OVERVIEW OF THE KSLA EFFORTS IN PARALLEL COMPUTING

G.A. van Zee
Koninklijke/Shell-Laboratorium, Amsterdam
PO Box 3003
1003 AA Amsterdam
The Netherlands

ABSTRACT

The parallel computing research in the Mathematics and Systems Engineering department of KSLA is concerned with applications, with programming methods and languages, and with hardware and systems software developments.
The parallel computers we are interested in consist of many processors with local memory. These machines offer supercomputing capabilities at such a favourable cost that we have started research to use them in a dedicated way for applications such as on-line plant optimisation, interactive fluid flow simulation for process design, combinatorial optimisation for supply and marketing decision support, and molecular modelling for product design. An important part of our research is concerned with the development of programming methods for parallel computers and with experiments to evaluate the use of parallel programming languages. The construction of a parallel program is difficult because the computational load must be divided over the processors in a well balanced way, without causing excessive communication overhead.

Despite of the fact that parallel programming requires highly specialistic skills, it is possible to make parallel computing available to the application oriented programmer or user. The main approach we have taken to achieve this is to develop libraries of parallel routines which can be called from programs in a sequential programming language like Pascal or FORTRAN, running on a sequential host computer.

1. INTRODUCTION

In the Mathematics and Systems Engineering (MSE) department of the Koninklijke/ Shell-Laboratorium, Amsterdam (KSLA) , a parallel computing research project was started in 1985 with a one manyear per year effort. This project has now grown out to an activity involving 4 people and in the next year this will increase to 7 or 8.
The research takes place in strong interaction with other activities in MSE such as mathematical physics, operations research and process control and optimisation. Further, there are joint activities with other departments such as Engineering Physics, Equipment Engineering and Separation Technology.

The parallel computers we are interested in are so called Multiple Instruction Multiple Data (MIMD) stream machines, consisting of many processors with local memory

which communicate by message passing over a network. An important property of these machines is that they can easily be made more powerful by increasing the number of processors and extending the communication network.

The objective of parallel computing research is to achieve very high speed computing in order to be able to solve larger problems than today in an acceptable amount of time. For many of our application areas (on-line plant optimisation, interactive fluid flow computation for equipment design), response time is so critical that a batch mode of operation (the common practice with conventional supercomputers) is not appropriate. Perhaps equally important as processor performance is the cost of a computer. By bringing down the costs, parallel computing leads to a dedicated interactive style of supercomputing.

The fields of interest of KSLA's parallel computing research are application algorithms, programming techniques and languages, and hardware and operating systems. Although our main interest is in algorithms, at present the other activities are necessary in support of this. In the following these fields of interest are discussed in reverse order.

2. HARDWARE AND OPERATING SYSTEMS

The progress in VLSI technology (miniaturisation, more efficient lay-outs, faster materials, three-dimensional chips) continues to enable the construction of faster computers. Parallel computers with the MIMD architecture take optimal profit from these developments, due to their scalability. In this context, hardware scalability is achieved by consistently applying the concept of locality: most operations that a processor chip performs require little or no long distance communications between the chip and its environment. As a consequence, further miniaturisation can be applied to the same chip: a corresponding increae in chip performance will result as the long distance communications will not become a structural bottleneck. The on-chip memory and floating point hardware of the T800 Transputer are a good example of this form of scalability. Examples of non-scalable architectures can be found in computers with vector co-processors and in shared memory computers. There is a second aspect of hardware scalability that MIMD architectures take full advantage of: an increase of the number of processors leads to an identical increase of the memory bandwidth and the inter-processor communication bandwidth.

In 1987 KSLA purchased a Transputer network with 40 processors, connected to an IBM-PC AT which is used for program development. Other computers (Sun, Vax) can also be used as a host machine.

Via our contacts with the BRC laboratory in Houston and with the Philips Natlab in Eindhoven, we have also access to a large Ncube and a forthcoming DOOM computer.

In the KSLA approach to parallel computing, systems software should provide support for program development and abstraction from the hardware. Time-sharing facilities to allow more than one user to run his parallel program on the same system are conflicting with the philosophy of dedicated computing.

To allow for abstraction from the architecture of a specific parallel computer, it is useful to have an infrastructure for the routing of messages over a network, for dynamic process creation and for domain decomposition. To provide full communication on Transputer networks, an Occam preprocessor has been constructed which transforms a given application program and adds the routing and multiplexing processes required. These additional processes are guaranteed to be deadlock free (1). Further, a limited form of process creation has been implemented on a Transputer network. Application programs that give rise to a dynamic chain of processes, can be mapped on a static ring of processors. However, it is still unclear how the cases should be handled where the processes communicate in more complex graphs, as it occurs for instance in tree search algorithms.

3. PROGRAMMING TECHNIQUES AND LANGUAGES

An important part of KSLA's research into parallel computing is concerned with the development of programming methods for parallel computers and with experiments to evaluate the use of parallel programming languages. The construction of a parallel program is difficult because the computational load must be divided over the processors in a well balanced way, without causing excessive communication overhead. Further, the correctness of the program (for instance the absence of deadlock) must be demonstrated.

At the present time, a single universally accepted method for parallel programming does not yet exist. At KSLA the preferred approach is to start with a functional specification, i.e. to formulate the problem to be solved by means of a formal logical expression, the postcondition. From the postcondition one can try to derive a parallel invariant, i.e. an invariant which contains no unwanted sequential restrictions. Like in the Gries-Owicki approach, this invariant expresses the properties of the parallel processes involved. By writing the invariant as the

conjunction of a set of local invariants, the exploitation of parallelism can be studied before an implementation is made. This study includes an analysis of the load balancing properties and the communication requirements. Finally, to show that deadlock cannot occur, simplifying program transformations can be used. These transformations never introduce or eliminate deadlock themselves, and they end up in a trivial program.

An important property of well-designed parallel algorithms that distinguishes them from algorithms for sequential or vector computers is their software scalability. In the software context, scalability means that the execution speed of algorithms depends linearly on the number of processors that are used. There are two important conditions for achieving software scalability: program texts have to be independent of the number of processors and the induced processor idle time due to load imbalance and communication overhead must remain small as the number of processors increases. The advantages of the combined hardware and software scalability are that computation times can be shortened and problem sizes can be increased according to the needs of the application. Unfortunately it depends upon the application how difficult it is to achieve software scalability. It is (and will remain for a long time) a programming problem as compilers and operating systems are not capable of dealing with efficient parallelisation.

Due to the many difficulties described above, parallel programming is considered as a specialism within computing science that within Shell will not be practised by application programmers in the foreseeable future. Despite this it is possible to make parallel computing available to the application oriented programmer or user. The approach we have taken to achieve this is to develop libraries of parallel routines that can be called from programs in a sequential programming language like Pascal or FORTRAN, running on a sequential host computer. Computational tasks that form the bottleneck in solving a problem (e.g. repeated solution of large systems of equations) are executed by a library routine running on a parallel computer, which returns its results to the host computer.

A first library that has been constructed in this way is a parallel linear algebra library (3). It is written in Occam and runs on a Transputer network. From a main program running on the host, a library function can be called on the parallel computer. Currently, we are testing language interfaces between Pascal and FORTRAN on the host computer and Occam on the parallel computer.

One of the important issues in the construction of a parallel linear algebra library is the choice of the process structure. The structure that has been chosen allows for simple functional specifications of the processes. The functional specifications of the library processes are given in such a way that they do not reflect any details of

the parallel implementation. They do not contain anything about communication between process instances. In the construction of the parallel library, it is an aim that the functional specifications of the linear algebra processes can be readily understood by a layman in the field of parallel computing. Controlability is achieved by requiring that the library processes can be verified in isolation. Compositionality is reached by imposing strict and uniform interface conventions on the library processes. One important aspect is the distribution of matrices and vectors across the processes.

Currently, the library contains routines for basic matrix and vector operations, LU decomposition with pivoting, QR decomposition and, by making use of these respective decompositions, linear equations solving and least squares solving. These routines are for dense matrices. A start has been made with the construction of the similar routines for general sparse matrices and structured sparse matrices such as band matrices. These types of matrices are most relevant in applications in linear programming and in differential equations solving, respectively.

The library approach is not suitable for applications where computation intensive parts cannot be concentrated in parallel routines, but rather appear everywhere in the application. This is for instance the case in the cellular automata implementation which is discussed in the next section (2).

4. ALGORITHMS

Currently, our major application areas are large scale scheduling of plant operation by using linear programming techniques and computational fluid flow. The scheduling problem studied is to find for a period of e.g. a week ahead, the settings of manipulable variables such as the throughputs of units, which optimise the profit of the plant operation. The approach taken is based upon the Karmarkar algorithm (4) to solve linear programming problems. The large sets of linear equations involved are repeatedly solved using the parallel linear algebra library functions (3). The status of the work is that a first version of the Karmarkar algorithm using parallel equations solving for dense matrices has been running since 1988. A version which makes use of sparse matrices in order to solve problems of a realistic size will appear during 1989.

In the computational fluid flow area two approaches are investigated at KSLA. One approach is to use the parallel linear algebra library together with differential equations solvers (5). The status of this work is that for dense matrices a running

version will be ready within a few months. However, for many practical problems it is necessary to exploit the special band structure appearing in the matrices involved, so that we will have to develop corresponding special routines in the parallel linear algebra library.

The other approach to fluid flow computation which is studied at KSLA is the cellular automata approach.

Stimulated by the opportunities offered by parallel computing, a completely new approach to fluid flow simulation has come up under the name of cellular automata. Cellular automata were originally developed as a theory to describe the basic mechanisms underlying fluid flow phenomena. Cellular automata form a microscopic world of cells that interact only with their direct neighbour cells. Mass and momentum are represented by the presence of a discrete number of particles per cell, each representing momentum in one of the directions along the grid.

In each of the cells particles can collide, redistribute their momenta and then move to a neighbour cell. This happens in a state transition for all cells at each step.

The theoretical research is directed at the discovery of the simplest interaction rules that satisfy on a microscopic scale the fundamental conservative equations and that show on a macroscopic scale the correct fluid flow behaviour. The formulation of fluid flow models with cellular automata is more direct and simple than the traditional approach with partial differential equations. Other advantages of cellular automata are the high speed and low cost of the computations on parallel computers, the unconditional numerical stability, and the flexibility to handle complex boundary conditions occurring in e.g. flow through porous media (catalyst bed, oil reservoir rock).

A first implementation of cellular automata on KSLA's Transputer network produced the solution of the time dependent Navier Stokes equations in two dimensions, using a hexagonal lattice. The accuracy achieved appeared to be acceptable and it was confirmed that boundary conditions with complex geometries can easily be imposed. Further, a three dimensional cellular automaton has been implemented which satisfies the Navier Stokes equations. The symmetry of the lattice has been used to reduce the size of the collision tables from 64 Mbyte to 40 kbyte of memory. To achieve this, the time to compute the collisions had to be allowed to increase by a factor 3, as compared to a full size table look-up.
The reduction of the collision table becomes even more important when two phase cellular automata are going to be implemented. Two phase flow can be simulated by

colouring the two kinds of particles. This will boost the state space of a cell from 2^{24} to 3^{24} possible states. It seems also possible to use different collision tables in order to vary the viscosity locally in the cellular automaton. This would result in a system that behaves according to some kind of turbulence model, such that experiments with higher Reynolds numbers can be done. We do not have any experience with this type of cellular automata yet. Currently, we are studying how the phenomenon of surface tension is related to the parameters of the two phase cellular automata.

At present we can simulate three dimensional flows with a Reynolds number of atmost 100. The computation times involved are still large. Over 95% of the total computation time is used for the collision of the particles. If we would equip our parallel machine with dedicated processors to perform the collision of the particles, a three dimensional two phase flow with a Reynolds number of well above 1000 could be simulated explicitly within a few hours of computation time (2).

Besides the scheduling and fluid flow applications, some early experience has been obtained with simulated annealing, a novel approach to solve integer optimisation problems (6). This kind of problem occurs in decision support systems for vehicle scheduling and for the allocation of resources in distribution networks for products such as gasoline.

In physics the process of annealing is known as the cooling of a liquid towards the solid state, such slowly that a perfect crystal structure establishes. An analogy with this physical process provides a framework for the solution of large integer optimisation problems.
When solving an integer optimisation problem with simulated annealing, first a set of states is defined, such that every possible solution of the integer optimisation problem corresponds to a different state. If the problem is non polynomial, the number of states in this set is exponentially large. Next a cost function is defined on the states, representing the costs of the solutions of the original problem. The aim is to find a state with an acceptably low cost, by evaluating the cost for only a polynomial number of states.
This is done by defining a neighbour relation between states, such that the diameter of the corresponding graph is only of polynomial length. Now the optimal state is searched by traversing a path through this graph, starting at an arbitrary initial state. In the beginning of this traverse not only steps towards neighbours with lower costs, but also towards neighbours with higher costs can be made, depending on a temperature control parameter T. But as the traversed path becomes longer, T is decreased, so that the probability of accepting a step towards a neighbour with

higher cost is decreased. Finally, if T approaches zero, only steps towards neighbours with lower costs are accepted, and a local optimum is found.
Whether this local optimum is the desired global optimal state, depends on the defined neighbour relation, the length of the traversed path, the cooling schedule used, and the probability density function for accepting states with higher costs. The results published thusfar look promising. The research at KSLA aims at finding a suitable parallel annealing algorithm, which is for some applications a difficult task.

Finally, there exist plans to start research into the design of parallel algorithms for molecular dynamics calculations and for multi component phase equilibria calculations. Both subjects require enormous computational efforts and are of significant importance for KSLA's product and process oriented activities.

5. CONCLUSIONS

Parallel computers consisting of many processors with local memory, interconnected by a communication network, can offer supercomputing capabilities at a relatively low cost and can handle problem types for which present day supercomputers are ineffective. However, in order to use such machines it is required to develop many new parallel programs. Parallel programming involves the solution of problems like workload balancing and will remain the domain of computer scientists for some time to come.

The KSLA approach to applications is to redesign algorithms fundamentally where appropriate and to provide interfaces to existing sequential programs. The first application areas considered are large scale plant scheduling and computational fluid flow. The results obtained so far look very promising and can be scaled up with the number of processors to achieve a break through in the application domains involved.

REFERENCES

(1) L.D.J.C. Loyens,
Full communication on a network of Transputers, Master thesis Eindhoven University of Technology, 1987

(2) J.A. Somers,
A parallel cellular automata implementation on a Transputer network for the simulation of small scale fluid flow experiments, these proceedings, 1988.

(3) J.G.G. van de Vorst,
A parallel implementation of the Karmarkar algorithm using a parallel linear algebra library, these proceeding, 1988.

(4) I. Adler, N. Karmarkar, M. Resende and G. Veiga,
An implementation of Karmarkar's algorithm for linear programming, Report ORC 86-6, Operations Research Center, University of California, Berkely, 1986.

(5) W.F. Ames
Numerical methods for partial differential equations, Academic Press, New York, 1977.

(6) P.J.M. van Laarhoven, E.H.L. Aarts
Simulated annealing: Theory and Applications
Reidel Dordrecht, 1987.