Is There Any Chance for Reconfigurable Processors Today?

Nelson Ezequiel Ferreira Nunes

Dep. Informática, Universidade do Minho
4710-057 Braga, Portugal
Mi5360@di.uminho.pt

Abstract: Reconfigurable processors are an emerging alternative to the increasing complex and fixed ISA (Instruction Set Architecture) approach: they contain hardware resources that are not pre-allocated to the execution of pre-defined functions, but instead can be configured with some logic commands. This communication shows the main differences between the traditional processor architecture and the reconfigurable processor, with some reference to their internal organization, performances, compatibility and to companies that are pursuing this novel approach. This communication also points out the motivations behind the current availability of commercial reconfigurable processors.

1 Introduction

Recently, the idea of using reconfigurable resources along with a conventional processor has lead to research into the area of reconfigurable computing and by this way, the concept of a *Reconfigurable Processor* comes from the idea of having a general-purpose processor coupled with some reconfigurable resources that allow implementation of custom application with some specific instructions.

Modern application programs often contain hundreds of thousands of lines of code. Operating systems are even more complex: Microsoft Windows 95 contains about 10 million lines, most of it written in C, and Windows NT contains more than 5 million lines written in C and C++. Imagine translating a million lines of C code into a set of instructions from 1 to perhaps 20 or 30 instructions each, and it is easy to see why software today is so complicated, and so difficult to debug [1].

The main goal is to take advantage of the reconfigurable processors improving is performance by migrating certain instructions of an application into reconfigurable hardware. While the processor takes care of all the general-purpose computation, the reconfigurable hardware acts as a specialized coprocessor that takes care of specialized applications. With such platforms, specific properties of applications, such as parallelism, regularity of computation, and data granularity can be exploited by creating custom operators, pipelines, and interconnection pathways. By allowing the programmer to change the hardware of the machine to match the task at hand, computational efficiency can be increased and an improvement in performance realized.

Following these lines, this communication presents a brief introduction on software and current compilation techniques that support reconfigurable processor architectures. It describes current programming models and compilation techniques that allow circuit implementation of instructions, as well as programmable logic configuration management issues. Finally, it presents a brief description of related university projects and currently available industry reconfigurable processor products.

A reconfigurable processor has the characteristic of being able to change its functionality at run-time according to the current application. This characteristic is called Run-time Reconfiguration and it is implemented in some chips that use a Dynamically Programmable Gate Array (DPGA) or Multi-Context FPGA. That is why FPGA is also referenced in this communication. But first, in the next section, it is presented some main concepts of Reconfigurable Processor Architecture.

2 Reconfigurable Processors Architecture

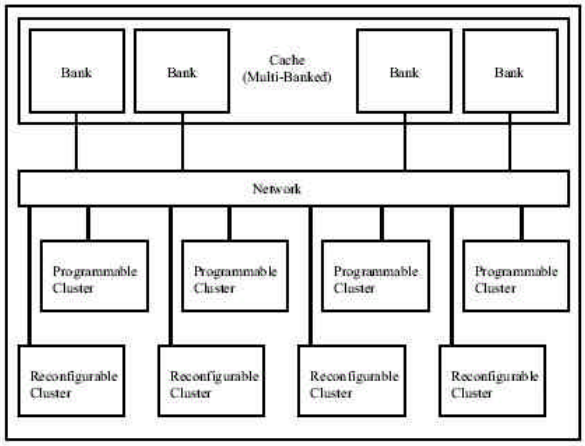

Figure 1 – A Reconfigurable Processor Architecture [2]

When measuring the performance of a processor, we could consider two things: execution time, which is the elapsed time between the start and the completion of a task; or throughput, which is how much work can be done in a given period.

Execution time is affected by three major factors: Instructions per program, clock cycles per instruction, and seconds per clock cycle as shown on the equation below. Sometimes by improving one of these, the others may be affected.

$$\text{Execution time} = \frac{\text{Instructions}}{\text{Program}} \times \frac{\text{Clock cycles}}{\text{Instruction}} \times \frac{\text{Seconds}}{\text{Clock cycle}} \quad \textbf{(1)}$$

For the purpose of this communication, the last factor is not a major concern because the clock rate is related to the hardware technology rather than the architecture.

Only the first two will be taken into account and so execution time measured in clock cycles per program. By reducing the execution time of a program, we are improving the performance of a system. The program will take less time to execute, so we say that we are obtaining “speedup”.

If we consider the ‘Execution time’ as a metric for measuring performance, we are concerned about reducing it to get better performance. Several ways to accomplish this will be briefly pointed.

There are several techniques used nowadays to speed up processors: pipelining, superscalar pipelining, dynamic scheduling and very long instruction word (VLIW) packing. The purpose is to take advantage of a potential execution overlap among independent instructions. This characteristic is called instruction-level parallelism (ILP).

In the next sections will be presented the main contributions to the appearance of a reconfigurable processor, it also be pointed out some reconfigurable processors architectures (See Section 4), and also some other reference points.

3 RISC Architectures

Now will be referred the development of processors but with focus to RISC processors, because the reconfigurable processors can not be limited like processors were, and to show the configuration flexibility that this kind of processors must have in order to maintain compatibility to every programs that were first created to work with the older Processors architecture.

When a program runs, the microprocessor reads, or *fetches*, the instructions one by one and executes them. RISC's original goal was to limit the number of instructions on the chip so that each could be allocated enough transistors to make it execute in one clock cycle.
But today's RISC chips often have richer and more complex instruction sets than CISC chips, with all the techniques for better performance referred above, like pipelining.

Something important to refer in the RISC architecture is that it uses uniform instruction lengths. On a Pentium, the length of one instruction varies. RISC designers, on the other hand, made all instructions the same length, usually 32 bits. This simplifies the instruction fetching and decoding logic and also means an entire instruction could be retrieved with one 32-bit memory access. It does not change the length to get the best performance.

In the RISC and CISC processors, the main difference between them are not the instruction sets, but of chip architectures. The designations RISC and CISC are no longer meaningful in the original sense; what count is how fast a chip can execute the instructions it is given and how well it runs existing software, and that's the purpose of the Reconfigurable Processors. These days, both RISC and CISC manufacturers are pulling out all the stops to get an edge on the competition. And that's why now is appearing the concept of Reconfigurable Computing, and so the concept of Reconfigurable Processors is gaining so much "adepts". And with this, the FPGA's, that will be referred next because they have a huge importance in the appearance of Reconfigurable Processors.

4 FPGA (Field Programmable Gate Array)

The Reconfigurable Processors have some reconfigurable hardware, but how it works? This is a subject that will be tried to be elucidated.

First lets describe an FPGA, a FPGA (Field-Programmable Gate Array) is a general-purpose, multi-level, programmable device with a very high logic density; it allows the user to implement logic circuits in a very short period. It consists of an array of unconnected logic blocks that can be connected by some interconnection resources. FPGA is used to refer to any form of reconfigurable hardware.

FPGA reconfigurable systems are implemented using arrays of reconfigurable processing elements such as the one shown below:

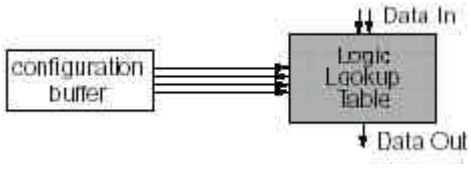

Figure 2 - Reconfigurable Processing Element [3]

The logic blocks are the building blocks for implementing a circuit in the FPGA. They consist of some logic gates, multiplexers, look-up tables, flip-flops or any other logic used

for implementing circuits. The structure and content of a logic block is what defines the FPGA architecture.

Programming elements controls some inputs in the logic blocks. These can be implemented with fuses, anti-fuses, EPROM or EEPROM transistors, or static RAM (SRAM) cells. The implementation used is what defines the FPGA programming technology. The SRAM programming technology is the one most commonly used by FPGA companies. These cells are referred to as configuration memory cells, which are distributed among the logic they control.

In the discussion of the architecture many points are can be analyzed, for example, although the chip area required by each SRAM cell is larger than the other implementations because of the number of transistors needed, the advantage of it is that it can be configured very quickly gaining more performance.

One variation of an SRAM-based FPGA that some researchers have looked into is a DPGA (Dynamically Programmable Gate Array), also known as Multiple-Context FPGA. This architecture can have more than one configuration cell multiplexed for each programmable element. The cells are arranged in a manner that they provide other new sets of configuration cells, which are known as a *configuration planes* or *contexts*. One configuration plane will be active at a time by selecting it through the multiplexors, and so defining the current configuration of the FPGA. This allows a quick context switch between configurations in the FPGA.

DPGAs also provide a desired characteristic for reconfigurable computing which is form of dynamic reconfiguration. At the same time that one configuration is active and executing, other configurations for the cells can be loaded or modified independently in the background without affecting the rest of the configurations like it is suppose to work with the reconfigurable processors.

The reprogramming capability of the SRAM-based FPGAs is what motivates their use in reconfigurable computing. By using this technology, we can build a compute engine in hardware to do some specialized computations. The goal is to have a shorter processing time with the hardware implementation than by executing a sequence of instructions on a general-purpose engine. In general, by coupling a general purpose processor with reconfigurable hardware, one could take advantage of the capabilities and features of both. While the processor takes care of all the general-purpose computation, the reconfigurable hardware acts as a specialized coprocessor that takes care of specialized applications.

But there are several ways in which an FPGA can be coupled to a CPU, as seen below.

Stand-alone Processor. Systems in this category are the most loosely coupled, where the FPGA acts as a stand-alone processor. This integration requires the FPGA to communicate with the CPU through an I/O interface. Because the communication though I/O interfaces is relatively slow, this integration is useful only on systems where the communication occurs with a very low frequency.

Attached Processor. In this category, the FPGA acts as an attached reconfigurable processing unit. It behaves as an additional processor in a multi-processor system. The communication between the FPGA and the CPU is done over a bus in the same way processors communicate in a multi-processor system.

Coprocessor. The FPGA can also be used as a coprocessor that aids the processor with certain computations. This allows the FPGA to do big amounts of computations in parallel with the CPU, as well as getting access to its memory resources.

Reconfigurable Functional Unit – RFU. Finally, the FPGA can be integrated into the CPU as a Reconfigurable Functional Unit.
This unit can be located inside the processor pipeline in parallel with the existing functional units of the CPU and has access to the processor's register file. This allows dynamic addition of application-specific instructions to the existing instruction set.

5 Compilers

Research in the area of reconfigurable computing has shown the performance benefits of using reconfigurable hardware for doing computations. However, there is a need of a software environment that lets a programmer use the reconfigurable resources without requiring much knowledge of the underlying configurable hardware.

The programmer usually uses a High-level Programming Language (HPL), such as C/C++, Java or any other, and the compiler takes care of the translation and generation of the correct machine code for the architecture. These compilers are specialized tools that take advantage of the CPU architecture they are designed to target.

There is a wide range of ways one compiler can program a reconfigurable system. On one side there is a manual implementation of a program and adaptation to the processor as well as the software running on it. In this approach, the knowledge required of both hardware and software is high. On the other end, there is a concept of an automatic compilation that detects pieces of code suitable for implementation in a custom reconfigurable hardware, which makes the adaptation transparent to a programmer, and for doing this the compiler must have some 'Executions Statistics' in order to get better performances from the processor, like is shown in the Figure 3.

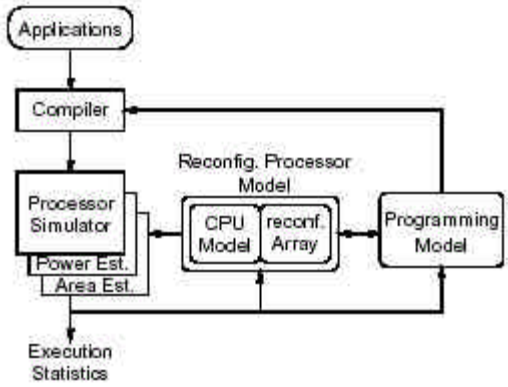

Figure 3–Simple Scalar Design methodology [3]

As been said, one of the major goals in reconfigurable computing is to adapt the hardware into a high level programming abstraction. Automatic compilation systems should be able to automatically analyze and detect which parts of an HPL abstraction of a program are suitable for implementing in hardware. Also, compiler technology may be used for exposing parallelism in loops with techniques such as loop unrolling, pipelining and trace scheduling depending on the processor capacities.

An example of a C-Compiler Technology for Reconfigurable Processors that is on the market is from the PACT Corporation (*http://www.pactcorp.com*). This compiler allows programmer to use existing C code and work in familiar development environments with virtually zero learning curve to program PACT parallel coprocessor for DSP/CPUs, using ANSI C, the industry-leading sequential software language.

PACT Corporation is a semiconductor and intellectual property vendor that has developed a wave-reconfigurable architecture that combines the performance of an ASIC with the flexibility of a DSP. PACT has developed a 32-bit processor core as a first implementation on the company's eXtreme Processor Platform (XPP™), which has demonstrated performance up to 50X greater than conventional sequential processors and 20X higher than DSPs. PACT provides XPP cores that can be easily tailored for next-generation mobile telephones, base stations, workstations and other high-performance devices.

There are others Compilers, witch some are referred in the next section.

6 Reconfigurable Processors Institutions

In this section will be referred some University Projects and some companies that have developed and implemented configurable processors.

In each subject will have a small introduction to the main architecture characteristics of each reconfigurable processor and is performance.

6.1 University Projects

Chimaera [5]. Chimaera was developed at Northwestern University. It is a reconfigurable array that is integrated to a processor as a Reconfigurable Functional Unit. The array has access to shadow registers and consists only of configurable combinational logic.

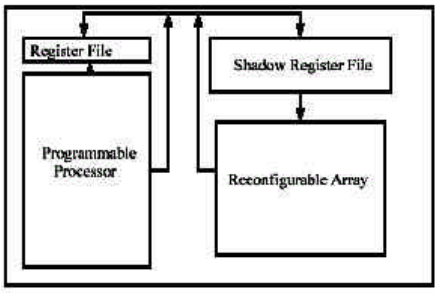

Figure 4 - Chimaera Architecture [2]

Reconfigurable array treated as customizable ALU. Configurable to implement 9-input 1-output operations. Compiler identifies common instruction sequences, defines as RFUOPs.. Compiler schedules RFUOPs like normal instructions

With this processor is possible to get an average of 21% improvement on media benchmarks over aggressive conventional processor [2]. To point that in this architecture is not exploit parallelism.

Garp [6]. Garp was developed at University of California Berkeley. It belongs to the family of Reconfigurable Coprocessors as it integrates a reconfigurable array that has access to the processor's memory hierarchy. The reconfigurable array may be partially reconfigured as it is organized in rows. Configuration bits are included and linked as constants with ordinary C compiled programs.

Relatively loose coupling requires large chunks of computation to map onto reconfigurable array. Compiler uses hyper block scheduling to coalesce long instruction sequences into atomic regions. Aimed at very compute-intensive applications: 24x speedup over UltraSPARC on DES encryption ; 9.4x speedup on image dithering; 2.1x speedup on sorting [2]. To point that in this architecture is not exploit parallelism.

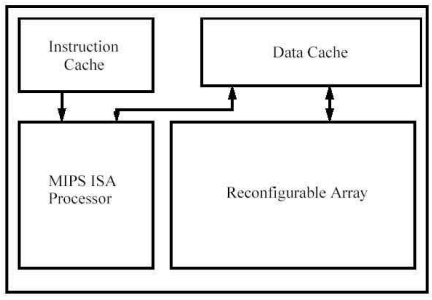

Figure 5 – Garp Architecture [2]

Relatively loose coupling requires large chunks of computation to map onto reconfigurable array. Compiler uses hyper block scheduling to coalesce long instruction sequences into atomic regions. Aimed at very compute-intensive applications: 24x speedup over UltraSPARC on DES encryption; 9.4x speedup on image dithering; 2.1x speedup on sorting [2]. To point that in this architecture is not exploit parallelism.

PipeRench. Developed at Carnegie Mellon University. It is a reconfigurable fabric that belongs to the family of Reconfigurable Attached Processors since it is interfaced with a processor through a PCI bus. PipeRench consists of an interconnected network of processing elements organized as pipeline stages. Each processing element consists of registers and ALUs. An intermediate language is used to generate the fabric's configurations.

MorphoSys. Developed at University of California Irvine. It is a Reconfigurable Cell Array architecture that includes context memory and belongs to the family of Reconfigurable Coprocessors. A DMA controller transfer data to the context memory, and to and from a frame buffer that holds the data the array will operate on.

Remarc. Developed at Stanford University. It is a Reconfigurable Coprocessor with 64 programmable units, which targets multimedia applications. Each 16-bit unit has an entry instruction RAM, ALUs, data RAM, instruction and several other registers. The reconfigurable array operates on the coprocessor data registers and a control unit transfers data between these registers and the processor.

Prisc. Developed at Harvard University. It integrates combinational reconfigurable logic as Reconfigurable Functional Units with two inputs and one output. The compiler analyzes opportunities the generated code and identifies sets of sequential instructions to execute on the PFU.

Sonic. A project from the University of London. It is designed to exploit parallelism in video image processing algorithms. It consists of a set of processing elements, called PIPEs, interconnected by a bus. It belongs to the family of Reconfigurable Attached Processors since it is interfaced with a processor through a PCI bus.

6.2 Commercial Systems

Chameleon Systems. The Chameleon CS2000 family combines a 32-bit embedded processor with a 32-bit reconfigurable processing fabric. A 128-bit, split-transaction bus links the two. The Chameleon compiler generates the final application from a C compiled object code linked with the fabric's previously generated bit stream configurations.

Annapolis. The Annapolis Wildfire family are Reconfigurable Attached multi-processor boards composed of an array of Xilinx FPGAs that are connected to a processor through a VME or PCI bus. The Wildcard is another Reconfigurable Attached Processor as it is PCMCIA sized card with an FPGA as processing element, memory and I/O connectors. They are programmed using standard C and VHDL tools.

7 Conclusion

The answer to the question "Is there any chance for Reconfigurable Processors Today?" is 'YES'.

Like it is described in this communication the architectures implemented in the earliest processors, like RISC processors, are "older" since they are limited to a fixed Instruction Set Architecture, and with no flexibility and witch are incompatible with earliest (other) architectures.

We give this answer with some backup support, because of the appearance of Institutions that are only focus the development and implementation of Reconfigurable Processors Architectures. Also, in this communication could be saw that they follow different architectures structures, but with the same purpose of creating a Reconfigurable Processor. Like for example 'Chimaera', that uses a architecture with a tightly-coupled Reconfigurable Functional Unit or a architecture with a Reconfigurable Coprocessor like used in 'Garp'.

Nowadays we are in the era of Reconfigurable Processors and Reconfigurable Computing in order to get the best performance of a Computer to work with whatever our needs are.

References

[1] PC Magazine PC Tech (RISC vs_ CISC The Real Story). See http://www.zdnet.com/pcmag/pctech/content/14/18/tu1418.002.html

[2] Professor Nicholas P. Carter, Coordinated Science Lab, University of Illinois. See http://www.crhc.uiuc.edu/ece412/lectures/lecture27.pdf

[3] Majd F. Sakr, Steven P. Levitan, C. Lee Giles, Donald M. Chiarulli, 'Reconfigurable Processor Employing Optical Channels', Proceedings of the 1998 International Topical Meeting on Optics in Computing (OC'98), Brugge, Belgium. See http://www.neci.nec.com/homepages/giles/papers/OC98.reconfigurable.processor-optics.channels.pdf

[4] Jonathan Hirshon, Horizon Communications, Pact Corporation. See http://www.horizonpr.com/pact/2001/011003-compiler.html

[5] Scott Hauck, Matt Hosler, 'The Chimaera Reconfigurable Functional Unit', Proceedings of IEEE Symposium on FPGAs for Custom Computing Machines (1997) 87-96. See http://www.ee.washington.edu/faculty/hauck/chimaera.html

[6] BRASS Research Group, 'Garp: Combining a Processor with a Reconfigurable Computing Array'. See http://brass.cs.berkeley.edu/garp.html

[7] Communication Systems Design, January 2000See http://www.csdmag.com/main/2000/01/0001top.htm