Fine grain parallelism on a MIMD machine using FPGAs

Frédéric Raimbault, Dominique Lavenier, Stéphane Rubini, Bernard Pottier

INRIA

INSTITUT NATIONAL DE RECHERCHE EN INFORMATIQUE ET EN AUTOMATIQUE

Fine grain parallelism on a MIMD machine using FPGAs

Frédéric Raimbault, Dominique Lavenier ,

Stéphane Rubini, Bernard Pottier

N° 1983

Mai 1993

PROGRAMME 1

Architectures parallèles,
bases de données,
réseaux et systèmes distribués

INRIA
RENNES

Fine grain parallelism on a MIMD machine using FPGAs

Frédéric Raimbault, Dominique Lavenier *,
Stéphane Rubini, Bernard Pottier ** ***

Programme 1 — Architectures parallèles, bases de données, réseaux et systèmes distribués
Projet API

Rapport de recherche n ° 1983 — Mai 1993 — 9 pages

Abstract: Current MIMD machines are used for coarse grain-parallelism and also offer messsage passing mechanisms to deal with inter-processor communications. But these mechanisms lack efficiency in fine-grain parallel applications such as systolic computation. This article presents the use of an FPGA chip to set up a fast systolic communication agent on a linear asynchronous network of TRANSPUTER processors; the machine is called ARMEN.

Key-words: fine grain parallelism, communication, systolic, FPGA, TRANSPUTER, ARMEN

(Résumé : tsvp)

A paraître dans *IEEE Workshop on* FPGA*s for Custom Computing Machines (FCCM'93)*

*{raimbaul,quinton,lavenier}@irisa.fr
**Département d'Informatique, Université de Bretagne Occidentale, BP 452, 29275 Brest Cedex, France, {rubini,pottier}@ubolib.cicb.fr
***This work is supported by Region Bretagne, PRC/GDR ANM and C^3

Unité de recherche INRIA Rennes
IRISA, Campus universitaire de Beaulieu, 35042 RENNES Cedex (France)
Téléphone : (33) 99 84 71 00 – Télécopie : (33) 99 38 38 32

Exploitation du parallélisme à grain fin sur une machine MIMD équipée de FPGAs

Résumé : Les machines MIMD actuelles sont utilisées pour le parallélisme à grain moyen. A cet effet, elles offrent des mécanismes à passage de messages pour les communications entre processeurs. Mais ces mécanismes sont peu efficaces dans le cas du parallélisme à grain fin tel que le calcul systolique. Cet article présente une utilisation des circuits à logique reprogrammable (les FPGA) pour la mise en place d'un noyau de communication systolique sur un réseau linéaire de Transputer; cette machine s'appelle ArMen.

Mots-clé : parallélisme à grain fin, communication, systolique, FPGA, Transputer, ArMen

1 Introduction

Custom machines and parallel architectures are frequently designed with TRANSPUTER [5] processors. The TRANSPUTER has integrated communication links which make it suitable for building parallel systems. Links provide inter-processor communication facilities and extensibility at low cost. Nevertheless due to the low communication / computation ratio, the TRANSPUTER is efficient only for coarse grain parallelism. To remedy this, short messages need to be gathered and transfered together, taking into account memory storage and input/output latency. As we are concerned with systolic algorithms we propose a way to efficiently simulate our systolic programs on a parallel architecture that supports word-by-word communications. The reconfigurable gates associated with the TRANSPUTER network of the ARMEN machine [10] offer a solution to this problem and allow fine grain parallelism to be exploited. This article presents the way we use FPGA chips to set up a fast systolic communication agent on a linear asynchronous network of TRANSPUTER processors.

Our work relies on the systolic programming environment RELACS, a close cousin to the C programming language. RELACS provides synchronous communication operators to simplify the programming of data transferts that occur in systolic algorithms. The RELACS compiler generates C programs that perform the computation process and the data managment process of a systolic network.

Historically, the developement occured in two stages. The first stage consisted of writing a library of communication functions using only the TRANSPUTER links. The C programs produced by the RELACS compiler used the messages passing services provided by the TROLLIUS [11] kernel running on each node of the ARMEN machine. In the second stage, parallel and synchronous communication operators were programmed into the reconfigurable gate array attached to each TRANSPUTER. These local hardware operators map directly to the systolic operators of the RELACS language. We also observed a great improvement in the transfer times since each word moved directly from one TRANSPUTER bus to one of its neighbours.

This article is organised as follows: the first section describes the ARMEN machine and its applications. The second section presents the design of a systolic operator on the configurable logic layer of the ARMEN machine. Section three deals with the programming environnement and describes the compiling of the RELACS language on ARMEN. Lastly, experimental results and future work are presented.

2 The ARMEN machine

ARMEN is a parallel machine, built at the Laboratoire d'Informatique de Brest [10]. The motivation for this machine was the desire to combine a reprogrammable logic layer with a conventional MIMD machine. This section describes briefly the ARMEN architecture and its execution modes.

2.1 The architecture

ARMEN is a realization of an architectural model where a processor node from an MIMD machine is tightly coupled to a programmable logic resource embedded in a shared configurable logic layer. The MIMD machine can be implemented either with distributed or shared memory.

An ARMEN node has a TRANSPUTER processor, a local memory and a XILINX 3090. This FPGA is a 16×20 logic cell array with four 36-bits ports. The north port is connected to the address and data multiplexed Transputer system bus. West and east ports are used to communicate with adjacent FPGAs in the configurable logic layer. The south port can be used to access various external resources.

The node processor reads or writes memory-mapped registers in order to control the FPGA behavior. These registers give access to FPGA functions such as initialization and status, writing a configuration or reading back cell contents. After a 100 ms configuration delay, the FPGA can act as a local coprocessor for its processor, or as a part of a global coprocessor for the whole MIMD machine. Its north port is addressable from the TRANSPUTER bus allowing the selection of different internal functions.

Figure 1 shows the current configuration of the ARMEN machine. ARMEN actually consists of a VME board connected to a SUN workstation. The upper ring has standard TRANSPUTER links and system signals

connected to a crossbar on a host workstation interface board. The lower ring allows FPGA west and east ports to be assembled into a 36-bit path using ribbon cables.

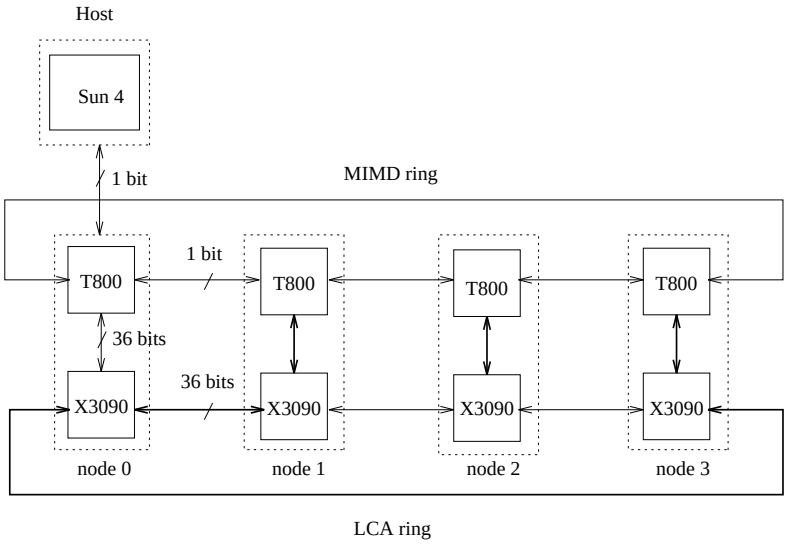


Figure 1: The ArMen machine

The Transputer network is controlled by the Trollius distributed operating system [11]. Transputers are programmed using the C language with libraries for the Trollius system and local FPGA control functions. In this environment, loading a FPGA with a configuration file is specified by a function call with a host file name as a parameter.

Configuration files are produced using Xilinx technology tools. A specific library provides easy and secure interfaces to the ArMen node with bi-directional blocking or buffered data exchanges and interrupt management capabilities.

2.2 Examples of the configurable-logic-layer use

Two circuit models illustrate the machine properties from the control and the computation points of view: (i) *global processors* and (ii) *shared operators*.

- *Global processors* are built using several transversal operator pipelines and centralized (or distributed) finite state machines. They can observe local states on node interfaces, compute global conditions and then write back these results to the interface. Alternatively interrupt waves can be used to signal conditions on the network. The global processors are built up as follow:
  - A pipeline has one stage in each node of the configurable logic layer. Each stage receives data from an adjacent previous stage, and possibly from local registers. It provides data to the next stage, and possibly to the node interface registers. The last stage of a pipeline can be read by the processor at node 0. An interface register can retain data information, or a control bit which generates an interrupt when it is set. A pipeline stage can be a processor, a logic operator or a simple register.
  - A control automaton is usually implemented on a special node (node 0) holding the first stage of each pipeline. The automaton can read results from the last stage and writes initial data to first stage.

- *Shared operators* provide micro-grain parallelism usable on large data arrays following the cellular automata model. Each FPGA is configured to work as the local coprocessor of its associated TRANSPUTER. It has a FIFO to receive the neighborhood information for a linear array of cells to be computed. TRANSPUTERs push data inside the FIFO, and then read back a transformed line. FIFOs from two adjacent FPGAs can exchange cell values and sequence information. Signals are used to implement write barrier local protocols in order to enforce processor synchronization.

  Many such coprocessors have been compiled using the CCEL language [3]. Implemented examples include low level image processing operators and lattice gas simulation.

3 Systolic communications

Having described the target architecture in the previous section, we examine here the principles of systolic communication and we explore two ways of applying them on ARMEN.

3.1 Basic concepts

The concept of systolic architectures was introduced by H.T. Kung in 1982 [8]. This concept aims at solving the problem of memory contention in sequential processors, commonly called the von Neumann bootleneck [1]. Systolic architectures feature the capability of exploiting a regular data flow through a network of identical cells with local memory. This characteristic is supported by an inter-processor communication mechanism which avoids the overflow of the local memory [7].

There exists one commercially available programmable machine based on this concept, the iWARP [2]. However this machine is much more expensive than a TRANSPUTER machine and its processor is not available as a component chip.

One could expect to use the TRANSPUTER links and the services of a distributed system for systolic communications. And this is the straightforward approach with which we experimented on ARMEN. While only neighbour to neighbour communications is needed, we choose the *data-link* layer service of the TROLLIUS software structure which avoids routing overhead. Figure 2 highlights the layered software and hardware structure that are concerned with neighbour to neighbour communications. Each processor of the MIMD network is loaded with an identical program and performs local calculations interleaved with neighbour to neighbour communications.

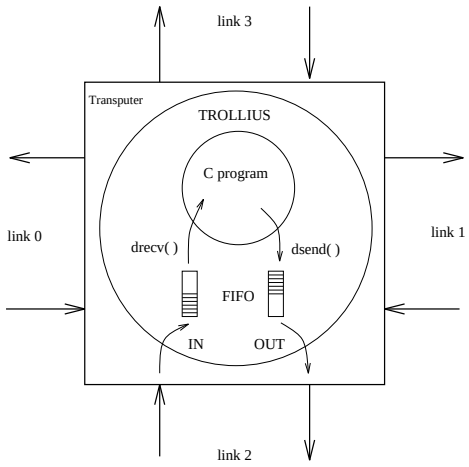


Figure 2: use of TRANSPUTER link

However the low granularity of communication of the resulting programs leads to inefficient executions. Our experiments indicate that it takes about 1.5 ms to exchange one 32-bit word between neighbouring nodes compared to the theoretical 3 μs given by INMOS. The overhead comes from the call to TROLLIUS system and the gathering of messages in memory.

The effective use of the systolic computational model requires register to register communications between neighbouring nodes. The second approach for programming systolic algorithms on ARMEN is to make use of its configurable logic layer and to implement a synchronous communication operator on each FPGA.

3.2 Systolic communication operators design

Implementing systolic operators in reconfigurable logic requires on one hand designing synchronous communication operators and programming them in the FPGA chips of the ARMEN architecture, and on the other hand the definition of a synchronization protocol between neighbouring TRANSPUTER processors through their associated FPGA chips.

Figure 3 shows the interconnections of the components of the ARMEN machine. Each FPGA contains one 16 bit-wide data register connected to the bus of its associated TRANSPUTER. Two parallel 16 bit-wide communication links allow fast data transfers between adjacent data registers. A multiplexer is provided with each FPGA to select the input data.

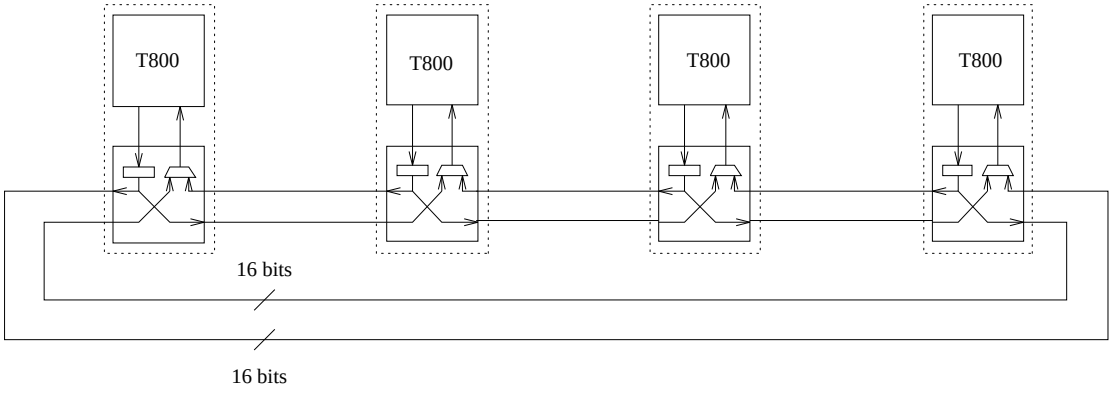


Figure 3: inter-FPGA connexions

The synchronization between neighbouring FPGA chips rely on two status bits (P_i^{left} and P_i^{right}) associated with the data register of each node i. This register holds the data to be sent during write operations to the left (WrL) or to the right (WrR). When the status bit P_i^{left} is set, the data register is full and the left neighbour is allowed to read it. While P_i^{left} is not set, the left neighbour cannot read it. The same rules apply to the P_i^{right} bit.

Write operation protocols are implemented using the following instruction sequences:

- node i write to left :

  `WAIT`($\neg P_i^{left}$)
  `WrL`(data)
  `SET`(P_i^{left})

- node i write to right :

  `WAIT`($\neg P_i^{right}$)
  `WrR`(data)
  `SET`(P_i^{right})

This protocol implies that writing a new value in the register is possible only after the associated status bit has been reset by the neighbouring processor reading the value. Read protocol is also performed by the following sequences:

- node i read from right :

  `WAIT`(P_{i-1}^{right})
  `RdL`(data)
  `RESET`(P_{i-1}^{right})

- node i read from left :

  `WAIT`(P_{i+1}^{left})
  `RdR`(data)
  `RESET`(P_{i+1}^{left})

P_{i+1}^{left} and P_{i-1}^{right} stands respectively for the left neighbour status bit P_i^{left} and for the right neighbour status bit P_i^{right}. While the status bit is not set (meanning that the data register is empty) the reading process is blocked.

The data register lying on a FPGA is accessible from the TRANSPUTER of the same node. Address decoding selects the operation to be done. Synchronization is assumed by the `WAIT` line control of the TRANSPUTER. It enables memory wait state during read or write access to a data register.

4 Systolic programming environment

Systolic computations using the designed operator approach and the address mapping defined in the previous section could be programmed in C. However the programmer has to deal with race conditions or communication protocols that are difficult to program correctly. This is the reason why we have adapted our systolic programming environment and its language, RELACS, to the ARMEN machine.

4.1 The programming model

The programming model we choose, makes the assumption of a fully synchronous execution of two processes (see figure 4):

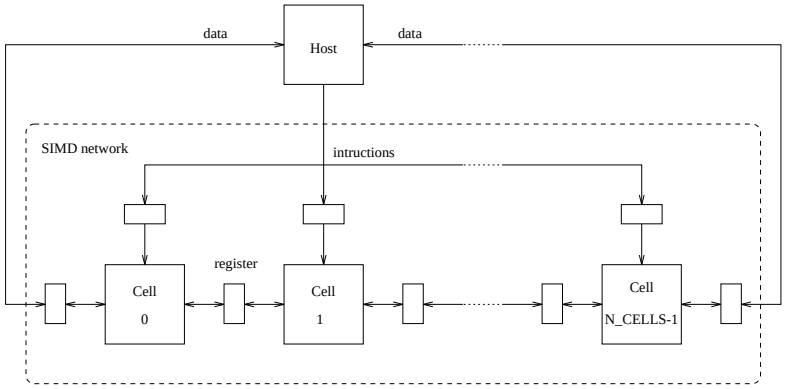


Figure 4: SIMD programming model

- The *computation process* executes the computational task in a systolic fashion with fine-grain access to the input/output ports. It takes place on a linear array of identical processors (the systolic array) controlled by a single controller which broadcasts the same instruction to each processor. Data transfers are synchronous and restricted to neighbouring cells.
- The *data management process* is responsible for correctly ordering data and for collecting results from the *computation process*. The *data management process* runs on an independant processor with its own controller. It communicates directly with the extreme ends of the systolic array.

Synchronizations between the processes occur during communication and conditional instructions.

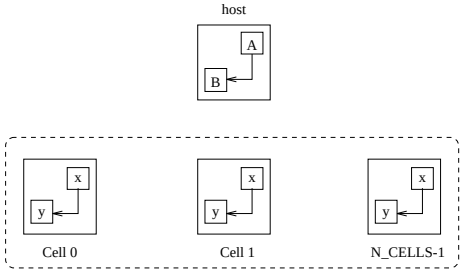
The machine on which the two processes are running is called a systolic machine. This machine can be considered as an accelerator for the host and is accessed through procedure calls from applications running on the host. Even though the data management process is implemented on the systolic machine, from the programmer's point of view, it appears to be running on the host.

This programming model can be mapped onto various machines: the synchronous execution and the synchronous communication simplify the model for the programmer. However the execution model stays asynchronous to exploit the concurrency between the *computation process* and the *data managment process*. The communications and execution modes employed (SIMD or SPMD) depend on the target architecture. On ARMEN node 0 is devoted to the *data management process* while the others all run the same *computation processes*.

4.2 The RELACS language

The RELACS language is designed to efficiently program parallel architectures that support systolic communications. Current target machines include iWARP, MICMACS[9] and now ARMEN. RELACS is a C -like language [6] augmented with parallel constructs. The user writes a single source program, from which the compiler generates code for both the host and each processor cell of the network. The partitionning is done by data types as described below.

The programmer of such a machine needs a way to differentiate between host variables (scalar variables), and objects which are located on the systolic array (systolic variables). The RELACS language defines a new storage class specifier, `systolic`, to allocate variables in each cell of the network. A statement operating on systolic variables implied the simultaneous execution of this operation on all the cells of the network. This programming model is referred to as *data parallel* [4]. An example is given in figure 5. This example also shows that, despite the fact that these operations are written sequentially, they can be executed in parallel according to the execution model.



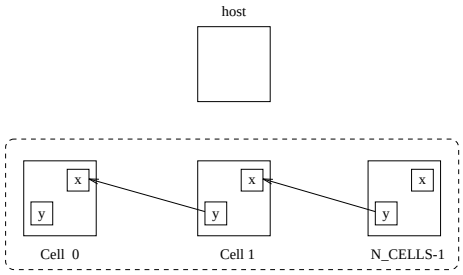

```
int A, B;
systolic int x, y;
B= A;
y= x;
```

Figure 5: assignments on the host and on systolic array

In systolic architectures, data transfers between processors are very important. Special care is devoted to this I/O mechanism in the RELACS language. New assignment operators match the hardware architecture and express the tight coupling between neighbouring cells.

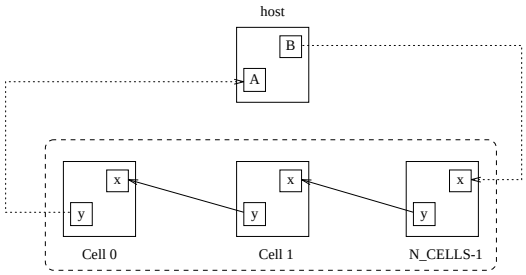
The key characteristics of the programming model are: the SIMD execution mode; and the synchronous communication mechanism, which forces each cell to execute a send followed by a receive during each communication cycle. As all cells execute the send / receive simultaneously, one can view the data transfer as a global shift operation on `systolic` variables involved in the communication sequence. In RELACS, this overall operation is expressed by the left (`=<`) and right (`=>`) systolic assignment. Figure 6 represents data flowing from right to left. Each cell sends the content of its variable `x` to its left neighbour and receives in variable `y` the value from its right neighbour.

Two optional parameters to systolic assignments handle the boundary conditions, i.e. the assignment of the array input and output to variables on the host. A scalar variable or a constant on the right side means inserting data from the host into the network. Adding a scalar variable on left side indicates the collection of a result on the host (see figure 7).




```
systolic int x, y;
x =< y;
```

Figure 6: internal network communications




```
int A, B;
systolic int x, y;
x : A =< y : B;
```

Figure 7: host extended communications

4.3 Compilation on ArMen

We now describe the use of hardware operators previously defined in the FPGA with the ReLaCS language. The systolic operators "=<" and "=>" are translated into memory accesses to the registers in the FPGA (see table 1).

ReLaCS	Host	Cell 0	Cell 1	Cell 2
$x =< y$			WrL(y)	WrL(y)
		x=RdR()	x=RdR()	
$x : A =< y$		WrL(y)	WrL(y)	WrL(y)
	A=RdR()	x=RdR()	x=RdR()	
$x =< y : B$	WrL(B)		WrL(y)	WrL(y)
		x=RdR()	x=RdR()	x=RdR()
$x : A =< y : B$	WrL(B)	WrL(y)	WrL(y)	WrL(y)
	A=RdR()	x=RdR()	x=RdR()	x=RdR()

Table 1: translation of the ReLaCS operator "=<"

We observe that in each case the number of read and write accesses is the same. This property and the fact that programs generated by the ReLaCS compiler are fully deterministic (no race conditions) ensure the validity of the protocol.

5 Conclusion

Our experimental results about word-by-word communications using FPGA chips indicate a great improvement in speed as compared with the message passing method. Figure 8 shows how time increases with the

number of exchanges. The angle between the two lines demonstrates the interest in terms of efficiency of performing synchronous communication operators in the FPGA layer of the ArMen machine. The ReLaCS language gives us a programming environment to develop and execute systolic algorithm on ArMen.

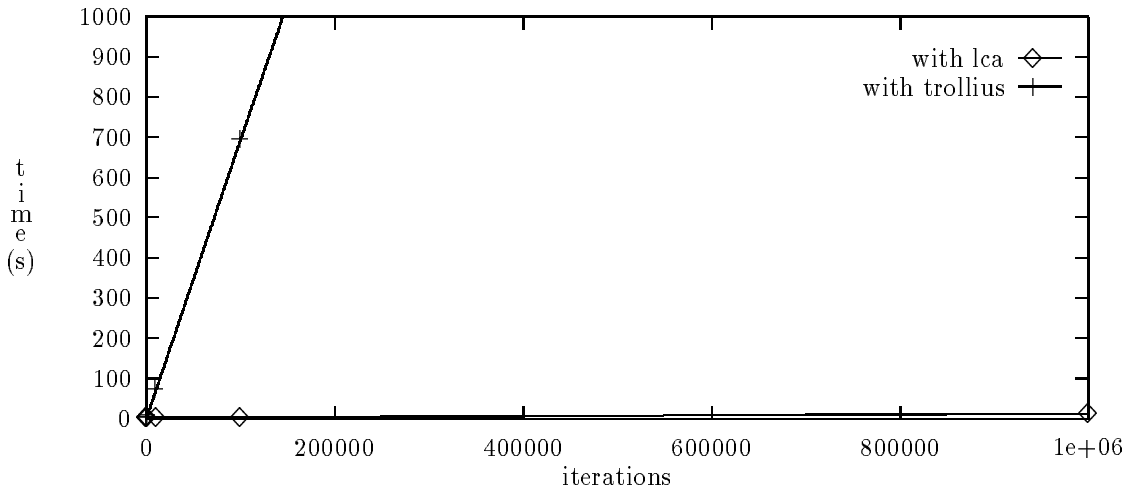


Figure 8: iterative executions of (`x=<y;`)

Ongoing investigation involves the design of arithmetic operators tightly coupled with the systolic communicating operators in the FPGA. The basic idea is to move down into the FPGA layer the most frequently used functions in systolic algorithms to increase the overall power of our machine. Such operators will have to be indicated and synthesized by the ReLaCS compiler. Another research interest concerns the execution model of the ArMen machine with the possibility of alternating distributed irregular computations and efficient systolic execution of global conditions, global score or global state over the network.

References

[1] J. Backus. Can Programming be Liberated from the Von-Neumann Style ? *CACM*, 8:613–641, 1978.

[2] S. Borkar, R. Cohn, G. Cox, S. Gleason, T. Gross, H.T. Kung, M. Lam, B. Moore, C. Peterson, J. Pieper, L. Rankin, P. Tseng, J. Sutton, J. Urbanski, and J. Webb. iWarp :An integrated Solution to High-Speed Parallel Computing. In *International Conference on Supercomputing*, 1988.

[3] K. Bouazza, J. Champeau, B. Pottier, and S. Rubini. Implementing Cellular Automata on the ArMen Machine. In P. Quinton and Y. Robert, editors, *Parallel Algorithms and VLSI Architectures II*, pages 317–322, Elsevier, June 1991.

[4] W. Hillis and G. Steele. Data Parallel Algorithms. *Communications of the ACM*, 29(12):1170–1183, December 1986.

[5] *The Transputer Data Book.* INMOS, 1989.

[6] Kernighan, W. Brian, and D. M. Ritchie. *The C programming language.* Prentice-Hall, 1978.

[7] H.T. Kung. Systolic Communication. In *International Symposium on Computer Architecture*, pages 695–703, May 1988.

[8] H.T. Kung. Why Systolic Architectures? *Computer*, 15(1):37–46, 1982.

[9] Dominique Lavenier. MicMacs : un réseau systolique linéaire programmable pour le traitement des chaines de caractères. Thèse de l'Université de Rennes 1, June 1989.

[10] B. Pottier. Armen, une machine parallèle intégrant un réseau de circuits reconfigurables. Thèse de l'université de Rennes I, June 1991.

[11] *Trollius Manual Set.* The Ohio State University, 1992.

Unité de recherche INRIA Lorraine, Technôpole de Nancy-Brabois, Campus scientifique,
615 rue de Jardin Botanique, BP 101, 54600 VILLERS LÈS NANCY
Unité de recherche INRIA Rennes, IRISA, Campus universitaire de Beaulieu, 35042 RENNES Cedex
Unité de recherche INRIA Rhône-Alpes, 46 avenue Félix Viallet, 38031 GRENOBLE Cedex 1
Unité de recherche INRIA Rocquencourt, Domaine de Voluceau, Rocquencourt, BP 105, 78153 LE CHESNAY Cedex
Unité de recherche INRIA Sophia-Antipolis, 2004 route des Lucioles, BP 93, 06902 SOPHIA-ANTIPOLIS Cedex

Éditeur
INRIA, Domaine de Voluceau, Rocquencourt, BP 105, 78153 LE CHESNAY Cedex (France)
ISSN 0249-6399