PROGETTO FINALIZZATO
SISTEMI INFORMATICI E CALCOLO PARALLELO

SOTTOPROGETTO 8

Iniziativa di Supporto al Calcolo Parallelo

Coordinatore Stefano Trumpy

R. Baraglia[1], D. Laforenza[1], R. Perego[1]
A. Laganà[2], O. Gervasi[3]
M. Fruscione[4], P. Stofella[4]

PORTING OF REDUCED QUANTUM REACTIVE SCATTERING CODES ON A MEIKO COMPUTING SURFACE

N. 8/ 20 Maggio 1991

[1] CNUCE, Italian National Research Council, Pisa, Italy
[2] Dept. of Chemistry, University of Perugia, Italy
[3] Academic Computing Center, University of Perugia, Italy
[4] Advanced Computing System, Milano, Italy.

Rapporto Tecnico

CNUCE - CNR, via S. Maria 36, 56126, Pisa, Italy

Abstract

The aim of this paper is to investigate the restructuring of a typical computing intensive application on a massively parallel architecture. A reduced dimensionality application to the calculation of reactive properties of atom-diatom systems has been restructured to run on the Meiko Computing Surface. By using the execution times analysis of the application made on IBM 3090-VF mod.18E we have reorganized the application according to a master-slave model. This paper presents the Meiko Computing Surface architecture, the parallel programming tool CSTools and the analysis of the restructure steps made to get the parallel version. Finally, the performance of the parallel code is discussed.

Keywords: *massive parallelism, massively parallel architectures, coarse-grain parallelism, speedup measurement.*

1. Introduction

The calculation of accurate (starting from first principles) estimates of physical observables of elementary chemical processes is of great importance both for pure and applied research advances. From the pure scientific investigation point of view, these calculations are, in fact, bound to the properties of the electron distribution around the nuclei and therefore are a severe test of the "ab initio" efforts to calculate electronic energies. In addition, the accurate calculations of physical observables is the key strategy for building a unified picture of the complex world of atoms and molecules[1].

From the applied research point of view, the calculation of these observables is the building material for the modelling of complex chemical systems which plays a vital role in designing several modern technological applications[2] such as laser sources, ion sources, space craft design, etc. In fact, very seldom the experiment can supply an estimate of physical observables of elementary chemical processes. This is due not only to the intrinsic difficulty of isolating an elementary process to the end of measuring its properties, but also to the fact that the range of initial conditions to be investigated in this way is rather restricted. On the contrary, for the modeling, a quite large interval of initial conditions has to be scanned when supplying physical observables.

The above mentioned intrinsic complexity of these techniques (associated with their "from first principle" nature) and the large amount of values to be calculated (due to the wide range of initial conditions) make the computational effort so heavy to render the use of parallel computing indispensible. A typical calculation of this kind is the evaluation of cross sections and rate costants for atom diatom reachers[3]. To this end use has been made in the past of suparcomputer[4] and of vector and parallel features of shared memory few processors machine. However due to the difficulty of obtaining all the time and processors needed on a large machine, our investigation has been focussed on highly parallel computers.

One of the goals of our subproject within the finalized project "Sistemi informatici e Calcolo parallelo" is, indeed, that of assessing the advantage of using non conventional architectures. For this reason we have ported the version of a reduced dimensionality program calculating atom diatom reactive cross sections working on a parallel IBM 3090 to a Meiko Computing Surface. The aim was to single out the features of distributed memory concurrent processor machines more important to exploit the potential parallelism of reactive scattering codes in particular.

The report is organized as follows: section 2 presents the Meiko Computing Surface architecture, section 3 describes the major CSTools's components, section 4 analyses the structure of the IOSA code, sections 5 and 6 illustrate the code restucturing strategy to running the code on a distributed parallel architecture and section 7 shows the speedups and efficiency values obtained after restructuring.

2. Meiko Computing Surface architecture

The Computing Surface exploits the concepts of parallel processing and advanced VLSI technology to provide users with a range of machines from entry level systems with a few processors delivering a few megaflops, up to systems with some hundreds, or even thousands, processors providing a supercomputing level of performance.

As illustrated in some details in figure 1, the Computing Surface is a distributed memory machine made of processors having their own local memory and working independently on their own programs and data. The communication between different processors occurs through the passing of messages. This kind of system architecture is based on CSP (Communicating Sequential Processes)[6] computational model described by Hoare. In the Hoare model the computation is seen as the output of an interaction between a set of processes communicating through point to point channels. The hardware implementation of this model is often able to go over the major problem of shared memory MIMD architecture, that is the communicating bus saturation when adding new processors[5]. The INMOS transputer and transputer based-machines are well suited hardware architectures for reproducing this computational model. The Computing Surface, a transputer-based machine integrated with Intel i860 nodes and a Sparc node, is a typical example of these architectures.

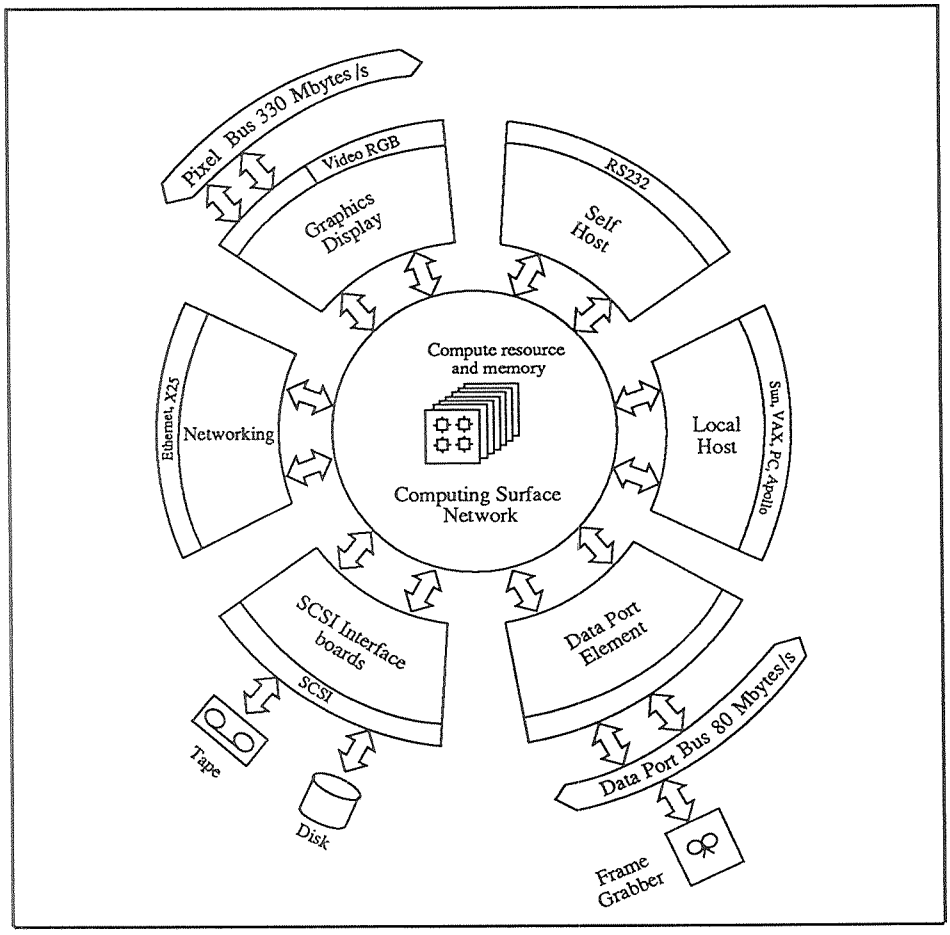

Figure 1: Computing Surface architecture

The Computing Surface can be provided as a set of modules, each of them containing either 10 or 40 boards. Each board is based on transputers used as processors dedicated to both communications and scalar computations. Each board may be dedicated to pure scalar computations, in which case it is made only of transputers; vector boards can be assembled by integrating transputers with Intel i860 chips. Specialized boards for parallel high performance I/O, display elements, frame grabber elements etc. are also available. Multiple modules can be connected in such a way that it is possible to assemble achitectures with teorically unlimited computational power.

The Computing Surface is not only designed to be a compute-intensive machine, but also to be both independent from, and adaptable to, requirements of any particular application. This adaptability is based on the idea that the Computing Surface does not need to have a fixed network topology. It is, in fact, designed to provide an adjustable interconnection topology between computational nodes, by making use of Meiko switching chips and an electronic configuration tool. Another important features of the Computing Surface is the System Supervisor bus, a global communication bus orthogonal to the inter-processor message routing network. Its function is to support the system housekeeping with an alternative way to the transputer link, by allowing a direct control over individual processing elements.

The building block of the system is the INMOS transputer, a RISC processor which in the T800 25MHz version delivers a power of 12 MIPS RISC and 1.5 Mflops on floating-point computation. It provides 4 bidirectional links for point to point communications with other transputers in the network. It can be profitably used for applications showing non-vector features. To achieve better performances on vector application, the i860-based Meiko board has to be used. This board, as described in figure 2, is based on two computational elements (seen as nodes by the remainder of the network). Each node is made of a i860 processor having a peak performance of 80 Mflops, and two transputers dedicated to communication services only. Transputers and i860 communicate via an up to 32 Mbytes shared RAM memory. Communications between nodes are supported by transputer links.

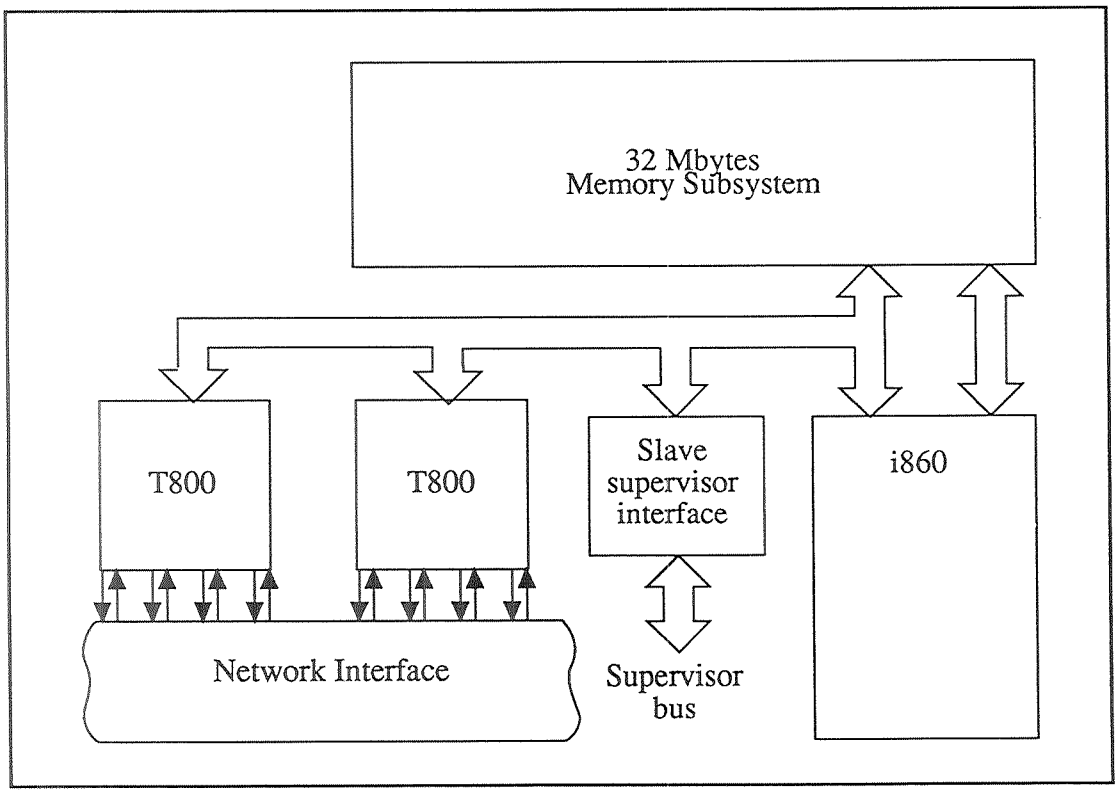

Figure 2: Meiko i860-based computational element architecture

To manage the whole system, a board based on a standard SPARC processor is used. This is able to provide a standard operating environment (SunOS), and make use of all the software running on it. It can also be integrated with parallel applications (by using CSTools) as well as with software development and debugging environments.

3. CSTools - a parallel programming tool

CSTools (Communicating Sequential Tools)[8] is a program development toolset for multiprocessor computer systems. It supports the programming of single and multi-processor applications using familiar development environments (UNIX, VMS) and standard languages (C, FORTRAN, PASCAL, LISP). It consists of cross development tools, compilers, configuration systems, and runtime facilities such as high level communication services and symbolic debuggers. CSTools is not a new parallel operating system; it yet provides a set of tools and facilities allowing cross-development of codes for parallel machines.

Parallel programming in CSTools is based on the CSP model, which structures a single application as a set of ordinary sequential programs, written in standard languages. They exchange data and synchronize the work being done by means of message-passing library routines. These routines are independent both from processor and communication hardware.

One of the most important feature of CSTools is the fact that it allows the programming of a large variety of hardware by using exactly the same techniques and the same source code. As an example a single multiprocessor application can run entirely on a network of transputers. By making small changes to configuration parameters and recompiling the source code for the proper processor without any alteration the same application can be run on a real network of transputers i860 or on a single SUN workstation, whose UNIX multiprocessing simulates concurrency and makes the parallel program development completely independent from specific parallel hardware.

The CSTools major components provide several distinct types of functionality:

- Communication Services (CSN);
- System services (RTE);
- Configuration tools;
- Runtime development tools.

CSN (Computing Surface Networking) communication services provide cushioning between the programmer and the hardware by presenting to the user a clean high level model of the parallel machine. The programmer interface to these services is a number of function libraries allowing an interprocess communication in the network regardless of whether a direct physical link can be established between related processors. The interprocessor communication is maneged by an efficient message router, providing message retransmission, multiplexing and buffering. Under CSN the programmer sees the hardware as an idealised, fully connected and homogenenous system. The porting of the code to a different machines very often requires only a recompilation and a linking to the library. The mechanism which underlies all CSTools communication services takes the form of a background process.

RTE (Run Time Executives) provides the standard operating system services to all nodes and processes of the parallel resource. These services can be local services, memory allocations or more often remote services (e.g. file and screen I/O). This is also taken care

by a background process allocated to the processors requiring RTE and having a size proportional to the kind of services requirements.

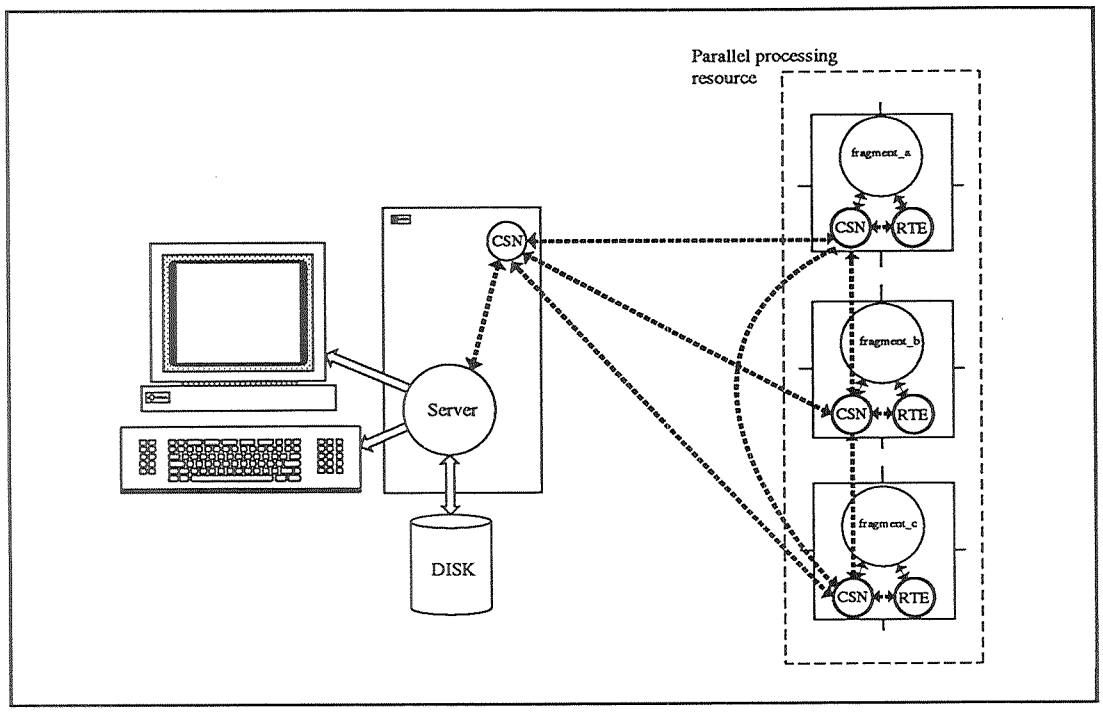

Figure 3: User processes and system services with CSTools

Configuration tools allow the programmer to describe the structure and the layout of the parallel application, to state where each process will run and how the processors will be connected. All these details can be supplied by editing a configuration file or alternatively by using the power and the flexibility of the CSbuild C function library that provides the user with the possibility of building his own parallel application loader.

An example of configuration file that loads an application consisting of 10 processes (one graphics, one master and 8 slave instances) running on different hardware processors, and chooses a binary tree in connection between the processors is given in the following scheme.

```
PAR
  processor 0 (proc_type sun4) graphics
  processor 1 (proc_type T8) master
  processor 2 for 8 (proc_type i860) slave
  networkis  binary tree
ENDPAR
```

Runtime development tools are particularly important during the construction of parallel programs. An important part of CSTools is TDB, an implementation of the standard UNIX dbx symbolic debugger. When debugging multi-process applications, a separate tdb has to be invoked for every process of interest.

4. Structure of the IOSA code

Nowadays the only pratical way of computing cross sections of elementary atom diatom reactions is by solving the nuclei Schrodinger equation under some dynamical

constraints. In this way the mathematical problem reduces to the solution of a set of differential equations in two variables. One of these approaches is the infinite Order Sudden Approach (IOSA) that approximates the cross section as an integral of fixed angle contributions. Details of the reduced dimensionality equations and of the numerical techniques for calculating the cross section and its fixed angle of approach components are given in appendix A.
The goal of the IOSA code, whose profile is shown in table 1, is the calculation of the fixed collision angle contribution to the reactive cross section for a given value of traslation energy.

Application Name	IOSA
Load Module Dimension	13 Mbytes
Lines of Code	2500 FORTRAN Statements
Number of Subroutines:	40

Table 1: Application profile

The structure of the original version of the application[4] is shown in figure 4 using a pseudo-Fortran code.

SECTION I

LOOP on *sectors*

Calculation of energy independent quantities needed for the propagation

END of the *sector*-loop

SECTION II

LOOP on *energies*

LOOP on *l-values*

SUBROUTINE CSL (arg1,.....argn)
LOOP on *arrangement channels*
LOOP on *Polar-sectors*
Step propagation
END of the *Polar sector*-loop

LOOP on *Cartesian-sectors*
Step propagation
END of the *Cartesian sector*-loop
END of the *channel*-loop
Asymptotic analysis and S matrix elements evaluation
END CSL

END of the *l-value* loop

Evaluation of fixed angle reactive quantities

END of the *energy*-loop

Figure 4: Structure of the IOSA application

In the first section, energy independent quantities needed for costructing the coupling matrix elements are evaluated for each point of the reaction coordinate grid. This section consists of one principal DO-loop running over all sectors. The second section of the program is devoted to the propagation of the solution through all sectors from the origin to the asymptotes and to the evaluation of the final quantities for an array of energies. This section contains four nested main DO-loops. The outer DO-loop runs over the specified energy values. The next internal DO-loop runs over the quantum number *l*.

The innermost DO-loop runs over the various steps of the propagation. Inside this DO-loop several calls to the routines performing different matrix manipulations (add, multiply, transpose, invert) are made. A higher level DO-loop repeats the propagation on both reactant and product channels. This double level DO-loop structure is incorporated into a single routine (CSL) making the structure of the code higly modular. A list of the main routines called and of the tasks they accomplish is given in table 2.

COOLEY	calculates the sector vibrational eigenfunctions;
MATINV	inverts complex matrices;
MMUL0	performs a matrix times matrix product;
MMUL1	performs a transposed matrix times matrix product;
MMUL2	performs a matrix times transposed matrix product;
OVLP	evaluates the overlap matrix between different sets of vibrational eigenfunctions;
POT	evaluates the potential energy value;
RMATRIX	propagates the R matrix along the potential energy channel;
SDMATQ	solves a system of real linear algebraic equations;
TQL2 and TRED2	find eingenvalues by tridiagonalizing the interaction matrix;

Table 2: Tasks accomplished by principal routines

A previous execution time analysis performed on IBM 3090-VF mod. 18E[4] has shown that the most time consuming part of the program is CSL. The routine performing the integration of the differential equation on the two reaction channels for a given value of the angular moment *l* and total energy E.

The dimension of the data structure depends on the size of the problem choosen for each specific run. When the reaction channel is segmented into 400 sectors about 4Mbytes of RAM are occupied if the basis set has dimension NV=15 and about 12Mbytes when NV=30.

5. Problem definition

As already mentioned the execution of the IOSA program needs the use of high peformance computing systems. The porting of the IOSA program on a MEIKO Computing Surface is the first implementation of the code on a massively parallel architectures, therefore it is interesting to discuss in detail the features of the software development environment of such a computer .

Execution times of the IOSA code measured on a IBM 3090-VF mod. 18E are shown in table 4 . The IOSA version "Vector + ESSL" was obtained substituting the most frequently called routines: MMUL0, MMLU1, MMLU1, TRED2, TQL2 and SDMATQ, with equivalent ESSL (Engineering and Scientific Subroutine Lybrary) routines[11].

Processor IBM 3090-VF mod.18E	CPU time (seconds)
Scalar version Vector version Vector + ESSL version	458.57 347.72 287.32

Table 4: Time consumption for single energy IOSA runs

The IOSA code was ported on a IBM multiprocessor VF-400. Parallel speedup were measured both for implicit and explicit parallelization. The first one was obtained by compiling the code with the option "AUTO" without parallelization primitives. The second one was obtained by using the parallelization primitives and a coarse-grain parallelism. The coarse-grain parallelism, being best suited for those sections of the program which are large enough to render negligible the syncronization times overhead, was applied to the CSL routine. Related time consumptions, speedups and processor efficiency are shown in table 5.

IOSA Version Vector+ESSL				
Processors	Elapsed times in seconds (implicit parallelism)	Elapsed times in seconds (explicit parallelism)	Speedup	Efficiency
1	335.78	329.38		
2	342.11	215.05	1.53	0.76
3	341.39	176.59	1.86	0.62
4	339.38	219.47	1.5	0.37

Table 5: Time consumptions for parallel runs in stand-alone mode

It is worth noticing from data reported in the table, that implicit parallelism leads to no time saving. It has also to be noticed that the elapsed times measured for explicit parallelism runs worsens when using 4 processors because the IBM 3090 VF-400 used had 3 VF only and calculations on different processors are syncronized at each batch of l value.

6. Code restructuring for parallel processing

The porting and the parallelization of the IOSA application on the Meiko parallel system have been carried out in four different steps:

a) recompilation and execution of the code on a single transputer;

b) analysis of the application structure and selection of the optimum parallelization strategy;

c) definition and implementation of the software structure to support the parallelization strategy;

d) parallel execution and analysis results.

a. Porting the code on a single processor

The compilation process to execute the code on a single node has needed only two main changes: one about the timing routines, the other to substitute the Fortran statements NAMELIST not supported by the Meiko compiler version used at this step (the actual version of the Meiko Fortran compiler supports this function). However, the porting on a single processor has confirmed that both the application and the Meiko Fortan compiler follow the standard specification of the Fortran language.

The available system configuration uses INMOS Tranputer T800 with 4 Mbyte RAM and Intel i860 with 8 Mbyte RAM. This allowed to run IOSA in sequential mode with a basic set of NV=15 and 400 sectors.

b. Parallelization strategies

As shown in figure 4, this application has a kind of natural coarse-grain parallelism. On massively parallel systems the use of the "coarse-grain" parallelism leads to an optimum ratio value of the computing-time/communication-time. Therefore, it was obvious to choose the coarse-grain parallelism as parallelization strategy.

The program execution time analysis indicates that only about 15% of the overall cpu time is spent in the section 1. Therefore, our effort was mainly focussed on reorganizing the second section of the program.

As for the IBM3090 it has been parallellized the DO loop running on the l quantum number. In fact the outermost DO-loop (running over the energy) might be too small (even one energy value), while the inner DO-loop over sectors is strictly sequential. On the contrary each iteration on l is independent from the others, and its time consumption is quite heavy. As will be discussed later on, both characteristics are very important also for the application scalability when the number of used processors increases.

The parallelization has been structured using a master-slave model with dynamic allocation of work. The model consists of a set of slave processes, all executing the CSL subroutine, and a master process that dynamically send to the slaves a quantum number l to start their execution. Whenever one of the slaves sends back to the master the result of its processing, the master process controls the convergency of the solution. If the convergency is not reached the master process sends to the free slave further work. When convergency is reached the master process ends the execution by stopping the work on the active slave processes.

c. Definition and implementation of the parallel structure

The development environment on the Meiko Computing Surface is CSTools. By using CSTools the programmer sees only the logical configuration choosen to perform his parallel application. He needs not to know the physical message-passing organization through the communication network. Moreover CSTools provides the possibility of defining mixed programming architectures allowing a structuring of the parallel code indipendent from the kind, the number and the nodes configuration, (both transputers and i860 processors can be used). Accordingly the code was first parallelized on a transputer based Computing Surface.Then it was run on a i860-based Computing Surface without any change of the software managing parallelism.

Figure 5 shows the master-slave logic configuration choosen to parallelize the application, while figure 6 shows the network configuration adopted. The network configuration is a ternary tree. Since each transputer has 4 communication links a ternary tree can be easily and efficiently mapped on the hardware architecture.

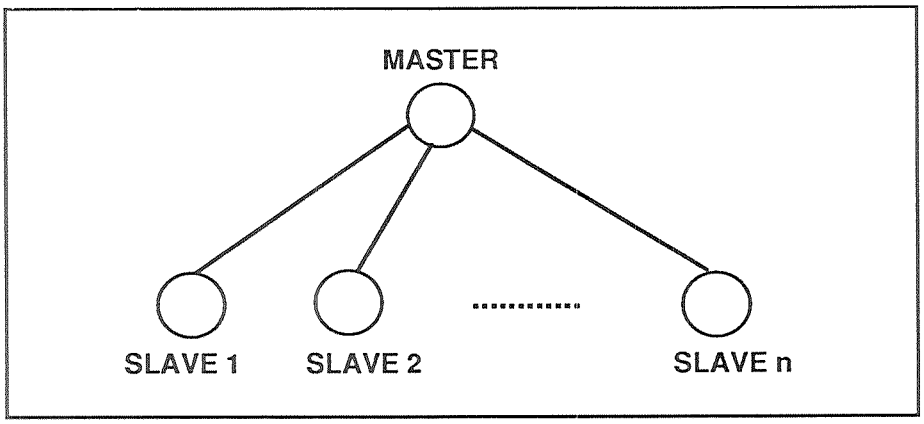

Figure 5: Data parallel master-slave model

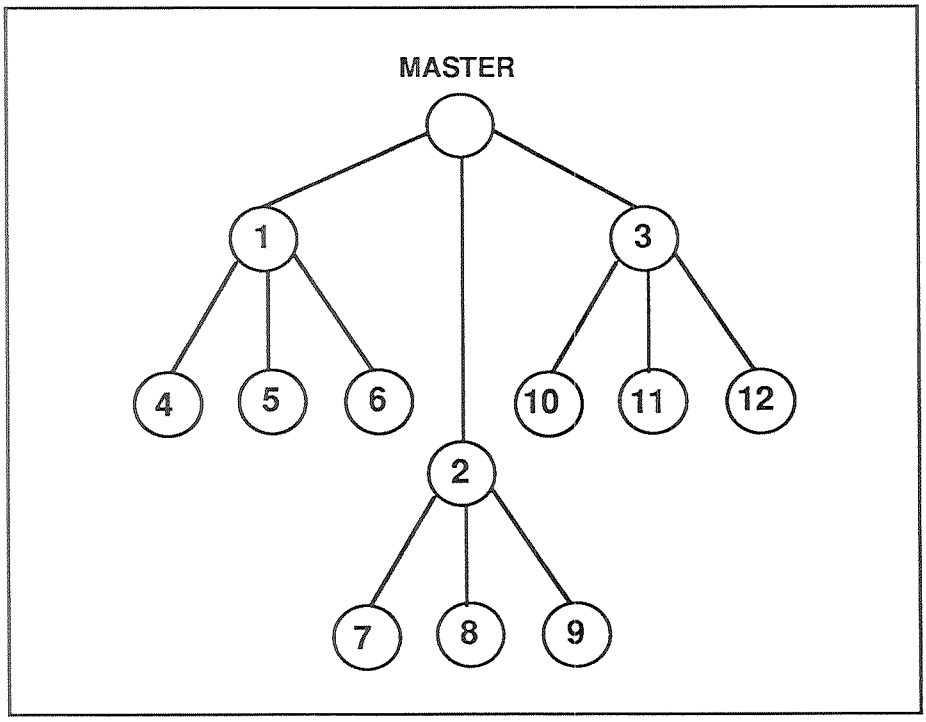

Figure 6: The adopted configuration

In those cases in which the communication links of the physical configuration do not match with the communication channel pattern, the message routing is arranged by CSTools. As a result the routing is trasparent to the programmer.

In figures 7 and 8 the master and slave process structure in which the application has been reorganized are shown.

```
Master Process

    Execution of the section 1
    Send constant data structure to the slave processor
    Quantum number allocation to the slave processor

    DO-loop on l-values

        Receive result from slave processor
        Convergency control (if converged go to stop processors)
        Send further work to the free processor

    END DO-loop

    Stop processors
    Output of the results
```

Figure 7: Structure of master process

```
Slave process
Receive data structure from the master process

DO-loop

    Receive quantum number L
    CALL CSL
    Send to the master process the risults

END DO-loop
```

Figure 8: Structure of slave process

7. Performance analysis and optimization strategies

Two important indicators of the effective use of parallel processing are the measured speedup and the efficiency. The speedup (S) is defined as the ratio between the time required to execute a given calculation on a single-node processor and that needed on concurrent processors. If T_s is the elapsed time of the application on a single-node processor and T_p is the elapsed time on a concurrent computer, the speedup is given by:

$$S = \frac{T_s}{T_p}$$

The efficiency (E) is defined by:

$$E = \frac{\text{Speedup}}{\text{Number of processors}}$$

and can be used to compare different parallel version of the same program.

As already mentioned the present investigation has been performed using two different configuration of the Meiko Computing Surface system. The first one has 25 Mhz transputer processors, each one with 4Mbyte of local RAM and with 6 MIPS VAX and 1.5 MFlops of peak performance. The second one has both transputer (for message routing) and i860 nodes. In our job the i860 processor has been used only in scalar mode, because the Intel vector compiler is not yet available. On experimental test base Meiko corporation states that it is possible to get a factor speedup of 2-3 by using the new vector compiler.

In table 5 are shown the execution times of the original version (problem size NV=15 and 400 sectors) performed on single transputer node and i860 node.

Executin times	T800	i860
IOSA (section 1 and 2)	13128,05	1722
IOSA (section 1)	2006,3	610
IOSA (section 2)	11121,75	1111

Tabella 5: IOSA execution times

To give a more appropriate evaluation of the limiting speedup reachable on the Meiko machine execution times limited to section 2 (subroutine CSL) are shown in table 6.

Number of processors	Subroutine CSL execution time (in seconds)	Speed-up	Efficiency
1 + 2 i860	595.84	1.86	0.62
1 + 4 i860	315.1	3.52	0.70
1 + 7 i860	181.59	6.02	0.76
1 + 16 i860	88.3	12.5	0.74
1 + 2 T800	5496.16	2.02	0.63
1 + 4 T800	2780.52	3.99	0.79
1 + 7 T800	1514.96	7.34	0.91

Tabella 6: CSL exection times

In the table the notation "1+n processor" means a configuration of processors consisting of a master processor and n slave processors.

Since the computational workload of the master node is poor, it can be allocated jointly with a slave process on the same node. That would lead to an insignificant increase of the execution time giving in return a speedup and efficiency increase because of the reduction of the processors involved in the program execution.

Judging from our results, the application presents a good scalability. A further increase of perfomance could be obtained by parallelizing the section 1.
For further comparison, figure 9 shows histograms for the execution time of the CSL routine on both IBM 3090 and Meiko Computing Surface systems.

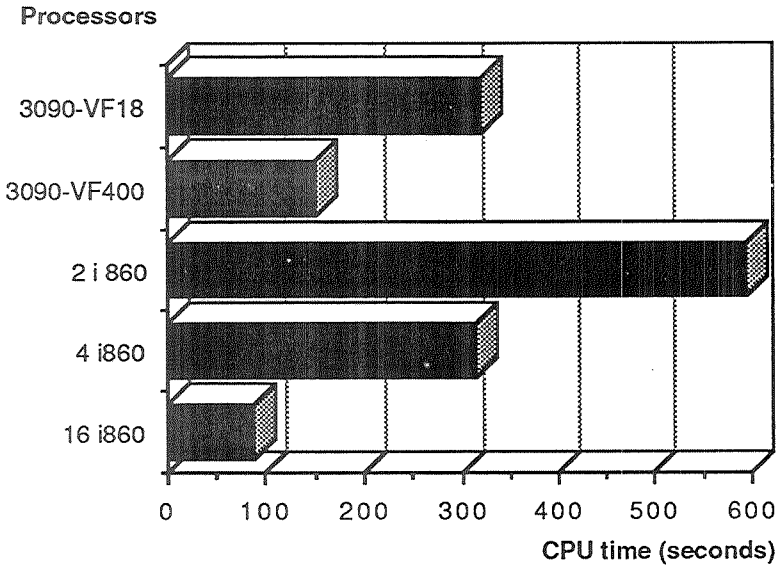

Figure 9: CPU times spent to run the subroutine CSL

8. Conclusions

In this paper we have investigated the restructuring of a typical intensive computer application on a massively parallel architecture. By using the execution times analysis of the application made on IBM 3090-VF mod.18E we have reorganized the application in according to a master-slave model .

The compilation process of the sequential version of the application has not required particular code modifications pointing out that the Meiko Fortran compiler works according to standard specifications.

The parallelization of the code has required the restructuring of the program for introducing the explicit parallelism management, without any alteration of the computational kernel.

As expected, the parallel version of the application, obtained by using CSTools, is indipendent from the number of processors, from the kind of processors (transputer or Intel i860) and from the network topology.

Speedups and efficiency measured for the IOSA code on the Meiko Computing Surface are excellent confirming that when coarse-grain parallelism can be exploited on highly parallel distributed memory architectures leads to top performances and unique price/performance ratios.

9 Appendix A: the theoretical approach

The most popular way of reducing the dimensionality of an atom-diatom reactive scattering problem is to apply both the energy sudden and the centrifugal sudden decoupling schemes (the so-called Infinite Order Sudden Approach - IOSA)[10]. In this way the exact three mathematical dimension Schroedinger equation is reduced to a set of fixed collision angle γ two mathematical dimension ones. These equations read as

$$\left[-\frac{h^2}{2\mu}\left(\frac{1}{Q_\lambda}\frac{\partial^2}{\partial Q_\lambda^2}Q_\lambda+\frac{1}{q_\lambda}\frac{\partial^2}{\partial q_\lambda^2}q_\lambda-\frac{L_\lambda}{Q_\lambda^2}-\frac{J_\lambda}{q_\lambda^2}\right)+V(Q_\lambda,q_\lambda;\gamma_\lambda)-E\right]*\Phi(Q_\lambda,q_\lambda;\gamma_\lambda)=0 \qquad (1)$$

Where λ labels the type of atom-diatom arrangement ($\lambda = \alpha$ indicates the reactant channel, $\lambda = \beta$ or β' indicates a product channel), E is the total energy, Q_λ and q_λ are the properly mass scaled atom to the diatom center of mass and the diatom internuclear distances, μ is the reduced mass of the system and L_λ and J_λ are costants the orbital related to l_λ and the rotational j_λ quantum numbers respectively.

Equations (1) can be solved by dividing the fixed collision angle γ_α (Q_α, q_α) plane into two regions using a straight line. Such a line originates at $Q_\alpha = 0$ and $q_\alpha = 0$ and follows the ridge separating the entrance from the exit channel. This choice makes the correspondence between γ_α and γ_β unique and ensures the proper matching of the reactants' and products' half-channel.

To integrate equations (1) the α and β half-channels are segmented into many small sectors. Within each i sector the global wavefunction $\Phi_\lambda(Q_\lambda, q_\lambda; \gamma_\lambda)$ is expanded in terms of NV products of two terms: a propagation $\chi i_{\lambda\nu}(Q_\lambda; \gamma_\lambda)$ and a bound $\phi i_{\lambda\nu}(q_\lambda; \gamma_\lambda, Q^i_\lambda)$ function. The value of NV is chosen large enough to include for each sector all open channels and the lowest closed ones. By averaging equations (1) over q_λ, a set of NV differential equations

$$\left[\frac{d^2}{d\,Q_\lambda^2} - D^i_\lambda\right]\chi_{i\lambda\nu}(Q_\lambda;\gamma_\lambda)=0 \qquad (2)$$

is obtained. The sector matrix $\underline{D}^i{}_\lambda$ plays the role of coupling the equations for different ν values. Equations (2) are integrated from the line separating the reaction channels to the asymptotes.

For our the IOSA procedure used here (which is limited to systems having a central atom much heavier than the external ones) rectants' and products' coordinates are exchangeable ($Q_\alpha = q_\beta$; $q_\alpha = Q_\beta$; $\gamma_\alpha = \gamma_\beta = \gamma$), the value of j' is linked to that of l ($l \rightarrow j'$) and the matching at the separatrix is straightforward. By imposing the appropriate boundary conditions[8] at both reactant and product asymptotes the detailed IOSA fixed γ S matrix elements ($S_{jlvv'}(\gamma, E)$) for the reactive process can be derived. Usually, because of the nature of the IOSA approach j is set equal to zero and therefore can be dropped

from the notation from these values, the fixed γ detailed state (v) to state (v') reactive cross section ($S_R^{vv'}(\gamma,E)$) can be calculated using the relationship

$$S_R^{vv'}(\gamma,E) = \sum_l \frac{\pi}{k_v^2}(2l+1)\,|S^l_{vv'}(\gamma,E)|^2 \qquad (3)$$

where, as usual, k_v^2 is $[2m(E - Evj)/h]^2$ and E_v is the energy associated with the considered rectant vibrational state. Then the corresponding global IOS 3D reactive cross section for a given total energy can be obtained by integrating ($S_R^{vv'}(\gamma,E)$) over cos γ

$$S_R^{vv'}(E) = \frac{1}{2}\int_{-1}^{1} S_R^{vv'}(\gamma,E)\, d\cos\gamma \qquad (4).$$

References

[1] A. Laganà, Supercomputer Algorithms for Reactivity, Dynamics and Kinetics of Small Molecules (Kluwer, Dordrecht, 1989).

[2] M. Capitelli and J.N. Barsdley, Nonequilibrium Process in Partially Ionized Gases (Plenum, New York, 1990).

[3] D.M. Hirst, A computational Aprroach to Chemistry (Blackwell, Oxford, 1990).

[4] A. Laganà, O. Gervasi, R. Baraglia, D. Laforenza, Vector and Parallel Restructuring for Approximate Quantum Reactive Scattering Computer Code, in: J.L. Delhaye, E. Gelembe, (ed.), High Performance Computing (Noth-Holland, 1989) pp. 287-298.

[5] M. Vanneschi, Architetture General Purpose, in: atti convegno AICA "Elaborazione parallela", 1988.

[6] C.A.R. Hoare, Communicating Sequential Processes, Communications of ACM, vol.21, Num. 8,1978.

[7] Meiko Hardware Reference Manual, Meiko 1989

[8] CSTools reference Manual, Meiko 1989

[9] Engineering and Scientific Subroutine Library Guide and Reference Version 2, IBM Order No. SC23-0184.

[10] Pack, R. T, J. Chem. Phys. 60 (1974) 653; McGuire, P.M. and Kouri, D.J., J.Chem. Phys. 60 (1974) 2488; Kouri, D.J., in: Berustein, R.B., (ed.), Atom Molecule Collision Theory: A Guide for the Experimentalist (Plenum, New York, 1979) pp.301-358; Kouri, D.J. and Fitz, D.E., J. Phys. Chem. 86 (1982) 2224.

[11] A. Laganà, E. Garcia and O. Gervasi J. Chem. Phys. 89, 7238 (1988).