Evaluation of Run-Time Reconfiguration for General-Purpose Computing

Adronis Niyonkuru and Hans Christoph Zeidler
Universität der Bundeswehr Hamburg
Holstenhofweg 85
D-22043 Hamburg, Germany
{niyonkur, h.ch.zeidler}@unibw-hamburg.de

Abstract: In order to investigate the impact of dynamic hardware reconfiguration on general-purpose applications, we present a superscalar micro-architecture that includes a variable number of execution units. The set of execution units available is updated dynamically according to the run-time behaviour of the application. Performance is evaluated using software simulation of the proposed micro-architecture while running real-world applications of the SPEC2000 benchmark suite.

1 Introduction

Positive results have been reported about performance enhancement of many configurable computing machines especially if dynamic reconfiguration is involved. The execution of well-suited applications such as signal processing applications, image processing, encryption/decryption, etc. using reconfigurable architectures showed a significant speed-up.

Many research projects have been carried out to make reconfigurable systems easier to use and to extend the positive results to general-purpose applications. One approach is the development of a kind of operating system [WKW02] which allows the user to run any kind of application without having to know the hardware micro-architecture. Another one consists of developing an enhanced micro-architecture for a processor architecture by adding a reconfigurable datapath as a coprocessor [HW97], as an extended functional unit [SGS98] or as a full configurable micro-architecture [MD96]. The challenge today is to build competitive reconfigurable systems for mainstream computing, without neglecting compatibility with existing computing models.

Although the latest FPGAs provide a high level of integration, hard-wired processors still have a considerably higher clock frequency than field-programmable devices. On the other hand, many enhanced micro-architectural techniques have been increasing progressively performance of systems based on conventional hardware. Fortunately, the same techniques can be used to enhance FPGA-based systems. Such techniques include pipelining, superscalar execution, branch prediction, multi-threading, etc. Our goal is to combine the well known advantages of partial and dynamic reconfiguration with the micro-architectural achievements to bridge the gap between the two technologies. The result is a reconfigurable processor micro-architecture that executes any general-purpose application without having to either change the programming model or to look inside the hardware.

In the following section we present some related work which makes use of reconfigurable hardware in processor architecture. Section 3 gives details about the run-time reconfigurable micro-architecture proposed. It also describes the method used to evaluate the performance achieved when using different ways of reconfiguration. Section 4 shows the performance results obtained when running real-world applications taken from the SPEC2000 benchmark suite. The last section contains a short conclusion and a preview of our future work.

2 Related Work

There is a wide range of research projects using reconfigurable hardware in processor architecture. The first group we want to refer to proposes a fixed micro-architecture based on a conventional processor or conventional functional units extended with reconfigurable processing units. The latter execute specialized hardware-coded instructions to speed up an application. They act as a hardware accelerator or as a coprocessor. Programs running on such configurable machines have to be partitioned or differently scheduled, so that a code portion is executed on the conventional execution path and another code portion on the reconfigurable path. Furthermore, the programmer has either to invoke hardware libraries or himself implements compute-intensive tasks as hardware macros and to map them dynamically to the reconfigurable processing units. This kind of computing has been realized in many research projects and performance gain was reported for selected applications (**PRISC** [RS94], **DISC** [WH95], **CoMPARE** [SGS98], **OneChip** [CC01], **T1000** [ZM00], ...).

A different way of computing is presented in the **SCORE**-architecture [CCH+00]. Applications to be executed with SCORE have first to be partitioned into consecutive tasks called operators (e.g. multiplier, FFT, FIR-Filter). These operators and their corresponding data memory segments are loaded statically or dynamically into the reconfigurable hardware. If there are enough hardware resources to implement all the needed operators corresponding to a given application, there is no need to dynamically reconfigure the hardware. The application will be completely loaded before program execution. Otherwise, the first operators that fit into the hardware resources available are loaded, and after having finished to compute their code portion they are swapped to memory to let the following operators be loaded. A conventional processor is also required to sequence the compute pages.

Another particular architecture is **MATRIX** [MD96]. This is a high flexible reconfigurable computing architecture which allows the definition of the most suitable micro-architecture for each application. Accordingly, the instruction memory, the instruction flow, the data memory as well as the data path are not fixed but are updated for every application within a multilevel configuration scheme. The basic architecture building component is an array of 8-bit basic functional units (BFU). Each BFU includes a local memory, an 8-bit ALU and some control logic. It can be configured to serve as an instruction store, a data store, a datapath or a control element. A hierarchical network implements the configurable interconnect and enables data transfers between BFUs. The programming model of MATRIX consists of a hand-coded mapping of an algorithm on the BFUs and setting up appropriated connections between them.

The **run-time adaptive flexible instruction processors** [SLC02] realizes a unique approach to implement a dynamic processor micro-architecture. It consists of processor templates which allow to implement different processor types dynamically by varying a set of predefined parameters. These parameters determine for example whether the processor is stack based or register based. Furthermore, they are used to customize data and instruction widths, to change the pipeline depth, to add or remove hardware resources, etc. By investigating the application behaviour at run-time and using dynamic reconfiguration, it is possible to adapt at run-time the processor micro-architecture to the application requirements. The application behaviour is determined by collecting some statistics such as the number of times functions are called or by counting the most frequently used opcodes.

3 Evaluation Method

3.1 A dynamic reconfigurable micro-architecture

Our approach is based on the idea that the hardware requirements of a running application may be determined at run-time, and accordingly an optimized micro-architecture can be updated dynamically as proposed in [SLC02]. However, if we consider the complexity of a processor micro-architecture, it seems to be almost impossible to configure all micro-architectural components at run-time. Thus, our micro-architecture takes into account current hardware restrictions when using dynamic reconfiguration and includes some fixed components as well as a dynamically reconfigurable amount of execution units [NEZ02].

We make use of run-time reconfiguration which means that we analyze at run-time the application behaviour in terms of hardware requirements before the most suitable hardware configuration can be determined. Thus, an application code undergoes the same compiler process as in the conventional way of computing. This is an evolutionary approach that may lead to a better software compatibility and to the use of existing programming tools. In the same way as an Intel processor executes the same program differently as does an AMD processor, our micro-architecture intends to only modify the way instructions are executed. This new way of executing instructions basically consists of making use of partial and dynamic reconfiguration to implement a conventional processor architecture with a reconfigurable micro-architecture.

The proposed micro-architecture (Fig. 1) consists of:

- Fetch Unit/Pre-decoder: provides a valid address to the Instruction Memory and to the Trace Cache. It gets a single instruction from the memory or an instruction

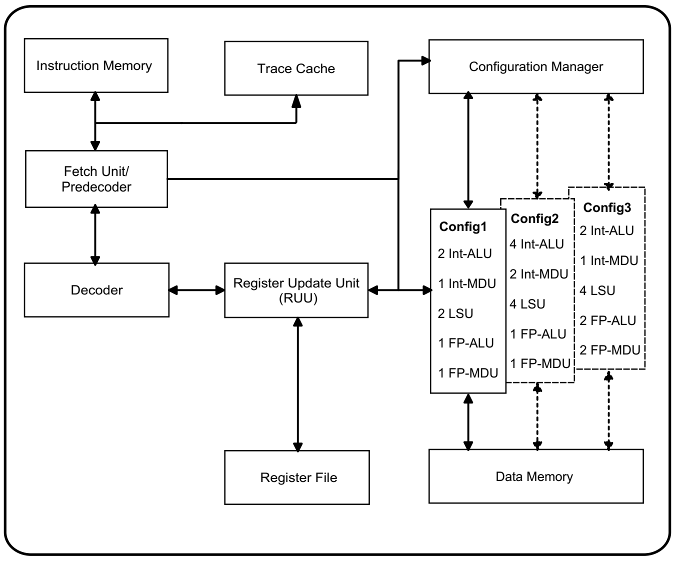

Figure 1: Processor Block Diagram

block from the Trace Cache if a corresponding trace line starting with that address has been updated previously. Pre-decoding determines the number of execution units required to execute this instruction block.

- Trace Cache: It holds already fetched instructions which may be executed many times within one application. The concept of a Trace Cache was originally introduced to avoid an instruction supply bottleneck [RBS96]. Meanwhile, Intel extended the concept as one of the key improvements of its Netburst micro-architecture and implemented it in the latest Pentium 4 Processor [In02].
- Decoder: It acts as a conventional instruction decoder
- Register Update Unit (RUU): This is the central unit to collect decoded instructions and to dispatch (issue) them to the different execution units.
- Register File: It stores operand values and operation results
- Config1/Config2/Config3: Some of the pre-defined hardware configurations with different numbers of execution units. One configuration may be substituted by another if a code portion of a running application requires more execution units of a particular type.
- Configuration Manager: It stores configurations currently not used and performs configuration swapping if necessary.
- Instruction/Data Memory: Separate instructions and data memories (caches).

We use the Trace Cache to store temporarily pre-decoded instructions which may be executed many times within an application. If an instruction is fetched for the first time it is also copied to the Trace Cache. Fetching an instruction is done by searching first the current instruction in the Trace Cache. A Trace Cache hit means that the corresponding trace line which includes a compact block of instructions is fetched according to the size of the fetch queue. It is possible to determine exactly how many integer operations, load/store operations and floating-point operations are included in a trace line. Hence, the number of corresponding execution units can be updated on-the-fly. If there is no trace line starting with the current fetch address, fetching is performed from the instruction memory as usual.

The advantage of such a run-time evaluation is that no additional software adaptation is needed. The results of the hardware evaluation are used to choose the best suitable configuration.

If we consider a high flexible reconfigurable platform with no hardware restrictions, the use of such a Trace Cache can help to dynamically reconfigure many micro-architectural components. Instead of configuring only the amount of execution units, one can decide to adapt dynamically the hardware resources deployed for the fetch unit, the decoder and the issue logic, the register file etc. to the variable size of an instruction block. Extending this concept to all micro-architectural processor components leads to the approach related in [SLC02] which means to create a complete micro-architecture at run-time.

Taking into account existing hardware restrictions when implementing this concept, especially focusing on partial and dynamic reconfiguration, we propose to split the micro-architecture into fixed and reconfigurable modules as proposed in [Xi00]. The Fetch Unit, the Decoder, the Register File, the Register Update Unit, the Configuration Manager as well as the Instruction and Data Memories form together the fixed module of the micro-architecture. The superscalar execution units are put together to form the reconfigurable modules. At one instant of program execution, only one reconfigurable module is active and can be replaced at any time by another one. Thus, applying the module-based partial reconfiguration design flow described by the referenced application note [Xi00], it is possible to dynamically integrate the fixed module and one of the reconfigurable modules to a complete processor micro-architecture.

We propose three reconfigurable modules with a different amount of superscalar execution units:

- A basic configuration which consists of two integer units for arithmetic and logical operations (*Int-ALU*), one integer multiplier/divider (*Int-MDU*), one load/store unit (*LSU*), one floating-point ALU (*FP-ALU*) and one floating-point multiplier/divider (*FP-MDU*).
- An enhanced configuration for integer-intensive operations with four integer ALUs, two integer multipliers/dividers, four load/store units, one floating-point ALU and one floating-point multiplier/divider.
- An enhanced configuration for the code portion with more floating-point operations. This configuration may consist of two integer ALUs, one integer multiplier/divider, four load/store units, two floating-point ALUs and two floating-point multipliers/dividers

Since only one of these pre-designed configurations can be active, the RUU reads an input signal from the configuration manager which shows the actual list of available execution units every time the configuration has been updated.

3.2 Performance Evaluation

An ideal way to evaluate the performance of the proposed micro-architecture for general-purpose computing is to implement it on real hardware (configurable, programmable or fixed) and to run standard benchmarks such as the SPEC2000 benchmark suite. Unfortunately, the key novelty of the micro-architecture, namely the dynamic hardware reconfiguration to map a variable number of execution units, is only supported by a few of SRAM-based programmable devices. Furthermore, the software support to enable partial and dynamic reconfiguration is still at an experimental level, so that it will take a long time to implement and test the micro-architecture in hardware. We are currently investigating the possibility to use the JBits-API as well as the module based design flow [Xi00], both from Xilinx.

Software simulation has been used in many areas of research as a method to evaluate models that are still under development. In this way it is possible to estimate in an early stage possible improvements of different models. Hence, only the model with the best simulation results will be implemented.

In this project, we used one of the SimpleScalar simulators, the *sim-outorder* that supports superscalar out-of-order program execution [BAB96]. We extended the simulator to model the additional Trace Cache and to support the run-time reconfiguration process. We do not model the configuration time penalty, even if this is the performance killer, because configuration time cannot be determined at the current development stage of the ongoing partial reconfiguration design flow. Nevertheless, the software flexibility offered by the simulator allows to evaluate the performance of different configuration models.

Moreover, we do not want to speed up only a given application but to evaluate performance benefits while using dynamic run-time reconfiguration for general-purpose computing. Therefore, we use the SPEC2000 benchmark suite as a workload. This benchmark is representative for general-purpose computing, since it consists of real-world applications including integer-based programs (CINT2000) as well as floating-point applications(CFP200).

In order to be able to run these benchmark programs on our extended processor simulator, they must first be compiled for the SimpleScalar architecture. This is made using the SimpleScalar toolset. After that, the different micro-architectures are compiled on a host machine. To run a simulation, a given micro-architecture is started within the simulator and executes the benchmark application. After the program is completely executed, the simulator delivers a detailed output including some performance parameters such as the number of instructions executed (*num_insts*), the average number of executed instructions per cycle (*IPC*), the average number of cycles per instruction (*CPI*), etc.

		Int-ALU	Int-MDU	LSU	FP-ALU	FP-MDU
sim-reference		4	1	2	4	1
sim-predef	config1	2	1	2	1	1
	config2	4	2	4	1	1
	config3	2	1	4	2	2
sim-dyn-reconf		1 - 4	1 - 4	1 - 4	1 - 4	1 - 4
sim-static-config1		4	2	4	1	1
sim-static-config2		2	1	4	2	2

Table 1: Simulated configurations with realistic hardware resources

3.3 Simulation results

We present simulation results obtained while simulating different processor micro-architectures. The first and basic micro-architecture (*sim-reference*) does not make use of reconfiguration. It is the default configuration from the SimpleScalar *sim-outorder* micro-architecture.

The second micro-architecture (*sim-predef*) makes use of dynamic reconfiguration by dynamically loading predefined configurations (*config1, config2, config3*).

The third micro-architecture (*sim-dyn-reconf*) is a highly dynamic reconfigurable one. It starts with the basic configuration in the same way as for the *sim-reference*-configuration. Every time the program execution requires more execution units of a special type (e.g. Int-ALU), the number of these execution units is updated to the evaluated amount. E.g. if a given trace line includes eight integer operations where each of them has to be executed on an integer ALU, the number of integer ALU available is increased up to eight. The amount of other execution units is not changed. This allows the software to determine the needed execution units (hardware-on-demand). We suppose to have enough hardware resources to implement all the execution units required. A more realistic assumption is to limit the amount to a maximum of execution units allowed, according to the hardware resources available.

The fourth micro-architecture is based on static hardware reconfiguration (*sim-static-config1*). For this micro-architecture, we assume that we have to execute a program that includes many integer operations (CINT2000). Therefore, the number of corresponding execution units is increased and the number of floating-point units is reduced before program execution starts.

The last micro-architecture we want to evaluate is also based on static reconfiguration, but intends to execute a program which includes more floating-point operations than integer operations (*sim-static-config2*).

Table 1 shows the number of the different execution units for the micro-architectures simulated. All these hardware configurations can be implemented with the amount of hardware resources available on today's FPGAs.

Using the software flexibility offered by the simulator, we are able to simulate other con-

		Int-ALU	Int-MDU	LSU	FP-ALU	FP-MDU
sim-reference		4	1	2	4	1
sim-predef	config1	4	1	2	4	1
	config2	8	4	4	1	1
	config3	2	1	4	4	4
sim-dyn-reconf		1 - 8	1 - 8	1 - 8	1 - 8	1 - 8
sim-static-config1		8	4	4	1	1
sim-static-config2		2	1	4	4	4

Table 2: Simulated configurations with extended hardware resources

figurations which are not realistic at the moment but which may be implemented in future. Table 2 summarizes the corresponding data:

When applying dynamic reconfiguration without predefined configurations (*sim_dynrec*), the amount of execution units continuously changes during program execution. The numbers are fixed at run-time varying between one and the maximum of eight execution units. The variable characteristics of the five simulated micro-architectures are:

⇒ the number of included execution units,

⇒ static, dynamic or without reconfiguration,

⇒ dynamic reconfiguration with predefined configurable modules or with run-time evaluation of the number of execution units.

Simulation results for some programs taken both from SPEC CINT2000 (integer operations benchmarks) and SPEC CFP2000 (floating-point operations benchmarks) are presented. For both benchmarks we used the test input set from SPEC to reduce the simulation run time. In order to compare performance benefits of different configurations one may use the number of instructions executed per cycle (Fig. 2) or the number of cycles needed to execute a single instruction (Fig. 3).

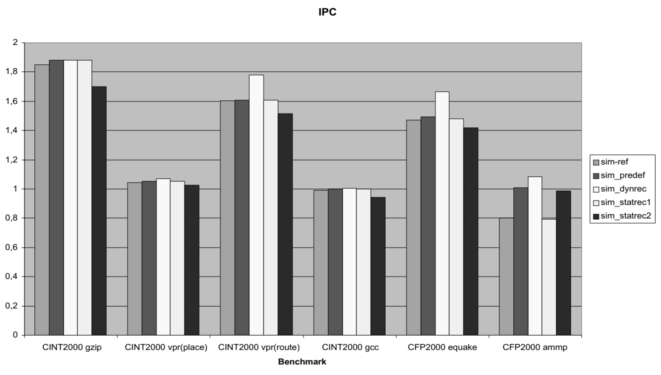

Figure 2: Instructions Per Cycle (IPC)

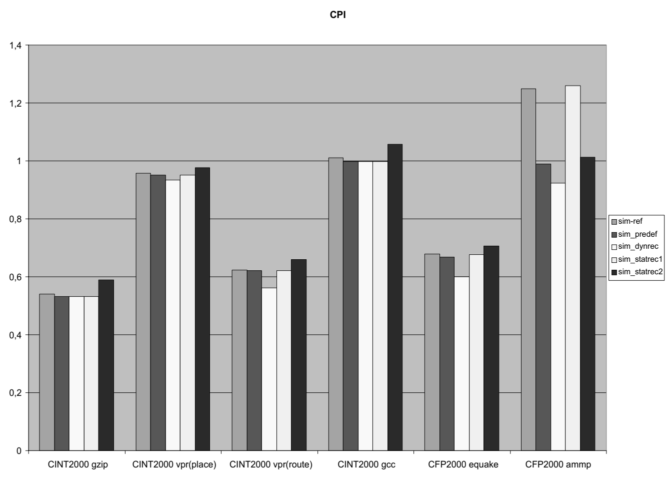

Figure 3: Cycles Per Instruction (CPI)

4 Conclusions

Reconfigurable hardware has been used to implement specialized processors with enhanced performance. However, the missing compatibility of these processors with each other as well as with general-purpose processors based on conventional hardware prevents their breakthrough. We propose an approach to use improved micro-architectural techniques together with partial and dynamic reconfiguration capabilities to implement a general-purpose reconfigurable processor. Our model consists of a superscalar reconfigurable micro-architecture, for which the amount of available execution units is dynamically updated during program execution. This is achieved by grouping some micro-architectural components to a fixed module, gathering the execution units to reconfigurable modules, and applying partial and run-time reconfiguration.

To get the first estimation of the expected performance a software simulation based on the SimpleScalar tool set with different reconfiguration schemes was run. Focusing on general-purpose computing, real-world applications from the SPEC benchmark suite were executed on the simulated micro-architectures. Some additional work is still needed to model also the real hardware characteristics such as reconfiguration time penalty, power consumption and utilization of hardware resources. The ongoing hardware implementation will give us realistic parameters to model these issues as well as the real performance gain.

References

[BAB96] Burger, D., Austin, T., und Bennett, S.: Evaluating future microprocessors: The simplescalar tool set. Technical Report 1308. University of Wisconsin - Madison. July 1996.

[CC01] Carrillo, J. und Chow, P.: The effect of reconfigurable units in superscalar processors. *9th ACM Int. Symp. Field-Programmable Gate Arrays*. S. 141–150. February 2001.

[CCH$^+$00] Caspi, E., Chu, M., Huang, R., Yeh, J., Wawrzynek, J., und DeHon, A.: Stream computations organized for reconfigurable execution (score). *Proc. 10th Int. Conf. Field-Programmable Logic and Applications*. S. 605–614. August 2000.

[HW97] Hauser, J. und Wawryzynek, J.: Garp: A mips processor with a reconfigurable coprocessor. *Proc. IEEE Work. FPGAs for Custom Computing Machines*. April 1997.

[In02] Intel: *A Detailed Look Inside the Intel NetBurst Microarchitecture of the Intel Pentium 4 Processor*. May 2002.

[MD96] Mirsky, E. und DeHon, A.: A reconfigurable computing architecture with configurable instructions and deployable resources. *Proc. IEEE Symp. FPGAs for Custom Computing Machines*. April 1996.

[NEZ02] Niyonkuru, A., Eggers, G., und Zeidler, H.: A reconfigurable processor architecture. *Proc. 12th Int. Conf. Field-Programmable Logic and Applications*. S. 1160–1163. September 2002.

[RBS96] Rothenberg, E., Bennett, S., und Smith, J. E.: Trace cache: a low latency approach to high bandwidth instruction fetching. *Proc. 29th Int. Symp. Microarchitecture*. S. 24–34. December 1996.

[RS94] Razdan, R. und Smith, M. D.: A high-performance microarchitecture with hardware-programmable functional units. *Proc. 27th Ann. Symp. Microarchitecture*. S. 172–180. November 1994.

[SGS98] Sawitzki, S., Gratz, A., und Spallek, R.: Compare: A simple processor architecture exploiting instruction level parallelism. *Proc. 5th Australasian Conf. Parallel and Real-Time Systems*. S. 213–224. April 1998.

[SLC02] Seng, S., Luk, W., und Cheung, P.: Run-time adaptive flexible instruction processors. *Proc. 12th Int. Conf. Field-Programmable Logic and Applications*. S. 545–555. September 2002.

[WH95] Wirthlin, M. J. und Hutchings, B. L.: A dynamic instruction set computer. *Proc. IEEE Work. FPGAs for Custom Computing Machines*. S. 99–107. April 1995.

[WKW02] Wigley, G., Kearney, D., und Warren, D.: Introducing reconfigme: An operating system for reconfigurable computing. *Proc. 12th Int. Conf. Field-Programmable Logic and Applications*. S. 687–697. September 2002.

[Xi00] Xilinx: *Two Flows for Partial Reconfiguration: Module Based or Small Bit Manipulations*. November 2000.

[ZM00] Zhou, X. und Martonosi, M.: Augmenting modern superscalar architectures with configurable extended instructions. *7th Reconfigurable Architectures Work. 2000/Proc. 15th Int. Parallel and Distributed Processing Symp.* S. 141–150. May 2000.