

# A Data-Flow Functional Computer for Real-Time Image Processing

Georges M. QUÉNOT, Bertrand ZAVIDOVIQUE

Laboratoire Système de Perception  
DGA/Etablissement Technique Central de l'Armement  
16 bis Av. Prieur de la Côte d'Or, 94114 Arcueil CEDEX (FRANCE)  
Tel: (33 1) 42 31 93 48 Fax: (33 1) 42 31 99 55 Email: quenot@etca.fr

## Abstract

*We present a data-flow computer, constituted of an array of data-flow processors, dedicated to real-time image processing. A custom processor, able to perform up to 50 millions of operations per second on 25 Mbytes/s data-flows, have been developed for low-level computations and T800 transputers are used for high level ones.*

*The data-flow computer is programmed in a pure functional style (Backus) programming language with a single formalism for both types of processors.*

*An experimental system including 256 low-level data-flow processors and 1 transputer has been built and several image processing algorithm have run on it in real time at digital video speed. This system will be extended very soon to 1024 low-level processors and 32 high level processors.*

## 1 Introduction

This paper describes a data-flow functional computer (DFFC) dedicated to real-time image processing. Image processing computations may be roughly divided in two classes: low-level and high level.

By low-level image processing, we mean calculations where the 2D image topology still prevails (a fixed number of operations is involved for each pixel). Such processing is very demanding on computing power and generally involves very simple and very repetitive types of calculations. For example: filters, convolution, histograms, summations, momentums, edge points detection, region growing and labelling.

By contrast, high-level image processing is characterized by the manipulation and interpretation of extracted features such as contours, polygons, sets of points and symbolic trees. Such calculations generally require less raw computing power but also involve much more algorithmic complexity.

For such calculations, Von Neumann style or other classical architectures fail to provide the computing power required by real-time processing. Von Neumann

style programming languages also fail to provide an adequate environment for real-time image processing application development.

Many innovative computers have been proposed as alternatives to Von Neumann's. Among them, data-flow architectures [1] [2] appear very well suited to real-time image processing.

Alternative programming languages have also been proposed to the Von Neumann style. Among them the concept of functional programming [3] seems very efficient for the expression of image processing algorithms.

Having exhibited a family of the most frequent operations in image feature extraction, our approach is to integrate the data-flow architecture principle and the functional programming concept together.

This is done through a unification principle which expresses the natural duality between a Control Data-Flow Graph (CDFG) and a functional expression. Both are ways of describing an image processing algorithm. The first representation is relative to a procedure hardware implementation, the second one is its natural software expression.

A functional programming language was defined in which the primitive functions are the basic real-time operators implementable on the physical processors of a data-flow machine. By so doing, a bidirectional and automatic translation between CDFGs and functional expressions becomes possible. The automatic CDFG implementation onto the data-flow machine is feasible using a place and route algorithm (Figure 1)

## 2 Data-Flow Computer Architecture

We call "data-flow" a structured data set moving serially along a physical link. Our approach is to achieve computations on data-flows, coming directly from digital video cameras, at the pixel acquisition rate and to couple one physical processor with each elementary operation defined in the algorithm.

The core of the data-flow functional computer is a network of processors. Processors may be used simultaneously as data-flow operators or as a cross-bar

**ALGORITHM:**

$$Y(n) = a.X(n+1) + b.X(n) + c.X(n-1)$$

**INVOLVED FUNCTIONAL OPERATORS:**

$$D : (X(1) X(2) \dots X(n)) = (0 X(1) \dots X(n-1))$$

$$id : (X(1) X(2) \dots X(n)) = (X(1) X(2) \dots X(n))$$

$$A : (X(1) X(2) \dots X(n)) = (X(2) X(3) \dots X(n) 0)$$

$$xa : (X(1) X(2) \dots X(n)) = (a.X(1) a.X(2) \dots a.X(n))$$

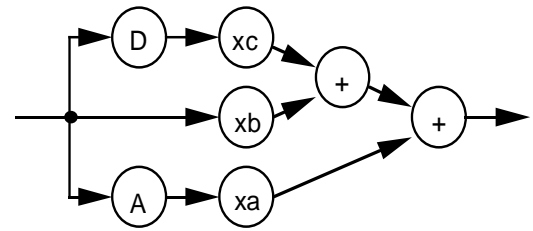
$$+ : ((X(1) X(2) \dots X(n)) (Y(1) Y(2) \dots Y(n))) = (X(1)+Y(1) X(2)+Y(2) \dots X(n)+Y(n))$$

**FUNCTIONAL EXPRESSION:**

$$DEF F = +o[xa \circ A, +o[xb \circ id, xc \circ D]]$$

$$(Y(1) Y(2) \dots Y(n)) = F : (X(1) X(2) \dots X(n))$$

**DATA-FLOW GRAPH:**



**MAPPING ONTO THE DATA-FLOW COMPUTER:**

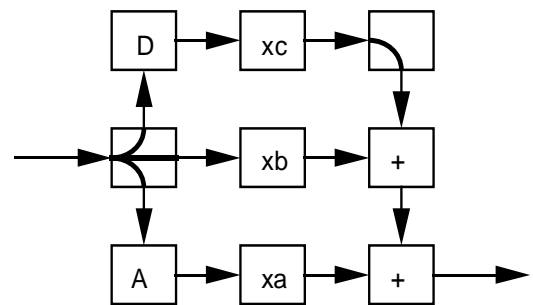


Figure 1: Functional programming and data-flow implementation

routing elements. Two different kinds of processors are integrated in the network. They are respectively dedicated to low-level and high-level image processing. This computer also incorporates digital video input/output subsystems and a global controller coupled with an host workstation (Figure 2).

For the execution of low-level functions, we developed a custom data-flow processor (DFP) [4]. Two coupled DFPs have been included in a single chip. Each DFP has 6 input-output ports (it has been designed to be mesh-connected in 3D networks). Each port is made up of 10 bidirectional lines (9 data and 1 acknowledge). The core of the data-flow processor is interfaced to the outside world through 3 input stacks and 3 output stacks, where each one is a synchronous FIFO with 8 usable 9-bit words, and with a 25 MBytes/second bandwidth. Each I/O port contains a receiving part that can be connected to any input stack and a sending part that can be connected to any output stack (Figure 3).

A datapath is inserted between the input and output stacks. It splits into three stages of pipe-line. The first stage decodes the input data type and generates commands for the following stages. 8-bit operations (input shifts and multiplications) are performed during the second stage, and 16-bit operations (2901-type ALU, absolute value, minimum or maximum, output shift) are performed at the third stage. This datapath is able to perform up to 50 Million of 8- or 16-bit operations per second.

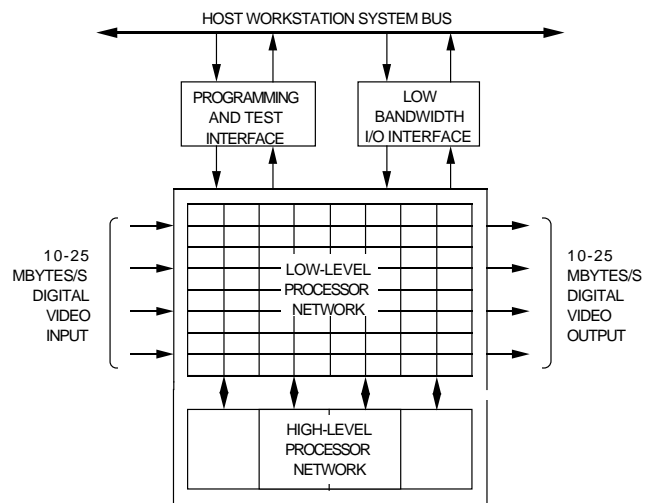


Figure 2: Data-flow computer architecture

A 256 9-bit words data RAM is included in each DFP. It can be used as a dual port RAM operator, as a 256 words FIFO, or as local memory for specific operators such as histogramming. The control part has been designed on the basis of a programmable state-machine for which a 64 32-bit word program RAM has been included in the DFP.

A wide variety of data-flow operators can be implemented using the state machine. Among them are additions, subtractions, multiplications, line or pixel

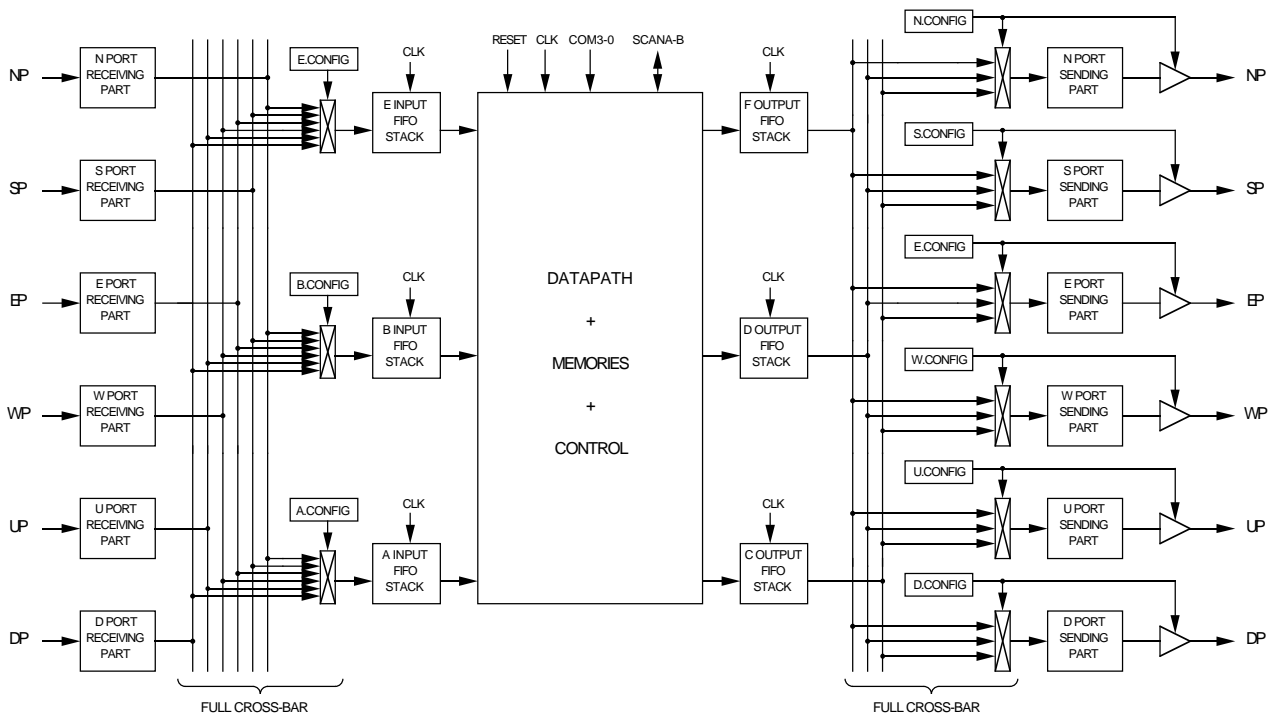


Figure 3: Data-flow processor internal architecture

shifts, derivatives, sommations, scalar products and histograms. More complex operators (convolution for example) can be defined as macro-functions using combinations of basic operators mapped on groups of processors.

For the execution of high-level functions, we choose the T800 transputer because it is a general purpose processor which directly incorporates communication links. It is therefore very well suited for the implementation of complex functions into data-flow operators. Transputer modules including 1 Mbyte of memory are the basic element of the high-level processing network.

Any kind of algorithm may be implemented on a high-level processing element as a C program. There are only two constraints: the first is to follow the data-flow principle (output data-flows are only functions of input data-flows, the control is done locally and the execution is independant of what may occur in other processors) and the second one is to satisfy real-time.

The following table summarizes the characteristics of both processors. Low-level and high-level processors communicate using bidirectional parallel/serial link adaptors.

DFP 8-bit	Low-level Simple operators	High bandwidth $6 \times 25$ Mo/s, 50 Mops
T800 32-bit	High level Unlimited complexity	Low bandwidth $4 \times 2$ Mo/s, 3 Mflops

### 3 Functional Programming

The execution of programs on the data-flow functional computer is fully data-driven. There is no global control. Each operator is activated by a completely local firing rule, and operations are performed as soon as all the required data are available (and if, simultaneously, destination processors for output flows are ready to receive new data). However, control flows may be inserted in data flows.

Every processor is programmed individually. The program of a processor contains two distinct and independant parts. The first one defines the function of the operator assigned to it. The second one defines the routing of data-flows between the physical ports and the logical input-output of the operator. This is the same for low-level or high-level processors.

The first task to perform in order to program the data-flow computer is to define a library of basic operators.

For the low-level processors, the basic operators are defined by the programming of the internal state machine. This is done using a specific assembler.

For the high-level processors, the operators are defined as a C program that uses transputer links as input and output files and that runs as an infinite loop. These operators also appear as data-flow ones.

After the library has been developed, every operator becomes a basic primitive in a proprietary data-flow

functional programming language (DFFPL), [5]. Low-level and high level primitives have exactly the same interface relative to this language. The language also incorporates the functional forms necessary to build any legal combinations of the primitives.

An image processing algorithm is described in the DFFPL language as a combination of basic primitives and macrofunctions using functional forms.

A natural duality between Data-Flow Graphs and Functional Expression has been exhibited [6]. Simple transformation rules permits an automatic translation of the whole algorithm in a complete data-flow graph in which the nodes corresponds to operators and macro-functions.

The data-flow graph is then mapped onto the physical network (Figure 1). This can be done automatically by using a place and route algorithm based on simulated annealing. After mapping has been completed, the processors are used simultaneously as operators or as cross-bar routing elements.

The algorithm is executed through a gigantic pipeline where every involved operation is assigned to a physical processor. Its complexity is therefore limited by the size of the available array but it is always executed in real-time.

Normal users generally will not need to develop their own basic operators since the machine will be delivered with a software package that includes a large library of currently used operators and macro-functions. Users will only need to write their image processing algorithms with the functional programming language. The library may be extended as and when required according to the users needs.

Programming the data-flow functional computer is then easy and efficient. Users may describe their algorithms in a text form or in a graphic form. They may place and route their data-flow graph themselves using a graphic editor or let the system do it.

From the theoretical point of view, it may be noticed that we make a pure use of the functional programming concept since the concept of variable never appear in our functional language. Similarly, we make a pure use of the data-flow principle since there is never an address-flow associated to a data-flow.

The common elimination of the variable and address concepts reflects a strong theoretical background in the duality between pure data-flow architecture and pure functional programming.

## 4 Experimental Results

We have developed a data-flow computer which has 256 DFPs (with 128 biprocessor chips) in a  $8 \times 8 \times 4$  tridimensional network and 1 transputer modules. This system is able to run with a 25 MHz clock

Several real-time image processing applications have already run at digital video speed onto this computer including convolutions, histograms, edge detection, region extraction and target tracking.

For example, Figure 4 shows the implementation of a macro-function performing edge point detection on a group of 8 processors and the result of its application in real time of this macro-function on a 25 Hz sequence of  $512 \times 512$  images.

The principle is to get, for each pixel, the absolute value of the normalized derivatives in four directions (1,0), (1,1), (0,1) and (-1,1), to take the maximum of them and to threshold the result.

14 operations per pixel are involved here (4 subtractions, 4 absolute value, 2 multiplications, 3 maximums and 1 threshold). DFPs are able to perform on the fly 2 arithmetic operations per pixel.

The data-flow functional computer will be extended soon to 1024 DFPs (in a  $8 \times 8 \times 16$  network) and 32 transputer modules. Its theoretical peak power will be of 50 billions of arithmetic operations per second.

## 5 Conclusion

This paper has described a data-flow functional computer for real-time image processing. Our approach integrates efficiently the data-flow architecture principle and the functional programming concept together.

It leads to the design of a very simple and regular hardware. It also provides a very nice and user-friendly software interface for the development of image processing algorithms. This approach can be maintained even with a system that includes processors of different granularity.

We have developed an experimental data-flow functional computer which has 256 DFPs (with 128 biprocessor chips) in a  $8 \times 8 \times 4$  tridimensional network and one T800 transputer.

Convolutions, histograms, edge detection, region extraction and target tracking have run at digital video speed onto this computer. Real-time color image processing algorithms are under development.

The data-flow functional computer is currently being extended to 1024 DFPs in a  $8 \times 8 \times 16$  tridimensional network and 32 T800 based transputer modules.

Our data-flow functional computer is mainly dedicated to image processing. However its concept is much more general. The basic processors may be adapted to address many other problems. For example, a 64-bit floating point data-flow processor could be similarly developed and used for the design of a data-flow functional supercomputer dedicated to the resolution of partial derivative equations.

**Basic operators:**

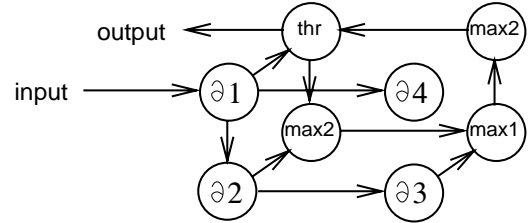
$$\begin{aligned} \partial 1 : [x_{i,j}] &= [| x_{i,j} - x_{i+1,j} |] \\ \partial 3 : [x_{i,j}] &= [| x_{i,j} - x_{i,j+1} |] \\ \partial 2 : [x_{i,j}] &= [| x_{i,j} - x_{i+1,j+1} |] \\ \partial 4 : [x_{i,j}] &= [| x_{i,j} - x_{i-1,j+1} |] \\ \text{max1} : ([x_{i,j}], [y_{i,j}]) &= [\max(x_{i,j}, y_{i,j})] \\ \text{max2} : ([x_{i,j}], [y_{i,j}]) &= [\max(x_{i,j}, y_{i,j}/\sqrt{2})] \\ \text{thre} : [x_{i,j}] &= [(x_{i,j} > th)] \end{aligned}$$

**Functional expression:**

$$\text{edge} = \text{thre}(\text{max2}(\text{max1}(\text{max2}(\partial 1, \partial 2), \partial 3), \partial 4))$$

$$\text{output} = \text{edge}:\text{input}$$

**Mapping on a 2 × 2 × 2 DFP array:**



**Result of real-time execution (on a view of the data-flow functional computer):**

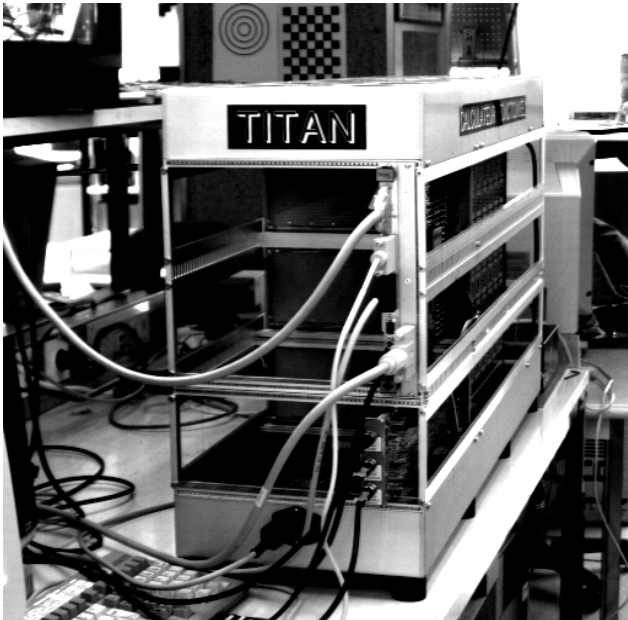


Figure 4: The edge-point detection macro-function

**References**

- [1] J.B. Dennis, "Data Flow Supercomputers," *Computer*, Vol 13, Nov. 1980, pp. 48-56.
- [2] K.P. Arvind, D.E. Culler, "Dataflow Architectures," *Annual Reviews In Computer Science*, Vol. 1, Palo Alto CA, Annual Reviews Inc 1986, pp. 225,253.
- [3] J. Backus, "Can programming be liberated from Von Neumann Style? A functional style and its algebra of programs." *Comm. of ACM*, Vol. 21, No 8, August 1978.
- [4] G.M. Quénot, B. Zavidovique, "A Data-Flow Processor for Real-Time Low-Level Image Processing," *IEEE Custom Integrated Circuits Conference*, may 1991, San-Diego, CA.
- [5] J. Serot, G.M. Quénot, B. Zavidovique, "Real-Time Image Processing using Functional Programming on a Data-Flow Architecture," *CAMP-91*, 15-18 december 1991, Paris, France.
- [6] E. Allart, B. Zavidovique, "Image Processing VLSI Design through Functional Match between Algorithm and Architecture," *IEEE Symposium on Circuits and Systems*, 7-9 june 1988, Espoo, Finland.