock-matrix update and therefore some processors are idling, because they cannot return their results immediately.

3.3 Local communication management and distributed Fock-Matrix update

Due to the above considerations, the algorithm of our distributed DSCF program had to be changed. In the new version, each server node builds its own partial Fock-matrix. At the start of the iteration, the density matrix is broadcasted to all server tasks. From the density matrix and the calculated integrals each server builds a partial Fock-matrix. In a synchronization step after the calculation of all two-electron integrals the client process adds up all partial Fock-matrices. The resulting flowchart of the new distributed DSCF program is shown in Fig. 6. Maintaining the pipeline concept, program versions with up to four pipelines have been developed. Routines for the transmission of complete matrices were added to the pipeline routines described above. The resulting subroutine package provides basically the same functions as the set of routines developed by Harrison [17, 18] for shared memory machines. Similar routines based on the TCP/IP-Protocol were used successfully for large molecules by Lüthi [19] on a wide area network of Cray-processors and Brode [20] on a cluster of ethernet coupled workstations.

On our relatively small transputer network the new version showed linear speedup for the two-electron integral calculation. Therefore the benchmarks were continued on a PARSYTEC Supercluster with 64 processors. With the HCOOH benchmark used to date, quasilinear speedup for up to thirty nodes has been observed. Above 48 processors the speedup stagnated, since – as described above – this example is too small for large networks. With the larger P_4S_3 benchmark, however, we observed nearly linear speedup for up to 48 nodes using four pipelines. The PARSYTEC Supercluster used in this investigation was heterogeneous, containing processors of different clock rate and hardware error correction. Therefore, an effective processor number had to be calculated. This number means the number of 25 MHz T800's equivalent to the actual network. Using 41.7 effective processors, the speedup for the calculation of the two-electron integrals was 39. This is equivalent to an average processor utilization of 93.5%. The speedup factor for one complete SCF iteration on 48 processors is reduced due to the scalar program parts (Amdahl's Law) to 29.1, corresponding to an overall processor utilization of 69.8%.

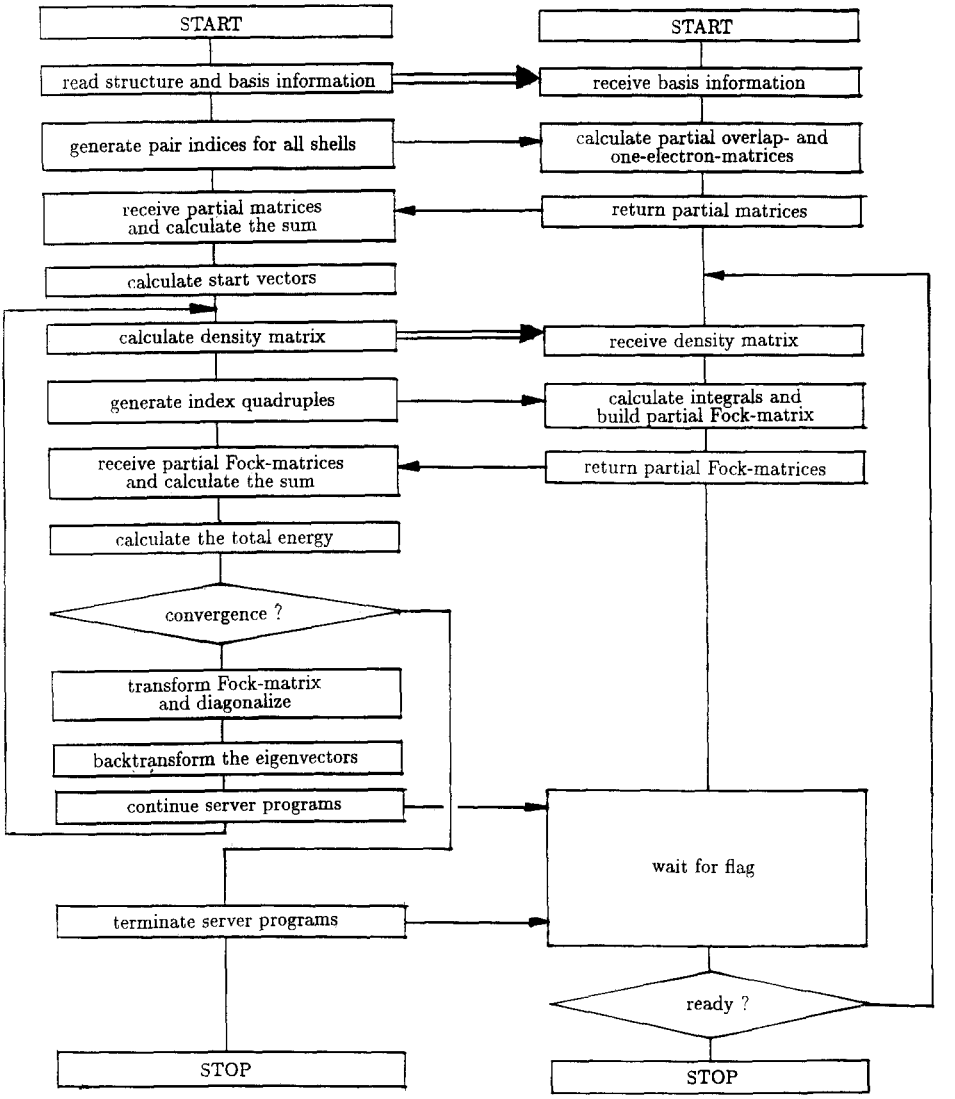

Fig. 6. The flow chart of an distributed SCF-program with distributed Fock-matrix update

4 The IBM PPCS implementation

4.1 Intention

To prove the portability of the programs and algorithms described in the last chapter, we adapted the program to the Parallel Processing Compute Server (PPCS), an experimental MIMD computer developed by IBM. We tested the program on two PPCS versions. The first version consisted of 16 IBM/370 processors – the so-called 'Satellites' – connected by a VME bus with an additional/370

Table 1. SCF iteration and 2*e*-integral calculation for P_4S_3 (DZP) on the IBM PPCS

		SCF iteration		Two-electron integrals		
Packet size	Processor number and type	Satellite utilization	Speedup factor	Satellite utilization	Speedup factor	Average execution time
100	16/370	61.73%	9.87	64.32%	10.29	5035.34
500	16/370	89.62%	14.33	95.15%	15.22	3481.55
2000	16/370	93.03%	14.89	99.10%	15.86	3283.36
2000	32/390	84.88%	27.16	93.19%	29.82	374.79
8000	32/390	87.63%	28.04	97.59%	31.22	363.53

processor, the so-called 'I-Host'. The second version consisted of 32 IBM/390 satellite processors and a /390-I-Host, connected by a fast crosspoint switch. The operating system was a modified diskless VM/SP and the programming language was 'Distributed VM/FORTRAN' with extensions for parallel programming (CS/L-Library) [21]. The interprocessor communication is managed exclusively by the I-Host node. The data to be exchanged is copied to and from COMMON blocks in the I-Host and the addressed satellite.

4.2 Global communication management and distributed Fock-matrix update

The communication routines had to be reformulated using the subroutines of the CS/L-library. Since both PPCS versions offer sufficient local memory on the satellite processors, we implemented the distributed Fock-matrix update. Due to the fact that only the I-Host processor is controlling the communication we had to use global communication management. The adaption of the Helios program to the PPCS was finished within 40 man-hours. It is noteworthy that the PPCS version was written entirely in FORTRAN, mixed language programming as for Helios was not necessary. On the two PPCS versions we ran the benchmark calculations described earlier (HCOOH and P_4S_3). The formic acid benchmark turned out to be to small for the PPCS versions, the number of jobs was – as in the 64 transputer network – to small to allow efficient farming. The results of the P_4S_3 calculation, which are summarized in Table 1, show the excellent performance of our DSCF program on the IBM-PPCS.

5 Portability and further benchmarks

From the point of view of a conventional programmer the poor portability of distributed codes is the main disadvantage of the available massively parallel systems. Our experience, however, is that once the algorithmic work – namely the proper division into sequential program modules – is done, porting the code to various distributed memory computers is relatively easy. Aside from the previously mentioned PPCS version we have also created a program version for

the UNIX-like PARIX Operating system [22] and a hardware independent version based on PVM (Parallel Virtual Machine) [23]. PVM is a library with communication primitives for distributed programs which is available for heterogeneous workstation clusters as well as for massively parallel computers. All these conversions were finished in less than 3 days. The resulting distributed programs are still scalable up to the theoretical limit set by Amdahls law. On a 128-processor Parsytec GCel parallel computer a test calculation (P_4S_3 DZ2P with 210 basis functions) achieved an efficiency of 80.4% for the calculation of the 2-electron-integrals. This is equivalent to a speedup of 102. The execution times of the sequential parts of the SCF cycle for this configuration are of the same magnitude, reducing the overall efficiency to 44.2%.

To exploit the limits imposed on the problem size by the lack of virtual memory management on the transputer nodes we performed a test calculation for a large organic molecule. Restricted to 4 MBytes of local memory we were able to perform a full SCF-calculation for the rhamnolipide $C_{32}H_{58}O_{13}$ with a split-valence basis set on carbon and oxygen and a STO-3G basis set on hydrogen, summing up to 463 basis functions. Using nine integral nodes instead of one reduced the execution time from 39 days to less than 5 days, due to a speedup of 8.7 for the 2*e*-integral calculation.

6 Summary

In the previous sections we have described the development of a direct SCF program for several distributed operating systems and discussed the various

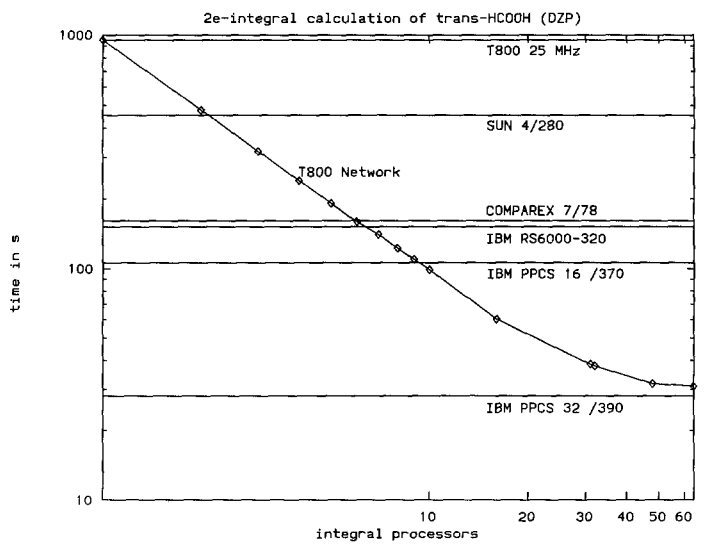

Fig. 7. Measured execution times for the calculation of the 2*e*-integrals of *trans*-HCOOH (DZP basis set) compared to the CPU times of various scalar computers. In the PARAMOLE program the MELD integral routines used in the PARAMELD version are augmented by the fast Obara-Saika integral routines. SEQUIMOLE is a scalar version of this program

parallelization strategies. While global communication management combined with a distributed Fock-matrix update was successful on the IBM PPCS, global communication combined with sequential Fock-matrix update turned out to be less favourable (Helios Load Balancer version). The speedup achievable this way is limited since the Load Balancer task is a bottleneck for the extensive communication required. As a consequence, we developed subroutines for local communication management for Helios and PARIX based on bidirectional pipelines. Retaining the sequential Fock-matrix update, these versions use up to 16 integral nodes efficiently and require only 1MB local memory on the integral nodes. To raise the processor number limit a new algorithm was implemented using a distributed Fock-matrix update. This version was tested on up to 128 transputers and shown to use them efficiently. Since two matrices must be held in local memory on each server processor, the program is limited to 470 basis functions, if 4 MB local memory are available. The results of the detailed benchmark calculations for this version in comparison to various workstations and supercomputers is shown in Fig. 7 and Fig. 8 and the underlying CPU- and execution times are summarized in Tables 2 and 3.

To prove the portability, the program has been adapted to the transputer operating system PARIX, Parallel VM/SP and to the PVM library, thus extending the possible hardware platforms from transputer networks to heterogeneous clusters of workstations and mainframes. The PPCS program portation (5000 lines of FORTRAN code) was successfully finished within 40 man-hours. All subsequent transformations needed less than 3 days. This fact demonstrates the value of transputer networks to develop software for MIMD computers.

The performance of all versions is increasing with growing packet size. Therefore the program is well suited for challenging SCF calculations with several hundreds of basis functions.

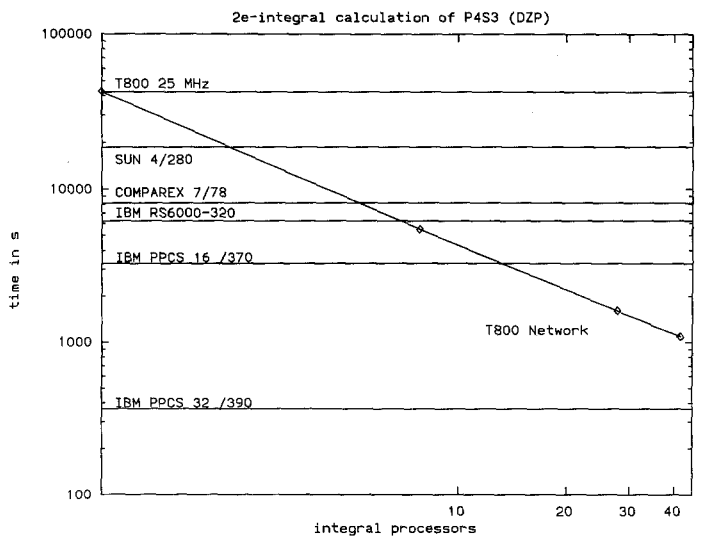

Fig. 8. Measured execution times for the calculation of the two-electron-integrals of P_4S_3 (DZP basis set) compared to the CPU times of various scalar computers

Table 2. Measured execution times for the calculation of the 2*e*-integrals of HCOOH (DZP) compared to the CPU times of various scalar computers

Computer	Program	Time	
MicroVAX II	MELD	1948.0	a)
Comparex 7/78	MELD	125.0	a)
CRAY-X/MP (1 processor)	MELD	76.0	a)
i386/387 25 MHz	SEQUIMOLE	3183.5	a)
T800 25 MHz	SEQUIMOLE	951.6	a)
SUN 4/280	SEQUIMOLE	458.2	a)
COMPAREX 7/78	SEQUIMOLE	161.1	a)
ESV-3 (MIPS 3000)	SEQUIMOLE	154.6	a)
IBM-RS/6000-320	SEQUIMOLE	151.8	a)
IBM PPCS 16/370	PARAMOLE2	106.1	b)
IBM PPCS 32/390	PARAMOLE2	21.7	b)
1 T800 integral node	PARAMELD1	1217.0	b)
11 T800 integral nodes	PARAMELD1	122.0	b)
20 T800 integral nodes	PARAMELD1	78.0	b)
1 T800 integral nodes	PARAMOLE2	1111.1	c)
12 T800 integral nodes	PARAMOLE2	93.4	c)
24 T800 integral nodes	PARAMOLE2	53.3	c)
32 T800 integral nodes	PARAMOLE2	42.3	c)
48 T800 integral nodes	PARAMOLE2	35.4	c)
63 T800 integral nodes	PARAMOLE2	34.7	c)

[a] CPU times.
[b] execution times (fixed topology).
[c] execution times on a PARSYTEC Supercluster 2 (variable topology), scaled to 25 MHz T800.

Table 3. Measured execution times for the calculation of the 2*e*-integrals of P_4S_3 (DZP) compared to the CPU times of various scalar computers

Computer	Program	Time	
T800 25 MHz	SEQUIMOLE	42373.4	a)
SUN 4/280	SEQUIMOLE	18683.2	a)
COMPAREX 7/78	SEQUIMOLE	8104.1	a)
ESV-3 (MIPS 3000)	SEQUIMOLE	6750.2	a)
IBM-RS/6000-320	SEQUIMOLE	6215.7	a)
IBM PPCS 16/370	PARAMOLE2	3283.4	b)
IBM PPCS 32/390	PARAMOLE2	363.5	b)
1 T800 integral node	PARAMOLE2	42422.5	c)
7.8 eff. T800 nodes	PARAMOLE2	5471.4	c)
27.8 eff. T800 nodes	PARAMOLE2	1601.7	c)
41.7 eff. T800 nodes	PARAMOLE2	1087.8	c)

[a] CPU times.
[b] execution times (fixed topology).
[c] execution times on a PARSYTEC Supercluster 2 (variable topology), scaled to 25 MHz T800.

Since, up to now, parallel computing is still widely an experimental area, we summarize our experiences with the following guidelines for the parallelization of existing software using the farming concept:

- The optimization and the minimization of the scalar program parts is required to maximize the number of server processes (Amdahl's Law).
- Additional optimization of the communication is the crucial step towards an efficiently distributed program for MIMD computers.
- If the number of jobs is reduced due to the communication optimization, the farm cannot retain a good load balance. The losses due to a load inbalance, however, are usually larger than losses due to communication overhead.

Acknowledgements. We wish to express our gratitude to PARSYTEC GmbH in Aachen and Dr. Peter Blömecke for the permission to test our program on a PARSYTEC Supercluster 2. We are also indebted to Dr. R. Janssen, Dr. W. Koch and U. Schauer from the IBM Scientific Center in Heidelberg and Dr. H. Bleher from the IBM Development Laboratory in Böblingen for support and the access to the IBM-PPCS prototypes.

References

1. INMOS Ltd (1987) The Transputer Family 1987, INMOS Product Information
2. Almlöf J, Faegri Jr K, Korsell K (1982) J Comp Chem 3:385
3. Almlöf J, Taylor PR (1984) Dykstra C (Ed.) Advanced theories and computational approaches to the electronic structure of molecules, NATO ASI Series 133:107, Reidel, Dordrecht
4. Hey AJG (1989) Comp Phys Comm 56:1
5. Glendinning I, Hey AJG (1987) Comp Phys Comm 45:367
6. Guest MF, Harrison RJ, Lenthe JH van, Corler LCH van (1987) Theor Chim Acta 71:117
7. Dupuis M, Watts JD (1987) Theor Chim Acta 71:91
8. Gadre SR, Kulkarni SA, Limaye AC, Shirsat RN (1991) Z Phys D 18:357
9. Transputer Development System (D 700 C) Inmos Ltd Bristol
10. Multitool 5.0 Parsytec GmbH, Aachen
11. INMOS Ltd. (1988) Occam 2 Reference Manual, Prentice Hall International Series in Computer Science. Prentice Hall, London
12. McMurchie L, Elbert ST, Langhoff SR, Davidson ER et al. Program MELD, University of Washington, Seattle. Modified version Wedig U
13. Cooper MD, Hiller IH (1991) J Comput-Aided Mol Des 5:171
14. Wedig U, Burkhardt A, Schnering HG von (1989) Z Phys D 13:377
15. Perihelion Software Ltd (1989) The helios operating system. Prentice Hall, London
16. Wedig U, Burkhardt A, Schnering HG von (1990) Harms U (ed) Supercomputer and chemistry. Springer, Berlin
17. Harrison RJ, Kendall RA (1991) Theor Chim Acta 79:337
18. Harrison RJ (1991) Intl J Quantum Chem 40:847
19. Lüthi HP, Mertz JE, Feyereisen MW, Almlöf JE (1992) J Comput Chem 13:160
20. Brode S (1991) Harms U (ed) Supercomputers in Chemistry 2:61, Springer, Berlin
21. Ammann EM, Berbec RR, Bozman G, Faix M, Goldrian GA, Pershing JA, Ruvolo-Chong J, Scholz F (1991) IBM J Res Develop 35:653
22. PARIX Operating System, Parsytec GmbH, Aachen
23. Geist A, Beguelin A, Dongarra J, Jiang W, Manchek R, Sunderam V (1993) PVM 3.0 User's Guide and Reference Manual, Oak Ridge National Laboratory, Oak Ridge TN