CROATICA CHEMICA ACTA CCACAA **69** (3) 899–925 (1996)

ISSN-0011-1643
CCA–2350

Original scientific paper

Development of a Quantum Chemical Two-Electron Integral Program for a Hierarchical Distributed Shared Memory Multiprocessor System (MEMSY)

Joachim Nedvidek and Peter Otto

Chair for Theoretical Chemistry, Friedrich-Alexander-University Erlangen Nürnberg, Egerlandstr. 3, 91054 Erlangen, FRG

Received November 8, 1995; revised April 24, 1996; accepted June 14, 1996

A quantum mechanical integral program has been implemented on a multiprocessor system with a hierarchical architecture, having at the same time a global memory and a locally distributed memory. Due to this hardware concept the possibilities of communication are manifold and therefore more complex in comparison with other multiprocessor systems, *e.g.* Intel iPSC/860 or workstation clusters. Nevertheless, the efficiency obtained using a simulator or the real system are of comparable quality. It is expected that this variety of interprocessor communications can be employed to its full extent in the second part of the program in which hermitian eigenvalue problems have to be solved many times.

INTRODUCTION

Analysis, modelling, prediction and optimization of molecular and macromolecular structures with the help of computers is one of the challenging classes for scientific supercomputers. The basis for understanding biological processes in the cell is the investigation of reaction mechanisms of biomolecules, *e.g.* DNA and proteins. Theoretical studies can provide important contributions to biochemistry at the molecular level.

Organic polymers are of increasing importance as materials for new technologies. Due to the variety of composition, structure and configuration, materials with a selected combination of physical and chemical properties can be designed. However, to obtain results which can be compared with experimental results, it is necessary to apply accurate quantum mechanical *ab*

initio methods. The use of these theoretical methods for investigations on the molecular level can in principle serve two purposes:

* theoretical models can be very useful for explaining new physical phenomena and for verifying postulated quantum mechanical interpretations versus experimental evidence. Current topics of high interest are *e.g.* the high-temperature superconductivity of ceramic materials, nonlinear optical properties and electrical transport properties of organic polymers.
* the application of theoretical models can be very useful in designing new materials with specific selected chemical and physical properties *e.g.* using structure-activity relations of biochemical molecules to predict new compounds with improved pharmacological efficiency. The necessary data can be obtained from quantum chemical calculations. In this way expensive and time-consuming experimental investigations can be avoided or at least reduced.

The size of chemical systems that can be treated numerically depends on the specific computer and its equipment. To investigate chemically interesting systems, which are usually very complex, supercomputers with large core memory, external disk space and an efficient I/O-system are needed. Therefore, new attempts should be made to scale problems to computers as well as to develop supercomputers with extreme capacity. These requirements, also arising from many other scientific areas, where heavy numerical operations are required, have stimulated the development of the new computer generation, mainly following three directions:

* large-scale multi-purpose computers with one or more scalar processors and additionally equipped with vector facilities (CDC CYBER 995E, IBM 3090 VF)
* vector computers with one or more (<32) processors (CRAY Y-MP, CRAY-T90)
* massively parallel computers with many (up to more than 1000) processors (CRAY, Intel, CONVEX, IBM, workstation cluster).

The computer industry enforces the strategy of massively parallel computers because systems appropriate for individual needs and financial resources can be designed. The hardware of a massively parallel system can be built up of nearly any number and any type of processors in very different architectures. The currently available multiprocessor systems differ not only in the architecture but also differ significantly in the interprocessor communication (synchronous/asynchronous), the storage and access of data (shared/distributed memory, local disks) and the software supporting parallel computing. Almost every multiprocessor system has its own communication possibilities and specific commands. Having worked out the proper partitioning of tasks of the mathematical problem under consideration, the user has to write a program for each parallel system to be used. Only in the last years attempts were undertaken to standardize parallel program software (PVM – Parallel Virtual Machine[1]).

Since the early development of parallel computers by Clementi on the lcap (loosely coupled array of processors) system,[2–4] many parallel implementations of quantum chemical programs have been reported in the literature,[5] *e.g.* the parallelization of a Direct Hartree-Fock program by Almlöf[6] for shared MIMD (Multiple Instruction Multiple Data) architectures and by Wedig for a transputer system.[7] Since our first attempts to reformulate our polymer program for the lcap system we have succeeded in improving its performance[8] by introducing a dynamical partitioning scheme. Later, we have developed an ab initio Hartree-Fock Crystal Orbital (HF-CO) program for SUPRENUM, Intel iPSC/860 and a workstation cluster.[9–11]

In this work we report the properties and results of the two-electron integral program for polymer calculations, which is the first time-consuming step of the HF-CO method obtained for MEMSY. It is a multiprocessor system with completely different architecture from that of the Intel or SUPRENUM.

In the following section, we summarize briefly the theoretical physical problem, *i.e.* the quantum mechanical method for calculating energy band structures and electronic wave functions for polymers. In the next section the MEMSY hardware and its operating system are presented. Then, a description follows of the computer program, its structure and ways of communication. Finally, results are reported and discussed and a short outlook for future developments is given.

THE PHYSICAL PROBLEM

Energy band structures and electronic wave functions of quasi one-dimensional periodic systems can be calculated with the help of the Hartree-Fock-Crystal-Orbital (HF-CO)-Method.[12–14]

The basic numerical problem of the HF-CO method is the solution of the generalized hermitian eigenvalue problem in matrix form.

$$\boldsymbol{F}(k_i)\,\vec{C}_n(k_i) = \varepsilon_v(k_i)\,\boldsymbol{S}(k_i)\,\vec{C}_n(k_i) \tag{1}$$

n = 1, ..., NBF basis functions

i = 1, ..., NKP points in reciprocal space

The overlap and Fock matrices in eq. (1), respectively, are defined as the Fourier transforms of the corresponding matrices in direct space

$$\boldsymbol{S}(k) = \sum_{J=-NEIG}^{NEIG} e^{iR_J k}\boldsymbol{S}^{0J}(q) \tag{2a}$$

$$\boldsymbol{F}(k) = \sum_{J=-NEIG}^{NEIG} e^{iR_J k} \boldsymbol{F}^{0J}(q) \tag{2b}$$

The elements of the respective matrices are defined as:

$$\boldsymbol{S}_{ab}^{0J} = \langle \chi_a^0 | \chi_b^J \rangle \tag{3}$$

$$\boldsymbol{F}_{ab}^{0J} = \boldsymbol{H}_{ab}^{0J} + \boldsymbol{G}_{ab}^{0J} \tag{4}$$

where the upper indices show that the basisfunction χ_a and χ_b are located in the reference and Jth cell, respectively. The Fock matrix element of $\boldsymbol{F}^{0J}$ is the sum of one-electron integrals (representing the kinetic energy, electron-nuclear attraction and, if necessary, effective core potential interaction)

$$\boldsymbol{H}_{ab}^{0J} = -\frac{1}{2}\langle \chi_a^0(\underline{r}) | \Delta_i | \chi_b^J(\underline{r}) \rangle - \sum_{H=-NEIG}^{NEIG} \sum_{\alpha=1}^{N} \langle \chi_a^0(\underline{r}) \left| \frac{Z_\alpha}{|\underline{r} - \underline{R}_H - \underline{R}_\alpha|} \right| \chi_b^J(\underline{r}) \rangle \tag{5}$$

and two-electron integrals (Coulomb and exchange integrals).

The overlap matrices $\boldsymbol{S}^{0J}$ occur due to the non-orthogonality of the basis functions. NEIG is the number of cells interacting with the reference cell.

$$\boldsymbol{G}_{rs}^{0J} = \sum_{H,L=-NEIG}^{NEIG} \sum_{c,d}^{NBF} P_{cd}^{HL} \left[\langle \chi_a^0(\underline{r}_1)\, \chi_c^H(\underline{r}_2) \left| \frac{1}{r_{12}} \right| \chi_b^J(\underline{r}_1) \chi_d^L(\underline{r}_2) \rangle - \right. \tag{6}$$

$$\left. - \frac{1}{2}\langle \chi_a^0(\underline{r}_1)\, \chi_c^H(\underline{r}_2) \left| \frac{1}{r_{12}} \right| \chi_d^L(\underline{r}_1)\, \chi_b^J(\underline{r}_2) \rangle \right]$$

The first time consuming step consists of the calculation of the two-electron integrals shown in eq. (6).

$$\left\langle \chi_a^0 \chi_c^H \left| \frac{1}{r_{12}} \right| \chi_b^J \chi_d^L \right\rangle = \iint \chi_a^0(\vec{r_1})\, \chi_c^H(\vec{r_2})\, \frac{1}{r_{12}}\, \chi_b^J(\vec{r_1})\, \chi_d^L(\vec{r_2})\, d\vec{r_1} d\vec{r_2} \tag{7}$$

$a, b, c, d = 1, \ldots, \mathrm{NBF}$

$J, H, L = -\mathrm{NEIG}, \ldots, \mathrm{NEIG}$

The basis functions (atomic orbitals) are written as a linear combination of n_i so called primitive Gaussian functions ($n_i \approx 2 - 10$). The superscripts denote one of Ngroup possible combinations of cells where the basis functions are localized. The value of Ngroup increases rapidly with increasing NEIG

(NEIG = 1, 2, 3; Ngroup = 5, 15, 35). The total number of integrals to be calculated then yields

$$\text{Ngroup} * \text{NBF}^4$$

(without taking into account symmetry restriction with respect to basis functions and negligible integrals smaller than a given integral threshold).

Since the calculation of the two-electron integrals can be performed independently a high degree of parallelism can be realized.

One of the main problems is the partitioning of tasks in such a way as to achieve a nearly equal load balancing of the processor nodes.

THE MODULAR EXPANDABLE MULTIPROCESSOR SYSTEM (MEMSY)

The high-performance multiprocessor system MEMSY (**M**odular **E**rweiterbares **M**ultiprozessor-**Sy**stem)[15] has been designed, developed and tested within the Sonderforschungsbereich 182 (multiprocessor- and network configurations) of the Deutsche Forschungsgemeinschaft. It belongs to the group of MIMD-systems (multiple instruction stream – multiple data stream) and its processor nodes have additional communication memory, which can be used by other nodes. In this way, MEMSY unifies elements of a parallel system with global memory and of a parallel system with local distributed memory. The real system is built up of two planes of processor units as shown in Figure 1. Identification of the nodes can be done in two different ways: using either coordinates, or characteristic node numbers (0–3 for the units in plane B; 4–19 for the units in plane A).

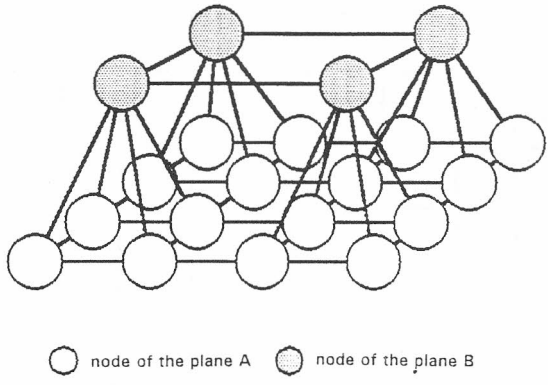

Figure 1. The MEMSY-architecture.

The configuration of a single node is:

4 processors, Motorola 88100, 25 MHz

512 KByte Cache

32 MByte working memory

Ports to the communication memories

Ports to the interprocessor-interrupt-network

Port to the global 1-bit-communication system

Port to the FDDI-network

local SCSI-harddisk 512 MB

Each unit can use its communication memory and has access to the communication memory of its four nearest neighbours (the nodes at the edge are directly linked with a torus connection to the opposite nodes). Figure 2 shows the connection of the processor nodes and the communication memories with linking elements.

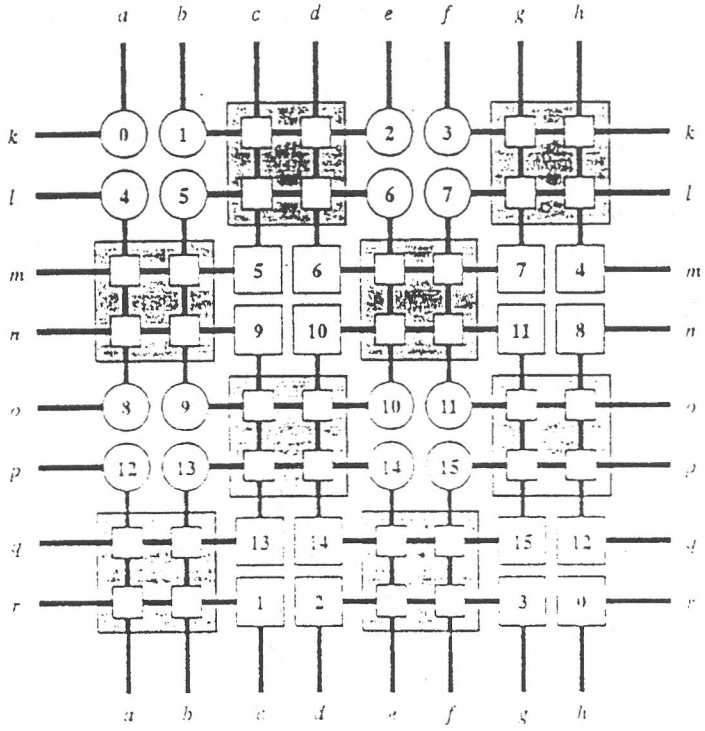

Figure 2. Use of the linking elements in a plane with 16 processors. Connection of processor-nodes and communication-memories.

The numbered circles symbolize the processor units, the grey boxes the linking elements and the numbered squares the communication memories. Different switching positions of the linking elements enable the required connections.

The recently developed operating system MEMSOS,[16] is based on UNIX and contains specific 'shared memory' commands.

The architecture of MEMSY is completely different from that of other parallel-systems, for which the Hartree-Fock Crystal-Orbital program had already been developed. The Intel iPSC/860 parallel-system consists of up to 128 processors connected in a hypercube. The communication between the nodes can be realized by using message-passing commands. In contrast to the SUPRENUM, the message-passing commands call subroutines to do this work. Therefore, the standard programming languages C and FORTRAN can be used without specific expansions.

Communication Between the Nodes

As mentioned above, MEMSY is a mixed form of parallel systems with global memory and parallel systems with distributed shared memory and is equipped with distributed memory and fragmented global memory (communication memory). Therefore, a communication between two nodes can be done either with message-passing, or using the communication memory, or in other ways, briefly indicated in the following paragraphs:

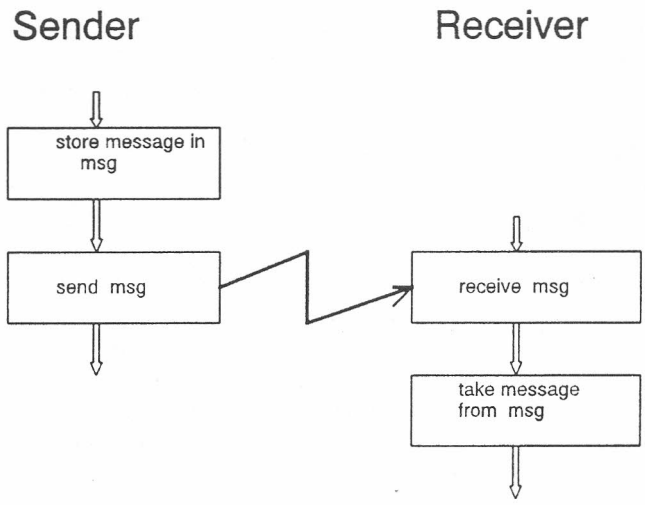

Figure 3. Sending of a short message to a receiver.

Messages are short informations, which a node can send to a neighbouring node. The possibility of sending messages to distant nodes is installed at the simulator and planned for the real system. One message contains only three data words and an identification-number, stored together in a structure. The reception can be blocking or non-blocking. This possibility of communication is appropriate for a small number of data, which should be transmitted fast. However, in comparison with the message-passing system of Intel iPSC/860, the message-passing system of MEMSY is very restricted:

* The Intel system can send messages up to 256 KBytes instead of only three data words.
* The sequence of the received messages in the MEMSY system is prescribed. The receive-command in the program of the receiver gets the next message and works with it. The Intel system, on the other hand, opens the possibility of crossing messages. One can characterize a message, by attributing a type reception will only be done, if the receiver command is of the same type; all others will be kept in a mail-box until they are required.
* In the program for MEMSY, one has to create a loop over all nodes to send a message to all others, whereas for the Intel system it is sufficient to send a message only to node '–1'.

The weakness of the message-passing system of the MEMSY is founded in the basic concept that has been followed in its development. MEMSY has a completely different architecture, predestinated for shared memory communication, and a comparison restricted to this communication possibility cannot be the only criterium of its value.

The global harddisk permits a node to send informations to any other node. There is no restriction in the length or type of information. The only drawback of this way of communication is the slowliness of this method. Therefore, only large informations should be send by using the global harddisk. Writing and reading files on the global disk has to be organized very carefully because only one node should write or read the same file of the global disk at a given time. Therefore, several other communications have to be done to coordinate this procedure. Figure 4 shows an easy way of sending information to a receiver by using the global harddisk. In this way, it could be guaranteed that the reading of the receiver starts after the writing of the sender has finished. To prevent complicated and inefficient communication structures, one can use 'global files with characteristic names'. For example, one can create a global file with a name that contains the number of the receiver. In this way, several receivers can read their global files simultaneously after having identified themselves with the symbolic constant M_LOCAL.

Data exchange can also result from the use of the *communication memory.* This way is the fastest possibility of transmiting information but it can

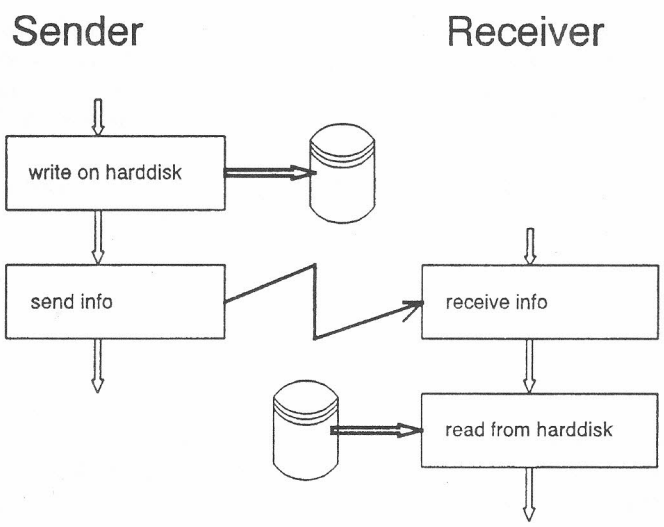

Figure 4. Sending of a long message to a receiver.

be done only with the nearest neighbours. Firstly, one has to create a segment that can be attached to other processes. Following the use of this segment all attachments can be removed and the segment released. The segment is created when a request occurs in the program. The segment has to be generated in the communication memory of that processor node, which is a neighbour to those who need to use this segment. Only if a segment that satisfies the requests exists, it can be created.

Other communication-possibilities e.g. semaphore, signal and spinlock, well-known from other multiprocessor systems are also implemented on MEMSY.

An easy and comfortable way for communication is planned, called 'transport'. The operating system itself chooses the best way for the sending, considering the type of the data, the partner of the data transfer and defects of the system. The only information the user has to supply is the node number of the receiver.

The MEMSY-Simulator / The MEMSY-System

At the beginning of our investigations, while the real computer system was still in the testing phase, we used the simulator of the MEMSY running on a Sequent Symmetry system (a multiprocessor system with global shared memory). After the realization of the real MEMSY-system, the program was adapted, transferred and tested on this system. The simulator maps the architecture with 20 processor nodes (see Figure 2), the message-passing sys-

tem, the use of communication memory, semaphores, signals, spinlocks and the connection mentioned above. In the simulator, each node consists of one processor. A program developed for the simulator can be easily ported to the real system. The only difference is that the message-passing in the real system can be done only with the nearest neighbours.

The simulator needs a program that branches after identification of the number of the node (M_LOCAL), since the same program has to start on all nodes.

STRUCTURE OF THE PROGRAM

The program was developed with the help of the programming language C, because C and C++, which indicate excellent tools for working on memory management, were the only compilers implemented on the system. The program uses the MASTER-SLAVE-method together with the FARMING-concept. One node, the defined MASTER, distributes the tasks to the other SLAVE nodes. If any SLAVE has nothing to do, he asks for a new task and gets it from the master. This concept is responsible for the fact that in the real system only 7 nodes can be used, since the master-node had to be the nearest neighbour to the slaves, as shown in Figure 5. In this way, one elementary pyramid and two additional nodes of the B-plane can be used.

The program for the simulator and also for the real system does not support the architecture of MEMSY and neglects the particular properties of communication memory, which is in this case more advantageous than pushing the problem into the architecture. The calculation of two-electron integrals does not need any local data exchange which is supported by MEMSY.

To restrict the number of slaves, a symbolic constant NSLE was introduced, which gives the number of the last working slave.

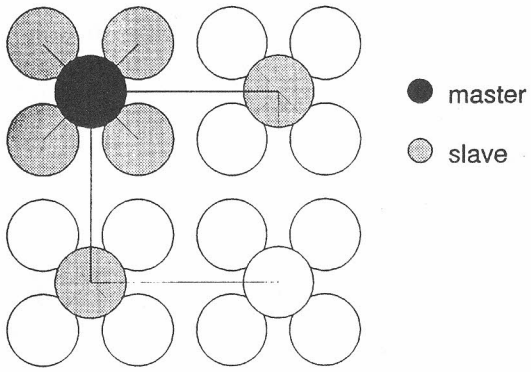

Figure 5. Connection of master and slaves in the program for the MEMSY-system.

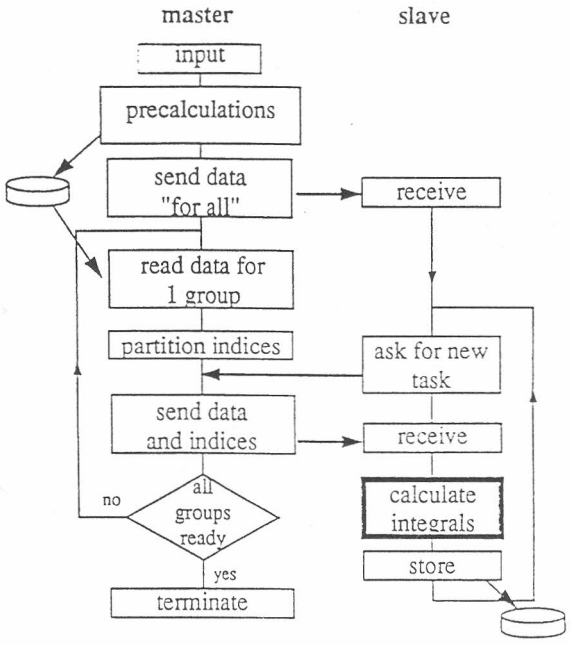

Figure 6. Master-slave concept of the integral program.

Running-structure of the Program

Figure 6 shows the main idea of the program. The left side demonstrates the master-program and the right side the slave-program. To keep the figure easily understandable, the last part (not necessary for the calculation) of the program does not appear. The following scheme describes in more detail the main steps of the program as it is indicated by the boxes in Figure 6.

master-node:

* Reading the input.
* Calculation of some informations and storing on harddisk.
* Sending the first results to all nodes.
* Reading from harddisk the informations for one integral group.
* Distribution of the work into tasks
* Sending one task to a free slave, from which the master got the corresponding information. All tasks are distributed in this way.
* If all groups are not calculated yet, the master reads the information for the next group, distributes the tasks, and so on.
* If all groups are calculated, the master gathers all control results and finishes his work.

slave-node:

* Reception of the results, which have been calculated by the master.
* Request for a new task.
* Getting a task from the master.
* Calculation of the integrals of this task.
* Storing the integrals on the local harddisk.
* New request for a task.

If the slave does not get a task,

* termination.

Loop Structure of the Integral Calculation

For all relevant combinations of H, J and L Ngroup groups were built up and in each group the variables a,b,c,d, are given by the loops over the contracted Gaussian basis functions.

igroup = 1, ..., Ngroup
  ab = 1, ..., NBFa * NBFb
    c = 1, ..., NBFc
    d = 1, ..., NBFd

$$< \chi_a^0 \chi_c^H \left| \frac{1}{r_{12}} \right| \chi_b^J \chi_d^L > =$$

<end of all loops>

The values of NBFi indicate possible symmetry restrictions with respect to the function indices a, b, c and d. The partitioning of the integral-calculation task to realize a parallel calculation is done in the ab loop.

Communication Structures of the Program

In comparison with the corresponding program for the Intel iPSC/860, the program for MEMSY is much more complex. The reason is the more advantageous message-passing system of the Intel iPSC/860, which meets better the requirements of the program. MEMSY needs in addition for the same problem the use of the global harddisk. Furthermore, the MEMSY architecture is completely different from the architecture of the Intel, so it is nessecary to rewrite the program.

With respect to the communication requirements in the program, three cases can be differentiated, which are discussed shortly in the next paragraphs.

The first part is the transport of the primarily calculated results from the master to all slaves. The calculation of the integrals is done in the second part. The master generates the individual tasks and distributes them. The third part represents the collection of the control results by the master.

The Master Sends Precalculated Information to All Slaves

The master precalculates data from the input and sends this information to all slaves. If every node calculated these data, there would be no advantage. It is true that this part of communication could be avoided, but other problems would appear, like the coordination of the reading of the input or the synchronization of the nodes after having performed the calculations. Two different possibilities can be distinguished:

The Master Sends a Short Information to All Slaves

It is very easy to send a short information to all slaves, which can be done in a loop structure counting message-passing commands. For example, to send a seven integer vector to all slaves, the master has to perform a loop over all slave numbers which includes three send commands. The slave function requires only three blocking receive commands. The following scheme explains in more detail Figure 7 and its individual steps.

master:

m1: start of the loop;
m2: junction;
m3: any slave without information?
    yes: continue m4;
    no: goto m8;
m4: send message to slave imemsy;
m5: as m4;
m6: as m4;
m7: increase imemsy;
    goto m2;
m8: continue program;

slave:

s1: blocking reception of a message;
s2: same as s1;
s3: same as s1;
    continue program

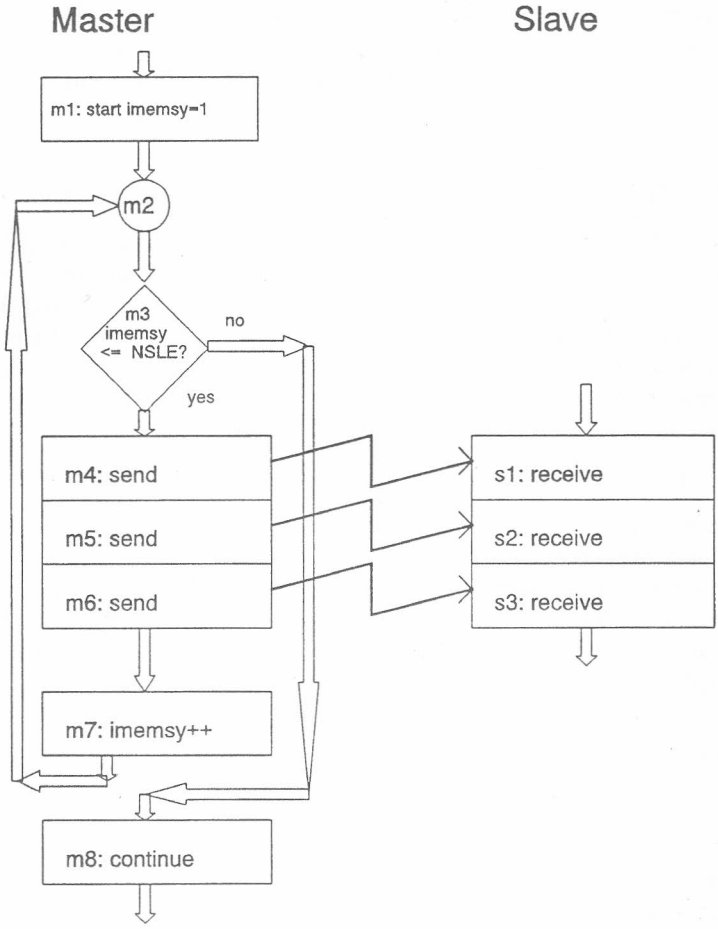

Figure 7. The master sends a short information to all slaves.

The Master Sends a Long Information to All Slaves

To send a long information to all slaves, the master first has to write it in one or more global files. Then, he has to order one single slave to read this information. The master has to wait until the reading has been finished. Afterwards, he can order the next slave to read the information. When all slaves have got their information, the master has to unblock the barrier at which the slaves are waiting. This barrier is necessary because a confusion of the 'reading finished message' and subsequent messages of the program may produce serious errors. The following scheme is explained in Figures 8 and 9 and its individual steps.

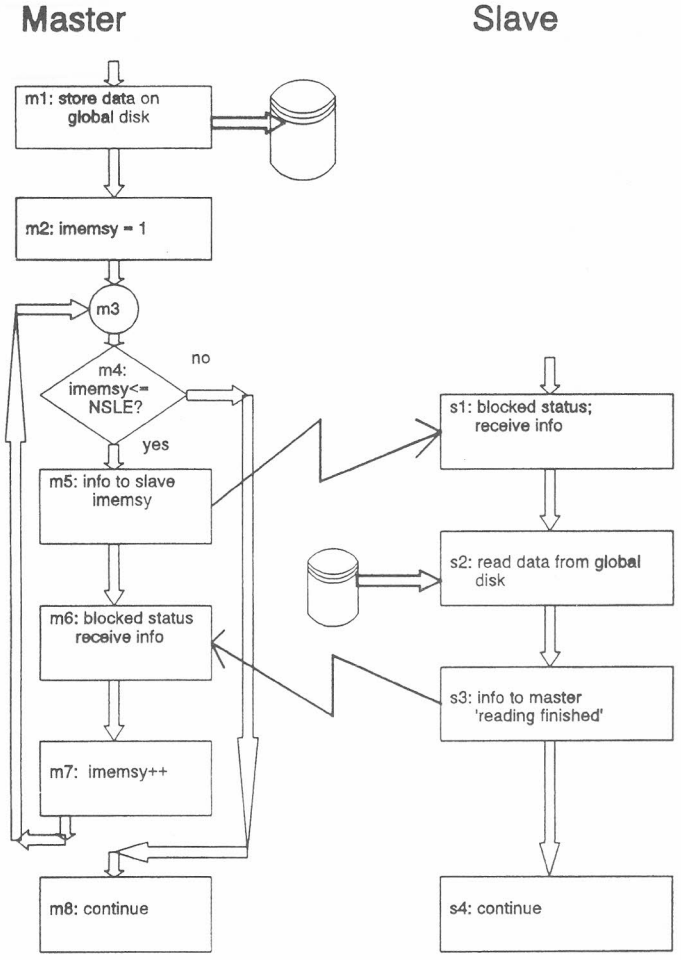

Figure 8. The master sends a long information to all slaves (part 1).

Master:

m1: open global files;
write global files;
close global files;

m2: start of the loop; imemsy=1;

m3: junction;

m4: any slave without information?
yes: continue m5;no: goto m8;

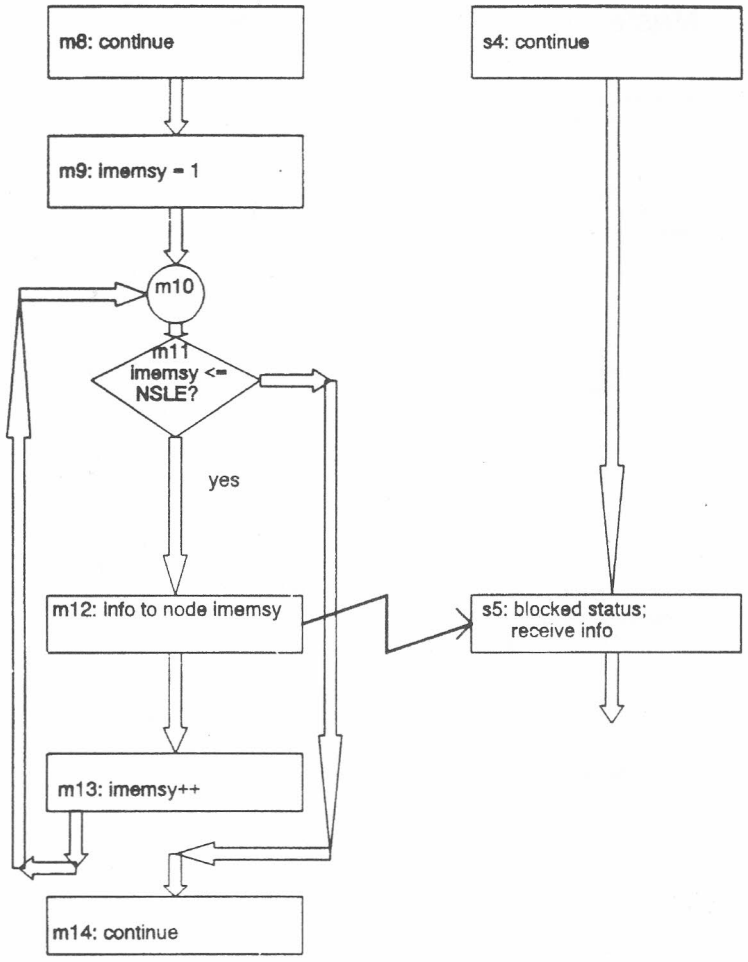

Figure 9. The master sends a long information to all slaves (part 2).

m5: send message to slave imemsy (–> s1) to start reading global files;
m6: blocked waiting until reading has finished;
receive message from slave imemsy (all others wait at s1 or s5);
m7: increase imemsy; goto m3;
m8: continue m9;
m9: start of the second loop; imemsy=1;
m10: junction;
m11: any slave without barrier-unblocking-message?
yes: continue m12;no: goto m14;

m12: send unblocking message to slave imemsy (–> s5);

m13: increase imemsy; goto m10;

m14: continue program;

Slave:

s1: blocked waiting until master sends information (<– m5) to start reading global files;

s2: open, read and close global file;

s3: send information to master (–> m6), that the reading of global files has been finished;

s4: continue s5;

s5: blocked waiting (barrier) until master orders (<– m12) to continue program;

The Master Distributes the Tasks to the Slaves, Which Calculate the Integrals

This is the most difficult part of the program. The master has to send tasks to a free slave until the complete work is done. The slaves have to calculate the integrals of the tasks and then ask for another task. After finishing all tasks, the master has to send a relevant signal to each free slave. Because of the size of the task information, a data exchange with the global disk is absolutely nessecary.

A serious problem is the fact that a free slave should send in this part of the program only one message to the master because a second message could cause errors. The master cannot differentiate between the request of a free slave for a new task and other messages (see above).

These difficulties can be solved by using global files with characteristic names. The master creates files with filenames, which are built up of some characters and the number of the free slave, sent to the master by asking for a new task.

The master node receives the node-number of a free slave. He tells the slave that there is a remaining undone task and sends the information for this task with the help of messages and global files to the slave.

To prevent the long waiting time until the slave has finished the reading of the global file, the master creates a receiver-characteristic global file and informs the slave. Finally, the master is ready to receive the number of the next free slave, while the first is still reading 'his' global file. After all work has been done, the master sends to the asking slaves a relevant message.

The slave node tells the master his readiness by sending him his node number. Now he gets information to receive a new task and to calculate integrals or to terminate this part of the program. To calculate a new task, the slave obtains a message-send vector and information about his specific global file. He reads this global file, calculates the integrals, stores them on the local disk and asks the master for a new task. If all tasks have been distributed, the slave terminates this section. The following scheme is explained in Figure 10.

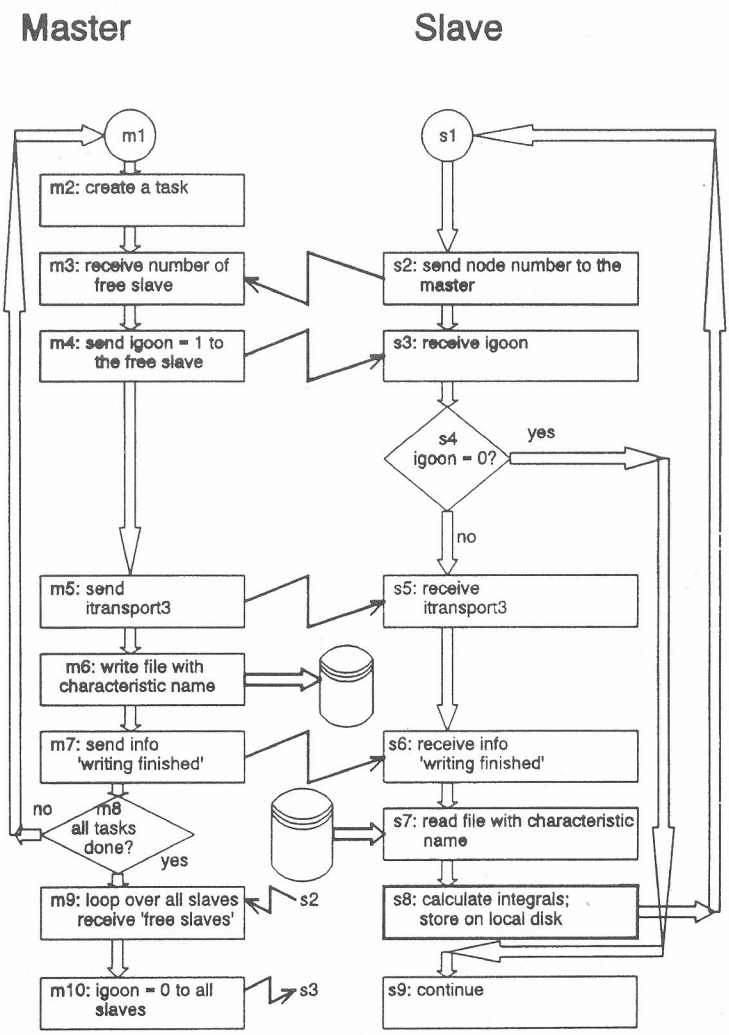

Figure 10. The master distributes the tasks to the slaves which calculate the integrals.

Master:

m1: junction;
start of all loops in which distribution of the tasks is done;

m2: creation of a new task;

m3: the master receives the number of a free slave (<– s2);

m4: Sending a message to the free slave that there are still undone tasks (igoon=1 to –> s3);

m5: Transmitting a vector itransport3, which consists of indices for integral calculation, with the help of three messages to the free slave;

m6: creation of a global file with a name characteristic of the free slave, writing and closing it;

m7: Sending a message to the slave (–> s6) to start reading this file;

m8: All tasks distributed?
no: goto m1;yes: continue m9;

m9: Getting a task-asking message from all slaves (loop with reception of the slave numbers);

m10: Sending to all asking slaves a message that all tasks have been distributed (loop with message igoon=0 –> s3);

Slave:

s1: junction;
start of an endless loop (this loop can only be left by if-branching);

s2: Sending the own node number to the master (–> m3);

s3: Getting igoon from the master (<– m4 or <– m10);

s4: igoon = 0 ?
yes: goto s9;no: continue s5;

s5: Receiving of the vector itransport3 with the help of three receive-commands (<– m5);

s6: Blocked reception of the information that the slave can start reading global file;

s7: Reading of the characteristic global file;

s8: Calculation of the integrals and storing them on local disk;
goto s1;

s9: continue program;

The Master Collects Control Results of the Slaves

This part of the program is not necessary for the integral calculation. It is induced only to control the correct run of the program but, due to the interesting communication structure, it will be discussed here. While calculating the integrals, each slave creates control results, which are sent to the master, where they are summed up. To prevent errors in the subsequent use of the integrals by the master, again the method of global files with node specific names is chosen. Each slave writes his control results on a file, whose name includes his node number. Now, he sends his node number to the master as information that he has finished writing. In a loop over the number of all slaves, the master receives the number of the slave, who has just finished, reads 'his' global file and sums up the results. Note that the loop variable need not be equal to the actual slave number. It only accomplishes that each slave has a data exchange with the master. Figure 11 explains the following scheme and its individual steps.

Master:

m1: start of the loop; imemsy=1;
m2: junction;
m3: any slave unconsidered?
yes: continue m4;no: goto m8
m4: blocked reception of the number of the finishing slave;
m5: reading of the characteristic global file;
m6: summation of the control results;
m7: increasing of imemsy;
goto m2;
m8: continue program;

Slave:

s1: Storing of the control results in a global file with a node-number characteristic name;
s2: sending the node-number to the master;
s3: continue program;

RESULTS AND DISCUSSION

The performance of a multiprocessor computer system depends on the capacity of the processors, the hardware architecture, the operating system including communication and synchronization tasks and, of course, on the

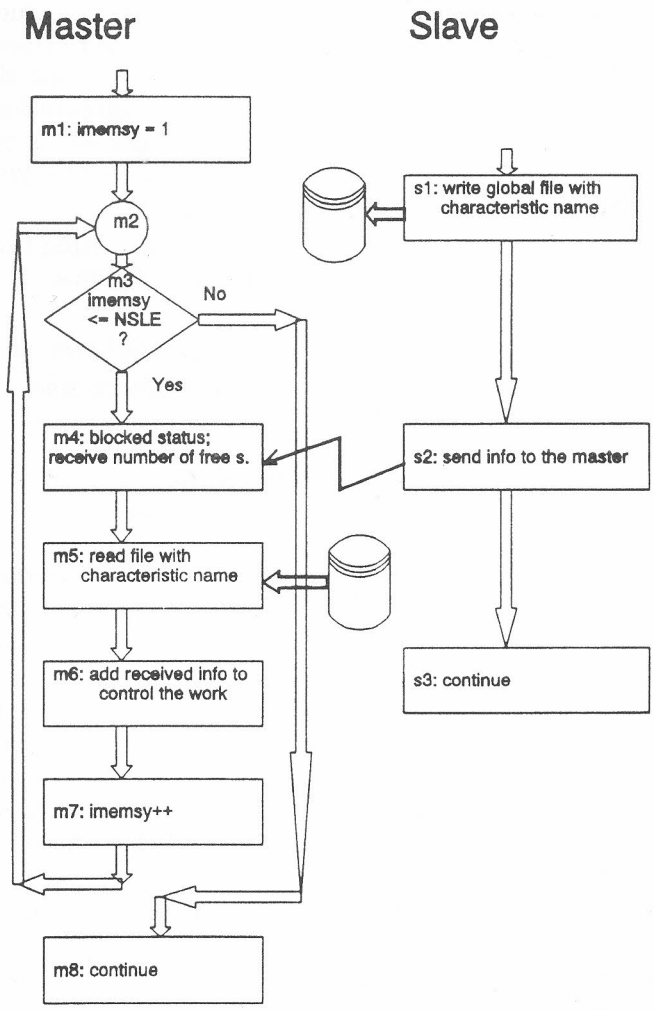

Figure 11. The master collects control results of the slaves.

quality of the user-supplied programs, *i.e.* the partitioning of tasks to achieve an optimal load-balancing of the participating processors.

Detailed measurements are necessary to evaluate the efficiency of the parallel computer system. Firstly, distinguished model calculations have to be worked out and, secondly, CPU- and wall times are required for a sequential version of the program. These performance times on a sequential system can then be used to compare with and to analyze the results on a parallel computer with respect to CPU, communication and synchronization time contributions.

At present, it is not possible to configurate MEMSY as a true sequential machine using only one processor. Therefore, we have performed the calculation for a configuration consisting of a master and only one slave processor. This choice is not unreasonable because in this configuration one of the nodes performs the complete calculation while the other one is in the standby state, and appropriate measurements of time differences yield the necessary values for comparison.

The quality of a parallel program can be evaluated using the two quantities, speed-up and efficiency, which are defined as follows:

$$speed\text{–}up = \frac{T_1}{T_p} = \frac{CPU\text{–}time\ of\ the\ sequential\ version}{CPU\text{–}time\ of\ the\ parallel\ version}$$

$$efficiency = \frac{Sp}{p} = \frac{speed\text{–}up}{number\ of\ processor}$$

In contrast to the efficiency, the speed-up is not characteristic of the performance of the whole system but of the degree of parallelization in the program. This analysis has been carried out on the MEMSY Simulator and on the real system, respectively.

We succeeded in running the program on the simulator with up to 13 nodes for calculating $(CH_2)x$ (NBF=7) in the first neighbours interaction approximation (NEIG=1). Due to the fact that the processors of the simulator were not equal and that the program could not select the processors, the time measurements were not reproducible and, therefore, no reliable conclusions could be drawn.

Therefore, we transfered and implemented the program to the real MEMSY computer. Due to the restriction of the interprocessor communication with the help of message-passing, which is allowed only between nearest neighbour processors, not more than seven nodes (one master and six slaves) could be used. However, in a near future project, software tools will be available to circumvent the present limitation. Nevertheless, it is possible to investigate the speed-up and efficiency as functions of the number of nodes, the number of tasks to be distributed and the size of the individual tasks.

Model calculations have been performed for polyethylene (number of basis functions NBF=7) in the neighbours interaction approximation ranging from 1 to 4 (NEIG=1,2,3,4; Ngroup=5, 15, 35, 65). The results are summarized in Table I.

The Table contains information on the number of slave processors, the total CPU time, speed-up and efficiency. Figure 12 shows the resulting speed-up as a function of the number of tasks and the number of processors

TABLE I

Running time, speed-up and efficiency for the calculation of $(CH_2)_x$, in the NEIGth neighbours' interaction approximation

	number of slaves	tins	speed-up	efficiency in %
NEIG = 1	1	203	1.00	50.0
	2	121	1.68	55.9
	3	86	2.36	59.0
	4	67	3.03	60.6
	5	59	3.44	57.3
	6	55	3.69	52.7
NEIG = 2	1	808	1.00	50.0
	2	427	1.89	63.1
	3	293	2.76	68.9
	4	223	3.62	72.5
	5	187	4.32	72.0
	6	163	4.96	70.8
NEIG = 3	1	2105	1.00	50.0
	2	1084	1.94	64.7
	3	731	2.88	72.0
	4	562	3.75	74.9
	5	460	4.58	76.3
	6	394	5.34	76.3
NEIG = 4	1	4374	1.00	50.0
	2	2233	1.96	65.3
	3	1509	2.90	72.5
	4	1150	3.80	76.1
	5	934	4.68	78.1
	6	793	5.52	78.8

for NEIG=1 up to 4 *vs.* the ideal straight line. The efficiency for the same series of measurements is graphically diagrammed in Figure 13. It has to be mentioned that the capacity of the processors (MOTOROLA 88100 25 MHz) implemented in MEMSY is too low to permit calculation of polymers with large elementary cells or many basis functions, respectively.

In the case of model calculations with a large number of tasks to be distributed (NEIG=3 and 4), the speed-up is linear from nslave=1 up to 6 whereas for smaller problems the linearity is no longer fulfilled. The decreasing speed-up for the larger number of processors finds its explanation in the fact that the number of integrals per task to be calculated by each slave processor becomes very small, so the total time will be determined

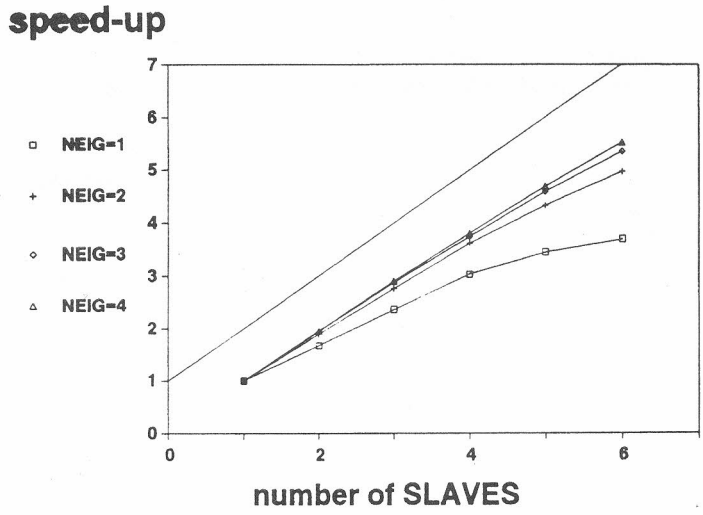

Figure 12. Speed-up for the calculation of $(CH_2)_X$.

more and more by communication processes, waiting and synchronization times. From the obtained results we can conclude that, for heavy numerical applications, the speed-up will increase linearly for more processors. The qualitative results of the speed-up are also reflected in the graphic of the

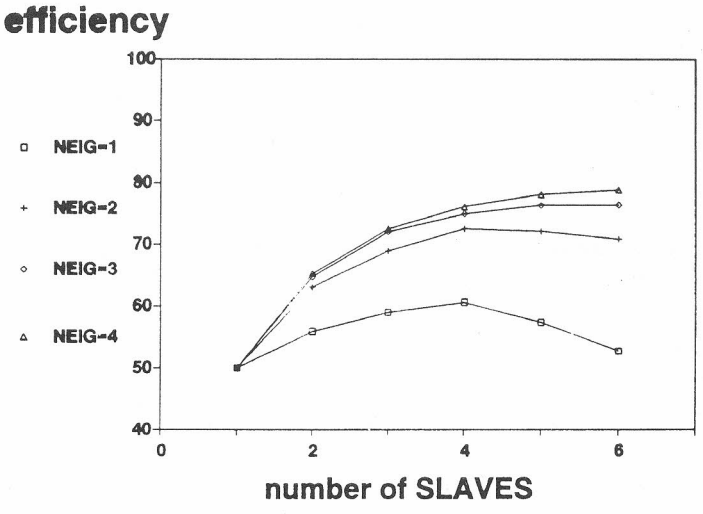

Figure 13. Efficiency for the calculation of $(CH_2)_X$.

efficiency *vs.* the number of slaves, shown in Figure 13. The efficiency reaches the value of 0.8 for the largest calculation that could be performed for six slaves. Again, for numerical applications of complex chemical systems, we can expect even a higher efficiency.

One important point has to be mentioned in evaluating the respective efficiencies. At present, the program running on the master processor serves only to precalculate and to distribute data to the slaves but does not take part in the actual calculation of integrals. This strategy has, of course, an important effect in the case of only a few processors. This effect has been investigated for a workstation cluster developing a version of a parallel program in which the master program is equivalent to the slave program except for the initialization of tasks. A significant improvement of the parallel performance has been observed. However, this increase in efficiency becomes less important when the parallel configuration unifies many processors and the influence of the »inactive« master processor will be strongly reduced.

SUMMARY AND PREVIEW

In this work, we have investigated the implementation of quantum mechanical programs – taking as an example the two-electron integral program as it is used for *ab initio* Hartree-Fock crystal orbital calculations – on a multiprocessor system with a hierarchical architecture with distributed and global storage possibilities. The comparison with other parallel computers shows, on the one hand, that despite the essentially different communication and synchronization procedures, a comparable speed-up and efficiency of the program could be obtained. It has also to be mentioned that the manifold ways of communication could be used only partially for the program under investigation, *e.g.* no advantage has been taken from the communication-memory available for nearest neighbouring nodes.

On the other hand, we expect that especially the variety of communications available in MEMSY can be employed in a very efficient way in the second part of the program, the iterative solution of hermitian eigenvalue problems. We have already developed parallelization algorithms and the corresponding computer programs for the diagonalization based on the QR algorithm. Detailed investigations show that, for the size of our problems, a small number of processors would be suitable and therefore the local memory could be efficiently used.

The restriction of message-passing with only the nearest neighbours leads to the desire of using all 20 nodes of MEMSY for the discussed program. This wish was an interesting aspect for those persons who developed the operating system of MEMSY. An optimal configuration which allow to calculating with one master and 20 slaves is shown in Figure 14. One node

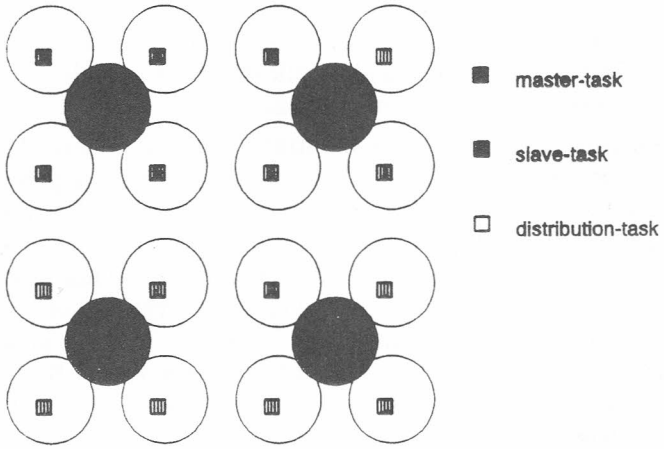

Figure 14. The use of 20 nodes with master-slave concept.

of the B-plane is defined as the master, all four nodes of the B-plane are distributers and, in addition, all 20 nodes of the A-plane and the B-plane act as slaves. The hardware to solve this problem already exists because each node consists of four processors and the connections are the same as those still used. Therefore, more than one task can be started on one node.

The realization of this possibility by extending the operating system is in progress and will be a long-term task for the computer science.

Acknowledgement. – The authors are very grateful for the financial support of the German Research Council (Sonderforschungsbereich 182 »Multiprozessor- und Netzwerkkonfigurationen«).

REFERENCES

1. PVM 3.0: *Parallel Virtual Machine System 3.0*, University of Tennessee, Knoxville TN. Oak Ridge National Laboratory, Oak Ridge TN. Emory University, Atlanta GA. Authors: A. L. Beguelin, J. J. Dongarra, G. A. Geist, R. J. Manchek, B. K. Moore, and V. S. Sunderam.
2. E. Clementi, G. Corongiu, J. Detrich, S. Chin, and L. Domingo, *Int. J. Quant. Chem. Symp.* **18** (1984) 601.
3. E. Clementi, S. Chin, and D. Logan, in: *Lecture Notes in Chemistry*, Vol. 44, M. Dupius (Ed.), 1986.
4. E. Clementi, G. Corongiu, and S. Chakravorty, in: *MOTECC-90*, E. Clementi (Ed.), ESCOM Publisher, Leiden, 1990.
5. R. J. Harrison and R. Shepard, *Ann. Rev. Phys. Chem.* **45** (1994) 623.
6. H. P. Luethi, J. E. Mertz, M. W. Feyereisen, and J. E. Almlöff, *J. Comput. Chem.* **13** (1992) 160.

7. U. Wedig, A. Burkhardt, and H. G. v. Schnering, *Z. Phys. D.* **13** (1989) 377.
8. S. Kindermann, E. Michel, and P. Otto, *J. Comp. Chem.* **13** (1992) 414.
9. P. Otto and H. Früchtl, *Computer & Chemistry* **17** (1993) 229.
10. H. Früchtl and P. Otto: *Quantum Mechanical Programs for Distributed Systems: Strategies and Results*; in Bode A., Dal Cin M., *Parallel Computer Architectures, Theory*, Hardware, Software, Applications; Lecture Notes in Computer Science, Berlin 1993.
11. G. Fritsch, W. Henning, M. Peric, M. Schäfer, E. Schreck, H. Früchtl, and P. Otto: *Numerische Anwendungen auf Verteilten Rechensystemen*, in: Wedekind H., Verteilte Systeme, Grundlagen und zukünftige Entwicklunge; Mannheim, 1994.
12. G. Del Re, J. Ladik, and G. Biczo, *Phys. Rev.* **155** (1967) 997.
13. J.-M. André, L. Gouverneur, and G. Leroy, *Int. J. Quant. Chem.* **1** (1967) 427, 451.
14. J. Ladik, *Quantum Theory of Polymers as Solids*, Plenum Press, New York, 1988, 9.
15. F. Hofmann and C. U. Linster, *Modular erweiterbares Multiprozessor-System, Multiprozessor- und Netzwerkkonfigurationen*, Arbeits- und Ergebnisbericht des Sonderforschungsbereichs 182, Friedrich-Alexander-University, Erlangen 1992.
16. MEMSOS Programmer's Manual, Erlangen 1991, 1993.

SAŽETAK

Razvoj programa za račun kvantno-kemijskih dvoelektronskih integrala za hijerarhijski ustrojen višeprocesorski sustav sa zajedničkom memorijom (MEMSY)

Joachim Nedvidek i Peter Otto

Program za kvantno-mehaničke integrale prilagođen je za višeprocesorski sustav s hijerarhijskim ustrojem koji je opremljen globalnom i lokalno pridruženm memorijom. Zahvaljujući ovakvoj koncepciji, mogućnosti komuniciranja su mnogostruke i složenije u usporedbi s drugim višeprocesorskim sustavima kao što su npr. Intel iPCS/860 ili grupirane radne stanice. Ipak, efikasnost procijenjena uz pomoć simulatora ili stvarnog sustava podjednako je dobra. Očekuje se da će raznolike međuprocesorske komunikacije doći do izražaja u drugom dijelu programa u kojem se mnogo puta računaju vlastite vrijednosti hermitskih matrica.