

A NEW PROGRAMMING METHOD FOR HIGHLY PARALLEL ARCHITECTURE

Eric PAYAN, Guy MAZARE

IMAG - L61

Laboratoire de Genie Informatique
INPG

46 Avenue Félix Viallet
38031 Grenoble Cedex

France

Email: payan@civa.imag.fr

With the recent development of massively parallel architectures (from large Transputer Network to the XILINX) there is a growing need for new programming methods. We propose here a new method to program highly parallel architecture, which can be used to solve many problems as : neural network programming, digital signal processing[1], systolic array,...

This job is part of a larger project : a programmable highly parallel architecture which will also be presented.

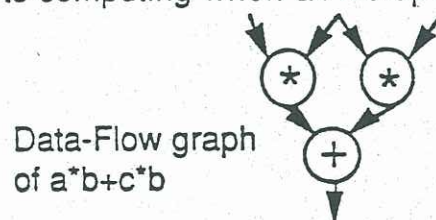
1. INTRODUCTION

The progress of VLSI technology are now allowing the realization of highly parallel architectures (more than 1000 processing elements).

Many topologies and architectural designs for processing arrays (including globally synchronous systolic arrays [2] and globally asynchronous wavefront array [3]) have recently been proposed. Most existing algorithms for such arrays were developed for problems with inherent regularity (vector and matrix operations, FFT...). But many algorithms do not have underlying regularity and, therefore are not suitable for these arrays.

The classical low level programming problems still exist on these architecture : poor instruction set, high debugging complexity; but parallel processing adds new problems : process synchronization, interlock... All these problems make the manual programming of a large network very difficult.

On a data-flow computer data processing is represented as a graph, where each node is an operator which starts computing when all his operands are available.



We suggest that mapping a Data Flow Graph extracted from a LUSTRE [4] program on a cellular network with global communications [5] could be an easy and efficient way to program highly parallel architecture.

2. THE HARDWARE CONCEPT

2.1 The network

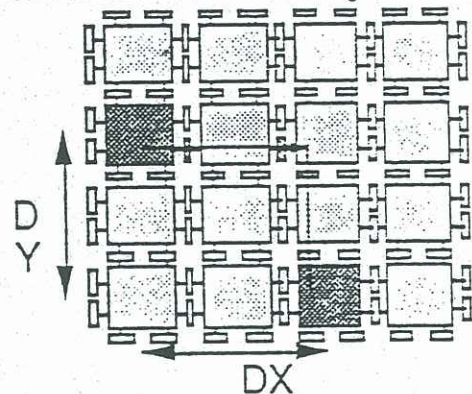
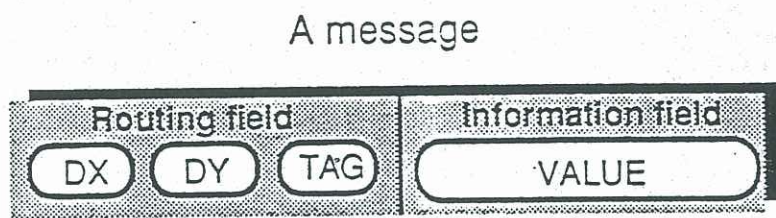
The hardware architecture is a regular network of asynchronous cells. Each cell performs a simple local function and is surrounded by eight one-way-driving buffers, one in each way of the four directions. A flip-flop based mechanism performs the mutual exclusion of the two asynchronous cells sharing a buffer.

2.2 Message transmission mechanism

Each cell includes a routing mechanism which allows the transmission of the message to any cell through the network. A message is made of two parts :

- an information field carrying the value intended to the application implemented on the network.

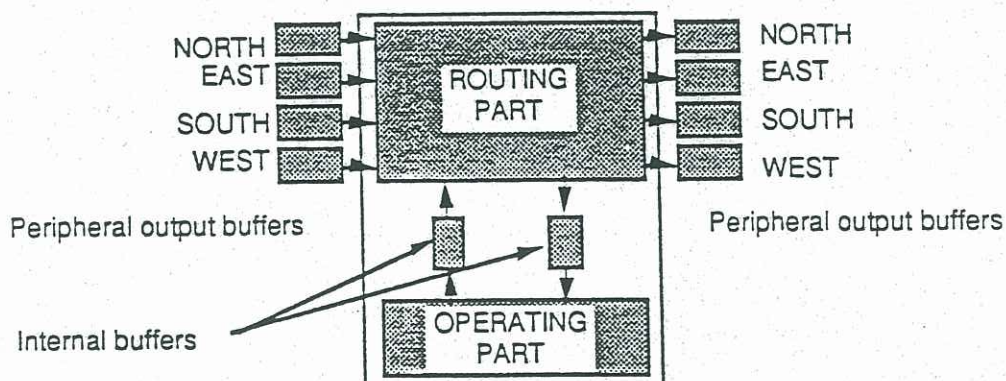
- a routing field carrying information required by the message transmission mechanism. The routing algorithm needs the relative displacement dx, dy to the addressed cell and a memory address in the destination cell to store the message.



2.3 The cell

Each cell performs simultaneously two concurrent tasks :

- the routing of messages from input to output buffers
- the processing part is a 8 bits microprocessor with a small memory (typically 256 bytes).



Some other applications using this network have already been studied by our team:

- logical simulation : a chip including 4 cells has been realized [6] .
- Image reconstruction : a chip has been realized[7]
- neural network with a delta learning rule [8]

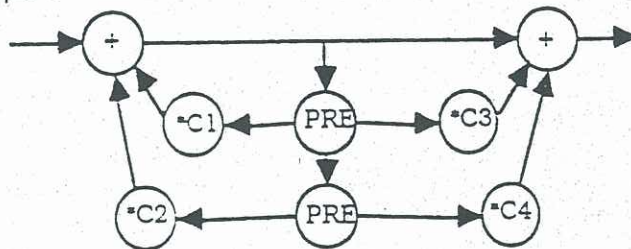
Computing a data flow program onto our array of processors consists of four phases:

- 1 : DF - program translation :
- 2 : DFG mapping onto array
- 3 : Array initialization
- 4 : Program execution

The first step is the translation of a dataflow program, written in LUSTRE, in a dataflow graph. This is general and does not depend on the specific architecture, it includes syntactic verifications and array expansion.

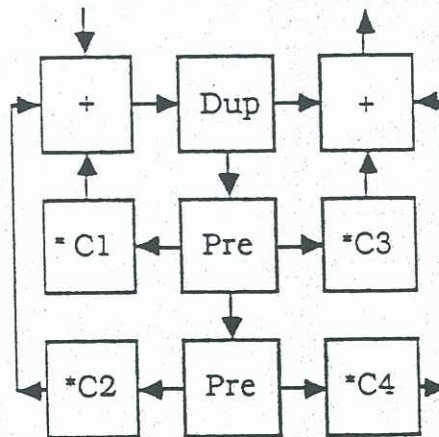
The LUSTRE program of the biquadratic filter:
 $I = \text{Input} + \text{Pre}(I) * c1 + \text{Pre}(\text{Pre}(I)) * C2$
 $\text{Output} = I + \text{Pre}(I) * C3 + \text{Pre}(\text{Pre}(I)) * c4$

The data flow graph :



For the second one many algorithms have been written. Actually we are using a simulated annealing methode.

The mapping off the bi-quad on the network :



The host computer uses the routing part of the cells for sending to each one its how algorithm (with depends of the operators mapped on the cell).

The asynchronous cell operations and the variable length of the message paths introduce significant speed differences in the message routing . Some cells may receive their inputs and send their outputs faster than others. As we want to be sure that each message sent can be received, and because there is no FIFO file between cells, we have to synchronize them.

The use of acknowledge messages seems to be the best compromise between hardware complexity and maximum speed.

3. MAIN FEATURES OF LUSTRE

In the past few years, some computer scientists have tried to find more formal way for programming. The LUSTRE language answers these problems by giving to programmers mathematical tools for program proving.

In a real system a physical measure have a different value at each instant. The LUSTRE language has the property (As in LUCID[9]) than a variable X can be seen as an infinite sequence $x_0, x_1, \dots, x_n, \dots$ of values; this makes easier digital signal processing programming.

In LUSTRE, each value belong to a domain $D(X)$ characterized by the type of X . Each domain contains an undefined value called nil. A variable X can be defined by means of an equation " $X=E$ ", where E is an expression defining a sequence e_0, \dots, e_n, \dots of values belonging to $D(X)$, whose interpretation is :

for every $n \geq 0$, $x_n = e_n$

or, from a timed point of view, at any time n , the value of X equals the value of E .

3.1 Synchronous Operators

Right hand side expressions may be built by means of variables, identifiers, constants (considered as infinite constant sequences) and the operators defined below:

Data Operators: All the operators over domains of values (for instance, boolean or arithmetic operators, if-then-else and case-of conditional operators) are extended to pointwisely operate on sequences:

for any operator op , of arity i , the n -th term of the sequence defined by $op(X_1, X_2, \dots, X_i)$ is the result of applying op to the n -th terms of X_1, X_2, \dots, X_i .

Synchronous Sequence Operators : If E is an expression, defining the sequence $e_0, e_1, \dots, e_n, \dots$. Then $pre(E)$ defines the sequence $nil, e_0, \dots, e_{n-1}, \dots$

If F is an expression of the same type as E , defining the sequence f_0, \dots, f_n, \dots then $E \rightarrow F$ (read E followed by F) defines the sequence $e_0, f_1, \dots, f_n, \dots$. For instance, the equation

$$X = 0 \rightarrow pre(X) + 1$$

defines the variable whose n -th value x_n satisfies

$$x_n = 0, \text{ if } n=0$$

$$x_{n-1} + 1, \text{ if } n>0$$

and, so, $x_n = n$

3.2 Program Structuring

A node is a subprogram. It receives input variables, computes output variables, and possibly local variables.

3.3 Arrays

In LUSTRE arrays can have as many dimensions; in order to make possible static check of semantic consistency, their use is restricted by the following constraints:

- Array bounds must be known at compile time
- Array must not be indexed by variables :

index must be an expression made of simple operators (+, -) applied to constants and indexes of "for...let...tel" constructs. Such a construct allows global definition of array elements. For instance, we can write:

$$PX[0] = X; \text{ for } i \text{ in } 1..n \text{ let } PX[i] = pre(PX[i-1]) \text{ tel;}$$

which defines $PX[i]$ as $pre^i(X)$, for i in $\{0..n\}$. This definition allows the programming

of computing arrays (ie systolic [10]).

3.4 Clock Changes

Until now, programs are written in strong connection with their basic execution cycle: all variables in a program evolve with the same frequency. However, there is a strong need of being able to define variables with slower frequency than the basic cycle.

Sampling : We define a clock to be any boolean variable. If C is a clock and E is an expression. "E when C" is an expression defining the sequence whose n -th term is the value of E at the n -th instant when C is true. "E when C" is said to be on clock C .

Current value : If E is an expression of clock C , then $\text{current}(E)$ is an expression whose value at each cycle is the value taken by E at the last cycle when C was true.

The following table (where tt and ff stand for true and false) illustrates these operations:

$C =$	tt	ff	tt	tt	ff	ff	tt	ff	tt
$E =$	e0	e1	e2	e3	e4	e5	e6	e7	e8
$X = E \text{ when } C =$	e0		e2	e3			e6		e8
$Y = \text{current}(X) =$	e0	e0	e2	e3	e3	e3	e6	e6	e8

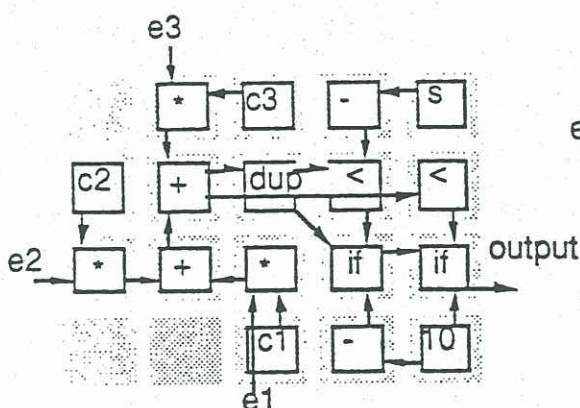
4. PERFORMANCE STUDIES

4.2 Performance evaluation

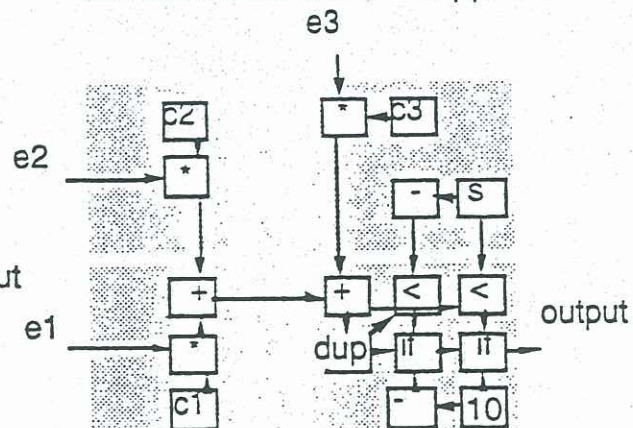
For larger (already more than 1000 cells) and more precise network simulation we have programed a complete simulation tool, including : Data flow graph extraction, mapping (by a simulated annealing method), assembling language generation and network simulation.

As the network is asynchronous, each operator has a different computing delay (depending of its function). If we use the 1cell=1node; mapping faster cell will have to wait for the slower one. By mapping several "fast" nodes on one cell we can increase the network activity.

1 Cell = 1 Node mapping

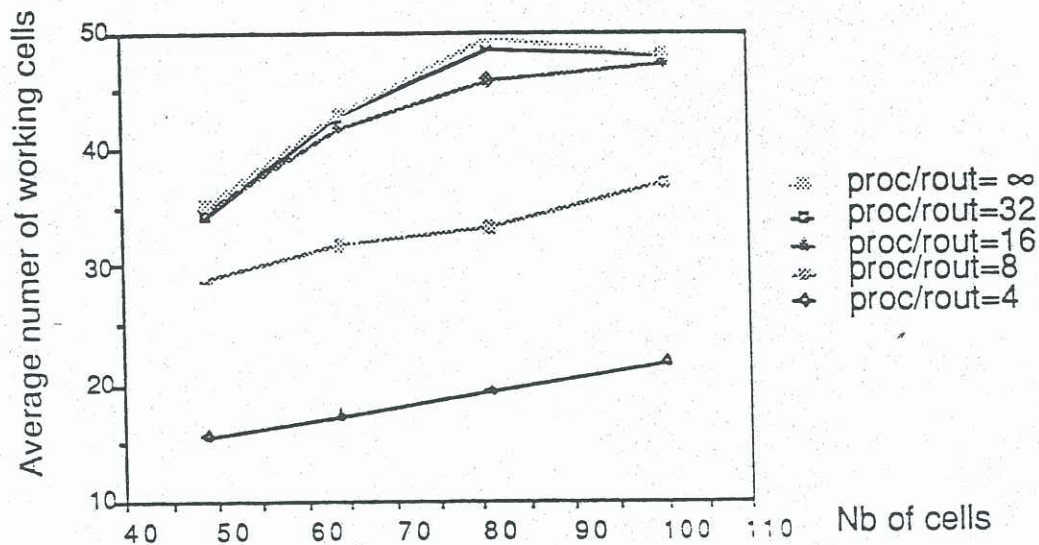


Several nodes can be mapped on one cell



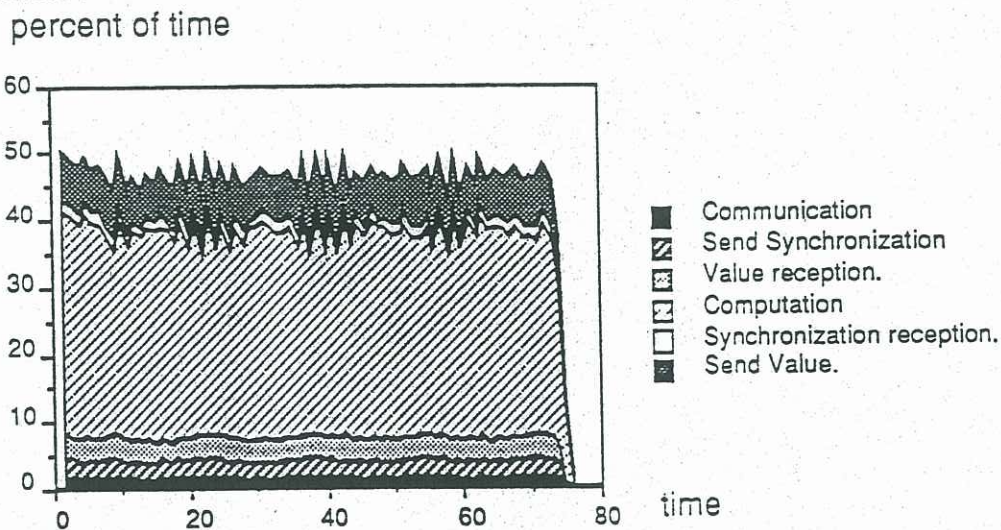
One of the main results we obtain from simulations was the answer to the question : how many data-flow nodes shall we map on one cell to obtain the best result?

We simulated a fast Fourier transform with several mappings (from 304 nodes on 4 cells to 304 nodes on 100 cells) and with several communication delays (proc/rout=4 means that the average delay needed by an operator to compute his function is 4 time slower than the delay needed to route a message from one side of the cell to the other one). We obtain the following result:



The best result appears to be obtained with a few nodes on each cell.

Our simulation tools also give us some results on the cells utilization. We can observe on this graph (obtain with a convolution product exemple) what is the ratio between computation (which is the usefull part of the work) communication and synchronization :



4.2 Communication performance evaluation

On a massively parallel architecture, communication is one of the possible bottleneck. We had also studied a combinatory routing part with can achieve the following performances:

- 4 Messages can be route in parallel (since they did not collapse together).
- It take less than 40ns to route a message from one side of a cell to another.

5. CONCLUSION

We believe that highly parallel architecture programming can be a real problem without specific compilation technique (especially on non regular problems). With the complete compilation and simulation tool we had developed, we can already achieve the following goal:

- Independence between program and architecture (the same program can be mapped on various network size)
- The total mapping automatically done is totally automatized so that the programmer does not have to worry about network topology, parallelism extraction...
- Many optimization can be done (critical path detection ..) but the network utilization is already good.

REFERENCES

- [1]. G. Mazaré E. Payan "A programmable Highly Parallel Architecture for Digital Signal Processing" 1989 IEEE International symposium on circuits and systems.
- [2]. H.T. Kung "Why systolic architectures?," IEEE Computer, vol.15 January 1982
- [3]. S.Y. Kung, "On Supercomputing with Systolic/Wavefront Array Processors" Proceeding of the IEEE, Vol 72 No7 July 1984.
- [4]. J.A. Plaice, N. Halbwachs "LUSTRE V2: User's Guide and Reference Manual" Laboratoire de Genie Informatique Octobre 1987.
- [5]. P. Objois "Reseau de cellules integre: mecanisme de communication inter-cellulaire et application a la simulation logique." Thesis, Institut National Polytechnique de Grenoble septembre 1988.
- [6]. R. Cornu-Emieux "Reseau de cellules integre : Etude d'architectures pour des applications de CAO de VLSI" Thesis, Institut National Polytechnique de Grenoble septembre 1988.
- [7]. D. Lattard G. Mazaré "Image Reconstruction using an Original Asynchronous Cellular Array" 1989 IEEE International symposium on circuits and systems.
- [8]. B. Faure, G. Mazaré "A VLSI Asynchronous Cellular Architecture Dedicated to Multilayered Neural Networks" proceedings of nEuro '88.
- [9]. E. A. Ashcroft, W. W. Wadge "LUCID, the data-flow programming language", 1985 Academic Press
- [10]. N. Halbwachs, D. Pilaud "Use of a Real-Time Declarative Language for Systolic Array Design and Simulation" Systolic Arrays Design Methods and Tools