Scalable Run Time Reconfigurable Architecture

Abdellah Touhafi, Wouter Brissinck and Erik Dirkx
Erasmus Hogeschool Brussel nijverheidskaai 170 1070 Brussel Belgium
Vrije Universiteit Brussel Pleinlaan 2 1050 Brussel Belgium

Abstract: Currently multi-FPGA reconfigurable computing systems are still commonly used for accelerating algorithms. This technology where acceleration is achieved by spatial implementation of an algorithm in reconfigurable hardware has proven to be feasible. However, the best suiting algorithms are those who are very structured, can benefit from deep pipelining and need only local communication resources. Many algorithms can not fulfil the third requirement once the problem size grows and multi-FPGA systems become necessary. In this paper we address the emulation of a run time reconfigurable processor architecture, which scales better for this kind of computing problems.

1. INTRODUCTION

Currently multi-Field Programmable Gate Array (FPGA) reconfigurable computing systems are still commonly used for accelerating algorithms. This technology where acceleration is achieved by spatial implementation of an algorithm in reconfigurable hardware has proven to be feasible. However, research pointed out that the application must fulfil some requirements in order to achieve a high performance. The best suiting algorithms are those who are very structured, can benefit from deep pipelining and need only local communication resources. Many algorithms can not fulfil the third requirement once the problem size grows and multi-FPGA systems become

The original version of this chapter was revised: The copyright line was incorrect. This has been corrected. The Erratum to this chapter is available at DOI: 10.1007/978-0-387-35498-9_57
L. M. Silveira et al. (eds.), *VLSI: Systems on a Chip*

necessary. This paper addresses this problem by presenting a scalable run time reconfigurable processor architecture. The presented architecture uses the fine granular nature of FPGA's to benefit from fine granular parallelism and ads the flexibility of run time reconfiguration together with instruction based programming. An emulator for this architecture is built and tested on its performance. As a case study we have implemented an ATM switch fabric simulator and show that our architecture suits better for large computing problems that require a complex interconnection scheme between successive pipeline stages.

2. MOTIVATION

Multi-FPGA systems can not simply be tiled ever larger without considering hughe latencies for wires traversing many FPGA's. The problem encountered here is comparable to interconnect problems encountered in parallel and distributed computing systems. Amdahl pointed out the existence of this problem for distributed computing systems and has shown that there is an upper bound to achievable performance gain by simply adding more computing resources. When using connected FPGA's as processing elements, the problem is even more emerging since the number of needed processing elements is a function of the size (complexity) of the problem. This implies that large computing structures with complex interconnection requirement will not always achieve a huge performance gain. Researchers in the reconfigurable computing society have remarked this effect [1] [2]. In figure 1 this effect is computed for a very simple computation model where we suppose that the communication cost scales linearly with the number of nodes.

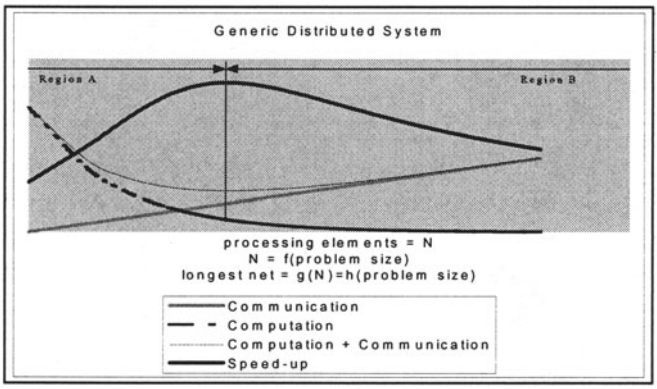

Figure 1. Generic distributed system

For pipelined systems a comparable graph will be achieved when considering the interconnected processing elements between two successive pipeline stages. If the spatial implementation of an application is located in Region A it is clear that there is potential to achieve faster implementations by exploiting more spatial resources. However, once an application enters region B it suffers from huge latency and throughput limitation resulting in larger execution times than would be possible in the ideal case. For applications located in region B a few approaches can be considered to enhance the performance. A first approach could be to provide more and faster routing resources (hierarchic)[3], a second approach is to make the basic computation cell larger [4] and a third approach is to provide a resource sharing mechanism in order to shorten virtually the communication lines. The three approaches are comparable in the sense that they all try to change the granularity of the computing system such that it fits better the application.

2.1 Computational model Targeted

A lot of research in the field of reconfigurable architectures targets solutions for general purpose computing[5][6]. Our research is more pushed by the need for scalable computing systems for high performance computing problems. As an example, we will address the simulation of large ATM switching fabrics [7] through this paper. NP complete problems like the Boolean Satisfiability problem [8] and Neural network implementations are also good candidates for the presented architecture.

The computational model targeted is based on three kinds of components. The first component is the source that provides input data to be computed. The second component is a process that executes some computation on the provided data and the third component is the sink that gathers the results. A computational problem can then be represented by a directed graph G(V,E) with vertices V and nodes E. The vertices represent communication channels and nodes are processing elements. Each processing element is characterised by its computation time and the number of input and output channels attached to it. We further suppose that each node has a "circuit" implementation, which can be mapped on runtime reconfigurable hardware. A node that represents a circuit is called a context. Independent nodes are organised in successive pipelined stages interconnected by unidirectional communication links. The nodes of a graph can then be interpreted as being contexts

and the vertices are context dependencies. A context can not be loaded for execution on a runtime reconfigurable device as long as all the other contexts on which it depends have not been processed yet.

2.2 Spatial Implementation

Given a directed graph of contexts it would be possible to implement its circuit equivalent completely in space using a multi-FPGA system. Such an approach has proven to be feasible for designs with deep pipelines. However once the interconnect between two pipeline stages becomes complex and nets between the different pipeline layers must traverse different FPGA's the well known Amdahl's bottleneck starts playing an important role. The accumulated net-delays will cause a frequency drop for the complete system. Although more gates are being evaluated in parallel for each clock cycle, the effect of the frequency drop can cause a net performance loss (i.e. less gates are being processed per time unit).

2.3 Temporal/Spatial Implementation

It is clear that other solutions must be searched for to keep gaining performance from scalable systems. The one proposed in this paper is the use of scalable (run time) dynamically reconfigurable processors with buffered communication between contexts. Runtime reconfiguration is used as a mean to make the circuit virtually less fine granular by putting some sequenciality in it. Run time reconfiguration is also used to virtually shorten the distance between computing nodes. With other words we start from a very fine granular graph specification which has a maximum amount of parallelism but at the same time can cause serious frequency drops due to accumulated net delay. From there on the fine granular nodes of the directed graph are grouped into larger nodes where the communication between the fine granular nodes is not visible anymore to the outside part of the directed graph. Further, the larger nodes are time-multiplexed on the available run time reconfigurable hardware, taking as a cost function to be minimised: Accumulated (communication time + reconfiguration time +computation time).

2.4 Buffered communication channels

Run time reconfigurable architectures are very sensitive to reconfiguration time. Some approaches proposed in [10] are based on the use of configuration prefetching, partial reconfiguration and compression techniques. Such approaches require rather difficult algorithms to be executed at runtime. Other approaches try to implement FPGA's with very fast context switch possibility [9]. For the computational model considered, a frequent reconfiguration scheme can be avoided by the use of buffered communication links. Other presented run time reconfigurable architectures

[9,12] do not provide- or provide only a very limited amount of memory cells to pass partial evaluated results from one configuration to another. This restriction has as effect that frequent reconfigurations are always necessary.

2.5 Cycle hiding

Avoiding frequent reconfigurations as explained in the previous subsection requires that communication links are not feedback. A possible solution to that restriction is to implement those contexts which contain feedback links in space as if they where one context. This requires that the reconfiguration of the computing nodes, which are used for the spatial implementation, can be synchronised with neighbour computing nodes. Whether a reconfigurable computing node is synchronised or not, with its neighbour nodes must be evaluated at each reconfiguration.

3. SUReCA: A SCALABLE ARCHITECTURE

In this paper, we introduce a processor with a scalable runtime reconfigurable computing layer and a scalable control layer, which gives a better base to deal with the computational model targeted. The SUReCA Architecture is organised as shown in figure 2.

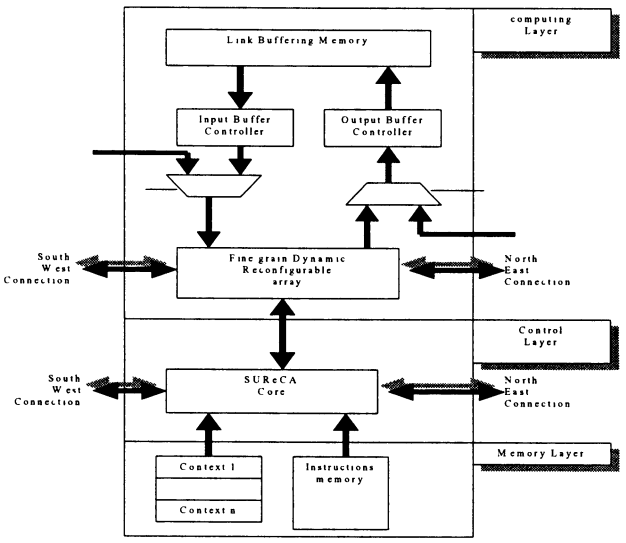

Figure 2. Scalable Uniform Reconfigurable Computing node Architecture

It consists of a computing layer, a control layer and a memory layer. A SUReCA node has communication links on the computing layer and on the control layer. Interconnected SUReCA nodes can then benefit from a large computing layer with a distributed tightly interconnected core. The virtual architecture layer contains the processor core that has full control over the processor resources.

3.1 Computing Layer

The computing layer contains a fine grain dynamic reconfigurable array like FPGA's. Further there is a fast 4 Mbytes memory and some memory control circuitry. The dynamic (run time) reconfigurable array is used as the computing medium. Different circuit contexts are being loaded sequentially onto the dynamic reconfigurable array, where they are activated for a certain number of clock cycles. The fast memory is used to pass partial results from one context to another and is organised as a set of FIFO buffers. For every circuit context their is an associated number of input and output FIFO buffers with each FIFO buffer being a communication link between two or more circuit contexts. The programmer has control over allocating buffer space and associating FIFO buffers with contexts. Memory management to handle this association is in hands of the input- and output buffer controller.

It is possible that contexts running on different SUReCA nodes share the same FIFO buffer. A multiplexer is used such that the local memory of a node can be connected to its neighbouring computing nodes. The flexibility of having such runtime routable memory can help to manage a better overlap between communication and computation.

The dynamic reconfigurable layer has near neighbour interconnections giving the ability to form a large fine granular tightly coupled computing plane. Reconfiguration of this large computing plane is synchronised and tested for I/O consistency on each reconfiguration cycle. Testing for I/O consistency is of major importance because inconsistent I/O connections can cause physical damage. When an I/O inconsistency is detected SUReCA nodes involved keep their I/O pins in high impedance state.

3.2 Control Layer

This layer forms the core of our architecture. The internal organisation of the core is given in figure 3. It is composed of an instruction decoder, a context loader, a link controller, and a condition evaluator and synchronisation circuitry. The control layer is responsible for the correct execution of the computation. The parts of the control layer are further explained in more detail.

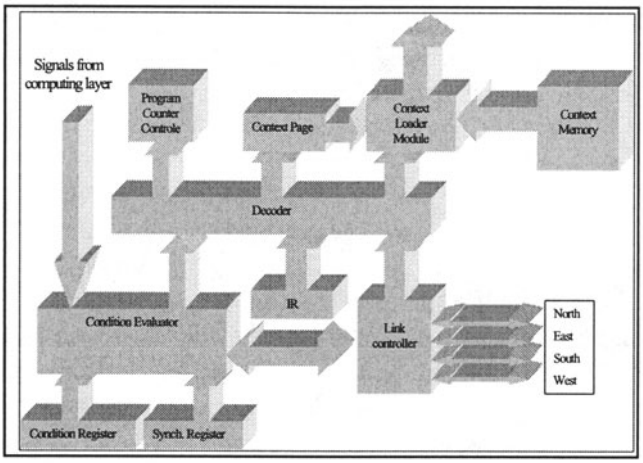

Figure 3. Sureca Core

3.2.1 Instruction Decoder

A SUReCA program is built up out of instructions and circuit contexts. The instruction set contains context-move instructions, memory management instructions, jump instructions, local resource initialisation instructions, and synchronisation and communication instructions. The instruction decoder controls the sequencing of the instructions and the circuit contexts. It also sets up the input buffer controller, and output buffer controller and link controller. The decoder is implemented as a state machine that runs in three phases. During the first phase an instruction is fetched. The second phase is used to decode the instruction and increment the program counter. The third phase is the execute phase of which the required number of clock cycles can vary between one and a few thousand depending on the instruction. In case of a context load instruction the next instructions will be already fetched decoded and eventually executed. If the instruction is a synchronisation, communication, initialisation or jump instruction it is executed. This way set up of the buffer controllers and synchronisation with the neighbour nodes can be overlapped with the ongoing context move instruction.

3.2.2 Context Loader

The context loader receives the context ID to be loaded from the instruction decoder. Before loading a new context into the dynamic reconfigurable layer all I/O pins are put into high impedance state to avoid inconsistent I/O set up. It then checks whether the context ID is the same as

the one, which is already loaded. In case they are different the required context is loaded from the context memory. The next step is to initialise the state registers of the loaded context.

3.2.3 Link Controller

The link controller is a smart mesh router that provides a programming interface to the chip and at the same time gives the ability to send and receive synchronisation information from its neighbour nodes. The first byte of an incoming data-stream is an address. The address is composed of a node number and a local resource address. If the receiving node recognises its node number and its local resource, the data stream is sent to that location. The parts that can be accessed by the link controller are the memory layer and the FIFO buffers in the computing layer. This has as a nice side effect that certain SUReCA nodes can reprogram or simply change initialisation values of other SUReCA nodes. A possible application for this are online trained neural network implementations where some SUReCA nodes implement the forward pass while other nodes contain the back propagation algorithm which send the newly computed neural network node weights to the appropriate neural network nodes.

3.2.4 Condition Evaluator

The evaluation of instructions can be blocked until a certain condition or a set of conditions is met. This blocking gives the possibility to perform a conditional context switch. Hence, a context switch can be performed because of many reasons. In the best case a context switch is done because the context has finished all available input data. In other cases the reason for a context switch can be that the output FIFO buffers are full. In other cases again the active context can have reached a state in which itself asks for a context switch. The condition evaluator gives the possibility to set up the required context switch conditions. It generates a "condition-met" signal when the required conditions are met. The instruction decoder then relieves the block and starts with a new fetch cycle. The condition evaluator receives signals from the dynamically reconfigurable layer, from the instruction decoder, the context loader, and the synchronisation controller and memory buffer.

3.2.5 Synchronisation

As proposed in the motivation, for circuits with feedback channels, a spatial implementation is still the best solution because reconfiguration cost

would be overwhelming. Supporting tight multi node implementations requires synchronisation with the involved nodes when working with runtime reconfigurable systems. The synchronisation control has the task to deal with this synchronisation. As can be seen in figure3, the synchronisation controller receives synchronisation information from the neighbour nodes and from the link controller. The synchronisation can be overlapped with the ongoing context loading. A synchronisation instruction is available to specify with which neighbours synchronisation is necessary.

4. SUReCA EMULATOR PLATFORM

An emulation system for the presented architecture is built and tested on performance. As shown in figure 4 our emulator is built up from of the shelf FPGA's, DPGA's, memory and connectors. Each emulation board can contain four SUReCA nodes, one master node, connectors in north, east, south and west direction and a parallel port interface to program the emulator. Two-dimensional computing arrays or a torus can be built by interconnecting emulator boards. The master controller contains a state machine that interprets incoming data streams from the hosts' parallel port; it is also the programming interface towards the four computing nodes. The master nodes of connected emulation boards can also interchange messages. Each board has a unique address, set up by external jumpers. The SUReCA node on the emulator is implemented concordant with the architectural description. For the implementation of the dynamic reconfigurable layer we've used an XC6264-2 FPGA from Xilinx. The core and buffer controllers are implemented using an XC4013E-2 FPGA also from Xilinx. We've further foreseen 4 Megabytes of 12 ns RAM organised as 1024K x 32 bits. The RAM organisation can be changed according to the requirements of the application. The board has a total of 16 Mbytes of fast static RAM divided over the four SUReCA nodes. This RAM is used as context memory, instruction memory and FIFO buffer memory.

A 16-bit mesh interconnects the computing layers of the four SUReCA nodes. To overcome the rather small bus-width we've foreseen 4 multiplex signals such that the mesh can be virtually 256 bits wide. An eight bit wide mesh interconnects the control layer. The emulation board has two clocks. A fixed 20 MHz clock for the control layer and a programmable clock for the dynamic reconfigurable layer. The clock speed can be changed at runtime on request of a SUReCA node. This is possible because each SUReCA node has access to the master node through the control layer. A program can send data to the master node requesting a certain clock period for the context that is

going to be loaded. The master node has not only control over the programmable clock but also over the multiplexing signals.

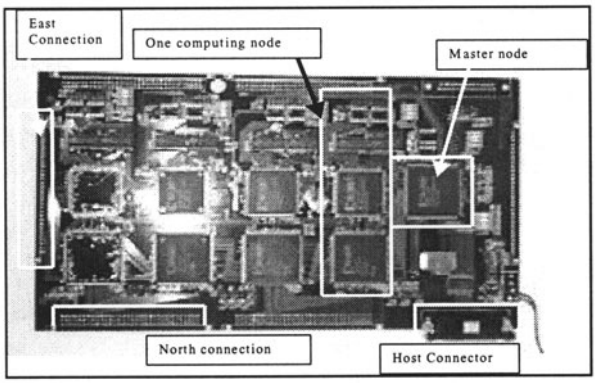

Figure 4. Emulator

Implementing large two or tree dimensional multi chip computing systems, which are modular and scalable with one global clock, such that all computing nodes can be synchronised is hard to realise. This due to clock skew appearing over connectors (Capacitive load) and clock power distribution (only limited fan-out is possible which would require buffering of the clock signal). An approach to deal with this restriction is the use of self timed modules. To enable such an approach, each computing node uses a local clock, for inter-node communication an asynchronous handshake protocol is used. Every board has one programmable clock distributed over the four SUReCA nodes. Although they have the same clock we have foreseen an asynchronous interfacing between the four computing nodes on the control layer.

5. EXAMPLES AND PERFORMANCE ISSUES

In the following example we compare the implementation of a high level simulator for a multistage ATM switching fabric on a mesh of FPGA's and an implementation on our emulator. Multistage ATM switches [7] are conceptually based on multistage interconnection networks (MIN). The switch architecture is built by using small shared memory switches as switching elements (SE) which are arranged in a MIN. The simulator built is that of an internally blocking switch of which the used MIN is classified as a Banyan network. The simulator for the ATM switch fabric is based on the hardware implementation of the queuing model of a switching element [11].

The following performance results where achieved for different problem sizes. The performance is given as the simulation time divided by the number of ATM packets handled. The simulations done handled 32.10^6 packets. We have chosen this number to make the computation time large enough, such that the set up and reconfiguration cost of the SUReCA system can be better amortised over the whole computation time.

To have comparable results we've used for the FPGA implementation the XC6264 but without reconfiguring it at runtime.

Problem size	FPGA's/ SUReCA nodes	Performance FPGA's	Performance SUReCA
16x16 switch	1 FPGA 1 SUReCA	3.75 ns/packet	3.77 ns/packet
64x64 switch	6 FPGA's 3 SUReCA	1.04 ns/packet	2.7 ns/packet
256x256 switch	40 FPGA's 20 SUReCA	0.91 ns/packet	0.85ns/packet

Interesting in these results is to see that the SUReCA system scales different from the meshed FPGA system. For the small problem size we have less performant results on the SUReCA system because only the half of the available parallelism is used. Once the size of the problem becomes large enough we remark that a better performance can be achieved on a SUReCA based system with less reconfigurable hardware. This due to the fact that the communication cost overwhelms the benefit of more fine grain parallelism in meshed FPGA systems.

6. CONCLUSIONS

We've tried to show that better performance can be achieved than the one obtained on meshed FPGA systems for large computing problems with highly complex interconnection requirements between successive pipeline stages. A processor architecture that gives a better scaling of the performance is discussed and an emulation system is built. The presented processor architecture consists of three layers and is programmed by means of instructions and circuit contexts. The first layer is a computing layer which contains an FPGA like run time reconfigurable compute medium and a FIFO buffering mechanism to exchange data between contexts. The input and output port of the FIFO buffer can be connected with the neighbour nodes, giving the possibility to have data exchanged between contexts ran at different moments on different computing nodes. The buffering mechanism

is also used as a mean to avoid frequent reconfigurations. The second layer is the control layer, which next to decoding instructions and controlling all the local resources also implements a synchronisation mechanism with the neighbour computing nodes. An emulation system for the presented architecture is built and tested out on performance. The used dynamically reconfigurable FPGA needs several ms for a complete reconfiguration. This due to its size (256x256 cells) and the rather slow reconfiguration interface (approximately 5 MHz). Although, the use of a deep FIFO-memory makes it possible to have the reconfiguration cost minimal. However note that this buffering technique is not applicable for general purpose computing systems. As a case study an ATM switching fabric is simulated on a meshed FPGA system and our emulation system. For those problems that are rather small the meshed FPGA system performs far better than the Emulator does. Once the problem size is large enough our emulator starts performing better.

7. REFERENCES

[1]Jonathan Babb et al, "Solving Graph problems with dynamic computational structures",SPIE photonics East: Reconfigurable Technology for Rapid Product Development and Computing, Boston,MA, November 1996

[2]Jonathan et al, ,"The RAW Benchmark suite: Computational Structures for General Purpose Computing", IEEE symposium on field programmable Custom Computing Machines

[3] M. Sliman Kadi et al, "A fast FPGA prototyping system that uses inexpensive high performance FPIC", in proc. ACM/SIGDA Workshop Field Programmable Gate Arrays,1994

[4] Elliot Waingold et al,"Baring it all to software: The RAW machine", MIT/LCS Technical Report TR-709, March 1997

[5] Andre DeHon "Reconfigurable architectures for General Purpose Computing",MIT artificial Intelligence Laboratory, Technica Report 1586, september 1996

[6]Scott Hauck,"The Chimaera Reconfigurable Functional Unit",IEEE symposium for custom computing machines, 1997

[7] Martin De Prycker, Asynchronous Transfer Mode, Solution for Broad Band ISDN, ISBN 0-13-342171-6

[8] Zhong P. et al.,"Solving boolean satisfiability problems with Dynamic Hardware Configurations",,8th international workshop,FPL'98 Tallin Estonia, September 1998, Proceedings

[9]Steve Trimberger et al, "A Time Multiplexed FPGA",IEEE symposium on Field Programmable Custom Computing Machines, FCCM 1997, NAPA, CA, april 1997,Proceedings

[10]Scott Hauck, "Configuration Compression for the xilinx xc6200", IEEE symposium on Field Programmable Custom Computing Machines, FCCM 1998

[11]Abdellah Touhafi et al, Simulation of ATM switches using Dynamically Reconfigurable FPGA's, 8th international workshop,FPL'98 Tallin Estonia, September 1998, Proceedings