A Reconfigurable Computing Fabric

Christophe Wolinski
Maya Gokhale
Kevin McCabe
Los Alamos National Laboratory
Los Alamos, NM, U.S.A.

Abstract ***We describe a new cellular array architecture and its implementation on a Configurable System on a Chip (CSOC). In this architecture, an array of computing cells communicate with a conventional processor over a global memory. The architecture is customizable to a class of applications by funtional unit, interconnect, and memory parameters. The architecture is flexible enough to accomodate a variety of parallel processing models including MIMD, SPMD, and systolic flow. The architecture supports dynamic reconfiguration.***

We describe a concrete realization of this architecture on the Altera Excalibur ARM which can deliver 7.6GigaOps/s (8 bit data) and show implementation of a systolic data mining algorithm on the Excalibur-based fabric.

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

1 Introduction

The notion of cellular computation, collective computation by an array of regularly interconnected nodes, has been a recurring theme in computer architecture since the early days of computer science [14]. Ranging from cellular automata to systolic arrays [10], a large body of algorithms and architectures have centered around the notion of a fabric composed of simple automata interacting with immediate neighbors in an n-dimensional mesh.

Fabric-based architectures are particularly attractive for efficient layout in VLSI, and specialized chips have been devised for solving specific computational problems, especially in image and array processing. However, specialized ASICs designed to solve specific instances of problems are not cost-effective, and relatively few of the scores of designs have actually been fabricated (see, eg. [3] for one such successful implementation), with virtually none seeing widespread use.

Recent proposals for fabric-based architectures include [15], [11], [7], [9], [16], and [2]. In this work, we build on design principles embodied in cited proposals, at the same time introducing two key new concepts. Our design consists of a parameterized cellular architecture in which a cellular fabric is connected to a standard microprocessor via a system bus. Each cell in the fabric has a local data memory and a separate program memory. The collection of local memories of all the cells form a global memory. It is possible to make this memory dual-ported so that it can be accessed concurrently by a cell and by the microprocessor. Parameters to the fabric include the number of nodes; the amount of local memory per node; the amount of global memory shared between microprocessor and fabric; the arithmetic unit functionality; dimensionality of the interconnect; and width of the interconnect.

In the next section we discuss related work and features that our design shares with previous proposals. Next we describe our parameterized fabric architecture for which we have created a reference implementation on the Altera Excalibur ARM Configurable System on a Chip (CSOC). The next section shows how a representative algorithm is mapped to the fabric and compares fabric-based performance to “native” performance on the Excalibur ARM. We end with conclusions and future work.

2 Related Work

Fabric-based architecture has been an attractive design point within the reconfigurable computing community. One of the first proposals for a fabric-based architecture using commercial FPGAs was the Programmable Active Memory from DEC Paris Research Laboratory [15]. Based on FPGA technology, a PAM (Programmable Active Memories) is a virtual machine controlled by a standard microprocessor. It can be dynamically configured into a large number of application-specific circuits. PAM introduced the concept of "active" memory. PAM is attached to some high-speed bus of the host computer, like any RAM module. Unlike RAM however, the PAM processes data between write and read instructions (see also [6] for another Processing in Memory approach).

Another important concept is that of bi-directional communication links such as the communication structure proposed by the Remarc project, a fabric-based mesh co-processor [11]. In our architecture, the communication network operates concurrently to cellular computation, allowing full utilization of the cell in the fabric. Programmable datapaths were proposed by RaPid [7] , a linear array of function units composed of 16-bit adders, multipliers and registers connectable through a reconfigurable bus. The logic blocks used are optimized for large computations. They will perform these operations much more quickly (and consume less chip area) than a set of smaller cells connected to form the same type of structure. Like RaPid, our design supports the design of flexible, application-specific datapaths by exploiting a reconfigurable communications network.

Our architecture also incorporates the idea of a configuration cache, such as that presented in GARP [9]. The GARP system had a configuration cache to hold recently stored configurations for rapid reloading (five cycles instead of thousands of processor cycles). Our architecture differs from the GARP in that our configuration cache holds instruction-set programs for the cells rather than FPGA-like configurations. In our system, representative programs range from 20 – 40 bytes in size.

Our architecture provides for runtime reconfiguration, a concept explored by many reconfigurable computing research projects, such as Chimera [8]. This allows the cell program to be modified during execution. In our architecture, the host processor can write to the instruction memory of each cell in the fabric, permitting dynamic reconfiguration at the instruction granularity.

Our system also uses a local microcontroller, giving each node the ability to run a different instruction stream. This concept has been proposed in numerous previous fabric architectures, including the iWarp, and more recently, RAW [16]. The RAW microprocessor chip comprises a set of replicated tiles. Each tile contains a simple RISC processor, a small amount of configurable logic and a portion of memory for instructions and data. Each tile has associated programmable switch which connects the tile in a wide-channel point-to-point interconnect. This architecture is estimated to achieve performance levels 10x-100x over workstations for many applications.

One final concept is the notion of systolic flow execution - that is the ability to flow the data 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. In the PACT architecture [2], a flow graph is automatically mapped to a processing element in the array. Here, the granularity of operation plays a vital role in system performance. PACT utilizes a pre-defined function unit. In our system, function unit granularity is a parameter of the system, allowing function units to be customized for an entire class of applications. In addition, we can accomodate other models such as MIMD or SPMD in addition to systolic flow execution.

3 Fabric Architecture

Our fabric architecture is based on a mesh-connected configurable network of runtime reconfigurable cells connected to a conventional microprocessor via a system bus. The host processor sends the program to the fabric; it can also send data. In addition, each cell can access data in a local memory and in a portion of the global (host-accessible) memory. The host can reset the fabric, send "start" and "stop" signals to the fabric, and can synchronize to any specific cell by testing the cell's internal status bit.

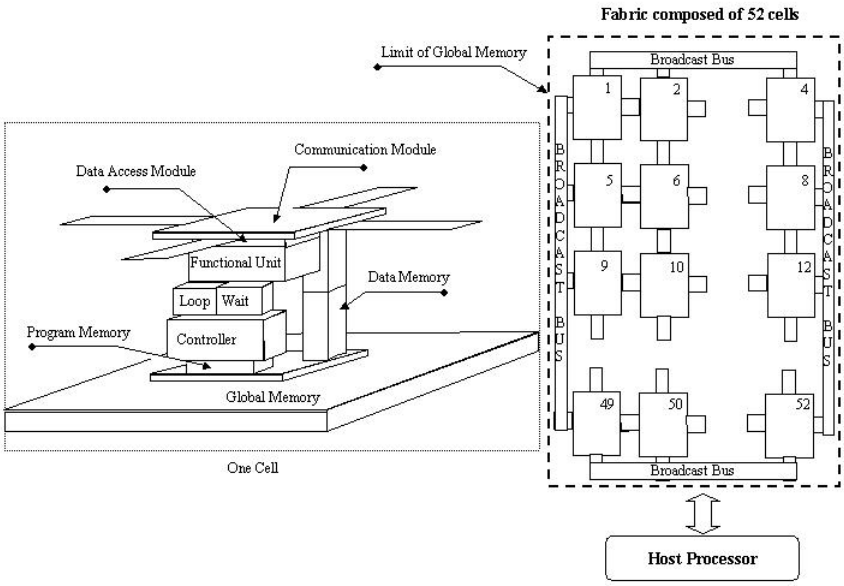

Figure 1. Layout of the Fabric and Cell

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. The layout of the fabric and architecture of a cell is shown in Figure 1.

Certain parameters of the architecture are defined before fabric generation. These include

- the size of the network
- the size of the communication busses
- the choice of functional unit for a given class of applications

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

4 Fabric Implementation

A reference implementation of this computing fabric has been created on the Altera Excalibur ARM CSOC. Our implementation uses 8-bit data paths for communication in a two-dimensional mesh. 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.

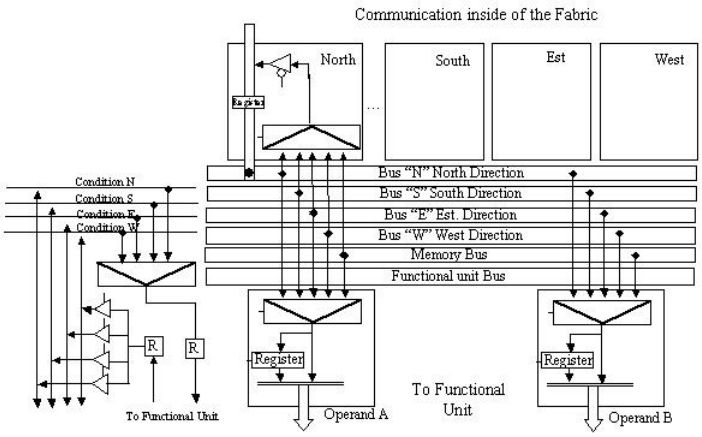

Figure 2. Busses to Access Operands from Neighbor Cells or Memory

North	0	0	0
South	0	0	1
East	0	1	0
West	0	1	1
Memory	1	X	X

Figure 3. Control Register Encoding for Operand Access Control

In addition to "traditional" parallel computing models, the processor/fabric combination enables several unique communication and computation modes. 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. For synchronization, the processor can broadcast a start or stop signal to the fabric, and can monitor any cell's status bit.

Each subsystem is described in detail below.

4.1 Communication and Access Control

The communication module is responsible for receiving data from and sending data to the cell's immediate neighbors in the interconnection mesh. This module interacts with a set of busses associated with nearest neighbor communication; with communication of the "condition" bus; with memory access; and with function unit communication (see Figure 2).

4.2 Function Units

The function unit performs operations on two operands "A" and "B." A three-bit control word to an access control unit determines the source for each operand as shown in Figure 3.

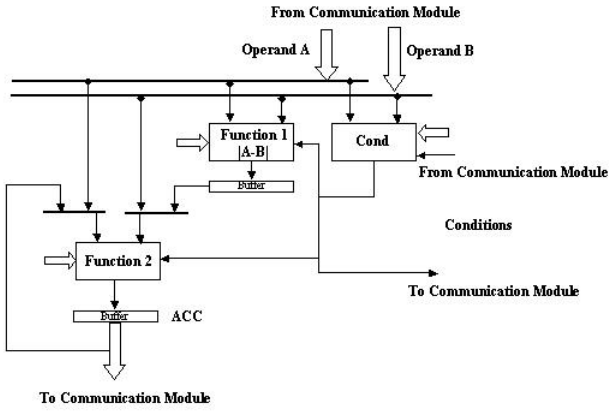

Figure 4. Function Unit 1: Optimized for Distance Calculation

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 two different function units, shown in Figures 4 and 5. Within Function Unit 1, Function 1 performs the absolute value of the difference of the two input operands, while Function 2 can do one of 6 different operations, as shown in Figure 6.

In Function Unit 2, Function 1 performs a multiply, while Function 2 can do one of four different operations, as shown in Figure 7. Function Unit 2 can be configured for efficient DSP filtering operations with a Multiply/Accumulate configuration.

A Condition module compares operands A and B, as shown in Figure 8.

4.3 Instruction Set and Controller

The microcontroller executes instructions from the Program Memory. The instruction set consists of eight instructions.

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, direction of optional memory access, 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 sequential memory 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. This can also be used to synchronize, so that the cell waits the specified number of cycles before continuing with the next instruction.

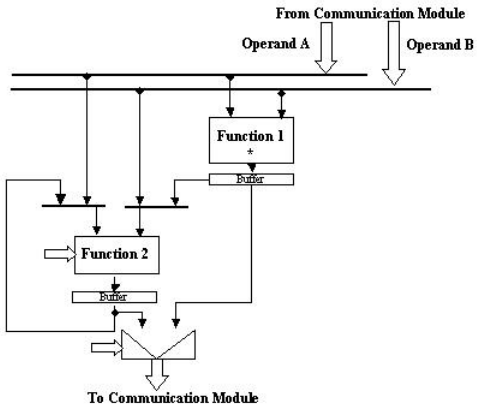

Figure 5. Function Unit 2: Optimized for DSP Operations

ACC = A + B	0	0	0
ACC = A − B	0	0	1
ACC = ACC + FU1-OUT	0	1	0
ACC = if Cond = 1 then A else B	0	1	1
ACC = ACC + B	1	0	0
ACC = B	1	1	X
ACC = B	1	X	1

Figure 6. Function Unit 1:Control Register Encoding for Function 2

ACC = A + B	0	0	0
ACC = A − B	0	0	1
ACC = ACC + FU1-OUT	0	1	0
ACC = ACC + B	1	0	0

Figure 7. Function Unit 2: Control Register Encoding for Function 2

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

Figure 8. Condition Module: Control Register Encoding

Control	1	Upper	Acc	Mem	Rx	Ry	BNe	BSe	BEe	BWe	W	A	I	T			
Reset	0	0	0	0			Rx	Ry	Racc	Rmem	W	A	I	T			
Jump	0	0	0	1						A	D	D	R	E	S	S	
Conditional Jump	0	0	1	0						A	D	D	R	E	S	S	
Configur	0	0	1	1	Function Select			Mdir	WBN	WBS	WBE	WBW	Tne	Tse	Tee	Twe	
	8/16b	B/nB	CTNe	CTSe	CTEe	CTWE	Condition Select				Cperand A			Operand B			
		Comm N Select			Comm S Select			Comm E Select			Comm W Select			Memoty Select			
Start Loop	0	1	0	0							Loop						
End Loop	0	1	0	1						A	D	D	R	E	S	S	
Stop and Wait for Start	0	1	1	0						A	D	D	R	E	S	S	

Figure 9. Instruction Set Encoding

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.

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.

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. The instruction set encoding is shown in Figure 9.

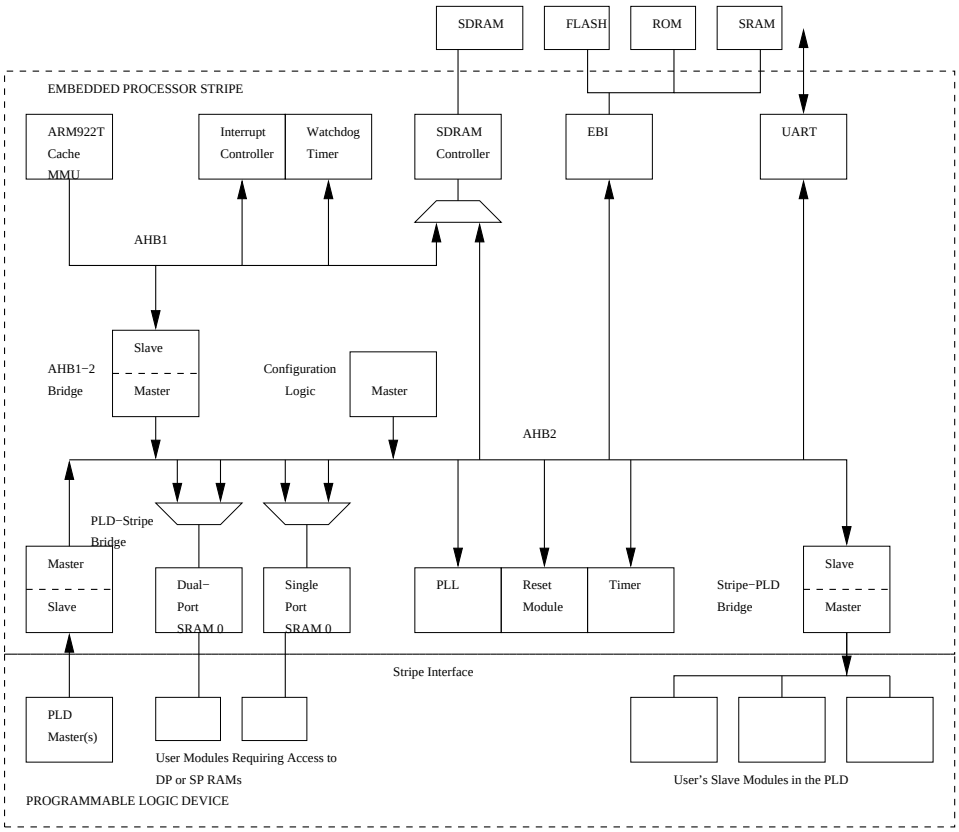

Figure 10. ARM System Diagram

5 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. The ARM portion of the Excalibur system is diagrammed in Figure 10.

In our implementation, the fabric is seen by the processor as a slave module. 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[1], 52 cells using Function Unit 1 were instantiated (13 rows X 4 columns) on the Excalibur ARM EPXA10F1020C2. Synthesis and Place/Route results are summarized in Figure 11. The clock frequency for Function Unit 1 is 29.17 MHz, giving peak performance of 7.6GigaOps/s. The clock frequency for the Function Unit 2 fabric is 30.78MHz, with performance of up to 1.5Giga MAC/s.

[1]Placement directives were generated by a script.

Clock pins	2
Fast i/o pins	0
Global signals	2
I/O pins	37
Logic elements	32131
ESBs	104 / 160 (65 %)
Macrocells	0
Flipflops	8788
FastRow interconnects	0 / 120 (0 %)
Maximum fan-out	3484
Maximum fan-out node	reset
Total fan-out	137997
Average fan-out	4.1
ESB pterm bits used	0 / 327680 (0 %)
ESB non-CAM memory bits used	212992 / 327680 (65 %)
ESB CAM bits used	0 / 327680 (0 %)
Total ESB bits used	212992 / 327680 (65 %)

Figure 11. Physical APEX resources used by Fabric

6 Example Application: Unsupervised Clustering

In this section we describe an algorithm mapped to the computing fabric. This algorithm is a popular data mining technique, unsupervised clustering, also called k-means clustering. The purpose of the algorithm is to classify a data set into a number of classes. Each element of the data set is a vector. This iterative algorithm has the following steps:

1. Randomly assign each data element A[i] to one of k classes
2. Compute the centers of the classes
3. Loop over the data set A
   (a) Let C = class of A[i]
   (b) Determine the class number K which has the minimum distance to C
   (c) if C is not equal to K, move pixel C to Class K
4. Recompute the centers of the classes K and C

A reconfigurable hardware implementation of this algorithm has been reported in [4] and mapping to a hybrid processor (Altera Excalibur NIOS) in [5]. The key compute intensive kernel of this algorithm is distance calculation between a class center and a data element. We use the distance metric suggested by [4], $|A[i] - C[i]|$ to approximate the more traditional Euclidean distance metric.

The fabric implementation of this algorithm parallelizes across classes and uses two cells for each distance calculation (see Figure 12 at cell pairs (4, 3) and (6, 5)). The first cell, located on the fabric periphery, computes the distance using the abs measure. The value for the center is stored in this cell's portion of the global memory, which is pre-initialized by the host. Cells 4 and 6 in Figure 12 perform this calculation. The second cell of the

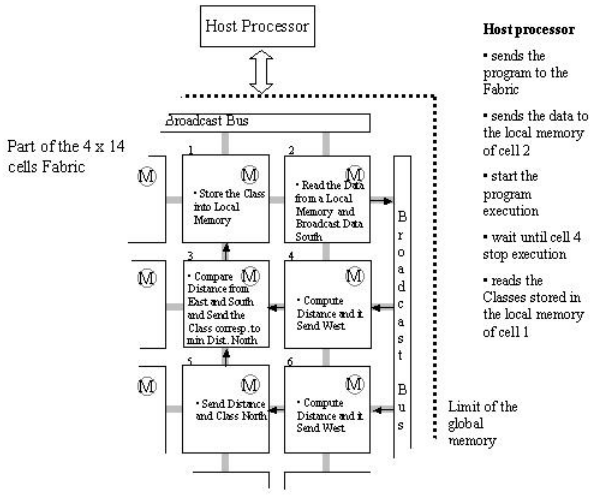

Figure 12. Data Flow and Computation on Cellular Array

pair compares the distance sent from its east neighbor with the distance from its south neighbor and sends the minimum of the two to its north neighbor. It also sends the class number associated with this minimum distance to its north neighbor. Cell 3 does this computation. Cell 5, since it has no south neighbor simply forwards the computed distance along with its class number from Cell 6 to its north neighbor, Cell 3. The host writes a data element into Cell 2, which broadcasts it to all the cells in its column. Cell 1 receives the class number associated with the minimum distance and stores it in local memory, from which location the host reads it back.

In the cellular implementation of this algorithm, there are four different programs: Cells 1 and 2 have very simple programs. Cell 1 reads the data element from local memory and broadcasts to all cells in the outer column. Cell 2 receives the class number from its south neighbor and writes it to local memory. The programs for Cells 1 and 2 have 7 instructions each. Cells 4 and 6 have a program to compute the distance and forward to the their west neighbors. This program has 11 instructions. Cell 5, using 10 instructions, simply routes its distance and class to its north neighbor. Cell 3 computes a minimum and then routes the appropriate center to its north neighbor. In order to do this, it must change its internal configuration to select either the south or east communication bus. This program has 23 instructions.

The time to process a data element of length N (recall each data element is a vector) and number of classes M is given as $\max(N+6, M*3+8)+M*3+8$ clock cycles. Each cell takes 617 Logic Elements and 2 ESBs, so that on the APEX20KC, it is possible to fit 52 cells. In contrast, a direct hardware implementation on the Excalibur ARM takes $\max(N+3, M+1)+M+1$ clock cycles and uses 202 cells and one ESB.

For values of N=224 and M=8, which are representative of hyper-spectral imagery, the fabric and the direct implementation are roughly equal in performance, while the cell uses about 3 times the area of the direct implementation. One advantage of the cell fabric approach is that the algorithm is written purely in software, with no hardware design and synthesis required. Another advantage is that the cells' programs are represented very compactly as compared to FPGA configurations, so that dynamic reconfiguration can be efficiently accomplished. For example, the largest of the cell programs occupies 38 bytes.

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 permits generation of the fabric for specific classes of applications. This approach is made possible by the modular architecture of the proposed fabric.

A novel aspect of this fabric is the use of global memory. This memory gives the host procesor 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 have been synthesized on the Excalibur ARM. Each can hold up to 52 cells. The fabric runs at 29MHz, giving peak performance of 7.6 GigaOps/s and 1.5 GigaMACs/s.

An unsupervised clustering algorithm has been demonstrated on the computing fabric. The implementation shows approximately the same performance as a direct hardware implementation, but holds fewer classes. However the fabric approach requires no hardware synthesis but instead uses a software program.

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*, pages 126–135, Darmstadt, Germany, Sept. 1996. 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.