LA-UR- 02-1799

Approved for public release; distribution is unlimited.

Title: A New Polymorphous Computing Fabric

Author(s): Christophe Wolinski, * **
Maya Gokhale,*
Kevin McCabe, *

* Los Alamos National Laboratory
** IRISA, ISFIC France

Submitted to: ISSS2002, Kyoto, Japan
October 2-4, 2002

A New Polymorphous Computing Fabric.

Christophe Wolinski* **, Maya Gokhale*, Kevin McCabe*

*Los Alamos National Laboratory
Los Alamos, NM, U.S.A.

**IRISA, IFSIC France

Keywords: reconfigurable computing, FPGA, Configurable System on a Chip, Cellular Array, Computing Fabric

Abstract

This paper introduces a new polymorphous computing Fabric well suited to DSP and Image Processing and describes its implementation on a Configurable System on a Chip (CSOC). The architecture is highly parameterized and enables customization of the synthesized Fabric to achieve high performance for a specific class of application. For this reason it can be considered to be a generic model for hardware accelerator synthesis from a high level specification. Another important innovation is the Fabric uses a global memory concept, which gives the host processor random access to all the variables and instructions on the Fabric. The Fabric supports different computing models including MIMD, SPMD and systolic flow and permits dynamic reconfiguration. We present a specific implementation of a bank of FIR filters on a Fabric composed of 52 cells on the Altera Excalibur ARM running at 33 MHz. The theoretical performance of this Fabric is 1.8 GMAC/s. For the FIR application we obtain 1.6 GMAC/s real performance. Some automatic tools have been developed like the tool to provide a host access utility and assembler.

1. Introduction

We have recently proposed a parametric cellular array architecture[18] with unique characteristics composed of regularly interconnected compute nodes accessible by the host processor. This paper is an extension of previous work and describes an enhanced version of this architecture that is more flexible, and thus more adaptable for a given class of applications. We give examples of some applications implemented on this innovative architecture showing outstanding performance.

The high parametric nature, the modularity of this architecture, and the concept of global memory used for data and program access are principal novelties, which distinguish our Fabric from previous architectures [15], [11], [7], [8], [9], and [2].

The recent development of the Altera Excalibur ARM Configurable System on a Chip (CSOC) has enabled us to build a system composed of a Polymorphous Fabric-based architecture and host ARM processor connected to the Fabric through the system bus. This system is characterized by the high performance at a low clock frequency as shown in Section 6.

Experience with the ARM CSOC[19] has shown how the data path architecture between the microprocessor and the Fabric is critical to performance. For this reason, in our design, each cell in the Fabric has a separate local data memory and local program memory. The collection of all the local memories constitute a global memory that is dual ported so that it is accessible by either the cell or the microprocessor.

In Section 3 and 4 we describe our Polymorphous Computing Fabric and its implementation on the Altera Excalibur ARM CSOC. The remainder of the paper shows how a representative algorithm is mapped to the Fabric and gives Fabric-based performance. We end with conclusions and future work.

2. Related Work.

In the past many different architectures were proposed introducing some important concepts such as: *active memory* PAM[15] where the re-configurable system appears to be memory to the host processor but unlike standard memory the data is processed between write and read operations, *bi-directional communication links* Remarc project [11] improve utilization of limited resources, *programmable datapath* RaPid[7] composed of blocks optimized for large computation, *configuration cache* GARP[9] which accelerates the configuration process, *runtime reconfigu-*

ration Chimera[8] where the memory contains all the configurations so that speculative execution is possible, *distributed control* RAW[16] that permits *MPMD* calculation, *transparent runtime reconfiguration* PACT[2] that utilizes a configuration controller and cache to apply the appropriate algorithms at precisely the right time.

Our architecture combines these different concepts while introducing others like a *flexible modular generic architecture* concept and a *global memory* concept where the Fabric is built above a global memory. The advantages are: the specific architecture is better matched to a given class of algorithms, simultaneously optimizing resource usage, and the data and program communication bottleneck is reduced by the fact that data are directly visible from separate cells. Run time communication between the cells is implemented by a bi-directional data network and by the host processor connected to the global memory running in the background.

3. Fabric Architecture

Our basic Fabric architecture of a mesh-connected configurable network of runtime reconfigurable cells is visible to the processor as memory containing instructions and data so the host processor can write (read) data into (from) any cell's local memory, allowing the host to both set and observe cell internal memory.

Each cell has its own micro-controller. Thus control of the Fabric is distributed across all cells. Each cell is capable of conditional execution, allowing local events to be propagated to other cells.

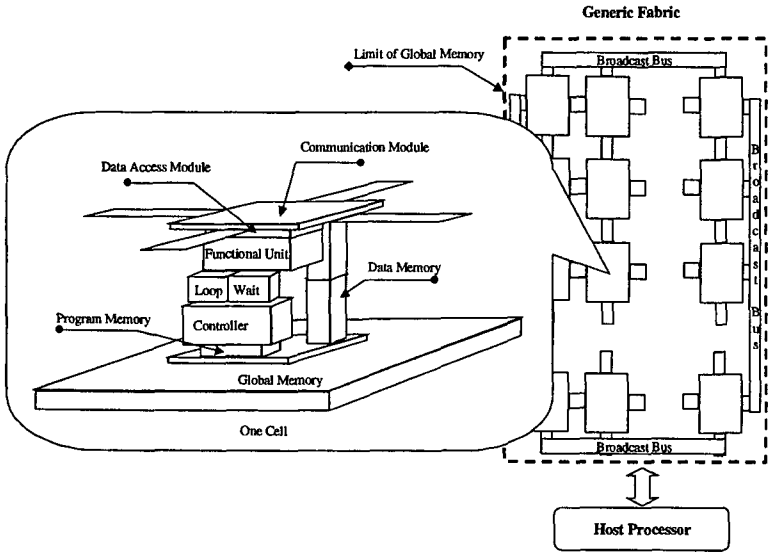


Figure 1. The layout of the Fabric.

This versatile computing Fabric supports a variety of computing and communications alternatives. Cells on the outer mesh border can broadcast to others on the border. Local nearest neighbor communications facilitates systolic computations.

Since each cell has its own program memory and control unit, *MIMD* computing is possible. If the same program is executed by all the cells, *SPMD* computing is accomplished. Moreover, *systolic flow* execution is possible. In this model of execution, data flows to the function units from an interconnection network in addition to the traditional mode of fetching operands from memory to execute in the function units.

For synchronization the host processor can broadcast a start or stop signal to the Fabric, and can observe any cell's status bit.

The layout of the Fabric and architecture of a cell is shown in Figure 1. The cell is composed of the following modules:

- Communication and Access Control module
- One or more Dual port Data memories
- One Dual port Program memory
- Controller with Wait and Loop modules

The cells are configured by parameters that are set prior to mesh generation and include:

- the size of the network
- the size of the communication busses
- the choice of functional unit for a given class of applications
- the number of local memories
- the type of access (random, sequential, circular)
- the instruction set layout
- communication direction

Other parameters can be modified during Fabric execution. These include:

- modification to the cell's communication pattern during application execution
- conditional execution
- conditional reconfiguration of the cell's memory access patterns

3.1. Communication and Access Control

The communication module connects the cell to a data network and a condition network.

It sends and receives data to the cell's immediate neighbors in the mesh, including its diagonal neighbors, and to the local memories and local functional unit. It feeds the functional unit with data coming from different sources. This module is high parameterized. It is possible to synthesize only the needed communication interfaces

and busses that optimize limited resource utilization. Figure 2 depicts an example where an unused datapath (Bus "S" South Direction) to an adjacent cell is removed and a connection to a second local memory (Memory Bus 2) is added.

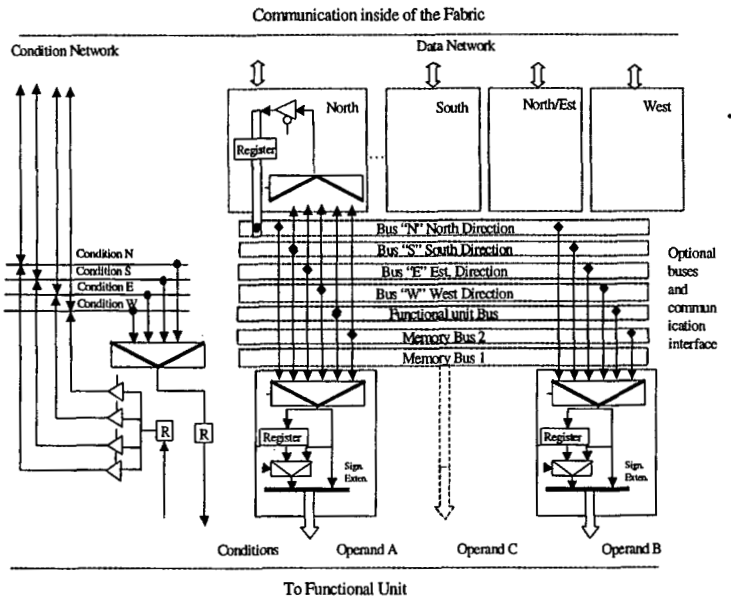


Figure 2. Communication and Access Control

3.2. Function Unit

The function unit performs operations on two or more operands.

Condition Module		
A>B	0	0
A=B	0	1
Take Cond from Cond Bus	1	0

Function 2			
OUT=ACC, ACC=A+B	0	0	0
OUT=ACC, ACC=A-B	0	0	1
OUT=ACC, ACC=ACC+FU1-OUT (A, B)	0	1	0
OUT=ACC, ACC=if Cond=1 then A else B	0	1	1
OUT=ACC, ACC=ACC+B	1	0	0
OUT=ACC, ACC=B	1	1	x
OUT=ACC, ACC=B	1	x	1

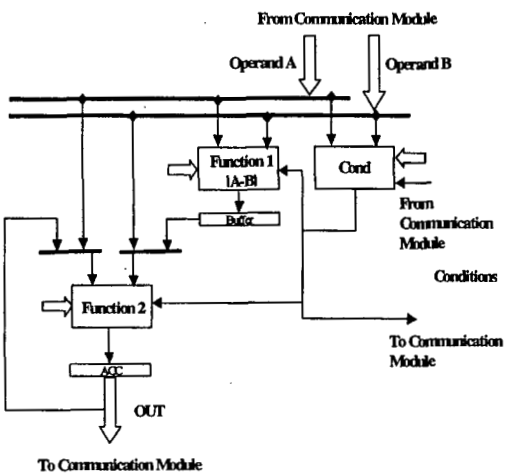
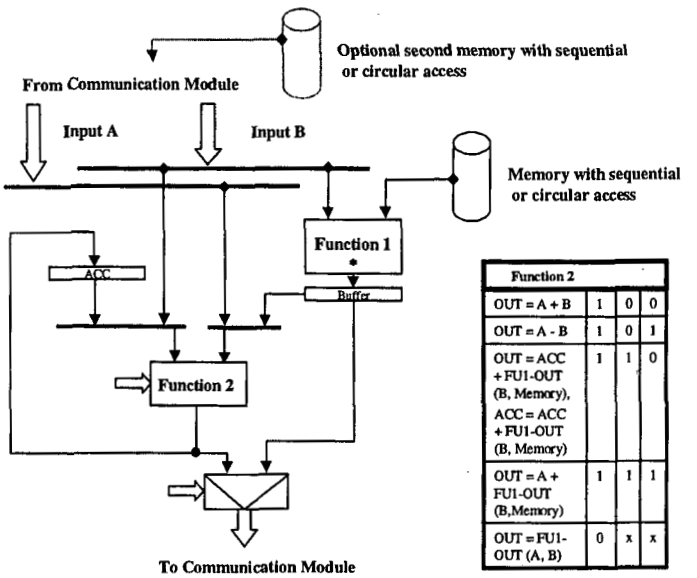


Figure 3. The layout of Function Unit 1.

In this modular architecture, function units are designed for application classes and can be replaced without affecting other aspects of the the Fabric.

In our experiments to date, we have designed many different function units, two are shown in Figure 3 and 4.



Function 2			
OUT = A + B	1	0	0
OUT = A - B	1	0	1
OUT = ACC + FU1-OUT (B, Memory), ACC = ACC + FU1-OUT (B, Memory)	1	1	0
OUT = A + FU1-OUT (B,Memory)	1	1	1
OUT = FU1-OUT (A, B)	0	x	x

Figure 4. The layout of Function Unit 2.

Within Function Unit 1, Function 1 is optimized for a distance calculation, while Function 2 can do one of the 6 operations.

In Function 2, Function 1 performs a multiply, while Function 2 can do one of 5 operations. This Function unit is optimized for DSP intensive processing. It has 3 inputs, the sources of two inputs are determined by the Access Control module and the source of the third is local memory. The local memories can be accessed using random, sequential or circular modes. The circular access mode is particularly good for a *delay line* implementation, sliding window, and repetitive coefficients access.

3.3. Instruction Set and Controller

The microcontroller executes instructions from the Program Memory. The instruction set consists of nine generic instructions where the instruction field depends on the specific cell implementation.

1. *Configure*. This instruction configures the Function 1 unit into one of eight possible operation modes. It configures the condition unit into one of eight modes. It selects communications busses, direction of communication, access mode (random, sequential, or circular) to local memories, and whether the function unit will operate in 8- or 16-bit mode.

2. *Control.* This instruction directs whether or not the function unit performs an accumulate; whether or not to perform a memory read or write access; controls which 8 bits of a 16 bit operand are to be accessed; and enables I/O to/from pipelined communication busses according to the communication pattern established by the *Configure* instruction. The "wait count" field of the control instruction serves a variety of functions. It can be used for looping: if a loop count field is non-zero, the instruction repeats the operation (such as accumulation) for the specified number of cycles. It can also be used for time synchronization, so that the cell waits the specified number of cycles before continuing with the next instruction.

3. *Jump.* This is the unconditional branch instruction to a 7-bit address.

4. *Conditional Jump.* A jump is executed if the Condition register is zero.

5. *Start Loop.* This instruction marks the beginning of a loop body. A 7-bit loop count is included in the instruction.

6. *End Loop.* This instruction marks the end of the loop body. If the loop count has reached zero, a branch is executed to the address supplied within the instruction. The combination of *Control* with *Start/End Loop* provide for two levels of nested loop. This instruction can also reset memory address counters and enable or disable memory and accumulator operations. That reduces the number of instructions necessary for a loop body implementation, in many cases by two cycles.

7. *Stop then Wait for Start.* The purpose of this instruction is to stop cell execution, setting the internal status bit to 1; and wait for the next start signal to arrive from the host processor. When the start signal is received, a branch is executed to the 7-bit address supplied within the instruction, and the status bit changes to 0.

8. *Reset.* This instruction selectively resets the specified function unit internal registers and then waits a specified number of cycles. As with the control instruction, the reset can be used for both looping as well as synchronization.

9. *Load.* Initialize the memory address counter and limit register for circular address generation. A combination of *Configure* and *Load* instructions are used for Random Memory Access.

This instruction set exposes the microarchitecture of a Fabric cell, and gives the programmer control over all the communication busses, memory, and function units. It is possible with this architecture to communicate independently from computation. Thus, a cell can compute using local memory and at the same time forward data through the interconnection network. The data distribution pattern can be dynamically reconfigured without affecting the state of computation. In addition, the architecture provides an optimized loop control mechanism for up to two nested loops. If a higher level of loop nest is desired, the host processor must coordinate the outer loops using the Fabric start/stop mechanism.

4. Excalibur ARM

Hybrid Configurable System on a Chip (CSOC) architectures have been proposed over the past several years ([12], [9], [13]). Recently these devices have begun to appear as commercial offerings ([1], [17]). In contrast to traditional FPGAs, these integrated systems offer a processor and an array of configurable logic cells on a single chip. We have implemented an instance of the general computing Fabric described above on the Altera Excalibur ARM hybrid system[1]. This chip contains an industry-standard ARM922T 32-bit RISC processor core operating at up to 200 MHz (equivalent to 210 Dhrystone MIPS). There is a memory management unit (MMU) included for real-time operating system support. This architecture builds upon features of the APEX TM 20KE PLD, with up to 1M gates.

In our implementation, the Fabric is connected to the dual port memory (on one side) and accessible by the ARM processor (from the other side) instead of directly to the slave bus. We have recently shown [19] that this solution is better because the AHB Master Port is about 10 times slower than a dual port memory for communication.

Our implementation uses 16-bit data paths for communication in a two-dimensional mesh of 52 processors.

Fabric/host communication is handled by a controller that manages both direct memory access to a 32K-16bit dual port RAM and to a 32bit bridge to the AHB2 bus of the ARM. Via the bridge the host can command the controller to:

- Send the program to given cells of Fabric
- Send Data to given cells of Fabric
- Receive Data from the Fabric

- Send Reset
- Send Start Strobe
- Read Stop Signal
-

The Fabric uses 8-bit data paths for communication and 8-bit registers, with instruction set support for 16-bit function unit operations. With manual placement (placement directives were generated by the script), 52 cells using Function Unit 1 were instantiated (13 rows x 4 columns) on the Excalibur ARM EPXA10F1020C2.

The clock frequency for Function Unit 1 and 2 is 33 MHz, giving peak performance of 10.2 GigaOps/s and 1.8 GMAC/s.

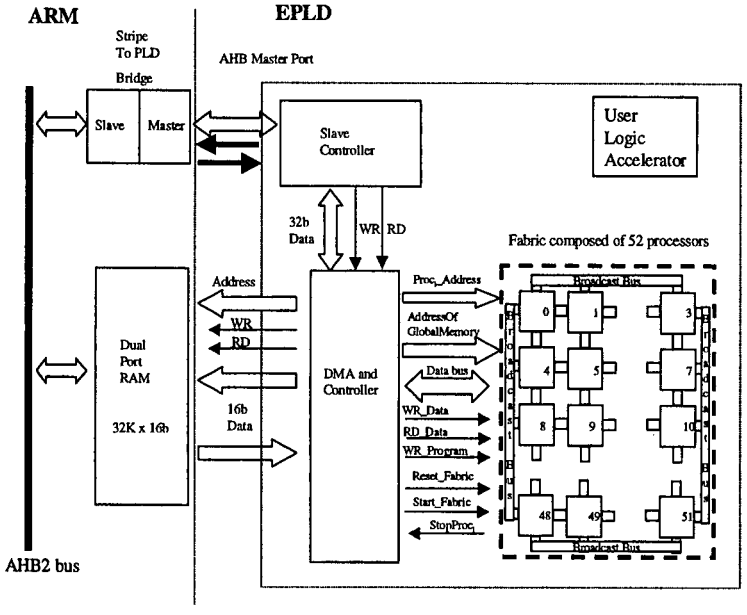


Figure 5. Communication between Fabric and ARM Processor

5. Example Application: FIR Filter Bank

For this application we can implement 50 parallel filters. Two cells out of the 52 are used for storage of the data samples and to implement the delay line of all the filters. This delay line is implemented by using local memory with circular access. Each cell implements a FIR filter, executes all the processing, taking the coefficients from one of the local memories accessed in a circular way and storing the results to another memory using sequential access. Our implementation stores up to 256 samples and results and realizes 256 filter taps. Figure 6 shows a C-program (left side) corresponding to the FIR Filter Bank application and its implementation on the Fabric (right side).

For 256 samples and a bank of 10 six-tap FIR filters the Fabric is 71 times faster than the ARM processor running at 200 MHz.

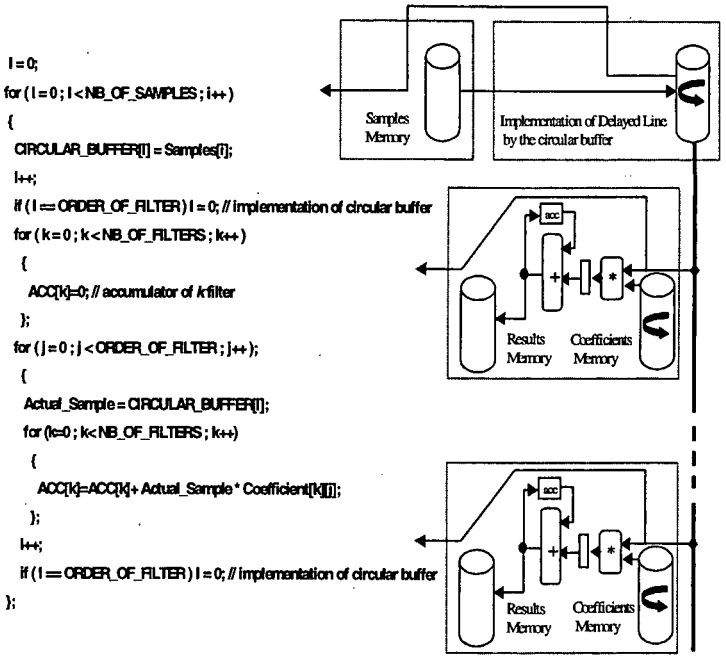


Figure 6. FIR Filter Bank application

6. Experimental results

Many applications are successfully implemented on the Fabric. The table 1 shows only some of them. Each of these applications posed different properties and demand different hardware solutions. The first one, the K-means clustering algorithm needs a large granularity function unit for distance calculation and conditional dynamic configuration for index processing. A bank of FIR Filters needs a large granularity function unit using circular access to memory and broadcast data capability. An N-tap FIR Filter application needs a small granularity function with multiple operands and a systolic flow execution mode. The vector by matrix calculation is a typical DSP application. It needs a large granularity function, systolic communication mode and a direct processor access to a local memory. As shows Table 1 our Fabric gives very good performances for each of this applications. For example, for vector by matrix multiplication the performance [20] is better than PAC 128 recently built circuits.

7. Conclusions

We have described a computing Fabric consisting of a parameterized cellular array connected to a host processor. The parameterized nature of the architecture allows generation of the Fabric for specific classes of applications. This approach is made possible by the modular architecture of the proposed Fabric. Another novel aspect of this Fabric is

Application	Performance measured on cycles	Configuration Average number of 16b words by used cell	Real performance at 33 MHz
K-means Algorithm Clustering Algorithm	{(NB_Bands + 6)*NB_Pixels + NB_Class * 3 + 8} cycles for NB_Bands + 6 >= NB_Class * 3 + 8 NB_Bands <= 256, NB_Class <= 24	16	NB_Pixels = 64, NB_Bands = 144, NB_Class = 8,252 M(ABS and ACC)/s NB_Pixels = 64, NB_Bands = 144, NB_Class = 24 756 M(ABS and ACC)/s
Bank of M, N-tap FIR filters and NB_Samples	{(N + 2) * NB_Samples + 12} cycles for N < = 256, M <= 50	11	NB_Samples = 128 N = 6, M = 50 1.2 GMAC/s
N-tap FIR Filter and NB_Samples	{NB_Samples + 4} cycles for N-tap FIR filter, N <=50, NB_Samples <= 256	6	NB_Samples=128, N = 50 1.6 GMAC/s
Vector[N] * Matrix[N, N]	{ N + 7} cycles for N <= 50	10	N = 50, 1.4 GMAC/s

Table 1. Results

the use of global memory. This memory gives the host processor random access to all variables and instructions on the Fabric's cells. The memory can be initialized in the same way as a standard memory in a computer. Programs and data can be dynamically loaded during processing on the Fabric because the global memory is dual ported. This reduces overhead for preparing the data and programs. The Fabric can reconfigure itself during processing using data generated during Fabric execution. Two Fabric instances using different function units (Section 3.2) have been synthesized on the Excalibur ARM. Each can hold up to 52 cells. The Fabric runs at 33 MHz, giving peak performance of 10.2 GigaOps/s and 1.8 GigaMACs/s. The presented work is an intermediate step in automatic generation of hardware accelerators from high-level specification. We are working on an abstract model describing a polymorphous Fabric and on the automatic generation of a Fabric instantiation and the configuration programs for a specific application.

References

[1] Altera Corporation. Excalibur. http://www.altera.com/products/devices/ excalibur/exc-index.html, 2001.

[2] V. Baumgarte, F. May, et al. Pact xpp - a self-reconfigurable data processing architecture. International Conference on Engineering of Reconfigurable Systems and Algorithms, June 2001.

[3] D. M. Dahle, J. D. Hirschberg, et al. Kestrel: Design of an 8-bit simd parallel processor. 17th Conference on Advanced Research in VLSI, pages 145–162, 1997.

[4] M. Estlick, M. Leeser, J. Szymanski, and J. Theiler. Algorithmic Transformations in the Implementation of K-means Clustering on Reconfigurable Hardware. ACM FPGA 2001, 2001.

[5] M. Gokhale, J. Frigo, K. Mccabe, D. Lavenier, and J. Theiler. Early experience with a hybrid processor: K-means clustering. ERSA 2001, June 2001.

[6] M. Gokhale, B. Holmes, and K. Iobst. The terasys massively parallel processor-in-memory array. IEEE Computer, pages 23–31, Apr. 1995.

[7] C. E. D. C. Green and P. Franklin. RaPiD – reconfigurable pipelined datapath. In R. W. Hartenstein and M. Glesner, editors, Field-Programmable Logic: Smart Applications, New Paradigms, and Compilers. 6th International Workshop on Field-Programmable Logic and Applications, 126–135, Darmstadt, Germany, Sept1996. Springer-Verlag.

[8] S. Hauck, T. Fry, M. Hosler, and J. Kao. The chimaera reconfigurable function unit. IEEE Symposium on FPGAs for Custom Computing Machines, Apr. 1997.

[9] J. R. Hauser and J. Wawrzynek. GARP: A MIPS processor with a reconfigurable coprocessor. In J. Arnold and K. L. Pocek, editors, Proceedings of IEEE Workshop on FPGAs for Custom Computing Machines, Napa, CA, Apr. 1997. To be published.

[10] H. T. Kung and C. E. Leiserson. Algorithms for vlsi processor arrays. In C. Mead and L. Conway, editors, Introduction to VLSI Systems. Addison-Wesley, 1980.

[11] T. Miyamori. A quantitative analysis of reconfigurable coprocessors for multimedia applications. IEEE Symposium on FPGAs for Custom Computing Machines, Apr. 1998.

[12] R. Razdan and M. D. Smith. A high-performance microarchitecture with hardware-programmable functional units. In Proceedings of the 27th Annual International Symposium on Microarchitecture, pages 172–80. IEEE/ACM, Nov. 1994.

[13] C. Rupp et al. The Napa Adaptive Processing Architecture. FCCM 1998, Apr. 1998.

[14] J. von Neumann and A. Burks. Theory of Self-Reproducing Automata. University of Illinois Press, 1966.

[15] J. Vuillemin, P. Bertin, et al. Programmable active memories: Reconfigurable systems come of age. IEEE Transactions on VLSI Systems, 4(1):56–69, Mar. 1996.

[16] E. Waingold, M. Taylor, et al. Baring it all to software:raw machines. IEEE Computer, pages 86–93, sep 1997.

[17] Xilinx Corporation. Virtex/powerpc. http://www.xilinx.com/prs rls/ibmpartner.htm, 2000.

[18] Christophe Wolinski, Maya GoKhale, Kevin McCabe: A Reconfigurable Computing Fabric.The International Conference on Engineering of Reconfigurable Systems and Algorithms June 2002 Las Vegas, Nevada, USA

[19] Maya Gokhale, Jan Frigo, Kevin McCabe, James Theiler, Christophe Wolinski, Dominique Lavenier: Experience with a Hybrid Processor: K-Means Clustering. special issue of the Journal of Supercomputing to be published in 2002

[20] Fredrik Gunnarsson, Christian Hansson, Denis Johnsson, Bertil Svensson: Implementing High Speed Matrix Processing on a Reconfigurable Parallel Dataflow Processor, ERSA'02 June 2002 Las Vegas, Nevada, USA