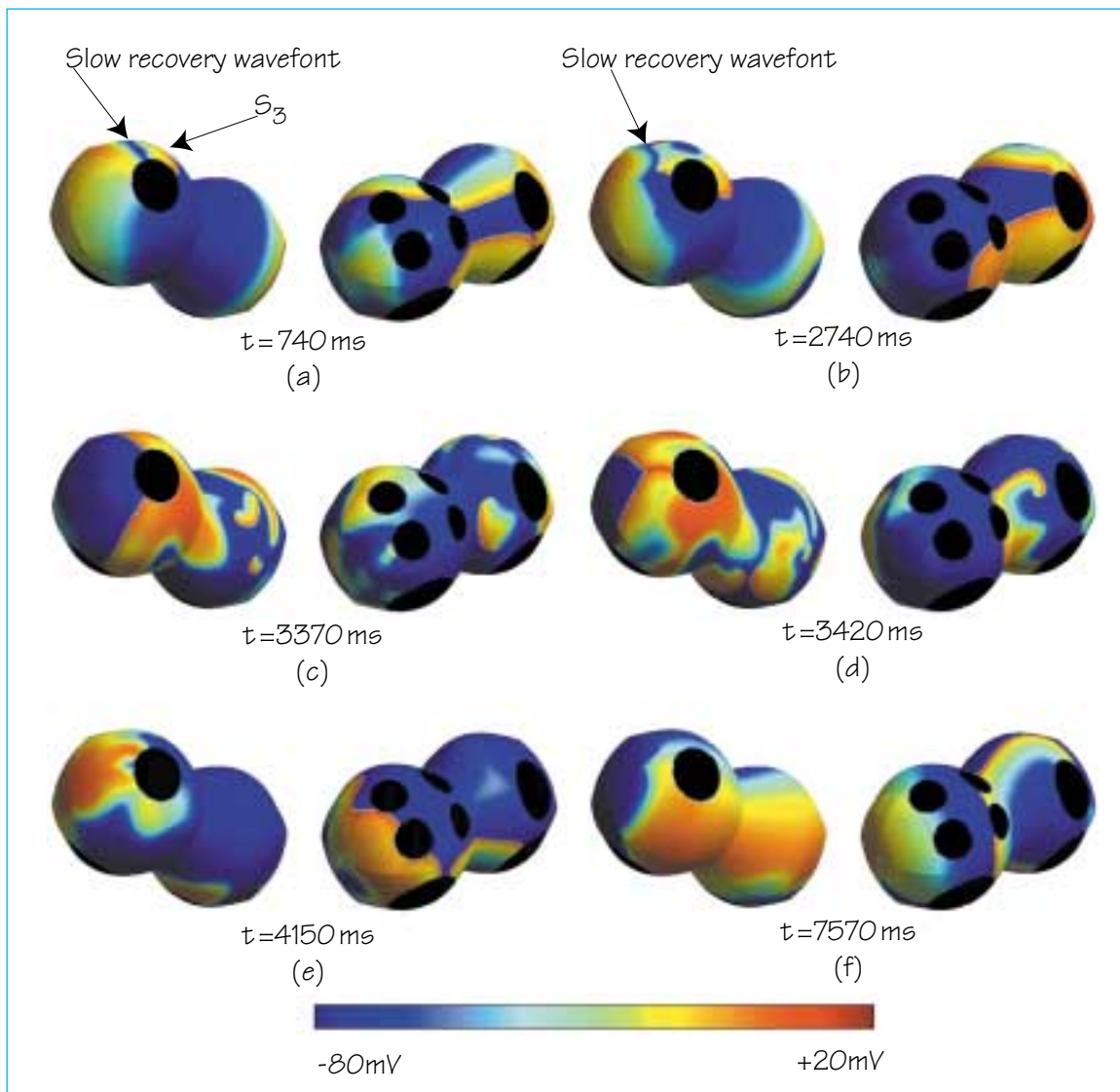


EPFL SUPERCOMPUTING REVIEW

NUMERICAL SIMULATION FOR SCIENCE AND TECHNOLOGY



Initiation of a nonsustained atrial fibrillation. First, one normal stimulus is applied followed by two ectopic beats S_2 and S_3 . We can see the evolution of the action potentials: (a) $t=750\text{ ms}$, shortly after the application of S_3 which collides with the slow recovery wavefront created by S_2 ; (b) $t=2740\text{ ms}$, just at the onset of atrial fibrillation; (c) $t=3370\text{ ms}$ and (d) $t=3420\text{ ms}$ show the atrial fibrillation with up to 6 independent wavelets, note the appearance of u-turns and anatomical reentry around veins; (e) $t=4155\text{ ms}$ transition between atrial fibrillation and atrial flutter, note the anchoring effect around SVC; (f) $t=7570\text{ ms}$ atrial flutter with a periodic pattern. (see article on page 32)

Contents

Table des matières

Parallel computer architectures for commodity computing and the Swiss-T1 machine <i>Architectures de machines parallèles construites avec des matériels standardisés et l'ordinateur Swiss-T1</i> Pierre Kuonen & Ralf Gruber	3
Communication Libraries for the Swiss-Tx Machines <i>Les bibliothèques de communication pour les machines Swiss-Tx</i> Stephan Brauss	12
Parallel File Striping on the Swiss-Tx Architecture <i>Entrées/sorties parallèles sur les disques locaux de l'architecture Swiss-Tx</i> Benoit A. Gennart, Emin Gabrielyan & Roger D. Hersch	15
A Parallel Discrete Element Method for Industrial Granular Flow Simulations <i>Une méthode d'éléments discrets parallélisés pour les simulations d'écoulements granulaires industrielles</i> Mark L. Sawley & Paul W. Cleary	23
Performance test of the SPECULOOS code on the T0-Dual parallel machine <i>Test de performance du code SPECULOOS sur l'ordinateur parallèle T0-Dual</i> Daniel Weill	29
Computer simulation of atrial arrhythmias <i>Simulation informatique des arythmies auriculaires</i> Olivier Blanc, Jean-Marc Vesin, Nathalie Virag, Olivier Egger, Jacques Koerfer & Lukas Kappenberger	32
IP3T a performance prediction tool for irregular parallel programs <i>IP3T, un outil de prédiction de performance pour applications parallèles irrégulières</i> Michel Pahud	36

*For more information about CAPA:
Pour plus d'informations sur CAPA
<http://capawww.epfl.ch/>*

Editorial

Depuis quelques années, l'idée d'assembler des machines destinées aux calculs à haute performance à partir de composants standard du marché est devenue attrayante dans la communauté des scientifiques. Ces machines devraient rivaliser en performance avec les super-ordinateurs actuels et offrir un rapport prix-performance plus avantageux. Ce défi a donné naissance à de nombreux projets dans le monde et parmi ces projets, il y a Swiss-Tx dans lequel l'EPFL participe intensivement. En effet, deux machines prototypes sont déjà à l'EPFL et une nouvelle machine pourrait être installée à la fin de cette année (page 3). Bien que les machines Swiss-Tx soient assemblées à partir de stations de travail, elles possèdent néanmoins un réseau de communication développé en Suisse qui supporte une implantation de la bibliothèque standard de communication MPI* (page 12) et un module NAFS de fichiers répartis utilisé pour les entrées-sorties parallèles (page 15). Les performances de deux des applications tournant sur ces machines sont montrées dans les articles des pages 23 et 29.

Par la nature multidisciplinaire de cette revue, nous vous proposons deux articles qui à notre avis méritent d'être soulignés. Le premier article concerne une simulation informatique réussie des arythmies auriculaires (page 32). Le deuxième article présente un nouvel outil de prédiction de performance pour les applications parallèles (page 36).

Editorial

Since a few years, building a machine dedicated to high performance computing made of computer commodities became an attractive endeavour for the scientific community. These machines should be as powerful as the existing supercomputers and should have a favourable price-performance ratio. With this challenging goal, many projects started around the world. The Swiss-Tx, in which EPFL has an intensive participation is one of these projects. Two prototype machines are already in use at EPFL and a new machine might be installed by the end of this year (page 3). Although the Swiss-Tx machines are built from commodity elements, they feature a Swiss-made communication network supporting MPI*, the standard communication library (page 12) and the NAFS striped files package used for parallel Input/Output (page 15). Performances of two applications running on these machines are presented in the two articles on pages 23 and 29.

As a multidisciplinary review, it is worth proposing two unrelated but interesting articles. The first one talks about computer simulation of atrial arrhythmias (page 32) and shows a successful simulation of some phenomena already observed in nature. The second one presents a new performance prediction tool for parallel applications (page 36).

* MPI: Message Passing Interface (interface pour les échanges de messages)

PARALLEL COMPUTER ARCHITECTURES FOR COMMODITY COMPUTING AND THE SWISS-T1 MACHINE

PIERRE KUONEN, EPFL, COMPUTER SCIENCE DEPARTMENT, PIERRE.KUONEN@EPFL.CH
AND RALF GRUBER, EPFL, COMPUTER SERVICES, RALF.GRUBER@EPFL.CH

La disparition, au cours des dernières années, des principaux constructeurs de superordinateurs parallèles a favorisé l'émergence d'un nouveau concept de machine parallèle basé sur l'assemblage d'ordinateurs utilisant des composants standards, tels des PCs ou des stations de travail, connectés au travers d'un réseau à haut débit et à latence faible. Dans cet article nous présentons et comparons différentes topologies envisagées pour réaliser de tels réseaux et nous justifions le choix de la machine Swiss-T1 que l'EPFL planifie d'installer à fin 1999 dans le cadre du projet Swiss-Tx. L'objectif de ce projet est de réaliser, à l'horizon 2000, une machine parallèle de faible coût délivrant un Teraflop/s.

Commodity parallel computing becomes more and more popular, as the specialised supercomputing companies stop their activity. In this new concept, a user is confronted with a machine built with mass produced fully equipped workstations or PCs which are interconnected through some high speed, low latency network. In this paper, we discuss and compare the different network topologies that are used to cluster those computational units. Then we justify the choice made for the Swiss-T1 parallel commodity computer that may be installed at EPFL by the end of 1999 in the framework of the Swiss-Tx project. Its final objective is to build a low cost, parallel commodity computer delivering one Teraflop/s by the year 2000.

INTRODUCTION

Since 6 years, most of the supercomputer vendors have been taken over by PC and workstation manufacturers (Cray, Convex), have stopped supercomputing (Intel), or stopped their business (Thinking Machines, KSR). There is no manufacturer now remaining for whom the main business is supercomputing. Users of high performance parallel machines can now choose among vector machines (Cray/SGI, NEC, Fujitsu) and SMPs (SGI, IBM, Sun, Compaq, HP, Hitachi). Besides the high prices, the vector machines demand data structures different to those chosen in cache based computers as PCs or workstations and the SMP machines with their customised architectures often do not scale with the number of processors. These are major

reasons why commodity parallel computing is now considered as an alternate road map towards high performance parallel computing. In this new approach, autonomous, high performance, shared memory computers are connected by an external high-speed network. Global communication between processors can be taken care of by message passing libraries such as MPI. Such parallel computers are often called message passing machines for which a user has to care about optimising the MPI implementation to become efficient on all those computer architectures. In contrary to vector machines, the use of commodity computers as computational units guarantees that local optimisation has not to be touched when porting a PC or workstation program to a commodity supercomputer.

In this paper we first discuss and compare the different popular network architectures chosen to cluster computational units with a special emphasis on circulant graphs. Secondly, we present the Swiss-Tx commodity parallel computer project [2, 3] that aims at delivering a Teraflop machine by the year 2000.

NETWORK ARCHITECTURES

The definition of efficient interconnection network topology is a major issue of parallel commodity computer designers. Since many years, several topologies have been studied and used to build parallel computers. For example, a few years ago, hypercube topology was largely used because of its apparently low diameter and its good mathematical properties. Besides the SGI Origin2000, this topology is not any more used today because of its lack of scalability. Topologies derived from trees are used in the IBM SP-2 and the fat-tree topology by the Compaq-Quadrics machines using a follow-up of the Meiko [7] network technology. On the other hand, there is a trend towards simple graphs such as grids (often used by Beowulfs) or the torus (SGI/Cray T3E). In the following paragraphs we will present the results of our studies concerning the topologies for interconnection networks realised in the framework of the Swiss-Tx project.

The graph theory is the main mathematical method applied in the field of interconnection networks. To well understand the content of this paper we start with a glossary:

- A graph is made of *edges* and *nodes*;
- The *size* of the graph is the number N of nodes of the graph. It is directly related to the maximum computa-

tional power of the machine. Typically, each node of the graph will be occupied by a fixed number P of processors;

- Two nodes are *adjacent* if they are the extremities of the same edge;
- A *chain* between two nodes x and y is a list of k nodes $x_1, \dots, x_k$ such that two consecutive nodes x_i and x_{i+1} , $0 < i < k$, are adjacent and such that $x_1 = x$ et $x_k = y$;
- A graph is *connex* if there exists a chain between each pair of nodes of the graph. In the following, we are only interested in connex graphs;
- The *length* of a chain is the number of its edges;
- The *distance* d_{ij} between two nodes x_i and x_j is the length of the shortest chain between them;
- The *diameter* D of a graph is the longest distance in the graph;
- The *average diameter* (or *average distance*) Dm of a graph is defined as:

$$Dm = \sum d_{ij} / (N^2 - N)$$

where N is the size of the graph (by definition, $d_{ii}=0$). The average diameter influences the transfer time between arbitrary nodes and the diameter influences the time used to broadcast information. In any case we will try to minimise these values with respect to the size and the degree of the graph;

- The *bisectional width* BiW is the smallest number of edges we have to cut in order to separate the graph in two parts of the same number of nodes (plus or minus one). It is an informal measure of the available bandwidth between the two half of the machine. Usually we will try to keep this value as high as possible;
- The *degree* d of a node is the number of nodes adjacent to it. The degree imposes the number of network communication (NC) ports we must have on each node. Usually this number is dictated by the status of the used technology. In any case, the price of the machine increases with the number of needed NC ports;
- A graph is *regular* if all the nodes have the same degree. In order to avoid special-case nodes, regular graphs are our favourite candidates. Special case nodes can become *hot-spots* and increase the contention phenomena;
- A *topology* is a class of graphs;
- A topology is *rigid* if for any given size N and degree d there exists only a few ($\ll N$) graphs of degree d and of size smaller or equal to N . An example of a very rigid topology is the hypercube. There exists only one hypercube with a given degree d and the size of this hypercube must be 2^d . Therefore, following the above definition, for any d and N there exists at most one hypercube of degree d and of size smaller or equal to N . To build computers of any power we need to have a great liberty on the choice of the size of the graph. Therefore we try to avoid rigid topologies.

To summarise, the ideal topology should have the following characteristics: *It should be regular and not rigid, we need a low average diameter, a low diameter and a high bisectional width for a large size and a small degree.*

Our objective is to compare different topologies. To do so, we need to define clearly what parameters of which graphs should be compared. As it can be seen in Fig. 1, the processing power of a machine is characterised by the number N of processing units (PU) or nodes as previously defined and the number P of processors per PU, and the network by the degree d , the diameter D , the average diameter Dm , the bisectional width BiW , and the cost. In our study, we will compare graphs with the same number of PUs and processors per node. In others words, the problem is to decide, for a given computing power, which are the topologies having the best connection characteristics.

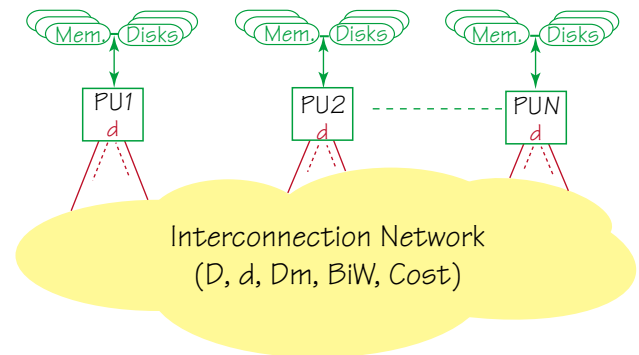


Fig. 1 – General representation of a commodity parallel computer

CIRCULANT GRAPHS

In [5] we analyse and propose the so-called K-Ring topology for the network of the Swiss-Tx machines. We show that this topology has better characteristics than the most currently used (hypercube, torus, fat-tree,...) ones. In the following section we are going to enlarge our study in order to extend our analysis to a topology called circulant graphs [1] that includes the K-Rings.

- A circulant graph: $C_N \langle a_1, a_2, \dots, a_K \rangle$ with $0 < a_1 < a_2 < \dots < a_K < (N+1)/2$ is a graph of size N where the nodes are numbered from 0 to $N-1$ and such that the node i is linked to the nodes $i \pm a_1, i \pm a_2, \dots, (i \pm a_K) \bmod N$.

Circulant graphs are regular graphs, but they can be non-connex (example $C_{12} \langle 2, 4 \rangle$). It has been demonstrated [1] that a circulant graph is connex if $\gcd(a_1, a_2, \dots, a_K, N) = 1$.

The condition " $\exists a_i, a_j$ such that $\gcd(a_i, a_j) = 1$ " implies that $\gcd(a_1, a_2, \dots, a_K, N) = 1$, but the reverse is not true. A simple example is: $\gcd(6, 10, 15) = 1$ but $\gcd(6, 10) = 2$, $\gcd(6, 15) = 3$ and $\gcd(10, 15) = 5$. If we impose that $a_1 = 1$, we obtain: $\forall i, \gcd(1, a_i) = 1$ and the corresponding graph is connex. Even if the class of circulant graphs having $a_1 = 1$ does not contain all the connex circulant graphs, we will restrict our analysis to circulant graphs having $a_1 = 1$ i.e. to $C_N \langle 1, a_2, \dots, a_K \rangle$.

It has to be noted that K-Rings are circulant graphs $C_N \langle a_1, a_2, \dots, a_K \rangle$ such that $\forall i, \gcd(a_i, N) = 1$. Consequently, K-Rings are included in $C_N \langle 1, a_2, \dots, a_K \rangle$ (see [4] for details).

FAT-TREES

Fat-tree topology is used to build multi-stage networks. In these networks some nodes of the graph are computing nodes (PU) while other nodes are switching nodes. More details on the fat-tree topology can be found in [6].

Fig. 2(a) present a fat-tree of size 12. Figs. 2(b) and 2(c) presents the two possible solutions for building an interconnection network starting from the fat-tree represented in 2(a). Squares represent computing nodes while circles represent 4×4 crossbar switches. Only computing nodes contain processors. In this paper we will assume that interconnection networks are built using the solution 2(c). This choice is motivated by the fact that fat-trees were designed for maximising the bisectional width. Solution 2(c) leads to a better bisectional width with respect to the number of computing nodes and it is very close to the solution used by Meiko [7].

As it appears in Fig. 2, fat-trees are not regular graphs. Indeed switching nodes have a degree that is the double of the one of computing nodes. In order to compare this topology with a regular one, we have to decide which degree we assume for fat-trees. In order to be fair in our comparison, we base our choice on the degree of the computing nodes. Indeed this degree determines how many NC ports must be present on the PUs. With this hypothesis the graphs presented in Fig. 2 have a degree of 2.

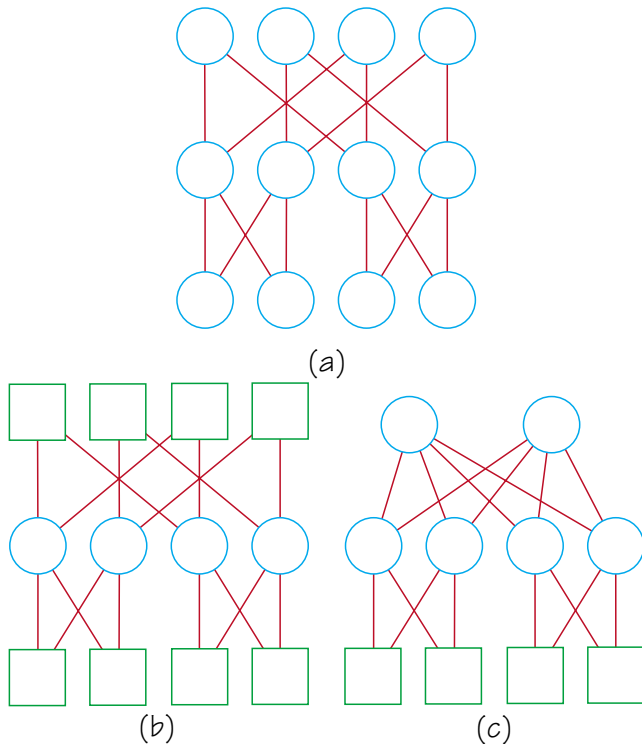


Fig. 2 – Fat-tree of size 12 and the corresponding interconnection networks

GRIDS, TORUSES AND HYPERCUBES

Toruses are periodic grids. Since their topologies are well known, we only remind that a torus of dimension K is a regular graph of degree $2K$.

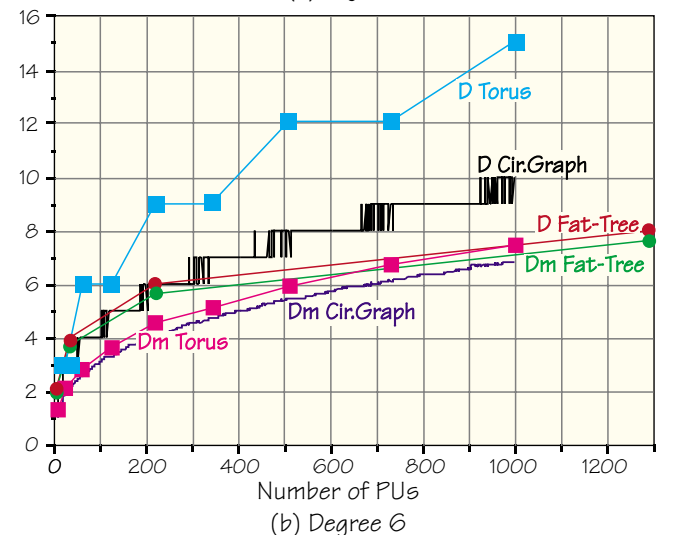
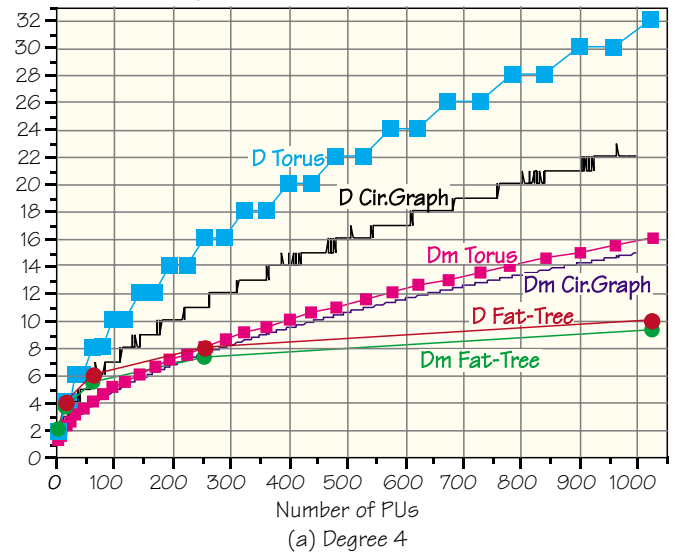
As our objective is to build the interconnection network of a parallel computer we are not interested by multi-graph (graphs that can have more than one edge between two nodes). More precisely, we consider a multi-graph to be equivalent to the graph obtained by replacing any multiple edges by one edge. With such a definition hypercubes are special case of toruses (toruses of dimension K and of size 2^K).

In the following sections we will compare the characteristics of toruses, fat-trees, and circulant graphs $C_N \langle 1, a_2, \dots, a_K \rangle$ for the same size and the same degree. Our comparisons are limited to the degrees 4, 6, 8 and 10 because, on the one hand, the degree of toruses and circulant graphs must be an even integer and, on the other hand, circulant graphs and toruses of degree 2 are simple rings.

COMPARISON OF THE CHARACTERISTICS OF TORUS, FAT-TREES AND CIRCULANT GRAPHS

Figs. 3 and 4 presents the comparisons of the measured values of the diameter, the average diameter and the bisectional width for degrees 4, 6, 8 and 10. For degree 10 the values are obtained with an approximate formula, since the computed values were not available for circulant graphs. These results show that:

1. Toruses always have the worst diameter;



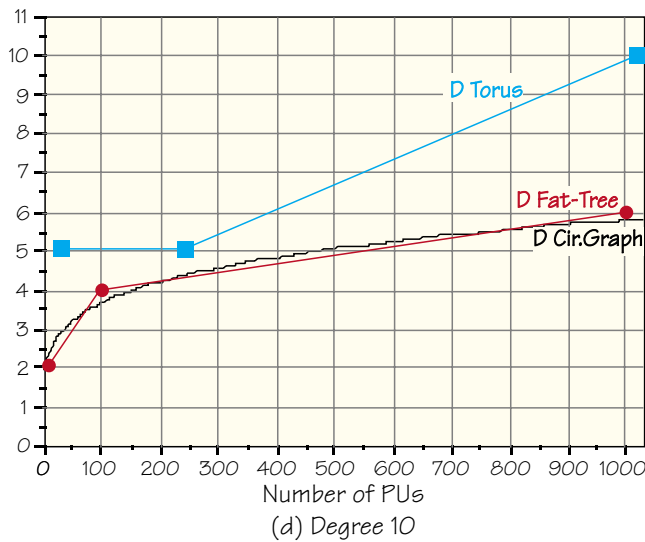
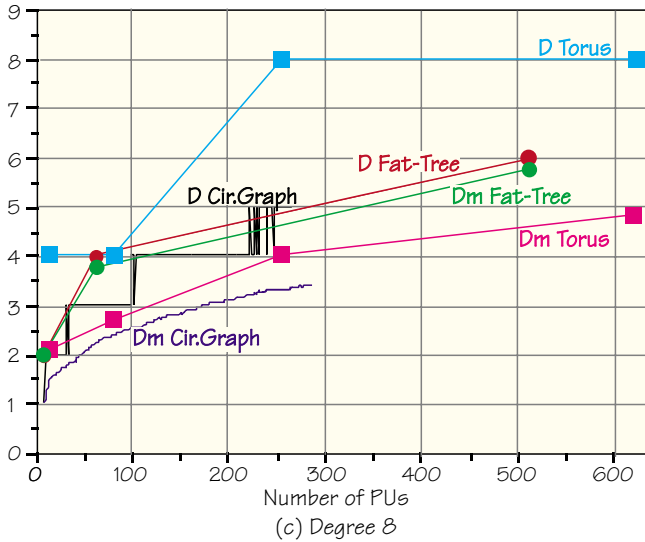


Fig. 3 – Comparison of the diameter (D) and average diameter (Dm) of toruses, fat-trees and circulant graphs

2. Fat-trees appear to have the best diameter but the difference with circulant graphs is decreasing with increasing degree;
3. The average diameter of fat-trees is very close to its diameter, as a consequence, for degrees greater than 4 and a size smaller than 1000, the average diameter of circulant graphs is smaller than the one of fat-trees;
4. For a number of PUs up to 1000 the diameter of circulant graphs is smaller or equivalent to the one of fat-tree as soon as the degree is greater than 6;
5. Fat-trees always have the best bisectional width, toruses the worst ones, and the bisectional width of circulant graphs is very erratic.

Based on these results we can discard the toruses that always have the worst diameter and bisectional width. Small degree fat-trees seem to be the best choice even if the difference with circulant graphs is not spectacular. Nevertheless, the drawback of fat-tree is that they are extremely rigid. We have the following properties:

- The number of fat-trees of a given degree d and of size $\leq N$ is equal to $\lceil \log_d(N) \rceil$. For $d=8$ and $N=1000$ this number is equal to 3;
- Performant circular graphs can be found for any number of PUs.

In order to decide whether or not fat-trees is a better choice than circulant graphs we are going to study how to build a communication network using these topologies.

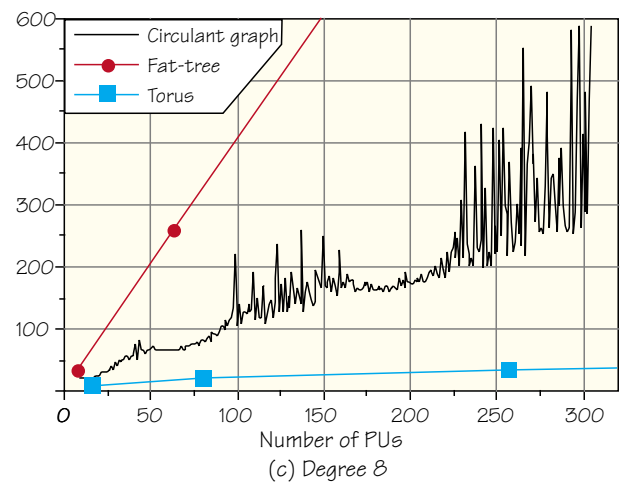
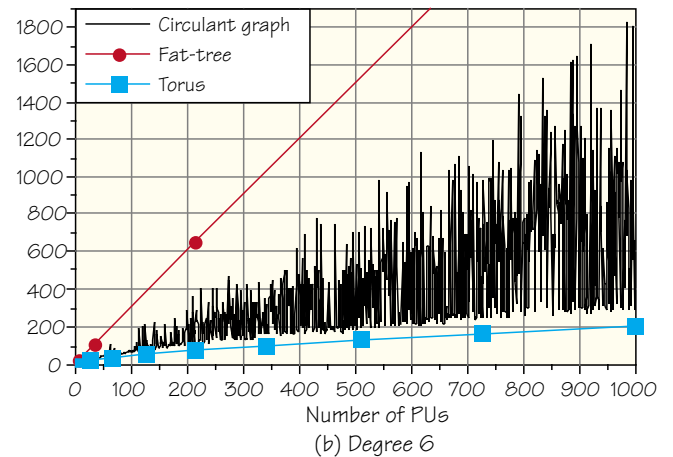
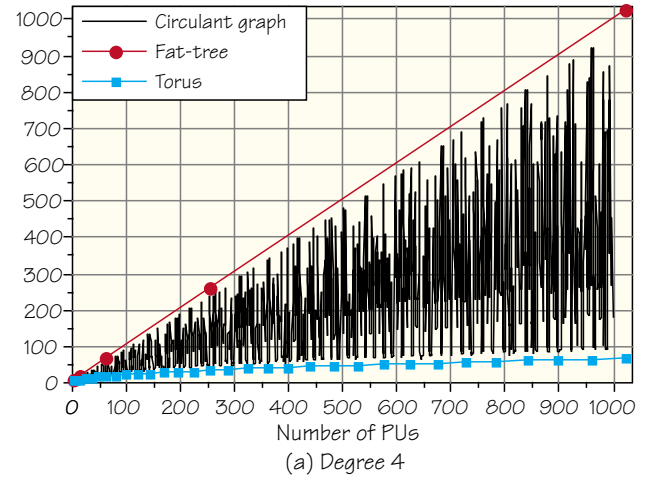


Fig. 4 – Comparison of the bisectional width of toruses, fat-trees and circulant graphs

COMMUNICATION NETWORKS AND CROSSBAR SWITCHES

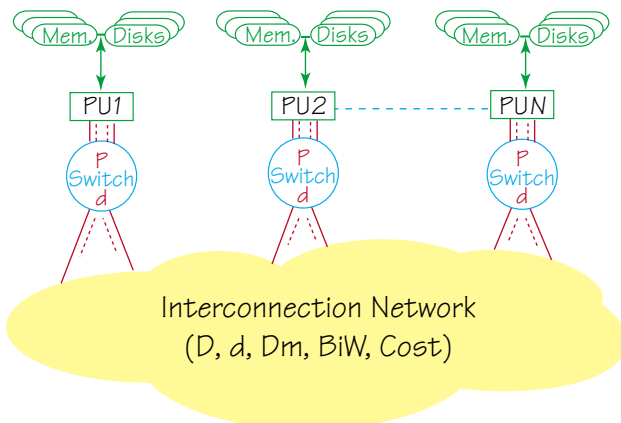


Fig. 5 – General representation of a parallel computer built using crossbar switches

Recent developments of high-speed crossbar switches have opened new possibilities for the design and the realisation of interconnection networks. Fig. 5 shows the general schema of a communication network built using crossbar switches. In the case of Swiss-Tx machines, the available technology is a 12x12 crossbar switch, called T-NET, designed by the company Supercomputing Systems AG (SCS). The objective of the Swiss-Tx project is to build a parallel machine having a peak performance of up to 1 Teraflop/s. Today's technology can provide processors of a peak performance of 1 Gflop/s (such as the DEC-Alpha 21264). For a parallel one Teraflop/s computer, 1000 one Gflop/s processors have to be interconnected using the high bandwidth, low latency 12x12 T-NET switches. In all the following considerations we make the assumptions that we need one link per processor. This assumption leads to the following possibilities:

- P=2 processors by PU, a topology of degree $d=10$ and a size of $N=500$;
- P=4 processors by PU, a topology of degree $d=8$ and a size of $N=250$;

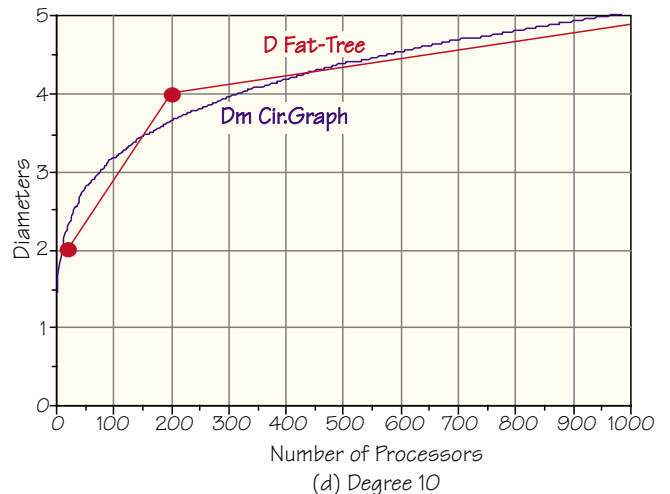
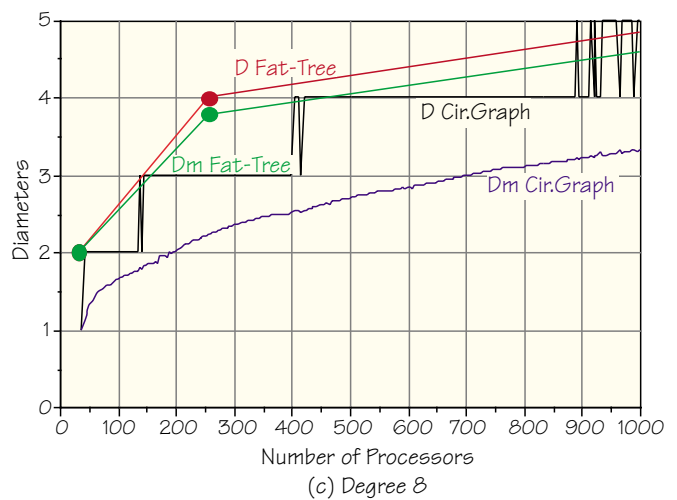
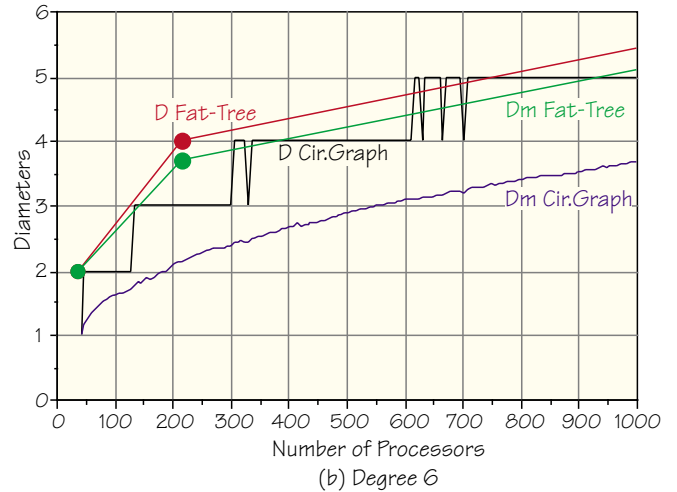
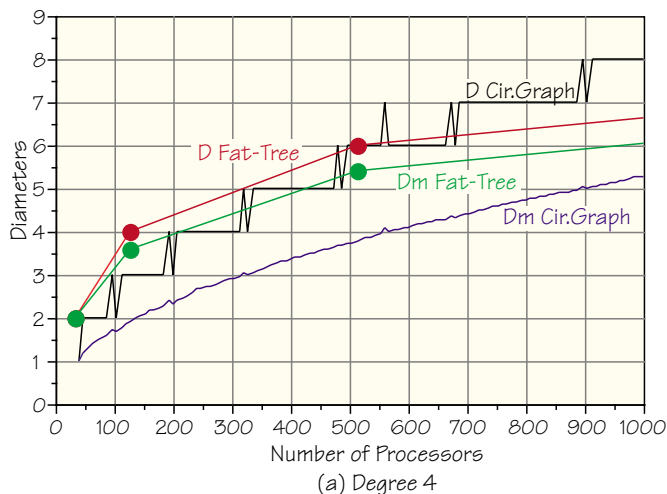


Fig. 6 – Diameter (D) and average diameter (Dm) of interconnection networks built using crossbars

- P=6 processors by PU, a topology of degree $d=6$ and a size of $N=167$;
- P=8 processors by PU, a topology of degree $d=4$ and a size of $N=125$.

All these situations can be realised with circulant graphs; no one can exactly be realised with fat-tree topology.

Fig. 6 compares the diameters of possible solutions using fat-trees and circulant graphs. For degrees 4, 6 and 8 the results are computed values, for degree 10 results are based on an approximate formula. Possible solutions using fat-trees are indicated with a dot. It clearly appears that circulant graphs always have a diameter smaller or equal to the one of fat-trees. Nevertheless, fat-trees have a better bisectional width.

At this stage of our analysis it is still difficult to choose between circulant graphs and fat-trees. Circulant graphs are much more flexible, have most of the time a better diameter and always have a better average diameter whereas fat-trees have a better bisectional width. The last criterion we have to analyse is the cost of the network.

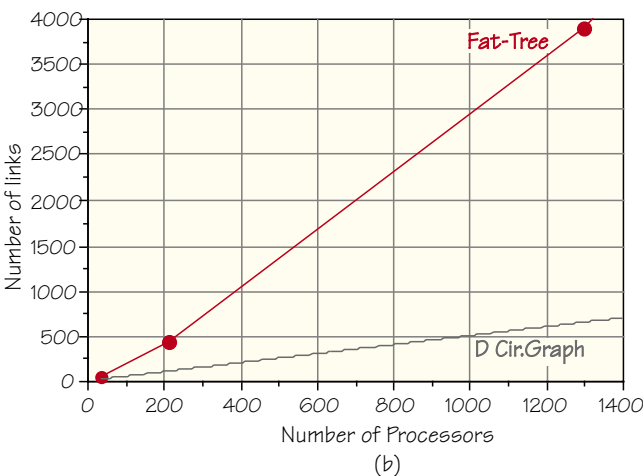
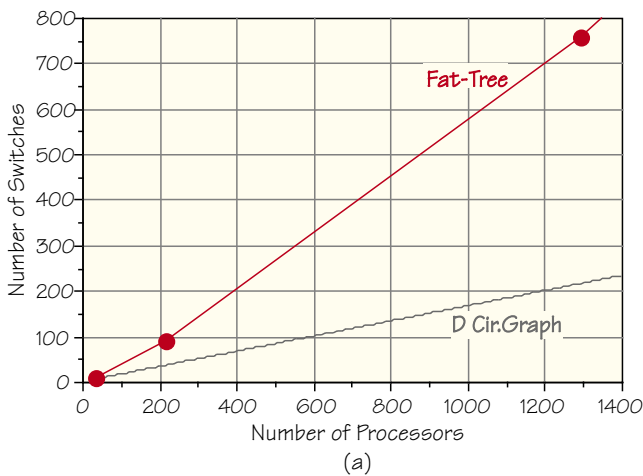


Fig. 7 – Cost of interconnection networks of degree 6 built using 12x12 crossbar switches

The cost of a massively parallel machine is roughly composed of the cost of the PUs and the cost of the communication network. Since $N \times P$ is chosen to be the same in all the cases, the cost for the PUs is the same for fat-trees and circulant graphs. The cost of the network is directly related to the number of switches and links necessary to build it. Fat-trees have the following property: if we use $n \times n$ crossbars to realise the switch nodes and the computing nodes then the degree of the fat-tree must be

$n/2$ (see Fig. 2). Therefore, if the available technology is 12x12 crossbars, we are limited to a degree of 6.

Fig. 7 shows the cost of fat-trees and circulant graphs of degree 6 for a number of processors up to 1400. The needed links and crossbars increase more rapidly for fat-trees than for circulant graphs. Let us compare possible solutions that are the closest to 1000 processors:

- Fat-Tree: 1296 processors (216 PU's), a diameter of 6, an average diameter of 5.63, a bisectional width of 648, 756 crossbars and 3888 links;
- Circulant graph: 1002 processors (167 PU's), a diameter of 5, an average diameter of 3.68, a bisectional width of 96, 167 crossbars and 501 interconnection links.

Of course the solution with a fat-tree has a much better bisectional width. But two remarks can be made:

1. The high bisectional width has its price. Thus, the ratios of the number of switches and links between the fat-tree solution and the circulant graph solution are 4.53 and 7.77, respectively;
2. The number of processors in a fat-tree must be a power of 6. Using circulant graphs, the number of processors must be a multiple of 6. Thus, there always exists a solution close to a given number of processors.

Because of their flexibility and of the erratic behaviour of their bisectional width, circulant graphs can always exhibit solutions having a good bisectional width for a number of processors close to a given value. In our case we can mention two possibilities:

1. 1260 processors (210 PU's), a diameter of 6, an average diameter of 3.98, a bisectional width of 362, 210 crossbars and 630 links;
2. 1356 processors (226 PU's), a diameter of 6, an average diameter of 4.07, a bisectional width of 394, 226 crossbars and 678 links.

Nevertheless, these two solutions still have a quite low bisectional width compared with the one for the fat-tree solution. But, as using circulant graphs we are not limited to a degree 6, we can also build solutions using a circulant graph of degree 8 (with 4 processors on each switch). Doing this we find the following solution:

- 1192 processors (298 PU's), a diameter of 5, an average diameter of 3.47, a bisectional width of 588, 298 crossbars and 1192 links.

In comparison with the solution of 1296 processors using a fat-tree, this solution exhibits a better diameter and average diameter, uses 2.5 times less crossbars, 3,26 less links and achieves a bisectional width per processor of 0.49 which is almost the same as the fat-tree solution (0.5).

Nevertheless, if the only important parameter is the bisectional width, regardless any other considerations and particularly the cost, we can design the interconnection network using circulant graphs of degree 10 with two processors per PU. Below are the characteristics of some examples of these possible solutions:

- 1000 processors (500 PU's), a diameter of 5, an average diameter of 3.54, a bisectional width of 570, 500 cross-bars and 2500 links;
- 1020 processors (510 PU's), a diameter of 5, an average diameter of 3.57, a bisectional width of 642, 510 cross-bars and 2550 links;
- 1080 processors (540 PU's), a diameter of 5, an average diameter of 3.61, a bisectional width of 778, 540 cross-bars and 2700 links.

For these solutions the bisectional width per processor is 0.57, 0.63 and 0.72, respectively, which is better than the solution using a fat-tree. Moreover they use significantly less crossbar switches and links.

As a conclusion we can say that the fat-tree is a topology especially designed for exclusively building very high performance interconnection network. The circulant graph topology allows to adapt the performance of the interconnection network to the user needs and, if needed, it allows to obtain performance equivalent or better to fat-tree for a lower cost. Moreover, the fat-tree topology is a very rigid topology, it cannot fully benefit from the increasing size of the crossbar switch technology, since the degree of a fat-tree must correspond to half of the crossbar size, and the number of PUs must be a power of the degree. With circulant graphs, networks of any numbers of PUs can be built.

For all those reasons we decided to use circulant graphs for building the interconnection network of the Swiss-Tx computers series. In the next section we present the architecture of the Swiss-T1 computer which is based on a circulant graph.

SWISS-T1 CONFIGURATION

HARDWARE CONFIGURATION

The first prototype Swiss-T1 machine will consist of 8 PUs connected using the T-NET 12x12 crossbar switches. Each PU consists of 4 dual processor, Alpha-based, DS20 servers, 2 links are used per server, and 8 links per PU (one link per processor). The four remaining links are used to connect the 8 PUs through the circulant graph: $C_8\langle 1,3 \rangle$ (Fig. 8). The diameter is 2, the average diameter is 1.43 and the bisectional width is 8. It has to be noted that we obtain a K-Ring for this particular case. An efficient routing for communications between all PUs is given in Table 1. This table has to be read in the following manner: for a direct link between the nodes, the routing number is identical to the destination node (blue background), for an indirect connection, the number (yellow background), denotes the crossbar number through which the routing passes. In Fig. 8 the numbers on the links indicate the number of packages that have to be sent in both directions for such an all-to-all message passing operation. The routing table has been set up to well distribute the charge on the different links during an all-to-all global communication.

The architecture is completed by a frontend consisting of 2 dual processor DS20 servers. The frontend is con-

nected to the computing nodes through the Gigabit Ethernet/Fast Ethernet switching system.

	1	2	3	4	5	6	7	8
1	-	2	2	4	4	6	8	8
2	1	-	3	7	5	3	7	5
3	2	2	-	4	4	6	8	8
4	1	7	3	-	5	7	7	3
5	4	2	4	4	-	6	6	8
6	1	3	3	7	5	-	7	1
7	8	2	8	4	6	6	-	8
8	1	5	3	3	5	1	7	-

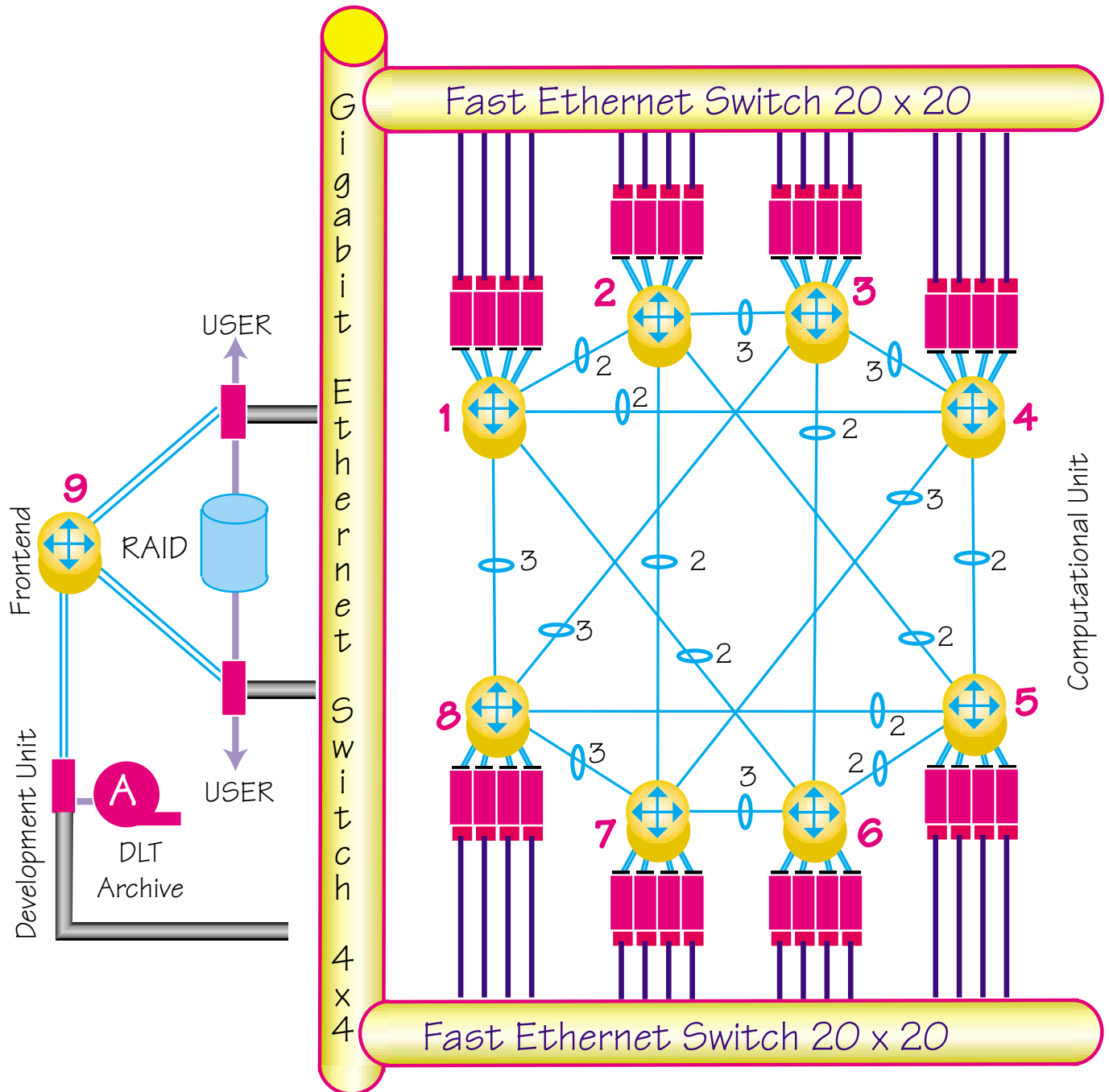
Table 1 – Routing table for the Swiss-T1 machine

The detailed hardware and software specifications are:

- Swiss-T1 consists of 8 PUs, each one including one 12x12 crossbar and 4 Alpha 21264 dual-processor boxes running at 500 MHz, giving 64 Gflops peak performance. One box has 1 Gbytes of main memory and 18 Gbytes of local disk space;
- Each box is connected to the 12x12 crossbar by two bidirectional 100 Mbyte/s links through PCI adapters;
- Four links interconnect the crossbars. The communication configuration is the circulant graph $C_8\langle 1,3 \rangle$;
- For an all-to-all communication, up to three messages in both directions have to be sent between crossbars if one follows the routing table given in Table 1;
- There is one frontend node consisting of 2 Alpha 21264 dual-processor boxes running at 500 MHz. Each box includes 2 Gbytes of main memory and 18 Gbytes of local disk space. During the installation phase it will be connected only through the Gigabit Ethernet/Fast Ethernet switching system, in a second phase, it will be directly linked to the T-NET as well;
- Two 20x20 Fast Ethernet switches interconnect the upper and lower half of the computational boxes;
- A RAID disk system of 300 Gbytes is connected to the frontend;
- The remaining frontend crossbar links can be used to interconnect other units;
- There is also a development box identical to a compute box separated from the production machine;
- A one Terabytes archive robot system is connected to the development box.

SOFTWARE CONFIGURATION

The different important software packages to be installed on the Swiss-T1 are (the programs marked with * will be available at T1 installation time):



$$N=8 \quad P=8 \quad d=4 \quad D=2 \quad Dm=1.43 \quad BiW=8$$

Fig. 8 – Swiss-T1 architecture based on Alpha 21264 dual-processors. The 8 PUs consist of 4 dual processor boxes. They are connected by a 12x12 crossbar switch, called T-NET. Each box, represented by a red rectangle, is connected by two links, the remaining 4 links connect to other crossbars. The diameter is $D=2$. For an all-to-all communication 3 bidirectional data exchanges are needed at most. The numbers on the links denote the number of these exchanges between these links. A Gigabit Ethernet/Fast Ethernet switching system directly connects the boxes to the frontend. An archive robot of 1 Tbytes and a RAID disk can be accessed by the 2 frontend boxes in a symmetric manner. There is a special development machine connected to the Fast Ethernet and the Gigabit Ethernet.

BASIC SOFTWARE IN EACH DUAL PROCESSOR BOX

*Tru64 Unix	Compaq	Operating system in each box
*F77/F90	Compaq	Fortran compilers
*HPF	Compaq	High performance Fortran
*C/C++	Compaq	C and C++ compilers
*DXML	Compaq	Digital math library in each box
*MPI	Compaq	SMP message passing interface from Compaq (only usable in a box)
*Posix threads	Compaq	Threading in a box
*OpenMP	Compaq	Multiprocessor usage in a box through directives
*KAP-F	Kuck Ass. Inc.	To parallelise a Fortran code in a multiprocessor box (preceeds OpenMP)
*KAP-C	Kuck Ass. Inc.	To parallelise a C program in a multiprocessor box (preceeds OpenMP)

Software to pass messages between the boxes and to use them in parallel

*LSF	Platform/ SIC-EPFL	Load Sharing Facility for resource management
Monitor	SIC-EPFL	Monitoring of system parameters
*Totalview	Dolphin	Parallel debugger
*Paradyn	Madison/ CSCS	Profiler to help parallelising programs
*MPI-1/FCI	SCS AG	Message passing interface between boxes running over T-NET
MPI I/O	SCS/ LSP-EPFL	Message passing interface for I/O
*MPICH	Argonne	Message passing interface running over Fast Ethernet
*PVM	UTK	Parallel virtual machine running over Fast Ethernet
*BLACS	UTK	Basic linear algebra subroutines
*ScaLAPACK	UTK	Linear algebra matrix solvers
NAG	NAG	Math library package
MEMCOM	SMR SA	Data management system for distributed architectures

CONCLUSIONS

The fat tree topology was especially designed to build a very high bandwidth network. As the fat-tree is a topology derived from trees, the diameter grows optimally with the logarithm of the number of PUs. The drawback of this topology is the extreme rigidity, the high cost, i.e. the very high number of links and switches.

We expect, but we have not proven yet, that the diameter of circulant graphs grows as $d/2\sqrt{N}$ which is, on a theoretical point of view, not as good as fat-trees. Nevertheless, due to its great flexibility we can fully benefit from the numerous possibilities offered by the use of the crossbar technology. Consequently, in the practice, it is always possible to find a solution using circulant graphs which have better character-

istics for a lower cost than the ones using fat-trees. Moreover, by using circulant graphs we can adapt the performance of the network to user needs. For a given number of processors and a given crossbar switch technology, we can choose the performance of the network. If, subsequently, the user needs to increase this performance we can increase the degree of the circulant graph without changing the number of processors. The opposite modification is also possible, we can increase the number of processors without changing the degree of the circulant graph. This flexibility which is not possible with other topologies, allows us to optimise the ratio price/performance according the user needs.

ACKNOWLEDGEMENTS

We would like to thank Martin Frey for continuous interaction to define the Swiss-T1 architecture and to Mario Romano for the graphical representation of it. The Swiss-Tx project is a co-operation between EPFL, ETHZ, CSCS, Supercomputing Systems and Compaq. It is financed by CTI (Commission for Technology and Innovation at Bern).

REFERENCES

- [1] F. Boesch, R. Tindell, *Circulants and their Connectivities*, Journal of Graph Theory, vol 8, p.487-499, 1984
- [2] S. Brauss, M. Frey, A. Gunzinger, M. Lienhard and J. Nemecek, *Swiss-Tx Communication Libraries*, HPCN'99 (Amsterdam) and this issue
- [3] Y. Dubois-Pèlerin, R. Gruber and Swiss-Tx Group: *Swiss-Tx, First experiences on the T0 system*, EPFL, Supercomputing Review, 10 (1998) 19-23 and <http://capawww.epfl.ch/>
- [4] P. Kuonen: *The K-Ring*, Proceedings of the European Research Seminar on Advances in Distributed Systems (ERSADS), April 1995 .
- [5] P. Kuonen, R. Gruber, A. de Vita and P. Volgers, *Parallel computer architectures for commodity computing*, keynote lecture at High Performance Computing and Networking (HPCN) Europe, Amsterdam, April 12-14, 1999
- [6] Leiserson C.E. Fat-Tree, *Universal network for hardware-efficient supercomputing*, IEEE Transactions on Computers, C-34, No. 10 (1985) 892-901
- [7] *Communication Network Overview*, <http://www.meiko.com/info/NetworkOverview/Network/Overview.html>. ■

COMMUNICATION LIBRARIES FOR THE SWISS-Tx MACHINES

STEPHAN BRAUSS, SWISS FEDERAL INSTITUTE OF TECHNOLOGY ZURICH (ETHZ), BRAUSS@IFE.EE.ETHZ.CH

Dans le projet Swiss-Tx, les composantes matérielles et logicielles pour la communication sont développées spécifiquement, car les produits standard disponibles ne peuvent pas offrir une latence, une bande passante, la fonctionnalité et la portabilité nécessaires. Une implémentation complète de MPI, standard pour les bibliothèques des échanges de messages, est disponible. Elle est basée entièrement sur la nouvelle interface de communication rapide, Fast Communication Interface (FCI). Cet article montre le concept de communication utilisé dans le projet Swiss-Tx incluant les bibliothèques FCI et MPI. Il compare ensuite cette bibliothèque MPI avec MPICH, une implémentation disponible gratuitement.

In the Swiss-Tx project, the communication hardware and communication software are custom-made, because available standard products do not offer the necessary bandwidth, latency, functionality, and portability. A full implementation of MPI, the standard for message-passing libraries, is available. It is designed entirely on top of the new Fast Communication Interface (FCI). This paper presents the Swiss-Tx communication concept including FCI and MPI and compares Swiss-Tx MPI with the freely available implementation MPICH.

INTRODUCTION

The highlights of the Swiss-Tx communication libraries are low-latency, high-bandwidth, portability, and compatibility. Portability means that all libraries run on various hardware platforms (currently PCs and Compaq Alpha-based Workstations and Servers), communication networks (currently EasyNet and T-NET¹) and operating systems (currently Compaq Tru64 UNIX, Linux, and Microsoft Windows NT). Compatibility means that user programs run without any modification on different platforms. For this reason, a high-level communication library offering the standardized Message Passing Interface (MPI) [3] is available. This MPI library is written entirely on top of the so-called Fast Communication Interface (FCI) [2]. See Fig. 1 for an overview of the communication hardware and software that is involved when a Swiss-Tx MPI application runs on two processing elements². The MPI application consists of two MPI processes running on two processing elements named PE₁ and PE₂. Each process consists of the application code (User Code) or a part of it and the communication libraries (that are the same on each processing element). Mainly, the following three libraries

are involved:

- Message Passing Interface Library (MPI Library);
- Abstract Device Interface Library (ADI Library);
- Fast Communication Interface Library (FCI Library).

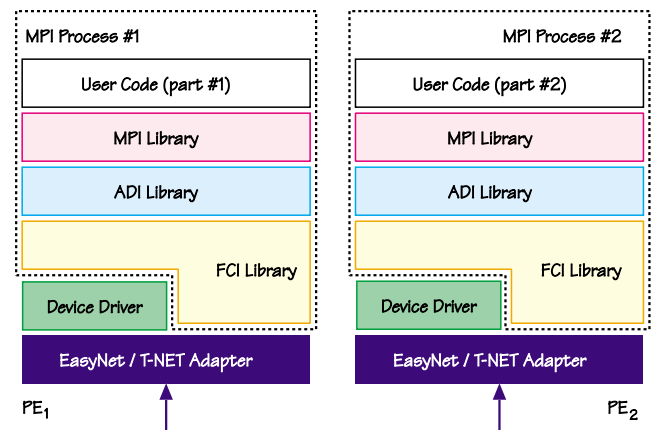


Fig. 1 – A Swiss-Tx MPI application running on two processing elements

The MPI library offers the high-level MPI routines, the ADI library does a part of the memory management and handles MPI data types. The FCI library is an implementation of the new communication architecture FCI. It offers two programming paradigms: Message Passing and the so-called Remote Store (for an introduction to Remote Store see [6]). The EasyNet or T-NET Communication Adapter is mainly controlled by FCI. The device driver is only used for startup and maintenance purposes and is easy to implement.

PORTABILITY

Portability is mainly attained by a good overall hardware and software concept and by smart programming. In this concept, we separate the hardware and operating system dependent from the hardware and operating system independent software parts. Most dependencies are covered by the new Fast Communication Interface FCI that is split into the following three interfaces:

- API (Application Programming Interface)
FCI defines the set of routines which can be used by a

- 1 EasyNet is the communication network used in Swiss-T0 and Swiss-T0-Dual, T-NET is used in Baby T1, Swiss-T1, and maybe also in Swiss-T2
- 2 in Swiss-T0, each box has one processing element, in Swiss-T0-Dual a box has two of it [1]

parallel application. Such a parallel application can be a pure FCI application (using only the routines offered by the FCI library) or a MPI application with the ADI and MPI libraries (see Fig. 1).

- DDI (Device Driver Interface)
FCI defines the functionality that must be offered by the network device driver residing in the kernel.
- NHI (Network Hardware Interface)
FCI defines the functionality of the network hardware. This includes mandatory and optional parts which can be added for better performance.

FCI guarantees that it is easy to port the communication libraries to new hardware and software platforms, which is a big advantage: We can reuse most of our software which reduces costs and development time:

- Replacing the communication hardware by a new one only affects the network device driver and some low level routines in the FCI library that directly control the communication interface adapter.
- Replacing the platform by for example Motorola Power PC based systems with Apple Mac OS only affects the network device driver (that has to be rewritten once for the new platform with the new operating system) and some low level routines in the FCI library that use the operating system and the device driver.
- Updating the operating system to a new release (for example to Compaq Tru64 UNIX V5.0) or replacing the hardware platform by a new one of the same architecture (for example a migration from Compaq Alpha 21164 Workstations to Compaq Alpha 21264 Workstations) normally needs no modification at all or only affects some small parts in the network device driver.

WHY SHOULD I USE SWISS-Tx MPI INSTEAD OF MPICH ON A SWISS-Tx MACHINE?

To make you understand why the Swiss-Tx communication libraries perform better than other products, we will compare MPICH [4] on the Swiss-Tx machines with Swiss-Tx MPI. MPICH is a freely available, portable implementation of MPI (Message Passing Interface) [3], the standard for message-passing libraries. On the Swiss-Tx machines, it uses the socket interface with Fast Ethernet to transfer data between processes running on different boxes (see [1]).

In Fig. 2 you can see a MPICH application running on two Swiss-Tx processing elements named PE₁ and PE₂ in a Compaq Tru64 UNIX environment. PE₁ wants to send a message to PE₂. The message is represented by a rectangle with a letter M inside. At the beginning, the message is held in the MPICH process on PE₁. When a transfer takes place, the message is copied into the I/O subsystem, which is a part of the UNIX kernel. Additional overhead of the TCP/IP protocol (e.g. the message has to be split up in smaller network packets and the network packets have a checksum that must be calculated in software) is illustrated by another copy step in the I/O subsystem, that is followed by a transfer

of the network packets to the Fast Ethernet Adapter. This adapter is a device in a PCI slot of the Compaq Alpha Workstation. The packets are transmitted to the adapter that is located in a PCI slot of PE₂ (by use of a Fast Ethernet Switch) and are stored in a block of memory in the I/O subsystem, where an additional copy step illustrates additional overhead of the TCP/IP protocol. At last, the MPICH process on PE₂ receives the message. Transferring data into the kernel and back to the MPICH process is a resource intensive job: The application has to call a routine in the kernel to initiate such transfers. As you can see, there are three main drawbacks of MPICH on the Swiss-Tx machines:

- messages have to be copied many times;
- the kernel is involved in the transfer of the messages (send and receive) and has to be called therefore;
- the big TCP/IP protocol overhead.

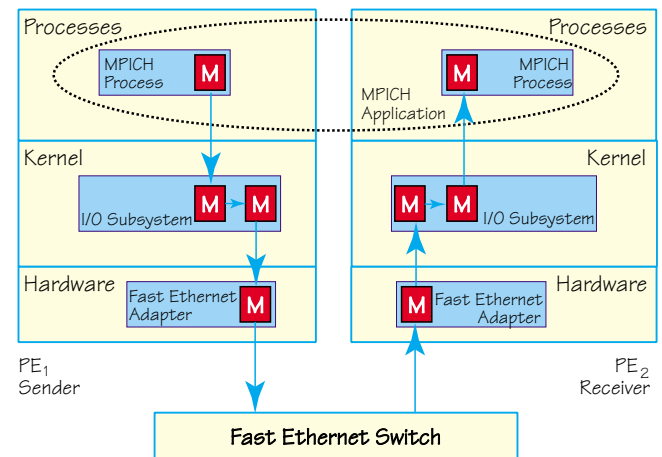


Fig. 2 – Two processing elements of a Swiss-Tx machine running a MPICH application

In Fig. 3 you can see a Swiss-Tx MPI application running on the same two processing elements PE₁ and PE₂ as shown above. We are still in a Compaq Tru64 UNIX environment and PE₁ wants to send a message to PE₂³. As before, the message is represented by a rectangle with a letter M inside. When the message must be transmitted from PE₁ to PE₂, it is directly transferred from the Swiss-Tx MPI process on PE₁ to the Swiss-Tx Network Adapter (currently EasyNet or T-NET) that resides in a PCI slot of the Compaq Alpha Workstation. This adapter sends the message to the destination adapter from where it is directly copied to the right location in the Swiss-Tx MPI process on PE₂ by the adapter itself. No CPU time is used to receive the message.

The Swiss-Tx FCI library avoids expensive kernel calls and avoids message copying. Send and receive operations are zero-copy. In addition to that, it has only a small protocol overhead. This can be achieved by an intelligent communication hardware that supports automatic multicast

3 Assumptions: the message is 4 Byte aligned, contiguous, and transmitted by a blocking send

capable routing⁴ and that guarantees secure data transmission which makes error handling in the FCI library unnecessary. All this advantages result in a better overall performance for real applications. A workstation cluster connected by Fast or even Gigabit Ethernet using TCP/IP can also offer reasonable bandwidth and latency. But often, one aspect is not taken into account: Such clusters waste a lot of CPU time for the communication between the processing elements because each processing element has to copy the message several times, has to call routines in the kernel, and has to run the TCP/IP. This time is lost and cannot be used for calculations. These networks often don't reach the possible peak bandwidth at all because the processing elements do not have enough CPU power.

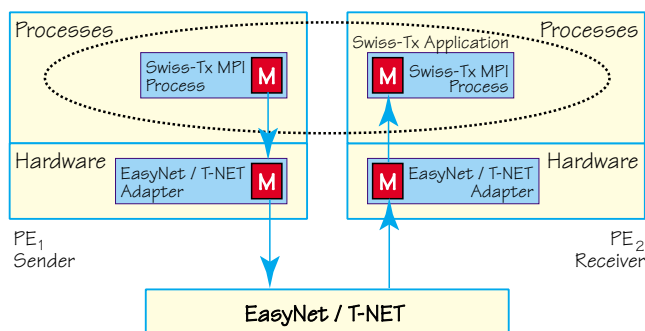


Fig. 3 – Two processing elements of a Swiss-Tx machine running a Swiss-Tx MPI application

INSIGHTS INTO THE SWISS-Tx COMMUNICATION NETWORKS EASYNET AND T-NET

Transferring messages directly from user space to the communication adapter in a UNIX environment requires that the communication hardware is specifically designed. The same is true for the automatic reception of messages, the Remote Store Concept [6], and other features. In fact, the communication adapters EasyNet and T-NET comply with the so-called Network Hardware Interface (NHI) Specification. This specification includes functionality that is mandatory to implement and other that is optional. Three important parts are the communication channels, the multicast capability and the page table support.

COMMUNICATION CHANNELS

One of the optional features is the number of supplied communication channels that are available on the communication hardware. Depending on the number of such channels, it is possible to attach one or more processes of an application to a communication adapter. Each process needs a private channel and is exclusively bound to this channel. Each process of a parallel application using the Swiss-Tx communication libraries should normally run on a private processing element using a private channel of a communication adapter. EasyNet has only one channel, T-NET is designed to have several, but only one has been

implemented yet. This is why only one process per box can use Swiss-Tx MPI on Swiss-T0-Dual. This machine has only one EasyNet adapter per box but each box has two processing elements. On Baby T1, each box has currently two 1-channel T-NET communication adapters, one for each processing element and therefore it is possible to run one process of a Swiss-Tx MPI application on each processing element. It is currently not possible to attach processes of different parallel applications to different channels of the same communication adapter. This restricts how the machine can be partitioned.

In the near future, T-NET will offer an additional channel that is reserved for the operating system. This channel is not part of the NHI. It can be used to transfer TCP/IP packets that are currently transported by Ethernet. This will speed-up applications like PVM [5], NFS [8], and MPICH.

MULTICAST CAPABILITY

An important feature of T-NET is the possibility to transport multicast messages. It is not fully implemented in the EasyNet hardware. Multicast means that a message can be sent from one communication adapter to a set of receiving adapters. The Swiss-Tx FCI library requires that the message is automatically copied to the right main memory locations of the receiving processing elements without any CPU interaction. This speeds up all collective routines in MPI significantly and reduces the network load.

T-NET PAGE TABLE SUPPORT

On Swiss-T0, Swiss-T0-Dual, and Baby T1 in the first phase, a programmer has to use the CMM Memory Manager for allocating memory suitable to store data that should be communicated to get the best possible performance. This will be obsolete in the near future. Modern operating systems use virtual memory with techniques called Paging and Swapping. The physical memory is partitioned into a set of pages of equal size that can be assigned to different processes. A process sees a contiguous virtual address space but the physical memory behind this addresses is located in distributed pages of physical memory or is even unallocated. This is managed by software and by the Memory Management Unit (MMU). So it is not guaranteed that a message that is larger than one page is located contiguously in physical main memory. It is even not sure that it is in main memory at all because it could have been swapped out to disk. See [7] for further explanations about Paging and Swapping. Because the communication adapters EasyNet and T-NET store received messages directly into the memory of a dedicated process, three solutions are possible:

- all main memory accesses of the communication adapter are routed through the MMU;
- the operating systems guarantees that the physical block of memory where the messages should be stored is contiguous;
- the communication adapter knows the page tables of the processes it is attached to and translates virtual addresses to physical addresses by itself.

⁴ not fully available in EasyNet

The first solution is platform dependent. It assumes that accesses from the communication adapter are routed through the MMU. Using this feature makes us incompatible to a large number of modern platforms. The second solution is implemented in EasyNet and also in the first release of T-NET. The CMM Memory Manager runs in a preallocated area of contiguous main memory that is marked as non-swappable. All memory allocated by this manager is taken out of this area. Such a memory block is guaranteed to be contiguous and available in the physical memory so that the communication adapter is capable to write to it. In the future, T-NET will offer page table support to be as flexible as possible. Memory allocation by the CMM Memory Manager won't be necessary anymore. It is still necessary that pages are fixed in main memory to prevent them from being swapped to disk. This is no real drawback. Applications running on a Swiss-Tx machine should not swap at all because swapping applications has normally a very bad performance.

CONCLUSIONS

The Swiss-Tx communication architecture guarantees that only a small overhead is included by the communication libraries and the communication network. Measurements on an experimental system based on two Compaq Alpha 21264 Workstations equipped with a non-optimized T-NET network are already available. The Swiss-Tx MPI

latency is less than 20 μ s and the bandwidth more than 50 MB/s (theoretical peak bandwidth of the network is 100 MB/s) ⁵. Both numbers will be improved.

REFERENCES

- [1] *Swiss-Tx Architecture*. Swiss Federal Institute of Technology Lausanne (EPFL), <http://lcapawww.epfl.ch/swiss-tx/index.html>
- [2] S. Brauss, J. Nemecek: *The FCI Reference Manual*. Swiss Federal Institute of Technology Zurich (ETHZ), <http://www.ife.ee.ethz.ch/hpclfci>
- [3] *MPI: A Message-Passing Interface Standard*. University of Tennessee, <http://www.mcs.anl.gov/mpi/index.html>
- [4] *MPICH - A Portable Implementation of MPI*. University of Tennessee, <http://www-unix.mcs.anl.gov/mpi/mpich/>
- [5] *PVM: Parallel Virtual Machine*. Oak Ridge National Laboratory, http://www.epm.ornl.gov/pvm/pvm_home.html
- [6] S. Brauss, M. Frey, M. Lienhard, J. Nemecek, A. Gunzinger: *Swiss-Tx Communication Libraries*. Lecture Notes in Computer Science 1593, HPCN Europe 1999 Proceedings, Springer (1999)
- [7] A. Tanenbaum: *Operating Systems: Design and Implementation*, pages 191-250, Prentice-Hall (1987)
- [8] NFS: Network File System Protocol Specification. Sun Microsystems, RFC1094, <http://www.cis.ohio-state.edu/htbin/rfc/rfc1094.html> ■

⁵ This measurements have been done at Supercomputing Systems AG in July 1999

PARALLEL FILE STRIPING ON THE SWISS-Tx ARCHITECTURE

BENOIT A. GENNART, EMIN GABRIELIAN, ROGER D. HERSCH, EPFL-DI, PERIPHERAL SYSTEMS LAB.
[HTTP://VISIBLEHUMAN.EPFL.CH](http://VISIBLEHUMAN.EPFL.CH)

La tendance actuelle dans le domaine des superordinateurs est de prendre avantage de l'augmentation constante et rapide de la puissance de calcul des stations de travail, et de créer des superordinateurs en empilant des stations de travail et des PC en les connectant par un réseau rapide. Cette approche minimise le coût du matériel en prenant avantage de l'effort de développement des composants grand-public (micro-processeurs, cartes réseau, disques).

Elle pose aussi de nouveaux problèmes : le modèle de programmation, les entrées/sorties, la gestion des utilisateurs et des tâches. Cette contribution discute les problèmes de conception d'un système d'accès aux données sur les ordinateurs à haute performance. Elle présente brièvement l'architecture matérielle, et les raisons de choisir un modèle de programmation distribué. Elle explique en détail la conception et l'implantation du module NAFS de fichiers zèbrés (striped files), dont l'implémentation est fondée sur le langage d'extension CAP.

The current trend in the area of high performance computing is to take advantage of the constant and rapid increase in the processing power of desktop workstations, and create supercomputers by piling up workstations and connecting them up with a high-speed network. Such an arrangement ensures minimal hardware cost by taking advantage of the development effort of commodity components (microprocessors, network cards, disks). It also raises new issues: the programming interface, input/output, user and job scheduling.

This paper presents the design issues surrounding input/output to disk on high-performance machines. It presents briefly the basic hardware architecture, and the reasons behind the choice of a distributed memory programming model. It also explains the reasons for offering an interface to distributed scalable data storage. It then describes in detail the design and implementation of the NAFS striped file package, based on the CAP parallel programming extension to C++.

INTRODUCTION

The current approach to high-performance computing is to use the available computing power in existing workstations and PCs and provide to users tools to extract the best performance out of multiple connected desktop computers. Low-end solutions simply consists of dividing a large task into multiple independent jobs, and submitting them to remote computers, as in the SETI@home project (<http://www.setileague.org/general/setihome.htm>). Higher-end solutions, for problems with more data dependencies, consists of improving the connectivity of an existing computer network, by using for example an Ethernet crossbar switch. Top-end solutions would include piling up dedicated computers and connecting them through a high-speed custom network, and adding a high-speed archive to handle large data sets.

While it is fairly easy to harness the computing power of multiple desktop computers connected through crossbar switches, achieving a high I/O throughput remains elusive. Solutions involving multiple RAID servers are possible, but involve additional costs. It is therefore tempting to use internal desktop-workstation disk-drives to mimic a single high-performance archive. The principle is to divide user files into multiple subfiles allocated on different disks in each desktop workstation. A parallel-striped-file software package ensures that the multiple files appear to the user as a single conventional file.

The main design goals of a parallel-striped-file package are portability and scalability. Portability is required to ensure that as many available computers as possible can be used regardless of their operating system and network, and leads to the selection of the most common network protocol, namely sockets. Scalability guarantees that the performance of the parallel striped-file package increases as computers are added to the architecture. Achieving high-performance in an I/O system is notoriously difficult: the latency of individual drives is high, and the program I/O operations tend to have small granularity. To improve the performance of the package, it is necessary to design an interface which allows multiple computers to coordinate their I/O activities in what is called collective operations.

Other design goals of a parallel-striped-file package are reliability, transparent job-scheduling, and client-server design. In terms of reliability, it is important for striped-files to survive the crash of a single computer/disk in the parallel architecture. Parallel I/O performance is also very dependent on the location of the files accessed by a program. A smart job-scheduler should be capable of scheduling jobs and moving striped files automatically so as to ensure maximum overall performance. In distributed systems, multiple user programs may access the same file. While this approach is required in the case of e.g. databases, it is often an overdesign in the case of high performance computing, where a single user completely controls the machine (or part of it) and the required files for the duration of the computation. At this point in the project, the reliability, transparency and client-server goals are secondary, and the main focus is on portability and scalability.

This paper describes NAFS, a parallel-striped-file package running under UNIX and WindowsNT.

- **The Swiss-Tx hardware architecture and I/O requirements** describes the Swiss-TX architecture and its effect on the I/O design;
- **The CAP Computer-Aided Parallelization tool** describes the precursor to the NAFS striped-file package project, the Visible Human Slice Server;
- **The Swiss-Tx I/O software design** addresses the issues surrounding the design and implementation of the NAFS striped-file package.

THE SWISS-Tx HARDWARE ARCHITECTURE AND I/O REQUIREMENTS

HARDWARE ARCHITECTURE

Fig. 8 in article: *Parallel computer architectures for commodity computing and the Swiss-T1 machine* (see on page 3) describes the parts of the Swiss-T1 architecture relevant to the I/O subsystem design. This machine consists of 8 processing nodes, each with 4 dual processor boxes, altogether 64 production processors, and of a four processor front-end subsystem. The front-end subsystem takes care of resource management and of all the external interactions. Two RAID servers connected to the front-end subsystem store the user files. They provide performance through striping and reliability through data redundancy. Each dual-processor box incorporates two 9GB disks, for system and scratch files. The boxes are connected through Ethernet for startup and system messages and through T-Net for high-speed transfers between user processes.

The issues to be addressed in the design of the Swiss-Tx architecture are: performance, scalability, reliability, portability. Performance of a N-processor architecture must be very close to N times the performance of a single processor, otherwise it is very difficult to justify the increase in size of the architecture. The architecture must be able to survive the failure of one or more nodes. And as a bonus, it would be nice if the software designed to improve the performance would work under various UNIX flavors (Solaris, Digital Unix, Linux) as well as under WindowsNT.

PROGRAMMING MODEL

The Swiss-Tx machine is a distributed memory architecture. Current implementation of shared memory models over distributed hardware are expensive and do not deliver higher performance for middle grain and small grain parallel applications [8, 18]. In fact, to achieve scalable performance, the programmer must handle data decomposition, allocate data subsets to the various nodes in the architecture and specify explicitly data transfers between the nodes of the architecture. Hence the selection of a distributed memory programming model and the use of a message passing interface for parallel programming. The standard message passing API (Application Programmer Interface) is MPI [4,7].

I/O DESIGN AND LIBRARY INTERFACE

I/O is a major bottleneck in many parallel applications. The main reason for poor application-level I/O performance is that I/O systems are optimized for large accesses, whereas parallel applications typically make many small I/O requests [1,2,9,12,16]. Leaving each parallel program thread to fend off for itself results in poor performance, as each thread performs comparatively small I/O requests, each incurring a high latency (see table 1). Much research has demonstrated the efficiencies of data organization and collective I/O [4,10,14]. In the MPI-IO interface, data types take care of data organization, and the API supports non-collective, collective, blocking and non-blocking operations. The Swiss-Tx project has selected the MPI-IO API as an interface to the striped file package.

THE CAP COMPUTER-AIDED PARALLELIZATION TOOL

To extract maximal performance from commodity component based architectures, and overcome the high latency of their communication network, it is necessary to overlap processing, communication, and data accesses. Writing such asynchronous parallel programs is tedious and error-prone. To facilitate the development of such programs, we have developed a computer-aided parallelization tool, CAP, and parallel file system components. CAP lets us generate parallel server applications automatically from a high-level description of threads, operations and the macro-dataflow between operations. Because of CAP's macro-dataflow nature, the generated parallel applications are completely asynchronous, without the need for callback functions. Each thread incorporates an input token queue, ensuring that communication occurs in parallel with computation. In addition, the CAP runtime environment executes disk-access operations asynchronously, also without the need for explicit callback functions [6].

We used the CAP tool in several applications, such as the Visible Human Slice Server which offers access to slices and surfaces within the 3D Visible Human dataset (13GBytes), and the RadioControl radio-rating project which correlates the content of radio programs with the content of wrist-held audio-data recorders (<http://www-imt.unine.ch/Radiocontrol>). Thanks to CAP the generated applications are flexible; it is easy to maintain and modify the parallel programs. Evaluation of the access times for the applications shows that their performance is close to the best that the underlying hardware can sustain [6]. The CAP tool is also used in a commercialization effort by A2I (<http://www.axsnow.com/>), aiming at providing components for manipulating large raster images.

While the development focus of the laboratory is on WindowsNT, the CAP tool is available on both WindowsNT and UNIX (Solaris, Digital). Among future developments is a wizard for specifying graphically the macro-dataflow between operations.

THE SWISS-Tx I/O SOFTWARE DESIGN

As a part of the Swiss-Tx effort, the authors implement a portable striped file package called NAFS (Not A File System). To the user, a NAFS striped file is a linearly organized set of bytes. The operations available to manipulate the files are the traditional file operations: create, open, close, delete, read from, and write to a NAFS file at specific offsets. The NAFS files are accessible both through an MPI API and an NAFS API. The aim of NAFS is to make the use of striped files as transparent as possible. To achieve this aim, the NAFS project will provide utilities to move, copy, and display (UNIX cat command) files. Both the NAFS and the MPI API hide the striping to the programmer. The striping information can be made available to the programmer who wishes to take advantage of the information to improve performance.

In the following paragraphs we address the following issues:

- performance considerations,
- striping and programming interface,
- miscellaneous design issues (need for dedicated I/O threads, network requirement, pipelining, file protection, redundancy),
- the implementation of NAFS using CAP,
- the current status of NAFS.

The CAP language extension is described in a separate box.

EXPECTED PERFORMANCE

	port to port bandwidth (Ethernet)	port to port bandwidth (T-Net/MPI)	local-disk throughput
latency	500ms	12μs	10ms
throughput (nominal)	12.5MB/s	100MB/s	8MB/s
throughput (aggregate)	100MB/s	1GB/s	512MB/s
throughput (2KB block)	5MB/s	62.5MB/s	0.195MB/s
throughput (50KB block)	7.5MB/s	97MB/s	2.5 to 5MB/s

Table 1 – Performance figures for the Swiss-T1 architecture

Table 1 presents the relevant performance figures for the Swiss-T1 architecture. We consider that each box in the architecture contains two processors and two disks. The measured Ethernet bandwidth per box is 5 to 8 MB/s per box. The nominal T-Net throughput is 200MB/s (100MB/s each way), shared between 4 boxes, or 50MB/s per box. We assume that both the Ethernet crossbar and the T-Net crossbar offer sufficient bandwidth to sustain the nominal throughputs at the box level. The next two paragraphs evaluate the distributed I/O design, and the centralized-server design.

In the distributed I/O architecture that we have chosen, each dual-processor box is both a producer and a consumer of data in an I/O operation. The processors produce data that is consumed by the disks. In a typical balanced I/O transfer, each box spends half the time sending data and half the time receiving data. Hence half the network bandwidth is available for distributed I/O operations. Assuming enough disk bandwidth, the I/O operations can be performed at the rate of 2.5 to 4MB/s per box through Ethernet, and 25MB/s per box through T-Net. In the case of the 32-box T1 architecture, the peak network throughput for I/O is 80 to 128MB/s through Ethernet and 1GB/s through T-Net. The nominal disk throughput is between 2.5 and 5MB/s for 40KB blocks depending on the locality of the data on disk. In the case of a 32 box architecture with 2 disks per box, the peak disk throughput is 160 to 320MB/s. This back-of-the-envelope analysis suggests that the Ethernet bandwidth is below the disk bandwidth, and that it is therefore necessary to use the T-Net for distributed I/O operations in order to achieve the maximum throughput.

These considerations suggest that the distributed approach offers high performance at low cost, if the T-Net is used. The Ethernet-based distributed design offers only acceptable performance. The back-of-the-envelope calculations in this section must of course be validated through experiments, and the overhead of various protocols (NFS, TCP sockets) taken into accounts.

NAFS DESIGN

In this discussion we assume that a parallel program consists of threads. Whether there are multiple user threads per processes as in CAP or a single user thread per process as in many MPI implementations is not important at this point in the discussion of the design. A striped file consists of one or more *subfiles*.

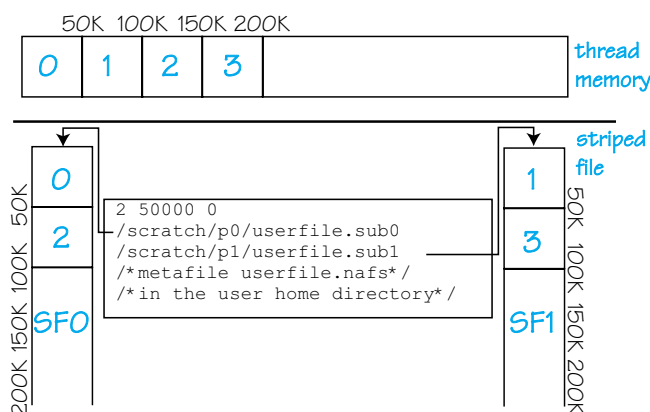


Fig. 2 – File striping

Each NAFS striped file consists of a metafile and one or more subfiles. The striped file is divided in extents (i.e. contiguous data sets of sufficient size to make a disk access worthwhile, typically 50KB) which are stored in subfiles in round robin fashion. A set of extents located at the same position in each subfile is called a stripe. Consider

(Fig. 2) a program consisting of a single thread which dumps the content of its memory (a single 1MB block) to striped file `userfile.nafs` with two subfiles `/scratch/p0/userfile.sub0` (short name: SF0) and `/scratch/p1/userfile.sub1` (short name SF1) and an extent size of 50KB. Striped file bytes `[0-50K[` are stored in `SF0[0-50K[`, striped file bytes `[50K-100K[` are stored in `SF1[0-50K[`, striped file bytes `[100-150K[` are stored in `SF0[50-100K[`, striped file bytes `[150-200K[` are stored in `SF1[50-100K[`, etc..

The metafile contains the number of subfiles, the extent size, the total file size and the list of subfile names (absolute OS paths, accessed through NFS on Unix machines or UNC's under WindowsNT). The metafile and subfiles are native OS files, for portability reasons. Each byte in the striped file is described by its offset (64-bit) and value.

When multiple threads are involved in writing to a striped file, each of them writes its own blocks to the file. The number of subfiles in the striped file need not be equal to the number of threads in the parallel program. Multiple requests to the same subfile are serialized. At the NAFS level, the operations read from and write to the striped file a *block-list*, each block being characterized by a size, an offset in the file, and a pointer in memory to the data to be transferred from/to disk. Arbitrary number of threads can take part to collective read/write operations. MPI I/O uses communicators to indicate the threads involved in collective operations, and datatypes to indicate the layout of data both in file and in memory. A software layer transforms MPI communicators and data types into block-lists and sets of threads.

NAFS ISSUES

Miscellaneous issues must be addressed in the design of the NAFS striped file package: the number of threads per dual-processor box, the high-speed network support, the pipelining of data transfers and disk accesses, the protection of files against multiple simultaneous accesses, and redundancy.

One dedicated I/O thread per I/O node

In the distributed I/O architecture, each user thread can request I/O operations from any dual-processor box in the architecture. In effect, it is acting as a client requesting I/O services. It is therefore necessary that each dual-processor box runs a thread dedicated for serving I/O requests from user threads. In this paper, we refer to user threads as compute-threads and I/O-request-serving threads as disk-threads.

Ethernet/socket communication vs. T-Net/MPI communication

The analysis of paragraph *Expected performance* suggests that it is necessary to use the T-Net to sustain the disk throughput of the distributed I/O design. And the comments of the previous paragraph indicate it is necessary to have a dedicated I/O thread in addition to the usual processing thread(s) in each box. However, many high-performance versions of MPI are not multi-threaded, and

CAP EXTENSION LANGUAGE OVERVIEW

This section presents a simple example describing the basic features of the CAP language extension to C++. CAP's main design goal is to implicitly implement asynchronous parallel behavior. To achieve these goals, it uses language constructs (1) to specify and group threads, (2) to specify the data structures exchanged between threads, (3) to specify the operations the threads and thread groups can perform and (4) to specify the macro-dataflow (pipeline/parallel scheduling) between thread operations.

We first illustrate the asynchronous semantics of the CAP. Consider a CAP program which converts data on one processor and writes it to one disk, chunk by chunk. For the sake of simplicity, we assume that the conversion operation does not change the size of the data chunk, nor does it require data in other chunks. Fig. A1 shows the syntax, graphical representation and timing diagram of such a program. In Fig. A1, program lines 1 to 6 specify the data structures specifying the data chunks moved between the program's two threads. Lines 8 to 17 specify the two threads (*DiskThreadT*, *ComputeThreadT*) in the program and the operations they can perform (*WriteChunk* and *ConvertChunk* respectively). Lines 19 to 26 logically group the two threads under the name *GlobalProcessT*, and specify which operation(s) they can perform as a group (*ConvertAndWriteData*). Line 28 instantiate *GlobalProcessT*, which automatically creates the two threads under the *GlobalProcessT* umbrella. In CAP, threads are instantiated upon program initialization. No threads are created during program execution, thereby reducing execution overhead.

Lines 29 to 44 specify the behavior of the program. The input of the program is a *DataT* token initialized with an open file descriptor *FileP*, an *OffsetInFile*, and a *Data* pointer to the memory data. The asynchronous parallel semantics of the CAP language is entirely handled by the parallel while expression (lines 39 to 44). The *SplitChunk* C++ function (passed as parameter to the parallel while expression) incrementally divides the *DataT* input token *inP* into several *DataT* tokens referencing consecutive 50KB blocks in memory. The *ComputeThread* converts the data chunks one after the other, and forwards each converted token to the *DiskThread* as soon as it is converted. The *DiskThread* writes the *P[i]* tokens to disk one after the other, and generates the *M[i]* void tokens. As soon as each of the *M[i]* tokens is available, it is merged by the *MergeChunk* function (not shown) into the *outP* void token. When all tokens have been merged, the *ConvertAndWriteData* operation is complete.

The graphical representation of the *ConvertAnd-*

WriteData operation (bottom of Fig. A1) matches the textual specification. It indicates that the input token is divided using the split function into several data chunks that are fed to the *ComputeThread*'s *ConvertChunk* operation. The output of the conversion is fed to the *DiskThread*'s *WriteChunk* operation. The output of the *WriteChunk* operations are used for synchronization purposes: when all outputs have been received, the *ConvertAndWriteData* operation is complete.

The *SplitChunk*, *MergeChunk* functions and the *WriteChunk*, *ConvertChunk* operations are all executed asynchronously, i.e. provided enough processors and disk they could all be executed simultaneously (albeit on different data chunks). It is possible to allocate the *DiskThread* and the *ComputeThread* not only on different processors in the same box, but also on different boxes. In that case, the token transfer over the network is automatic and asynchronous, that is, computation, communication and disk accesses are overlapped.

In the case where the split function generates many chunks of data, there is a risk of memory overflow, since the split function is typically much faster than the conversion operation. To work around this, CAP uses flow control modifiers to its parallel construct, limiting the number of tokens simultaneously active inside the parallel construct. Fig. A2 shows a modified *ConvertAndWriteData* where the number of simultaneously active data chunks is limited to 4.

To achieve asynchronous parallel behavior in CAP, the programmer replaces the disk- and compute-thread (lines 21 and 22) by arrays of disk- and compute-threads (e.g. *DiskThreadT*, *DiskThread[MAX]* and *ComputeThreadT*, *ComputeThread[MAX]*). The pipelining behavior explained in the previous paragraph is still available, but up to MAX data chunks can be compressed or written simultaneously, depending on the available hardware resources. The work of the CAP programmer is then to define the tokens and the operations required to achieve a given algorithm. The CAP language extension supports 8 predefined expressions, 3 for asynchronous parallel behavior (parallel, indexed parallel, parallel while), 3 expressions for iterative behavior (sequence, for, while), and 2 expressions for branching (if, ifelse). CAP programs based on predefined CAP expressions are deadlock-free by construction. CAP programs are reconfigurable without recompilation: the same executable can run on a 1-processor 1-disk low-end PC, on a 4-processor 4-disk shared-memory machine, or on an 10-processor distributed-memory architecture with 60 disks.

```

1 token DataT {
2   FILE* FileP ;
3   int OffsetInFile ;
4   int Size ;
5   char* Data ;
6 } ;
7
8 process DiskThreadT {
9 operations:
10  WriteChunk in DataT* inP out void* outP;
11 } ;
12
13 process ComputeThreadT {
14 operations:
15  ConvertChunk
16    in DataT* inP out DataT* outP;
17 } ;
18
19 process GlobalProcessT {
20 subprocesses:
21  DiskThreadT DiskThread ;
22  ComputeThreadT ComputeThread ;
23 operations:
24  ConvertAndWriteData
25    in DataT* inP out void* outP;
26 } ;
27
28 GlobalProcessT GlobalProcess ;
29
30 leaf operation DiskThreadT::WriteChunk
31   in DataT* inP out void* outP
32 { // C++ code to write data chunk to file }
33
34 leaf operation ComputeThreadT::ConvertChunk
35   in DataT* inP out DataT* outP
36 { // C++ code to uncompress data chunk }

```

```

37 operation GlobalProcessT::ConvertAndWriteData
38   in DataT* inP out DataT* outP
39 { parallel while (SplitChunk, MergeChunk,
40                  ComputeThread, DataT result)
41   ( ComputeThread.CompressChunk >->
42     DiskThread.WriteChunk
43   ) ;
44 }

```

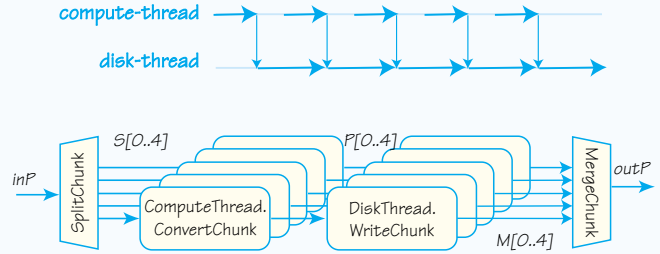


Fig. A1 – CAP specification and pipelining semantics

```

1 operation GlobalProcessT::ConvertAndWriteData
2   in DataT* inP out DataT* outP
3 { flow_control (4)
4   parallel while (SplitChunk, MergeChunk,
5                  ComputeThread, DataT result)
6   ( ComputeThread.CompressChunk >->
7     DiskThread.WriteChunk
8   ) ;
9 }

```

Fig. A2 – Flow-control modifier

therefore the I/O thread must be allocated in a separate process. This is often made difficult by the fact that the requirements of the high-performance network preclude the use of multiple processes. As a result, and also for portability reasons, NAFS and user threads communicate through sockets. In the second phase of the Swiss-Tx project, it is planned to adapt the NAFS parallel-striped-file package to use the high-speed T-Net network.

boxes and I/O transfers to disks are overlapped. The usual approach is to use the two-phase approach [14], where the next chunk of data is transferred over the network while the current chunk is transferred to disk. The CAP language semantics implies the pipelining of operations, as explained in paragraph *NAFS implementation using CAP*. This simplifies the programming of parallel I/O operations to striped files.

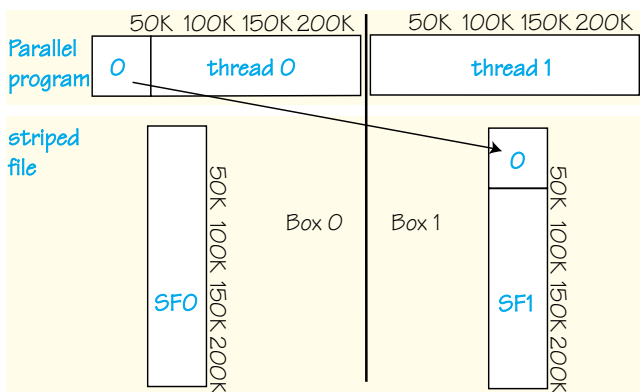


Fig. 3 – The need for dedicated I/O threads

Pipelining data transfers and disk accesses

To achieve peak I/O throughput, it is necessary to ensure that data transfers between different dual-processor

NAFS IMPLEMENTATION USING CAP

This section discusses the implementation of the non-collective/collective read and write operations in terms of a graphical representation of the behavior of the program. The parallel program consists of 5 compute-threads (*PC0* to *PC4*) and 5 disk-threads (*IO0* to *IO4*), and the parallel programs reads or writes data covering exactly 4 extents (*E0* to *E3*). As explained in section *NAFS design*, the data handled by a NAFS operation is represented as a single block-list linking all data blocks being written to or read from disk. The striped file written to consists of 3 subfiles *SF0*, *SF1*, *SF2*, handled by threads *IO0*, *IO1*, *IO2* respectively. Extent *E0* (resp. *E1*, *E2*, *E3*) is written to subfile *SF0* (resp. *SF1*, *SF2*, *SF0*), according to the round-robin rule.

Non-collective read

In this example, one compute-thread (namely *PC2*) reads a single block covering 4 extents in a 4-subfile striped file (Fig. 4). The compute-thread divides its memory into

four blocks covering the extents and sends the extent requests for each block (ER0 to ER3) to the appropriate disk-thread. Each disk-thread returns the data to the compute-thread which writes it to memory. All extents are read in parallel. However, to limit the memory requirements in the case of an operation involving many extents, a flow control modifier limits the number of extents processed by a single disk-thread to a small number (4). The non-collective write operation is similar to the non-collective read operation.

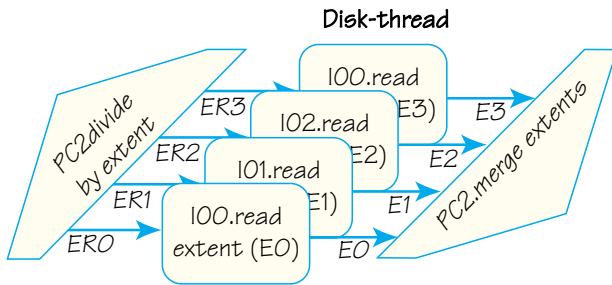


Fig. 4 – Non collective read operation

Collective write

In this example, the 5 compute-threads PC0 to PC4 collectively write the data covering the 4 extents E0 to E3 (Fig. 5)¹. Before starting the collective operations, all compute-threads synchronize, and compute-thread PC0 initiates the collective operation by creating extent-writing requests for each of the extents covered by the operation (ER0 to ER3)².

For each extent request, the disk thread controlling the subfile where the extent is stored sends requests to all compute-threads for their part of memory covering the

extent (GetBlocks operation). After receiving all blocks from the compute-threads, the disk-thread merges the blocks and writes the extent to disk. All extents are processed in parallel. However, to limit the memory requirements for the operation, a flow control modifier limits the number of extents *simultaneously* processed by a single disk-thread to a small number.

Collective read

In the collective-read operation, the 5 compute-threads collectively read 4 blocks from 3 subfiles (Fig. 6). The 5 compute-threads first synchronize and PC0 initiates the collective read operation by asking all compute-threads to divide their respective block-list list into separate block-sublists for each extent. Each compute-thread then sends the block-request sublists to the appropriate disk-thread. Each disk-thread waits for block-sublists. When it has received from all compute-threads the block-sublists corresponding to a given extent, it reads the extent from disk and fills the blocks in the corresponding block-sublist. The disk-threads then send the blocks to the compute-threads, where they are copied in memory.

CURRENT STATUS

The CAP environment has undergone extensive testing. The experimental performance results published in [6] show that the pipelining strategies of CAP are effective, that the performance achieved is close to the maximum of either computation-time, communication-time or disk access-time. The CAP environment runs on WindowsNT, Solaris and Digital Unix. An installation wizard is available under WindowsNT.

The NAFS parallel-striped-file package is implemented and partially tested. It supports blocks of arbitrary size

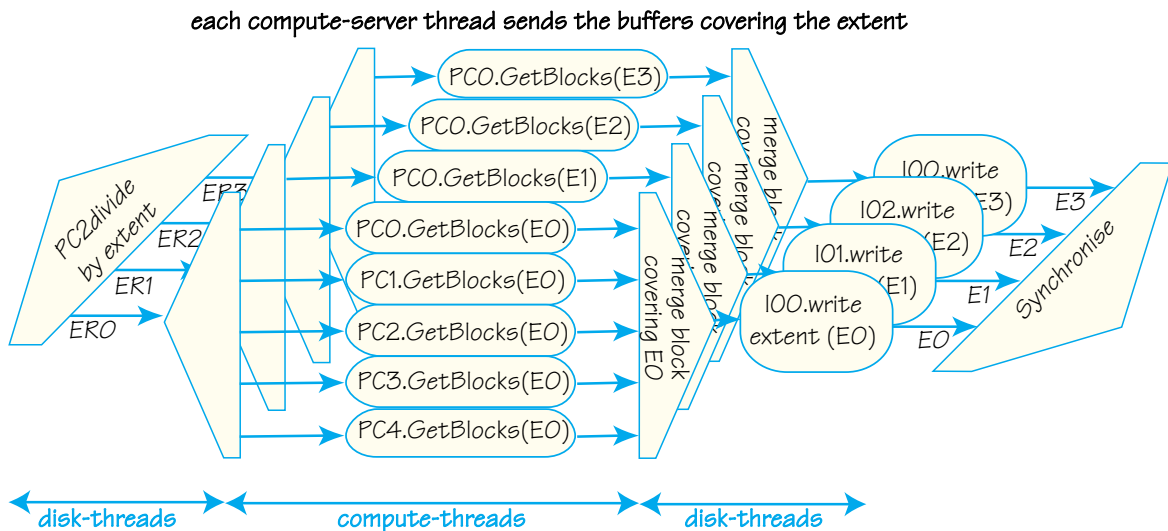


Fig. 5 – Collective-write operation

¹ Each compute-thread reads data from each extent.

² The requests are forwarded to the disk-threads and specify for each disk-thread the extent for which it must gather data from the compute-threads.

written in arbitrary positions in the striped file (support for MPI data types), arbitrary number of stripes in the striped-file, arbitrary number of processes taking part to collective operations (support for MPI communicators), and non-blocking function interfaces. Current performance on the Swiss-T0 machine is limited by the absence of a FastEthernet crossbar switch. Further testing will be conducted under WindowsNT and UNIX, depending on the available configurations.

The MPI interface to the NAFS parallel-striped-file package is currently under development, reusing as much as possible the ROMIO implementation [17].

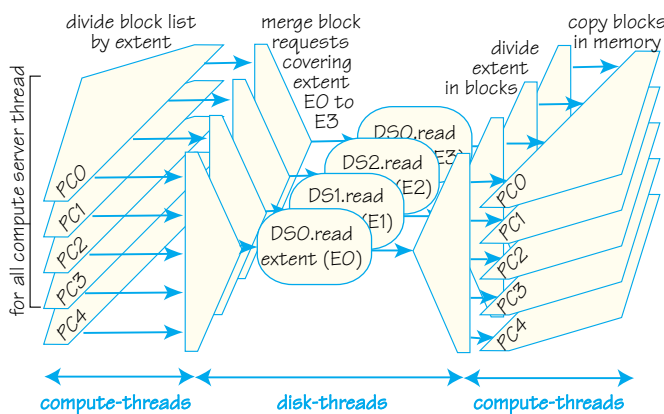


Fig. 6 –The collective-read operation

CONCLUSION

This document presented the issues involved in the design and implementation of a parallel-striped file package. It shows that the parallel-striped-file approach is viable, but requires careful implementation: collective operation interface, and pipelining between data transfers and data accesses. Future work will address the implementation of the MPI-IO interface, performance measurements, and possible optimizations of the existing package.

REFERENCES

- [1] S. Baylor, and C. Wu. *Parallel I/O workload characteristics using Vesta*. In R. Jain, J. Werth, and J. Browne, editors. *Input/Output in Parallel and Distributed Computer Systems*, chapter 7, p. 167-185. Kluwer, 1996.
- [2] P. Crandal, R. Aydt, A. Chien, and D. Reed. *Input-output characteristics of scalable parallel applications*. In Proc. Supercomputing'96. ACM Press, December 1995.
- [3] S. Garg. *Architecture and design of a highly efficient parallel file system*. In Proc. SC'98 High Speed Networking and Computing. Orlando, Florida, USA, November 7 - 13, 1998 (<http://www.supercomp.org/sc98/papers/index.html>).
- [4] W. Gropp, E. Lusk, A. Skjellum, *Using MPI*, The MIT Press, 1994.
- [5] D. Kotz. *Disk-directed I/O for MIMD multiprocessors*. ACM Transactions on Computer Systems, 15(1):41-74, February 1997.
- [6] V. Messerli, O. Figueiredo, B. A. Gennart, and R. D. Hersch. *Parallelizing I/O-intensive image access and processing applications*. IEEE Concurrency, 7(2):28-37. April-June 1999.
- [7] Message Passing Interface Forum, *MPI-2: Extensions to the Message-Passing Interface*, Technical Report, July 1997, <http://www.mpi-forum.org>.
- [8] C. M. Mobarri, T. Sterling, J. Crawford, and D. Ridge. *A comparative analysis of hardware and software support for parallel programming within a global name space*. In Proc. SC'96 High Speed Networking and Computing (<http://www.supercomp.org/sc96/papers/index.html>) 1996.
- [9] N. Nieuwejaar, D. Kotz, A. Purakayastha, C. Ellis and M. Best. *File-access characteristics of parallel scientific workloads*. IEEE transactions on Parallel and Distributed Systems, 7(10):1075-1089, October 1996.
- [10] J. del Rosario, R. Bordawekar, A. Choudhary. *Improved parallel I/O via a two-phase run-time access strategy*. In Proc. Workshop on I/O in Parallel Computer Systems at IPPS'93, pages 56-70, April 1993. Also published in Computer Architecture News, 21(5):31-38, December 1993.
- [11] K. Seamons, Y. Chen, P. Jones, J. Jozwiak, and M. Winslett. *Server-directed collective I/O in Panda*. In Proc. Supercomputing '95. ACM Press, December 1995.
- [12] E. Smirni, R. Aydt, A. Chien, and D. Reed. *I/O requirements of scientific applications*. In Proc. 5th IEEE Int. Symp. on High Performance Distributed Computing, p. 49-59. IEEE Computer Society Press, 1996.
- [13] J. Sturtevant, M. Christon, P. D. Heerman. *PDS/PIO: lightweight libraries for collective parallel I/O*. In Proc. SC'98 High Speed Networking and Computing. Orlando, Florida, USA, November 7 - 13, 1998 (<http://www.supercomp.org/sc98/papers/index.html>).
- [14] R. Thakur, and A. Choudhary. *An extended two-phase method for accessing sections of out-of-core arrays*. Scientific Programming, 5(4):301-317, Winter 1996.
- [15] R. Thakur, W. Gropp, and E. Lusk. *A case for using MPI's derived datatypes to improve I/O performance*. In Proc. SC'98 High Speed Networking and Computing. Orlando, Florida, USA, November 7 - 13, 1998 (<http://www.supercomp.org/sc98/papers/index.html>).
- [16] R. Thakur, W. Gropp, and E. Lusk. *An experimental evaluation of the parallel I/O systems of the IBM SP and Intel Paragon using a production application*. In Proc. 3rd Int. Conf. of the Austrian Center for Parallel Computation (ACPC) with special emphasis on parallel databases and parallel I/O. LNCS 1127. Springer-Verlag, September 1996.
- [17] R. Thakur, W. Gropp, and E. Lusk. *Users guide for ROMIO, A high-performance, portable MPI-IO implementation*. TR ANL/MCS-TM-234. Mathematics and Computer Science Division, Argonne National Laboratory, revised July 1998.
- [18] G. Bell, C. van Ingen. *DSM Perspective: another point of view*. In Proc. of the IEEE, 87(3):412-417, March 1999. ■

A PARALLEL DISCRETE ELEMENT METHOD FOR INDUSTRIAL GRANULAR FLOW SIMULATIONS

MARK L. SAWLEY, PAUL W. CLEARY. CSIRO MATHEMATICAL & INFORMATION SCIENCES, CLAYTON, AUSTRALIA

La simulation numérique d'écoulements granulaires, comme tant d'autres méthodes particulières, demande des ressources de calcul importantes pour nombre de problèmes de grande taille auxquels l'industrie s'intéresse. Le calcul parallèle a le potentiel pour repousser les limites de capacité des systèmes existants, permettant l'étude de systèmes granulaires plus proches de la réalité. Nous présentons ici l'implémentation d'un algorithme parallèle, ainsi que des mesures de performances d'un système parallèle formé de composants standard interconnectés par un réseau à basse latence et haut débit. Les résultats valident l'algorithme parallèle mais aussi le rôle important que ce type de système parallèle peut jouer dans les simulations d'écoulements granulaires industriels demandant de grandes puissances de traitement.

The numerical simulation of granular flows, like many other particle-based methods, is computationally intensive for large-scale problems of industrial interest. Parallel computation has the potential to alleviate current compute-based limitations, allowing much larger granular systems with greater physical reality to be analysed. A study of the implementation of a parallel algorithm is presented, together with performance measurements on a commodity cluster computer system. The results obtained validate not only the parallel algorithm, but also the potential role of such computer systems in industrial granular flow simulations.

INTRODUCTION

Granular or particulate materials, composed of a large number of loosely-packed individual particles or grains, are an integral part of our environment; common examples are sand, powder, rocks and grains in their many forms. The flow of such granular material plays a critical role in many industrial processes, such as separation and mixing (eg in the manufacture of glass and pharmaceuticals), rotary moulding of plastics, mineral production and processing (eg blasting and excavation, crushing and grinding of mineral ores), commodity sampling (eg from conveyor belts), stockpile construction and discharge, and flows in and from hoppers and silos. Since small reductions in energy consumption or increases in output represent substantial financial benefits for plant owners, significant effort is placed on improving the efficiency of such processes. Numerical

simulation of granular flows has an increasing role to play in this optimisation procedure.

Granular flows are known to exhibit strongly different behaviour than conventional continuum flows. This behaviour can be numerically simulated using the Discrete Element Method (DEM), which involves tracking the movement of all the individual particles, as well as their interactions with other particles and with their surroundings. DEM simulations – sometimes referred to as Granular Dynamics – can be viewed as a macroscopic-level equivalent of short-range Molecular Dynamics [1,2], in which the inelastic nature of particle interactions is taken into account. The simulation of simple flows using DEM has been established for several years [3-5]. More detailed flows have also been successfully computed, such as for geophysical [6,7], mineral processing [8-10], and bulk material handling [9-12] applications.

The realistic simulation of industrial granular flows may involve the tracking of many millions of particles, as well as a high level of complexity to describe particle interactions involving breakage, attrition, cohesion and aggregation. In addition, the presence of particles of non-spherical shape, which can significantly affect flow behaviour, increases the demands on contact detection. Such complex simulations require sizeable computational resources. Fortunately, particle-based numerical methods can generally be parallelised in a relatively straightforward manner.

For DEM simulations, parallelisation is facilitated by the fact that the particles interact via short-range collisions [13-17]. This leads to a high degree of data localisation and an associated modest level of data communication between processors. Nevertheless, there are a number of considerations that must be taken into account to obtain good performance from a parallel DEM code. In particular, granular flows can be highly dynamic, with particles that are in close proximity at one time rapidly becoming more distant. To maintain efficient use of all processors under such conditions, dynamic load balancing is essential.

In the present paper, a parallel DEM implementation that incorporates dynamic load balancing is described. Using this code, calculations have been undertaken for the flow of granular material from two different slot hoppers. Performance results obtained on a commodity cluster computer system, the Swiss-T0-Dual installed at the Ecole Polytechnique Fédérale de Lausanne [18], are presented.

THE DISCRETE ELEMENT METHOD

DEM simulations involve solving the equations of motion for the trajectory, spin and orientation of every

particle in the flow and modelling each collision between particles and between particles and the surrounding boundary objects. When appropriate, equations of motion are also solved for the boundary objects with which the particles interact in order to treat moving boundaries.

The DEM variant used in our studies is sometimes referred to as the *soft particle method*. The particles are allowed to overlap and the amount of overlap Δx , and normal v_n and tangential v_t relative velocities determine the collisional forces. There is a range of possible contact force models available that approximate the collision dynamics to various extents. A conventional linear spring-dashpot model, as shown schematically in Fig. 1, is used in these simulations.

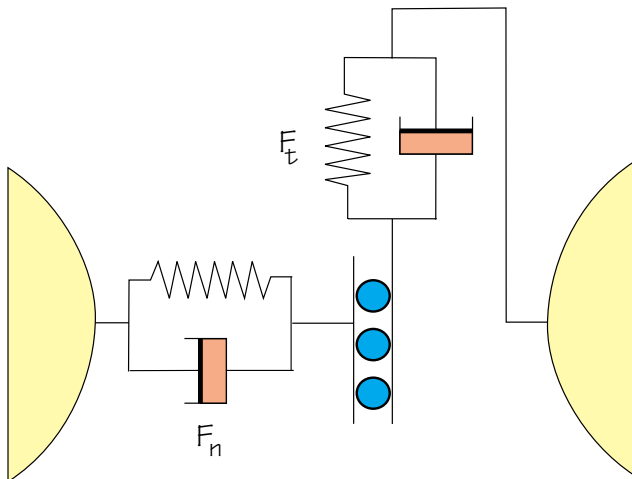


Fig. 1 – Schematic diagram of the linear spring-dashpot model for the collisional normal (F_n) and tangential (F_t) forces acting between particles

The normal force $F_n = -k_n \Delta x + C_n v_n$ consists of a linear spring to provide the repulsive force and a dashpot to dissipate a proportion of the relative kinetic energy. The maximum overlap between particles is determined by the stiffness k_n of the spring in the normal direction. Typically, average overlaps of 0.1-1.0% are desirable, requiring spring constants of the order of 10^6 - 10^7 N/m. The normal damping coefficient C_n is chosen to give the required coefficient of restitution ϵ , defined as the ratio of the post-collisional to pre-collisional normal component of the relative velocity.

The tangential force is given by

$$F_t = \min \{ \mu F_n, k_t \int v_t dt + C_t v_t \},$$

where the integral of the tangential velocity v_t over the collision behaves as an incremental spring that stores energy from the relative tangential motions and represents the elastic tangential deformation of the contacting surfaces. The dashpot dissipates energy from the tangential motion and models the tangential plastic deformation of the contact. The total tangential force (given by the sum of the elastic and plastic components) is limited by the Coulomb frictional limit (with μ the coefficient of dynamic friction) at which point the surface contact shears and the particles begin to slide over each other.

The DEM algorithm itself is relatively simple, and is comprised of three essential parts:

- A particle near-neighbour interaction list is constructed with the aid of an overlaying Cartesian grid. The size of the grid cells is chosen such that there is at most one particle in each cell, and the surrounding cells searched for neighbouring particles. This grid search technique [5,11,16] reduces the nearest neighbour search procedure to an $O(N)$ operation, where N is the total number of particles. This is essential for DEM simulations involving a large number of particles.
- The collisional forces on each of the particles and boundary objects are evaluated efficiently using the near-neighbour list and the spring-dashpot interaction model and then transformed into the simulation frame of reference.
- All the forces on the particles and surrounding boundary objects are summed and the resulting equations of motion are integrated to advance the positions in time. A second-order predictor-corrector scheme is used, with between 10 and 50 timesteps required to integrate accurately each collision. This gives very small timesteps, typically 10^{-3} to 10^{-6} s, depending on the length and time scales of each application.

Due to the relative motion between particles, it is necessary to update periodically the near-neighbour list. In addition, at selected times various physical quantities of interest are calculated, such as the particle velocity and force distributions and the forces exerted on the boundaries.

The above DEM algorithm has been employed to compute a wide range of two- and three-dimensional granular flow simulations [9,10]. Examples of three-dimensional DEM simulations of particular interest to the mineral processing industry are shown in Fig. 2.

PARALLEL DEM IMPLEMENTATION

DEM simulations involving a relatively modest number of circular or spherical particles (<100,000), such as those shown in Fig. 2, can be performed in a reasonable time on a single-processor workstation. As the geometric and physical complexity of the modelling increases, so does the computational resources required. The simulation of, for example, one million non-spherical particles in a complex 3D geometry, including the effects of breakage and/or cohesion, is thus not reasonably performed by a single processor. However, the required level of computational power can be obtained by parallel DEM simulations.

As a preliminary study of the development of a parallel DEM code, the parallelisation of two-dimensional simulations has been investigated. The goal was to produce a portable parallel implementation that would exhibit good performance scaling up to 100's of processors for large-scale simulations. The code is required to run on parallel computer systems having either shared or distributed memory.

A number of different algorithms can be successfully employed to parallelise particle-based computations [14-16]. The parallel algorithm used for the present study is of SPMD (Single Program, Multiple Data) style, based on

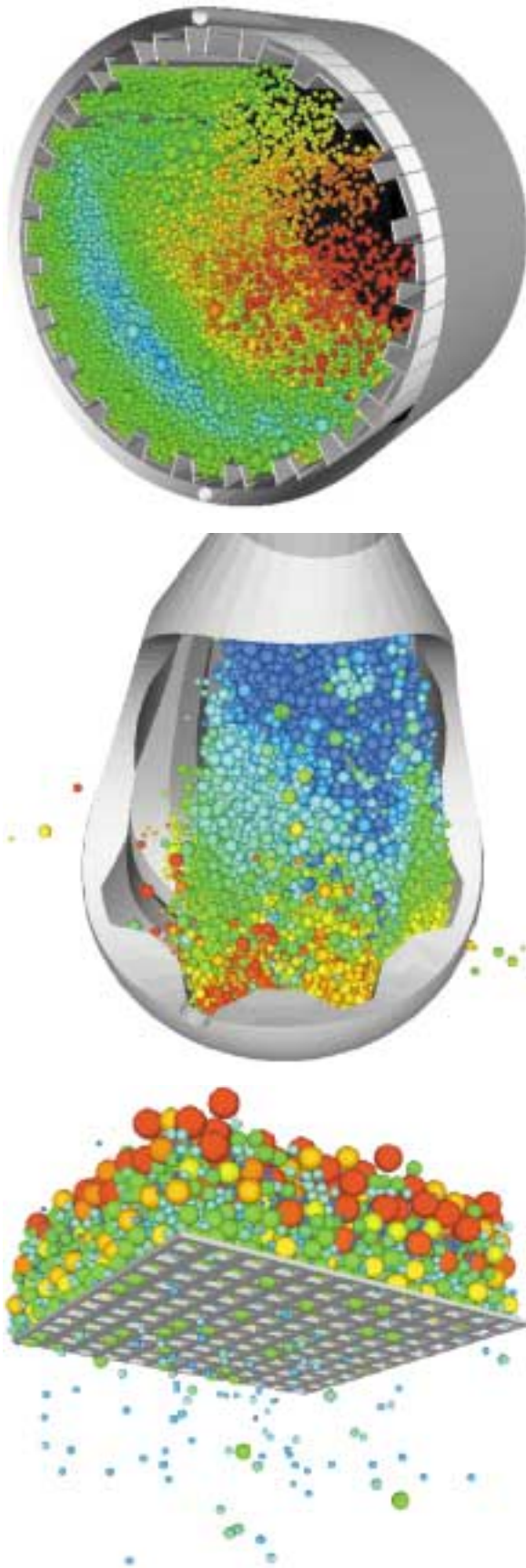


Fig. 2 – Examples of three-dimensional DEM simulations: charge motion in (top) a ball mill and (middle) the Hicom nutating mill, and (bottom) size segregation by a vibrating screen [10]

spatial domain decomposition. The simulation domain is divided into subdomains by a simple slicing parallel to either the x or y axis. Choosing the number of subdomains equal to the number of processors available, data associated with particles in the same subdomain are stored in the same processor's memory. The computational effort depends on the number of collisions between particles, which is closely related to the number of particles. Assuming processors of equal performance, to achieve load balance (in order to minimise synchronisation delays) the subdomains are chosen to contain the same number of particles.

It should be noted that a block decomposition may lead to a smaller amount of data exchange between processors than the strip decomposition that has been chosen. Nevertheless, a block decomposition is generally found to be only marginally more efficient, since its more complicated communication pattern involves a greater number of short length exchanges of data [13]. In addition, implementation of a dynamic load balancing scheme is considerably more complex.

Of the different parallel programming models available, message passing (using either the MPI or PVM library) has been chosen for the present implementation. This model provides both low communication overhead (leading to higher parallel performance) and a high level of code portability (since message passing is available on essentially all parallel computer systems).

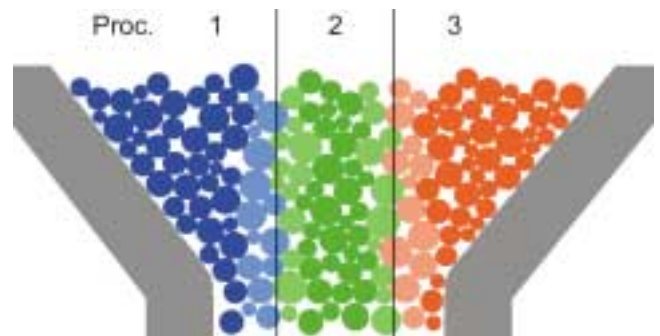


Fig. 3 – Particles coloured according to storage location in memory for decomposition into three subdomains
(Light colours denote ghost particles)

Data localisation – and hence inter-processor communication – is optimised by copying a layer of “ghost” particles from the neighbouring subdomains, as shown in Fig. 3. The width of this layer is chosen to ensure that all interactions can be calculated from local data. The use of ghost particles also simplifies the dynamic load balancing, since renaming ghost particles as physical particles is equivalent to moving particles between subdomains. It is important to note that the subdomain boundaries are not fixed in physical space, but are automatically moved to satisfy the load balancing requirements.

A flow diagram for the parallel DEM algorithm is presented in Fig. 4. After the input data is read and

discharge flow pattern has made them attractive for numerical simulation, using both discrete element and continuum methods. While both methods can qualitatively predict mass flow, neither method has been able to predict all the observed phenomenon [12]. In some situations – such as when the grains are not significantly smaller than the discharge port – the granular nature of the material cannot be neglected, and a continuum approach is not valid. DEM simulations that incorporate a sufficient level of physical modelling can provide valuable qualitative and quantitative insights into complex phenomena observed during hopper discharge [5,9-12].

Fig. 5 presents, for each of these two cases, plots of the particle locations at an early time during the discharge. Since the present paper is concerned primarily with the code performance, the reader is referred to [9] for more details of the physical interpretation of the simulation results.

The performance of the parallel DEM code implementation has been assessed on the Swiss-T0-Dual machine. This commodity cluster computer system consists of 8 dual-processor Digital Alpha 21164 (EV5.6) boxes [18]. Each processor has a 4 MB level 3 cache, with the total system having a distributed memory of 8 GB and a peak performance of 16 GFlop/s. The processors are connected via both an EasyNet bus and a Fast Ethernet switch; each dual-processor box has one PCI-based connection to the EasyNet bus and one Fast Ethernet port. Initial measurements [18] indicated a communication bandwidth between boxes of 35 MB/s for EasyNet and 10 MB/s for Fast Ethernet, and latencies of 12 μ s for MPI-lite (using EasyNet) and 500 μ s for MPICH (using Fast Ethernet). A maximum of 8 processors can be used with MPI-lite, and up to 16 with MPICH (however, the inter-box bandwidth is halved and latency doubled if more than 8 processors are used). The Digital UNIX operating system is run on each box.

For each of the hopper flows described in the previous section, computations have been performed for different numbers of processors, using the MPI code version for both the Fast Ethernet and EasyNet interconnect. For comparison, computations have also been undertaken using the serial code version on one processor. To obtain these performance measurements, the flow has been computed for only the first 1 s (about 55,000 timesteps) for the single-port hopper and the first 0.1 s (about 1100 timesteps) for



mpi_init	mpi_isend
mpi_comm_rank	mpi_recv
mpi_comm_size	mpi_wait
mpi_finalize	mpi_barrier

HOPPER DISCHARGE

Hoppers are common storage devices for granular materials. Their relatively simple geometry and well-defined

the dual-port hopper. Measurements have been made of the time required to perform the individual tasks of the DEM algorithm. This has enabled the determination of the amount of time spent in both computation and communication / synchronisation to be determined, and also the performance speedup (defined as the ratio of simulation times using the serial code and the parallel code).

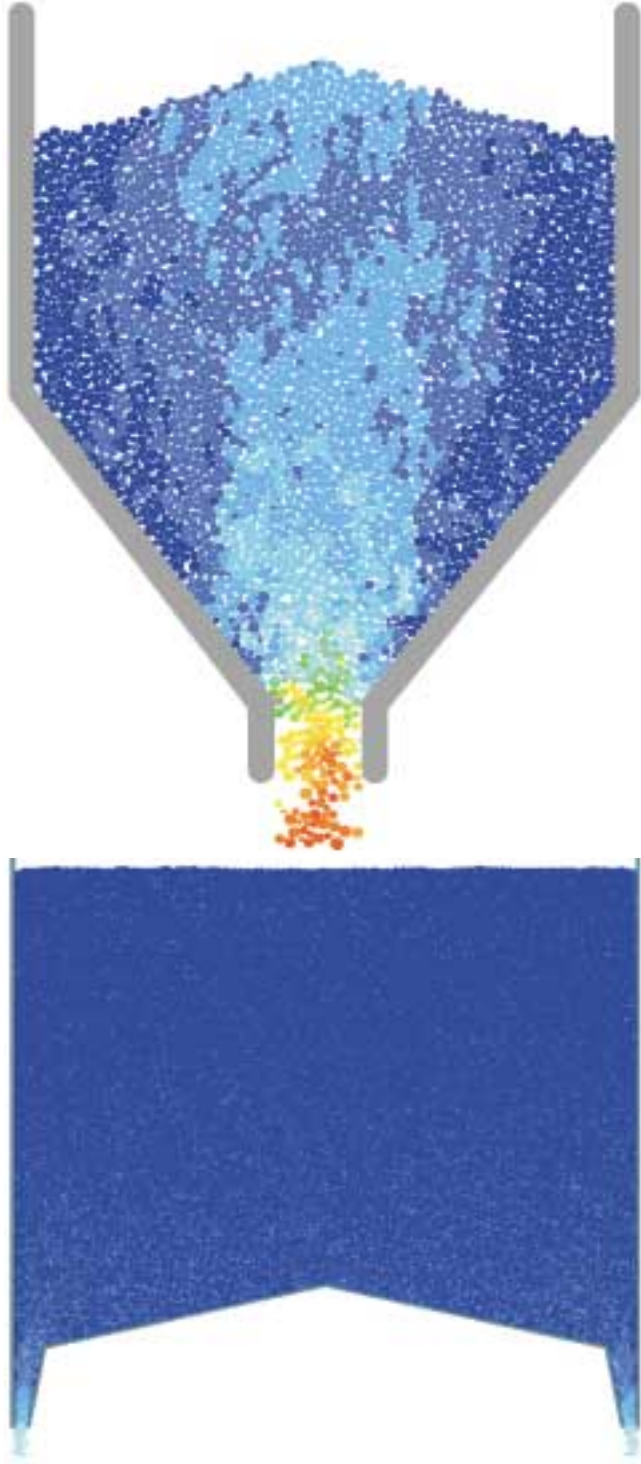


Fig. 5 – Discharge of (top) 3545 circular particles from a single-port hopper, and (bottom) 200,000 circular particles from a dual-port hopper (The particles are coloured by their velocity, increasing from blue to red)

The results of timing measurements for the code using MPICH are presented in Figs. 6 and 7. The times presented are averages of the values measured on the different processors used in the simulations. Only slightly different values were measured for simulations using MPI-lite (via the EasyNet bus).

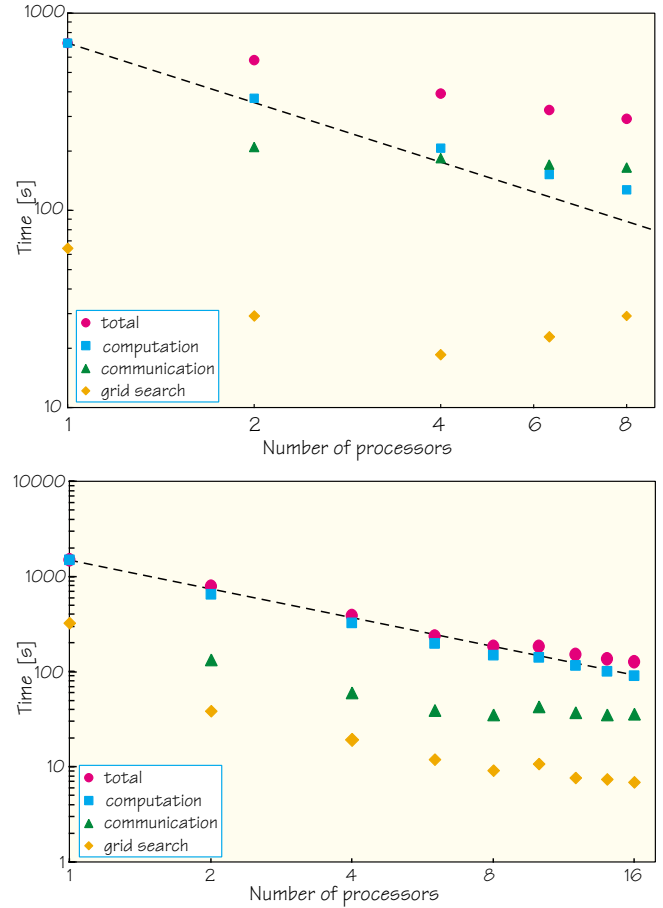


Fig. 6 – Computational time required for the individual tasks of the DEM simulation of (top) single-port and (bottom) dual-port hoppers (The dashed line represents linear scaling)

From the timing results presented in Fig. 6 it can be observed that as the number of processors used is increased, the simulation time required decreases. For both hopper flows the computation time decreases roughly linearly with the number of processors, whereas the time required for inter-processor communication remains approximately constant. This results in the flattening of the speedup curve observed in Fig. 7. It should be remarked that for the larger problem size of the dual-port hopper case, the ratio of computation to communication is substantially higher; this results in a higher parallel efficiency for larger number of processors. In fact, the single-port hopper case represents a very small problem size for which parallel efficiency is rather low using MPICH on the Swiss-T0-Dual. Nevertheless, the timing results for this case show that with a relatively modest reduction of the communication time (as expected from a faster interconnection network), good scalability should be attainable for up to around 8 processors.

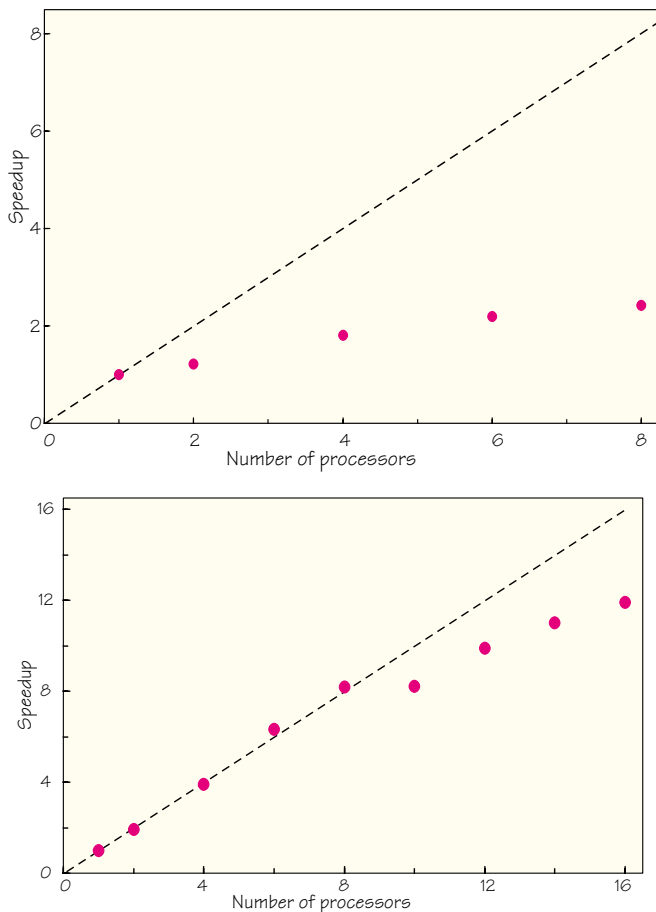


Fig. 7 – Measured speedup for the DEM simulation of (top) single-port and (bottom) dual-port hoppers (The dashed line represents linear scaling)

For the hopper flows considered here, the ghost particle data is updated much more frequently than the near-neighbour list. Thus, as seen in Fig. 6, the grid search requires only a very small proportion of the total simulation time. While this proportion may be somewhat larger for more dynamic flows involving very rapidly moving particles, it should never be the source of a computational bottleneck.

For the dual-hopper case, the performance is observed to scale linearly for up to 8 processors. The relative decrease in performance observed for more than 8 processors is attributed to the degradation of communication when more than one processor in at least one of the dual-processor boxes is used.

Given the substantially greater performance of EasyNet compared to Fast Ethernet, it is surprising that its use did not yield a significant improvement in code performance. To update the ghost particle data, about 5.5 kB of data per iteration is passed in each direction across each of the subdomain interfaces for the single-port hopper case, and about 30 kB for the dual-port hopper case. The communication time determined using the network performance data given above is, however, much smaller (by a factor of up to 5) than the time measured to perform the update of ghost particle data. Two possible reasons for this discrepancy

are that either there are unaccounted for communication overheads, or that the simulation is not well balanced resulting in significant synchronisation overheads. A close examination of the subroutine timings for each processor indicates that the simulation is well balanced, and therefore appears to rule out the second of these reasons. It is anticipated that the significantly improved performance of the TNet crossbar switch (to be installed on the next-generation Swiss-T1 machine, see page 3) will both shed light on this discrepancy, and provide a substantial increase in performance of the parallel DEM code.

FUTURE WORK

A number of extensions to the preliminary study reported here are planned. Firstly, while two-dimensional flows have been considered to date, the real benefit of code parallelisation is to be realised for large three-dimensional flow simulations [10]. The extension of the parallel DEM algorithm from two to three dimensions should be straightforward.

The hopper flows described above were concerned with poly-dispersed circular particles, having a moderate ratio of the largest to smallest particle sizes. For the strongly poly-dispersed non-spherical particles to be considered in future simulations (having particle size ratios of up to 250), the search algorithm used here is no longer appropriate. A more suitable search algorithm and contact detection has been implemented in the serial code version; parallelisation of this advanced search algorithm should not, however, require modifications to the techniques presented here.

Of particular importance to the accurate simulation of complex granular flow behaviour is a sufficiently refined modelling of inter-particle collisions. The inclusion of breakage, attrition and cohesion models into the DEM formulation is essential for certain industrial applications.

CONCLUSIONS

A preliminary investigation of the implementation of a spatial domain decomposition technique for parallel granular flow simulations has been presented. The parallel algorithm developed incorporates dynamic load balancing, which is essential to obtain good performance from a parallel DEM code. Performance measurements obtained on a prototype commodity cluster computer system for two hopper flow simulations have confirmed that good parallel performance can be obtained for sufficiently large problems. Improved scalability is to be expected on future computer systems having a greater communication performance.

The performance of parallel simulations increases as the ratio of computation to communication increases. For parallel DEM simulations, increasing the problem size and the complexity of the contact detection and physical modelling will require an increase in computational effort, but only a modest increase in inter-processor communication. It is therefore possible to perform numerical simulations

of large-scale industrial granular flow problems in an efficient manner on commodity cluster computer systems having substantially more processors than available on the machine employed in the present study.

ACKNOWLEDGEMENTS

The authors wish to acknowledge the Service Informatique Central of the Ecole Polytechnique Fédérale de Lausanne for providing access to the Swiss-T0-Dual machine and, in particular, to Nouredine El Mansouri for his technical support.

REFERENCES

- [1] M.P. Allen and D.J. Tildesley, *Computer simulation of liquids*, Oxford Science (1989).
- [2] D.C. Rapaport, *The art of molecular dynamics simulation*, Cambridge University Press (1995).
- [3] G.C. Barker, *Computer simulations of granular materials*, in "Granular Matter: An Inter-disciplinary Approach", A. Mehta (ed.), Springer-Verlag (1994).
- [4] O.R. Walton, *Numerical simulation of inelastic frictional particle-particle interaction*, in "Particulate Two-Phase Flow", M.C. Roco (ed.), Butterworth-Heinemann (1994) 884-911.
- [5] G.H. Ristow, *Granular dynamics: a review about recent molecular dynamics simulations of granular materials*, Annual Reviews of Computational Physics, **1** (1994) 275-308.
- [6] M.A. Hopkins, W.D. Hibler and G.M. Flato, *On the numerical simulation of the sea ice ridging process*, Journal of Geophysical Research, **96** (1991) 4809-4820.
- [7] C.S. Campbell, P.W. Cleary and M.A. Hopkins, *Large-scale landslide simulations: global deformation, velocities and basal friction*, Journal of Geophysical Research – Solid Earth, **100** (1995) 8267-8283.
- [8] B.K. Mishra and R.K. Rajamani, *The discrete element method for the simulation of ball mills*, Applied Mathematical Modelling, **16** (1992) 598-604.
- [9] P.W. Cleary, *Discrete element modelling of industrial granular flow applications*, TASK. Quarterly - Scientific Bulletin, **2** (1998) 385-416.
- [10] P.W. Cleary and M.L. Sawley, *Three-dimensional modelling of industrial granular flows*, Proceedings of the Second International Conference on CFD in the Minerals and Process Industries (Melbourne, 1999), to appear. See also: www.cmis.csiro.au/cfd/dem
- [11] G.H. Rong, S.C. Negi and J.C. Jofriet, *Simulation of flow behaviour of bulk solids in bins. Parts 1 and 2*, Journal of Agricultural and Engineering Research, **62** (1995) 247-269.
- [12] J.M.F.G. Holst, J.M. Rotter, J.Y. Ooi and G.H. Rong, *Numerical modeling of silo filling. I: Continuum analysis & II: Discrete element analysis*, Journal of Engineering Mechanics, **125** (1999) 94-110.
- [13] G.A. Kohring, *Dynamic load balancing for parallelized particle simulations on MIMD computers*, Parallel Computing, **21** (1995) 683-693.
- [14] S. Plimpton, *Fast parallel algorithms for short-range molecular dynamics*, Journal of Computational Physics, **117** (1995) 1-19.
- [15] J.-A. Ferrez, D. Müller and T.M. Liebling, *Parallel implementation of a distinct element method for granular media simulation on the Cray T3D*, EPFL Supercomputing Review, **8** (1996) 4-7.
- [16] G.A. Kohring, *Dynamical simulations of granular flows on multi-processor computers*, Computational Methods in Applied Sciences '96, J.-A. Désidéri et al. (eds), Wiley (1996) 190-196.
- [17] C.M. Dury, R. Knecht and G.H. Ristow, *Size segregation of granular materials in a 3D rotating drum*, in "High-Performance Computing and Networking", P. Sloot et al. (eds), Springer (1998) 860-862.
- [18] R. Gruber and Y. Dubois-Pèlerin, *Swiss-Tx: First experiences on the T0 system*, EPFL Supercomputing Review, **10** (1998) 19-23. See also: <http://capawww.epfl.ch/swiss-tx>
- [19] See Supercomputing Systems: Remote Store Architecture, <http://www.scs.ch/rsa.html> ■

TEST DE PERFORMANCE DU CODE SPECULOOS SUR L'ORDINATEUR PARALLÈLE T0-DUAL

DANIEL WEILL, EPFL-DGM, LABORATOIRE DE MÉCANIQUE DES FLUIDES

The use of spectral methods in the purpose of simulating unsteady flows in complex geometries implies increasing computational costs in terms of memory and CPU time. These methods being very expensive, the parallelisation of the codes using them becomes necessary. We propose in this article to study the performance of the code SPECULOOS parallelised using the MPI library. The tests are made on the new Swiss T0-Dual computer with 16 processors. Comparisons will be done with the former T0 computer with 8 processors.

L'utilisation des méthodes spectrales pour la résolution d'équations différentielles partielles, comme l'équation de Navier-Stokes, nécessite de plus en plus de moyens informatiques puissants afin de mener à bien les simulations d'écoulements non-stationnaires dans des géométries complexes. Ces méthodes étant très gourmandes en espace mémoire ainsi qu'en temps CPU, la parallélisation des codes qui les utilisent devient une voie nécessaire à suivre. Cet article propose de mesurer les performances en parallélisme du code SPECULOOS écrit en langage C++, en utilisant la librairie MPI. Les

tests sont effectués sur la nouvelle machine T0-Dual à 16 processeurs. Une comparaison sera effectuée avec le précédent T0 à 8 processeurs.

INTRODUCTION

Nous effectuons une étude de performance en parallélisme du code SPECULOOS, solveur des équations de Navier-Stokes incompressibles instationnaires pour un fluide visqueux par une méthode numérique d'éléments spectraux.

$$\begin{cases} \frac{\partial \mathbf{u}}{\partial t} + (\mathbf{u} \cdot \nabla) \mathbf{u} = -\nabla p + \frac{1}{Re} \nabla^2 \mathbf{u} + \mathbf{f} \\ \nabla \cdot \mathbf{u} = 0 + \text{conditions aux bords/initiales} \end{cases}$$

($\mathbf{u}$ est le vecteur vitesse, p la pression, $\mathbf{f}$ un terme de force de volume).

Plusieurs cas test ont été étudiés, parmi eux l'équation de Poisson tridimensionnelle dans un cube, un écoulement de Navier-Stokes bidimensionnel dans une cavité carrée entraînée à Reynolds 1000 ainsi qu'un écoulement bidimensionnel autour d'un obstacle de forme rectangulaire à Reynolds 100. A la différence de la méthode des éléments finis (MEF) ou des volumes finis (VF), la méthode des éléments spectraux est plus précise, ce qui en fait un atout incontestable pour la recherche fondamentale où l'on étudie souvent des phénomènes très sensibles et où l'on veut que les effets numériques, parasites et non-physiques, soient les plus réduits possible. De plus, contrairement aux méthodes spectrales classiques (comme celle de Fourier), cette méthode peut traiter de problèmes à géométrie complexe et non-périodique, ce qui la rend attrayante dans le domaine de l'ingénierie.

Son principe de base consiste à mettre les équations différentielles sous forme faible, à décomposer les grandeurs étudiées dans un espace polynômial adapté et à discrétiser ensuite les intégrales selon une quadrature de Gauss. La base polynômiale est constituée des interpolants de Lagrange-Legendre. La discrétisation se fait en prenant le produit scalaire des équations de Navier-Stokes avec les fonctions tests représentant la base polynômiale, que l'on choisit être les interpolants de Lagrange-Legendre $h(r)$ définis sur l'élément parent $[-1,1]$ et dont l'une des propriétés est que $h_i(\xi_j) = \delta_{ij}$, où $\xi_j \in [-1,1]$ sont les points de collocation de Gauss-Lobatto-Legendre (qui sont les zéros de la dérivée du polynôme de Legendre d'ordre n , $L'_n = 0$ plus les extrémités de l'intervalle). La vitesse u est approchée sur la grille constituée des points de Gauss-Lobatto-Legendre. La pression p est approchée sur la grille de Gauss-Legendre, constituée des points de collocation $\xi_j \in]-1,1[$ qui sont les zéros de $L_{n-1} = 0$, pour n fixé. Le solveur des équations discrétisées est de type itératif. Il s'agit d'un gradient conjugué avec préconditionnement.

En ce qui concerne la discrétisation temporelle, un schéma d'Euler d'ordre deux est utilisé pour le traitement implicite du terme visqueux, alors que le terme non-linéaire

est traité par une extrapolation explicite d'ordre deux. Une méthode de découplage vitesse/pression est mise en oeuvre et fait appel à une décomposition LU généralisée des équations discrétisées. Pour plus de détails concernant la méthode des éléments spectraux, on peut se référer aux articles suivants [4] et [3].

Un des aspects originaux de cette recherche est que SPECULOOS est écrit dans un langage de programmation par objet, le C++. Cette approche permet plus de clarté dans la lecture du code source, et laisse la possibilité de pouvoir réutiliser plus facilement des parties de code ultérieurement. Les méthodes d'éléments spectraux nécessitant beaucoup de puissance de calcul à cause de la précision qu'elles exigent, la parallélisation de ce code devient une étape incontournable pour son utilisation ultérieure. Ce code fait appel à la librairie de communication parallèle MPI. C'est aujourd'hui l'un des standards les plus utilisés dans le monde. Ceci permet de le porter sur une plus large gamme d'ordinateurs parallèles. Pour notre part nous utilisons le Swiss-Tx (T0-Dual à 16 processeurs DEC à 500 MHz distribué sur 8 boxes bi-processeurs à 1 Gb de mémoire et 5 Mb de mémoire cache [5]). Une version optimisée d'une librairie parallèle pour le T0-Dual est aussi utilisée et se nomme FCI.

TESTS

Dans le numéro 10 de la présente revue (novembre 1998,[1]) nous vous avons présenté une suite de tests effectués avec SPECULOOS dans différents cas de figure. Ces tests se sont faits sur la version précédente du T0. Nous vous proposons donc de refaire certains de ces tests dans le cas du T0-Dual. Nous utilisons 1, 2, 4 et 8 processeurs.

La librairie parallèle est MPI. Dans un deuxième temps nous effectuons les mêmes mesures en utilisant, cette fois, le protocole de communication FCI, qui est spécialement adapté au Swiss-Tx.

TEST 1: EQUATION DE POISSON 3D

Nombre de processeurs	Eléments par processeur	MPI TO	FCI TO	MPI (TO-dual)	FCI (TO-dual)
1	12x8x8	1	1	1	1
2	16x8x8	1.7	2.1	1.86	1.94
4	6x4x8	3.4	4.0	3.71	3.68
8	6x4x4	6.6	6.9	6.69	6.72

Tableau 1 – Equation de Poisson 3D

Le test No 1 résout l'équation de Poisson en 3D sur un cube $[0,1[x[0,1[x[0,1[$ contenant 12x8x8 éléments. L'ordre dans chaque élément est respectivement de 8, 10 et 11, selon x, y et z. Les résultats du speedup sont présentés sur le tableau 1. Le comparatif s'arrête à huit processeurs car pour l'instant FCI ne gère pas encore seize processeurs.

TEXT 2: CAVITÉ ENTRAÎNÉE 2D

Nombre de processeurs	Éléments par processeur	MPI TO	FCI TO	MPI (TO-dual)	FCI (TO-dual)
1	12x16	1.0	1.0	1.0	1.0
2	12x 8	1.7	2.3	1.85	1.79
4	6x8	2.8	4.2	3.67	3.95
8	6x4	5.1	7.9	5.86	7.61

Tableau 2 – Navier-Stokes 2D incompressible

Le deuxième test que nous avons retenu (test 4 de l'article mentionné en [1]) est celui de la résolution des équations de Navier-Stokes 2D instationnaires dans une cavité carrée $[0,1[x[0,1[$ dont le couvercle est entraîné à une vitesse donnée. Le nombre de Reynolds est fixé à 1000. Nous avons fait 20 pas de temps. Le domaine carré est constitué de 12×16 éléments avec un ordre polynomial respectivement de 9 et 7 dans chaque direction. Les résultats sont donnés sur le tableau 2.

On remarque de manière globale sur les deux tests que les performances sont légèrement meilleures lorsque nous utilisons FCI. Cependant nous ne pouvons rien déduire entre les performances du T0-Dual et celle du T0, car pour ce dernier le speedup est à dominante super-linéaire. Cela provient des effets d'accélération dus à la mémoire cache, et celle-ci n'est pas la même sur les deux machines

TEST 3: ECOULEMENT AUTOUR D'UNE CONSTRICTION 2D

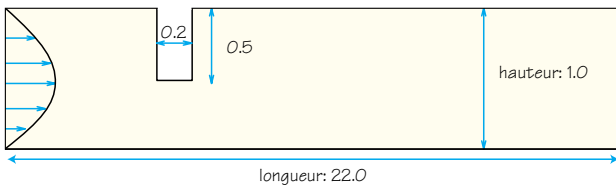


Fig. 1 – Géométrie du domaine (pas à l'échelle)

Nous avons aussi fait un test de performances sur le T0-Dual en prenant le cas de la constriction 2D. Le domaine de simulation est représentée schématiquement sur la figure 1. Il est constituée de 1024 éléments quadrilatéraux. L'ordre polynomial dans chaque élément est de 7×7 . Le nombre de Reynolds est de 100. Nous effectuons 200 pas de temps. Les conditions aux limites incluent un profil de vitesse de type Poiseuille à l'entrée du domaine, de type non-glissement sur les parois solides et des conditions naturelles homogènes à la sortie,

$$\mathbf{v} \frac{\partial \mathbf{u}}{\partial \mathbf{n}} - p \mathbf{n} = 0$$

Encore une fois nous nous intéressons au speedup du code.

La méthode des éléments spectraux est particulièrement bien adaptée à une implémentation sur ordinateur parallèle. Pour que la distribution des éléments spectraux sur les différents processeurs soit optimale, il y a deux contraintes que nous devons prendre en compte. La première est de distribuer la charge de calcul sur les processeurs de la

manière la plus équilibrée. La seconde exige que le minimum de communication se fasse entre processeurs lorsqu'un transfert de données est nécessaire. La solution optimale qui doit satisfaire à ces deux exigences n'est de loin pas triviale et demande de faire appel à des logiciels spécialisés dans cette tâche. Le résultat est montré sur la figure 2.

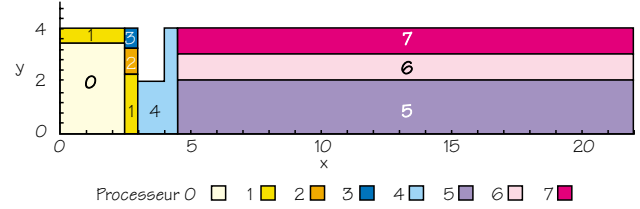


Fig. 2 – Distribution des éléments selon 8 processeurs

Les résultats sont montrés dans le tableau 3 (la simulation avec 1 processeur est prise comme référence) pour MPI et FCI. La figure 3 présente les lignes de courant de l'écoulement. On voit bien apparaître le tourbillon derrière la constriction.

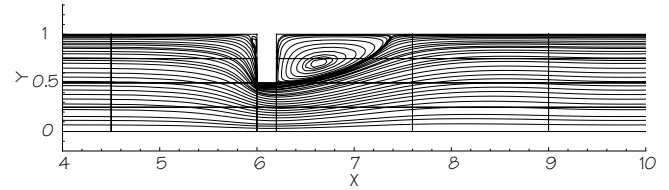


Fig. 3 – Lignes de Courant

Le tableau 3 laisse apparaître un speedup super-linéaire pour le cas à 2 processeurs.

Nombre de processeurs	temps MPI (sec.)	MPI (TO-dual)	temps FCI (sec.)	FCI (TO-dual)
1	3368.5	1	3497.3	1
2	1664.6	2.02	1744.7	2.01
4	929.9	3.62	891.7	3.92
8	507.6	6.64		
16	359.9	9.36		

Tableau 3 – Speedup vs. nombre de processeurs (TO-Dual)

Cet effet provient encore de la mémoire cache, dont les transferts de données se font beaucoup plus vite avec le processeur, que de celui-ci avec la mémoire conventionnelle. Une manière de contourner cette singularité serait de faire des simulations avec une taille de domaine beaucoup plus grande que la nôtre, alors l'effet de la mémoire cache en sera réduit. De manière générale cependant, le speedup n'est pas très bon. Le cas FCI est meilleur que celui avec MPI. La raison est due principalement au éléments suivants:

- la version du compilateur C++ n'est pas encore assez optimisée comme l'est Fortran77. Cela se comprend quand on sait que ce langage n'est que peu utilisé dans

la recherche scientifique. Un effort de la part de fabricants de compilateur est donc demandé.

- l'implémentation du parallélisme n'a pas encore été optimisée au niveau de la programmation. Il faut entre autre faire attention à ce que la répartition des tâches distribuées aux processeurs soient la plus homogène possible.

Malgré des performances qui ne sont pas encore maximales en terme de parallélisme, SPECULOOS reste un solveur des équations de Navier-Stokes très prometteur. Il est d'ailleurs plus que cela, c'est un outil très général de résolution de systèmes d'équations aux dérivées partielles qui peut être utilisé dans d'autres domaines de la science ou de l'économie. Sa souplesse d'utilisation en fait un atout incontestable pour des applications futures. En dépit de ces quelques critiques qui ne concerne pas le hardware du T0-Dual proprement dit, on peut sans nul doute espérer de très bonnes performances de la part de cet ordinateur et des prochaines versions qui sortiront bientôt.

RÉFÉRENCES

- [1] Gruber R., Dubois-Pèlerin Y., *Swiss-Tx: First experience on the T0 system*, Supercomputing Review, EPFL #10, pp. 19-23, 1998.
- [2] Dubois-Pèlerin Y., Van Kemenade V., Deville M., *An object-oriented toolbox for spectral element analysis*, J. Scient. Computing, Vol. 14, pp. 1-29, 1999.
- [3] Maday Y. & Patera A. T., *Spectral Element Methods for the Incompressible Navier-Stokes Equations*, In *State-of-The-Art Surveys on Computational Mechanics*, A.K. Noor and J. T. Oden (eds.), The American Society of Mechanical Engineers, pp. 71-143, 1989.
- [4] Rønquist E., M., *Spectral Element Methods for the Unsteady Navier-Stokes Equations*, von Karman Institute for Fluid Dynamics, Lecture Series, 1991.
- [5] <http://sewww.epfl.ch/SIC/SE/servcentraux/generalites.html>. ■

COMPUTER SIMULATION OF ATRIAL ARRHYTHMIAS

OLIVIER BLANC & JEAN-MARC VESIN, EPFL-DE, SIGNAL PROCESSING LABORATORY,
NATHALIE VIRAG, MEDTRONIC EUROPE S. A., OLIVIER EGGER, OASYA S. A. AND JACQUES KOERFER
& LUKAS KAPPENBERGER, CHUV-LAUSANNE, DIVISION OF CARDIOLOGY

La fibrillation auriculaire constitue la forme la plus fréquente d'arythmie, et provoque inconfort, malaises cardiaques et embolies artérielles. Du fait de l'absence de modèles biologiques stables de la fibrillation auriculaire, la thérapie est basée sur des observations empiriques. En contraste avec le myocarde ventriculaire, la surface et l'épaisseur limitées de l'oreillette permet le développement d'un modèle 3D du tissu auriculaire sur les ordinateurs actuels dans le but de développer de nouvelles stratégies thérapeutiques. Sur la base d'un modèle informatique de tissu cardiaque hétérogène et anisotrope qui a été prouvé être réaliste lors d'expérimentations antérieures, nous avons développé un modèle 3D anatomique de l'oreillette humaine. L'anatomie a été reconstruite virtuellement, et est composée d'environ 250 000 cellules cardiaques simulées. Des observations électro-physiologiques faites sur des êtres humains ont révélé l'existence de motifs anormaux d'activité électrique tissulaire ressemblant à des spirales et auto-organisants lors d'arythmies auriculaires. Nous avons été capables à l'aide de notre modèle de simuler ces phénomènes dans des conditions physiologiquement réalistes.

Atrial fibrillation is the most frequent form of arrhythmia, provoking discomfort, heart failure and arterial embolisms. As reliable biologic models do not

exist, therapy is based on empirism. In contrast to the ventricular myocardium, the limited surface and wall-thickness of atria should permit the development of a 3D model with up-to-date computer power in order to develop new therapeutic ideas. Based on a 2D heterogeneous and anisotropic computer model of cardiac tissue that has proved to be realistic in previous experiments, we have developed a 3D anatomical model of human atrium. The anatomy has been virtually reconstructed and is composed of about 250'000 cardiac cells. Electro-physiological observations made in humans have revealed the existence of anomalous, self-organizing, spiral-like patterns of tissue electrical activity during atrial arrhythmias. Using our model, we have been able to simulate these phenomena under physiologically realistic conditions.

INTRODUCTION

Research and development in any field of science is nowadays almost always supported by computer power. Several mathematical models of the heart based on an accurate representation of anatomy and cellular biophysics are currently under development [1,2], one of the main goals being to better understand the complex spatio-temporal mechanisms leading to cardiac arrhythmias. We present here our model of human atria, which is to our knowledge

the first one incorporating realistic anatomical features. Its ability to reproduce phenomena observed in clinical experiments is illustrated.

DESCRIPTION OF THE PROBLEM

Humans, like all other mammals, have a four-chambered heart, consisting of the left and right atria and the left and right ventricles. The two atria can be envisioned as the receiving chambers, or «priming» pumps, of the heart. The two ventricles represent the «power» pumps of the circulatory system. The right atrium receives the blood returning from the body through the veins, and the left atrium receives the freshly oxygenated blood from the lungs. Blood circulation depends on an appropriate contraction of the atria and the ventricles. This contraction is provoked by the propagation of a so-called action potential from cardiac cell to cardiac cell. This action potential relies on changes in the concentration of ions (calcium, potassium, and sodium) inside these cells. The excitation spreads rapidly through the tissue of an atrium or a ventricle, ensuring that all cells contract together. In the normal functioning of the heart, this excitation finds its source in a special set of pacemaker cells called the sinoatrial (SA) node. Atria are excited, and thus contract, first. The contraction of the ventricles is delayed because the excitation can reach them via an electrical connection formed by a group of special cardiac cells called the atrioventricular node.

What may (unfortunately) happen is that, either because of defects in the tissue (anatomical reentry) or of ectopic foci (functional reentry), spiral-like wavefronts of excitation appear in the cardiac tissue (atrium or ventricle) and start to wander through it, creating a continuous, self-organizing electrical activity. Defects in the tissue may be due for instance to prior infarction. Ectopic foci are groups of cardiac cells that become spontaneously excited. This type of phenomenon goes under the general denomination of arrhythmia, since the normal rhythm disappears, and its most extrem form (corresponding to a large number of randomly propagating wavefronts) is called fibrillation. As progressive, rhythmic excitation of the cardiac tissue is replaced by an anarchic excitation, the atrium or ventricle loses its efficiency as a pump. Atrial arrhythmias, the most frequent form of arrhythmias, are not lethal by themselves. However, they often cause disabling symptoms and severe complications such as heart failure and stroke [3]. Ventricular fibrillation leads to sudden cardiac death, and the only known therapy is defibrillation, which consists in applying an electric shock to the heart in order to «reset» it.

Arrhythmias are phenomena difficult to study *in vivo*. Observation on human patients is of course rare, and experimentation on animals is quite challenging: First, large animals (sheep, goats, pigs,...) have to be used for reasons of heart size similarity, and mice hearts are even too small to fibrillate. Second, it is impossible to impose the conditions leading to fibrillation with precision. Third, the most common data acquisition procedure is epicardial

mapping [4], i. e. the simultaneous recording of potentials at numerous locations of the cardiac tissue. Besides practical difficulties, this coarse spatial discretization yields only limited information upon the space-time evolution of the arrhythmia.

Since the most frightening fatal arrhythmias leading to sudden death are of ventricular origin and induced by ischemic events, most efforts of research in electrophysiology are made to understand ventricular tachycardias and ventricular fibrillation. Computer models of the heart have been used to simulate these arrhythmias but the major drawback is the heavy computational load needed for realistic models. Indeed, the development of a three-dimensional model of the whole heart is today still difficult due to the computation time. The most effective computer models of the heart based on detailed ionic models have about 1'000'000 cardiac cells [5,6], leading therefore to a limitation in the number of cardiac cells and size of tissue that can be simulated. While in the ventricles reentry is a three-dimensional process, even within an isolated wall segment, atria are constituted of thin walls and the arrhythmic process can be reduced to a two-dimensional phenomenon. Based on these premises the electrophysiology of the atrium can be considered as a 2D problem in a 3D structure with a total area of about 100 cm². This is a size that we can model with current computer power.

TWO-DIMENSIONAL MODEL OF THE CARDIAC TISSUE

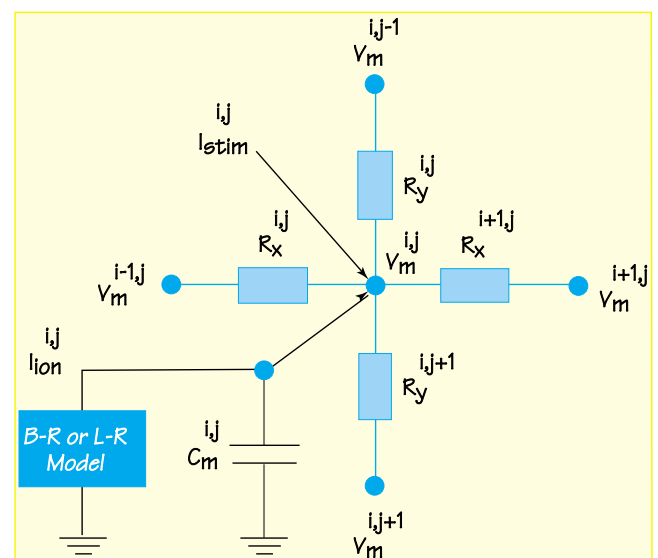


Fig. 1 – Representation of a cardiac cell with its four neighbors

A 2D heterogeneous and anisotropic model of cardiac tissue has been developed, where the modeling of each cell is based on membrane ionic channels. The cardiac tissue consists of a grid of individually calculated cells interconnected via resistors representing gap junctions. Membrane

ion kinetics is computed with either the Beeler-Reuter (B-R) [7] or the Luo-Rudy (L-R) [8] models derived from experimental data. These models have been chosen because they are based on physiologic parameters and are therefore well suited for the study of arrhythmias. To limit the computation time, we have implemented a monodomain model, where the current flow is described only in the intracellular region and where the extracellular region is assumed to be grounded (Fig. 1).

The electrical propagation in the intracellular domain is described by the continuous reaction-diffusion equation:

$$\frac{1}{S_v} \nabla \cdot (D \nabla V_m) = C_m \frac{\partial V_m}{\partial t} + I_{ion} - I_{stim} \quad (1)$$

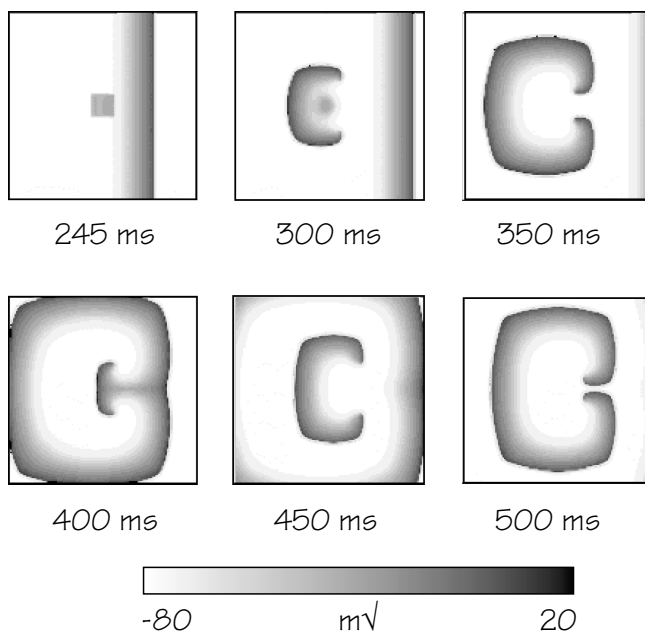


Fig. 2 – Functional reentry in a 120 * 120 cells homogeneous and isotropic 2D tissue

where S_v is the surface to volume ratio, D the conductivity tensor, V_m the transmembrane potential, C_m the membrane capacitance, I_{ion} the sum of the membrane ionic currents and I_{stim} a stimulus current. Equation (1) is discretized by a finite difference method and solved in two steps: firstly an explicit computation of the membrane ionic currents I_{ion} , and secondly a semi-implicit current diffusion solved by a classical ADI scheme. The choice of the time and space discretizations is the result of a tradeoff between computation speed and accuracy. All the simulations are conducted with $C_m = 1 \mu\text{F}/\text{cm}^2$ and $S_v = 0.024 \text{ cm}^{-1}$. The behavior of separate cells and the values of intracellular resistivities in the longitudinal and transversal directions are individually programmable for each cell via the conductivity tensor D , allowing us to introduce heterogeneity (ischemic zones, obstacles to conduction) and anisotropy in the cardiac tissue. Several experiments have been performed to show the realistic behavior of our two-dimensional tissue, which has been described in detail [9]. Fig. 2 shows an

example of functional reentry in a 120*120 cells homogeneous and isotropic tissue. The pattern resulting from the conjunction of normal excitation (vertical front) and an ectopic focus (point on the left) repeats itself in time in a stable way.

THREE-DIMENSIONAL MODEL OF HUMAN ATRIA

Since atria are constituted of thin walls, we have built our anatomic model of the human right and left atrium by folding the described two-dimensional cardiac tissue into a 3D model of one layer of cells. Holes are placed to simulate veins and valves like represented in Fig. 3.

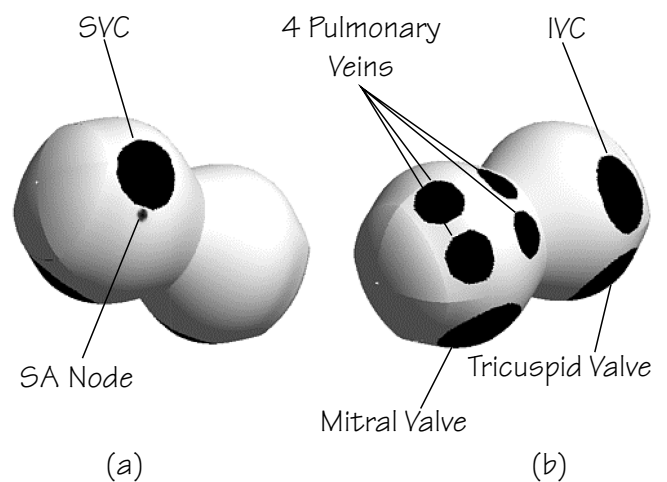


Fig. 3 – Proposed model for both atria with the holes represented in black: (a) top front view (right atrium on the left) with the superior vena cava and the sinoatrial node, (b) bottom rear view (right atrium on the right) with the four pulmonary veins, the inferior vena cava (IVC) and the tricuspid and mitral valves

The surface of the mitral and tricuspid valves is 4 cm², 2 cm² for the superior vena cava, 2.5 cm² for the inferior vena cava and 1 cm² for each pulmonary vein. Electrical activation is initiated from selected regions by injecting an intracellular current (simulated electrical stimulus) into the cells. Furthermore, different action potential modifications have been tested by a modulation of the ionic channels, simulating membrane defects or electrical disturbances. The simulated size of both atria is equivalent to the external surface of a 3cm x 3cm x 7cm parallelepiped-shaped box. The total area is 100 cm² with the orifices of the veins and the valves representing 20 % of this area.

We have used in our model $\Delta x = \Delta y = 200 \mu\text{m}$ and $\Delta t_{min} = 0.025 \text{ ms}$ for a resistivity ranging from $\rho = 80 \Omega \cdot \text{cm}$ to $800 \Omega \cdot \text{cm}$. This leads to a total number of cardiac cells of about 250'000 cells. The model has been implemented with double precision in C++ on a Silicon Graphics Onyx. This supercomputer runs with 8 R10'000 processors having an internal frequency of 195 MHz.

The software generates a color-coded movie of the electrical activation in the tissue, with 5 frames per second, and a simulation time of 5 milliseconds between each frame (we typically simulate 10 seconds of activation). Activation potentials of specific cells can also be recorded, as well as an electrocardiogram (ECG) corresponding to the global activation of the tissue.

EXAMPLE OF EXPERIMENT

Atrial fibrillation could be initiated from a normal sinus rhythm and two or three ectopic beats carefully timed and located. Although many attempts have resulted in unsustainable wavefront perpetuation, two ectopic beats S_2 and S_3 located in the high right atrium at 475 ms and 700 ms respectively have led to a fibrillation lasting for about 2 seconds, which then converted to a so-called sustained atrial flutter (see figure on the cover of this issue).

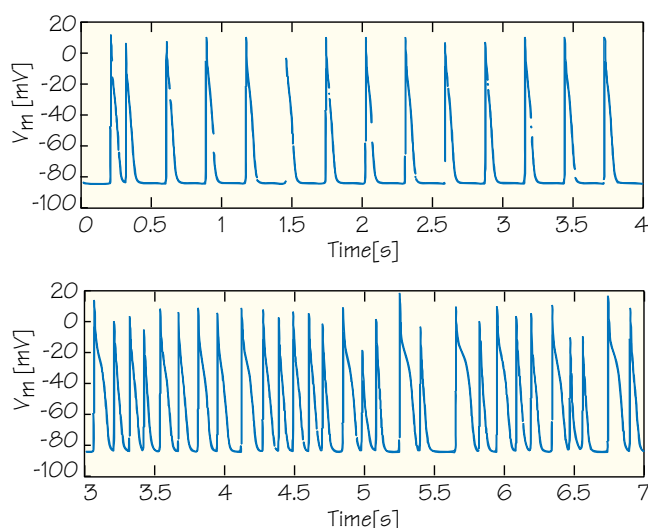


Fig. 4 – Comparison of the evolution of the action potential for a cardiac cell located between the tricuspid valve and the inferior vena cava during atrial flutter and atrial fibrillation: (a) atrial flutter with an average rate of 220 bpm; (b) atrial fibrillation with an average rate of 422 bpm

Atrial flutter is a sustained tachyarrhythmia characterized by a periodic pattern. We can clearly see in this example that atrial fibrillation is observed as multiple reentering wavefronts traveling randomly. It is to be noted that this conversion from atrial fibrillation to flutter has indeed been observed in experimental studies [3]. Fig. 4 (a) displays the temporal evolution of the action potential of one cardiac cell during an atrial flutter. The average rate is 220 beats per minute and the waveform looks distinctly quasi-periodic. Fig. 4 (b) displays the same information in the case of atrial fibrillation. The average rate is 422 beats per minute and the more stochastic aspect of the waveform is obvious.

CONCLUSION

We have developed a computer model of human atria which allows us to simulate phenomena observed in nature such as normal rhythm, atrial flutter, or atrial fibrillation. We can analyse the temporal and spatial evolution of the excitation, and we can also, like in real clinical experiments, record the action potential at a specific location.

As such, at least in what concerns atria, we have full access to a virtual electrophysiology device, on which we can study in full depth the mechanisms leading to arrhythmias. Our next goal is to simulate therapeutic actions such as defibrillation and ablation (creation of a zone of non-conducting tissue blocking the onset of the arrhythmia) in order to define optimal strategies for their use.

ACKNOWLEDGMENTS

This study was made possible by grants from the Theo-Rossi-Di-Montelera Foundation, Medtronic Europe and the Swiss *Commission pour la Technologie et l'Innovation* (CTI).

REFERENCES

- [1] P.J. Hunter, M.P. Nash and G.B. Sands, *Computational electromechanics of the heart*, in *Computational Biology of the Heart*, pp. 345-407. Edited by A.V. Panfilov and A.V. Holden, John Wiley & Sons, 1996.
- [2] Y. Rudy, *Insights from theoretical simulations in a fixed pathway*, *J. Cardiovasc. Electrophysiology*, Vol. 6, pp. 294-312, 1995.
- [3] P. Wolf, E. Benjamin, A. Belanger, W. Kannel, D. Levy, R. D'Agostino, *Secular trends in the prevalence of atrial fibrillation: The Framingham study*, *American Heart Journal*, Vol. 131, No. 4, pp. 790-796, 1996.
- [4] K. Konings, C. Kirchhof, J. Smeets, H. Wellens, O. Penn, and M. Allesie, *High-density mapping of electrically induced atrial fibrillation in humans*, *Circ.*, vol. 89, pp. 1665-1680, 1994.
- [5] W. Quan, S.J. Evans and H.M. Hastings, *Efficient Integration of a realistic two-dimensional cardiac tissue model by domain decomposition*, *IEEE Trans. on Biomed. Eng.*, Vol. 45, No. 3, pp. 372-385, March 1998.
- [6] D. Noble and R.L. Winslow, *Reconstruction of the heart: network models of SA node-atrial interaction*, in *Computational Biology of the Heart*, pp. 49-64. Edited by A.V. Panfilov and A.V. Holden, John Wiley & Sons, 1997.
- [7] G. W. Beeler and H. Reuter, *Reconstruction of the action potential of ventricular myocardial fibers*, *J. Physiol.*, Vol. 268, pp. 177-210, 1977.
- [8] D. Luo and Y. Rudy, *A model of the ventricular cardiac action potential*, *Circ. Res.*, Vol. 68, No. 8, pp. 1501-1526, 1991.
- [9] N. Virag, J.-M. Vesin, L. Kappenberger, *A Computer Model of Cardiac Electrical Activity for the Simulation of Arrhythmias*, *PACE*, Vol. 21, No. 11, Pt. II, pp. 2366-2371, Nov. 1998. ■

IP3T A PERFORMANCE PREDICTION TOOL FOR IRREGULAR PARALLEL PROGRAMS

MICHEL PAHUD, EPFL, DÉPARTEMENT D'INFORMATIQUE

Dans ce travail nous présentons tout d'abord notre modèle de prédiction de performance pour applications parallèles irrégulières (ou régulières). Ensuite, nous présentons notre outil de prédiction de performance, baptisé IP3T (Irregular Parallel Performance Prediction Tool), basé sur notre modèle. L'originalité de notre modèle et de l'outil est de permettre de prédire les performances d'applications réelles et industrielles. Pour réaliser cela, le modèle a été conçu pour être facilement extensible. L'outil IP3T a été testé avec succès pour des applications complexes sur machines parallèles et sur réseaux de stations et a prouvé qu'il est précis et fiable. Enfin, nous présentons quelques résultats obtenus avec cet outil.

In this work we first present our performance prediction model for irregular (or regular) parallel applications. Secondly, we present a performance prediction tool, named IP3T (Irregular Parallel Performance Prediction Tool), based on our model. The distinctive feature of our model and our tool is that it provides for predicting the performance of industry-oriented complex applications. To do that, the model was design to be easy to extend. The IP3T tool has been successfully tested with several complex applications on parallel machines and on networks of workstations and has proven to be accurate and reliable. Finally, we present some results obtained with this tool.

INTRODUCTION

During the last decade, performance prediction has been repeatedly quoted as a key factor to developing parallel systems [1]. Predicting the behaviour of a program performance as a function of the number of processors and of the problem size is essential to users:

- in order to choose the right implementation method.
- to manage execution of processes in supercomputer systems or in nondedicated networks of workstations/PC.
- for optimizations in parallelizing compilers.

Numerous prediction tools have been recently proposed in the literature [2,3]. Most of this work focuses mainly on applications with regular and static data structures. In contrast, the present work focusses on irregular applications

[4] and especially those exhibiting dynamic data structures and a data-dependent execution scheme. For this type of application, performance does not depend only on the number of processors and on the problem size. Other parameters have been taken into account, the value of which are data-dependent, and therefore unpredictable.

In this work, we present a performance prediction model for irregular (or regular) parallel applications on parallel machines and on nondedicated networks of workstations/PCs. We have developed a performance prediction tool to show that our model is intended for automatic use.

MODELING IRREGULAR APPLICATIONS

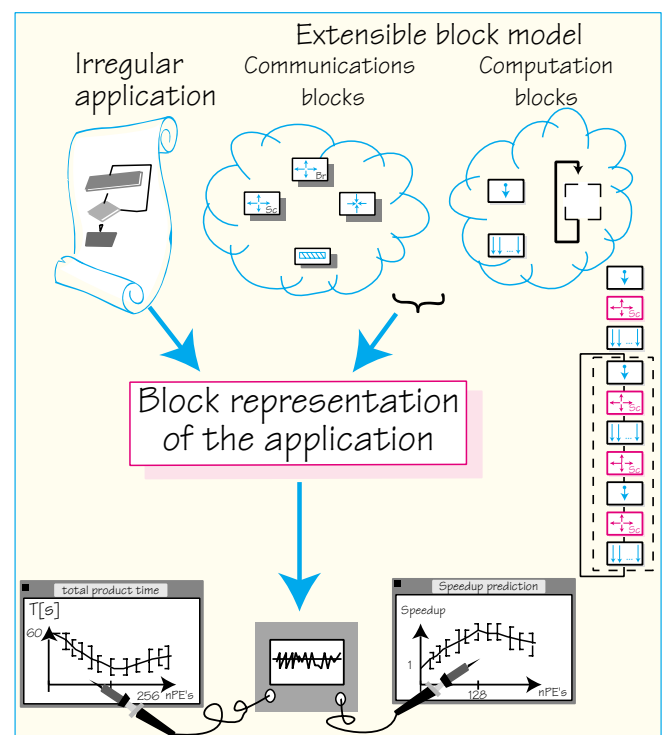


Fig. 1 – Our approach for performance modeling

In our approach, we began by assuming that the total execution time of an application is the sum of different times spent in computation, communication, synchronization, etc. In order to predict the behaviour of irregularly (or regularly) structured algorithms, it is necessary to find well defined templates or frameworks that can capture their behaviour. We have assumed that the algorithms to be

modeled can be decomposed into a sequence of blocks, where each block consists of either local computation (performed in parallel by the processors), or of a global communication operation, or a synchronization barrier. Once the application is decomposed into blocks, a tool can automatically predict its runtime performance (Fig. 1).

In our model, there are two main categories of blocks: computation blocks and communication blocks. The main types of blocks defined in our work and their respective symbols are given in table 1.

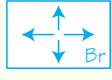
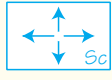
Computation blocks:		Communication blocks:	
Type of block	Symbol	Type of block	Symbol
Sequential block		Point-to-point	
Parallel block		Broadcast	
Iterative block		Scatter	
		Gather	
		Barrier	

Table 1 – Graphical symbols of the main blocks

For each of the proposed block types, a simple performance model is established as a function of basic parameters.

- Sequential block: A simple way of estimating single-processor execution time is to measure it on one of the processors of the target parallel machine.
- Parallel block: A global synchronization is supposed to occur at the end of the block. The time of the block is the time spent in the slower concurrent process of the parallel block.
- Iteration block: For N successive execution of code, time for loop is modeled as N times the duration of the loop body.
- Communication block: A set of benchmarks is established, to time these communication operations beforehand on the target multi-processor architectures or on networks of workstations/PCs, for different numbers of processors and different message sizes. These measurements have to be done only once for a given architecture and stored into a database.

Execution time of the whole program can then be modeled by the sum of the execution times for the consecutive blocks. In order to obtain accurate results, we have also taken in account statistical models [7]. To predict performance of irregular applications we add the possibility to model blocks where the length varies during the execution [4]. For example, some parts of the code of real applications

contain a block or a sequence of blocks where the execution time depends on environment parameters like iteration number of a surrounding loop. In order to model this kind of situation we have defined a concept of duration function. A duration function returns a duration (in seconds) for a given input value. The value given as parameter to the function is an environment parameter like the current iteration value of a loop. A duration function can be decomposed into two part: an architecture dependant part (named **load unit** or **U**) and an application dependant part (named **load function**). Fig. 2 shows how a duration function is composed.

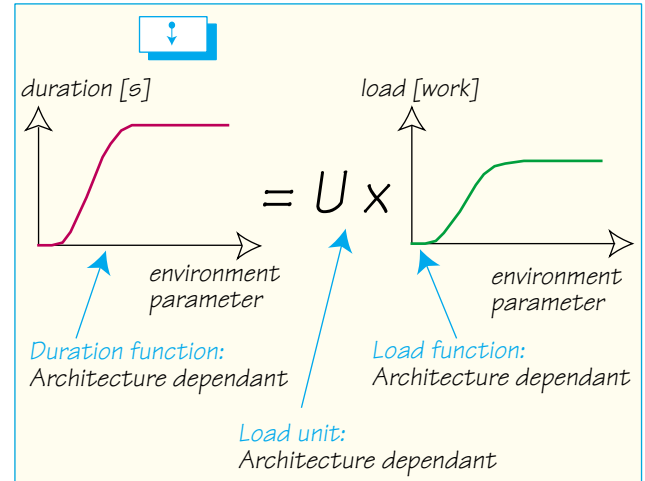


Fig. 2 – A duration function permits to model the irregularity of some applications

RESULTS

In order to verify the usability of the model in practice we carried out prediction experiments on custom-made, irregular, parallel programs, then on real programs. Among the programs used for these performance validation experiments are:

- application simulation of wave propagation in urban areas named ParFlow++ (parallel C++ application used for the STORMS European project: EPFL, University of Geneva, Swisscom, ...) [5].
- optimal networking decomposition algorithm in domains (parallel C application presented in NASA National Symposium) [6].
- a genetic algorithm based on the concept of individual islands (parallel C++ application used for the LEOPARD project: TDF, EPFL, ...) [7].

The good results of the validation experiments confirm the validity of the method [8]. For example, we show results with the ParFlow++ application where the general block decomposition is given in Fig. 3. This application contains a main loop which contains a parallel block. Each process is a loop which contains a sequence of a sequential block, a communication block and another sequential block.



IP3T, Irregular Parallel Performance Prediction Tool

The structure of the tool is in 3 layers (Fig.5):

- 1 a lower layer where the access to the target architecture is done for single-processor code measurements and database access is done for communication blocks.
- 2 a middle layer which works in 3 phases:
 - Analysis phase: in which the structure of the applications is recognized. Some directives are added into the source code in order to help the recognition of the block structure [8].
 - Measurement phase: in which the performances of sequential blocks are measured on one of the processors of the target parallel machine.
 - Prediction phase: in which the tool can compute the prediction with the model using the performance measurements of the sequential blocks and the database of the performances of communication blocks.
- 3 a user interface which allow the user to interact and modify some parameters of the prediction.

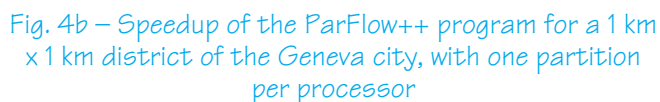
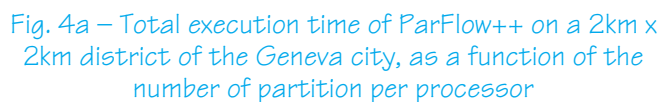


Fig. 4 – Some results of prediction and measurement with an application of wave diffusion in urban areas (Cray T3D)

In the first phase, IP3T can automatically recognize the block structure of applications (with the help of some special block directives) and display it in a window (Fig. 6). This window permits also directly to add or delete new blocks and/or change some parameters of blocks. Moreover, in this window, the user can select a subpart of the program to be analyzed.

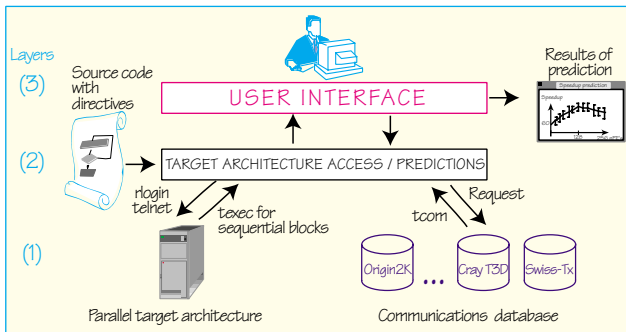


Fig. 5 – Structure of IP3T

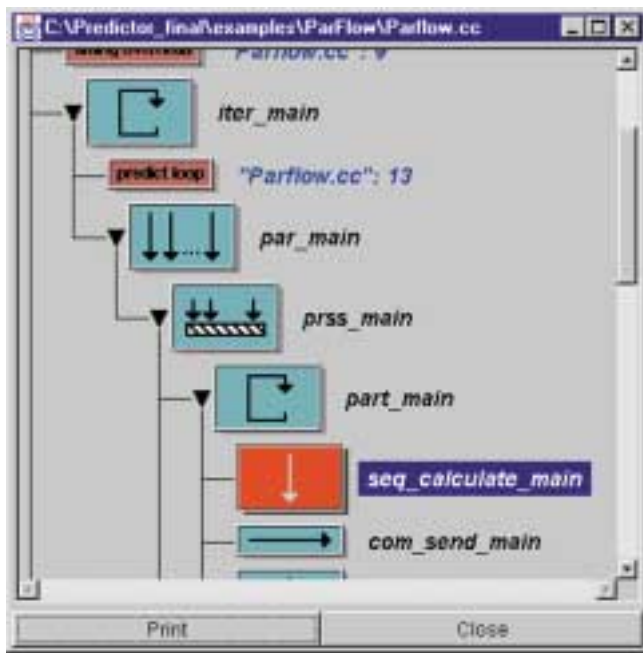


Fig. 6 – Window with block decomposition of the analysed application

In order to predict the performance of applications with many irregularities, the user of the tool can define some personal global block parameters and can interact with them for performance prediction through a dialog box generated automatically from these parameters. For example, we obtain the dialog box of Fig. 7 with defined global parameters of the ParFlow++ application.

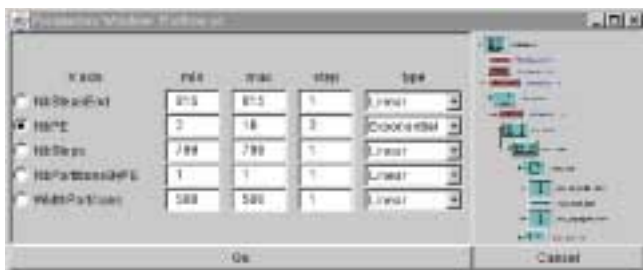
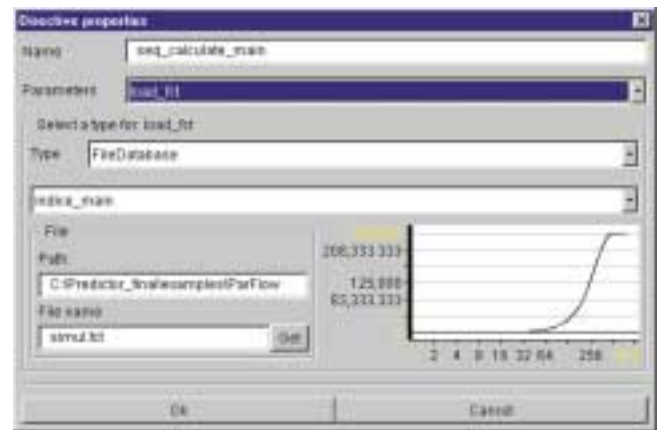


Fig. 7 – Dialog box with user defined global parameters with the ParFlow++ application

With IP3T, parameters depend on the type of a block [4]. For example, a loop block has an indice name, a step and boundaries. Every parameter of this block can be a constant or a variable value. A sequential block has a **load unit U** and a **load function**. There are also a divider parameter named **div load**. Fig. 8 presents the dialog box for a sequential block of the analysed application with a **load function**. The Fig. permits to feel how user-friendly it is to introduce a **load function** into a block.

Fig. 8 – Parameter window with the load function of one block of the analysed application. The x-axis of the "load function" is directly linked with the indice named *Index_main* of a surrounding loop

For the same block, Fig. 9 shows the **load unit** parameter which can be an integer value, a floating point value or a question mark. A question mark forces the tool to measure the execution time of that block on the target architecture. This is automatically done by the tool using some ftp and telnet primitives.



Fig. 9 – The Load unit U parameter configuration

There are also a **div load** parameter in order to model how the duration of a block changes when the number of processors vary (Fig. 10). The type of this parameter can be a **Global parameter**, an integer value, a floating point value or a function given by a file (FileDatabase).

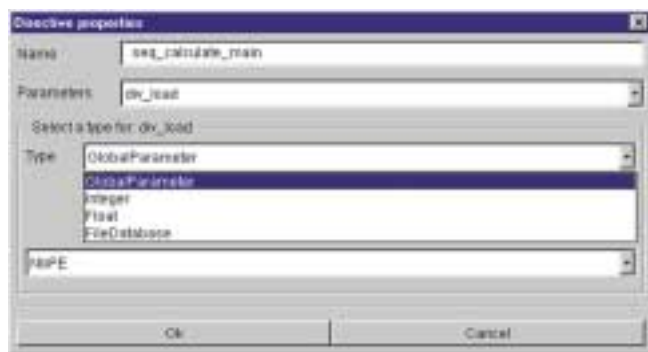


Fig. 10 – The Div load parameter configuration

IP3T has been successfully tested on the real applications already mentioned in the results section.

CONCLUSIONS AND PERSPECTIVES

In this project, we have successfully achieved:

- a performance prediction model for irregular (or regular) real parallel (or sequential) applications;
- a validation through experiments with industry-oriented applications in order to confirm the validity of the method;
- a complete prediction tool to show that our model is intended for automatic use on parallel machines or on nondedicated networks of workstations/PCs.

Future work is directed towards an integration of IP3T in a development environment useful in the Swiss-Tx project [9].

ACKNOWLEDGEMENTS

This work was a part of a Ph.D. funded by the Fonds National Suisse de la Recherche Scientifique. This Ph.D. work was advised by Prof. Jean-Daniel Nicoud, LAMI (Laboratoire de MicroInformatique), Prof. Giovanni Coray, LITH (Laboratoire d'Informatique THéorique) and Dr. Thierry Cornu, Simulog (Paris, France). Thanks especially to Dr. Pierre Kuonen (CAPA EPFL), Dr. Thierry Cornu, Dr. Frédéric Guidec and Dr. Nouredine Bouhmala for all

very nice help and precious suggestions during this work. Thanks also senior students David, Samir, Gilbert, Serge, Christian, Thomas and Boris for the participation to experiments and tool development.

REFERENCES

- [1] Kai Hwang. *Advanced Computer Architecture Parallelism, Scalability, Programmability*. Mc-Graw-Hill, 1993.
- [2] Mustafa Uysal, Tahsin M. Kurc, Alan Sussman, and Joel Saltz. *A performance prediction framework for data intensive applications on large scale parallel machines*. Technical Report CS-TR-3918, University of Maryland, College Park, July 1998.
- [3] Yu-Kwong Kwok, Ishfaq Ahmad, Min-You Wu, and Wei Shu. *Graphical tool for automatic parallelization and scheduling of programs on multiprocessors*. In Proceedings of Euro-Par'97, pages 294-301, August 1997.
- [4] M. Pahud and T. Cornu. *Predicting the behaviour of irregular parallel applications using code block decomposition*. In Proceedings of the SIPAR Workshop'96 on Parallel and Distributed Systems, Geneva, Oct. 1996.
- [5] M. Pahud, F. Guidec and T. Cornu. *Performance evaluation of a radio wave propagation parallel simulator*. In Proceedings of the Third International Conference on Massively Parallel Computing System, MPCS'98, Colorado Springs, USA, April 1998.
- [6] N. Bouhmala and M. Pahud. *A parallel variant of simulated annealing for optimizing mesh partitions on workstations*. In 4th NASA National Symposium, Williamsburg, October 1997.
- [7] P. Calégari, F. Guidec, P. Kuonen, and D. Kobler. *Parallel Island-Based Genetic Algorithm for Radio Network Design*. Journal of Parallel and Distributed Computing (JPDC): special issue on Parallel Evolutionary Computing, Academic Press, 47(1): 86-90, November 1997.
- [8] M. Pahud. *Une méthode de prédiction de performance pour les programmes parallèles irréguliers*. PhD thesis, École Polytechnique Fédérale de Lausanne, Switzerland, 1998. Number 1911.
- [9] R. Gruber, Y. Dubois-Pèlerin, EPFL-DGM, and Swiss-Tx Group. *Swiss-Tx: First Experiences on the T0 System*. EPFL Supercomputing Review, 19-23, November 1998. ■

RÉDACTEUR EN CHEF
MISE EN PAGE ET GRAPHISME
ADRESSE

TÉLÉPHONE
TÉLÉCOPIE
ADRESSE ÉLECTRONIQUE
ADRESSE WEB

Nouredine El Mansouri, SIC-EPFL
Appoline Raposo de Barbosa, SIC-EPFL
Service informatique central EPFL
MA-Ecublens Case Postale 121
CH - 1015 Lausanne
(021) 693 22 11
(021) 693 22 20
Nouredine.ElMansouri@epfl.ch
<http://sawwww.epfl.ch/SIC/SA/publications>

EDITOR
TEXT PROCESSING AND LAYOUT
ADDRESS

PHONE
FAX
E-MAIL
WEB LOCATION