Microprocessors and Microsystems 38 (2014) 509–524

Contents lists available at ScienceDirect

Microprocessors and Microsystems

journal homepage: www.elsevier.com/locate/micpro

A transparent and adaptive reconfigurable system

Antonio Carlos Schneider Beck [a,*], Mateus Beck Rutzig [b], Luigi Carro [a]

[a] Universidade Federal do Rio Grande do Sul, Instituto de Informática, Porto Alegre, RS, Brazil
[b] Universidade Federal de Santa Maria, Departamento de Eletrônica e Computação, Santa Maria, RS, Brazil

ARTICLE INFO

Article history:
Available online 20 March 2014

Keywords:
Computer architecture
Reconfigurable architectures

ABSTRACT

In the current scenario, where computer systems are characterized by a high diversity of applications coexisting in a single device, and with the stagnation in frequency scaling because of the excessive power dissipation, reconfigurable systems have already proven to be very effective. However, they all present two major drawbacks, which are addressed by this work: lack of transparency (the need for special tools or compilers that changes the original code) and no ability to adapt to applications with different behaviors and characteristics, so significant gains are achieved only in very specific data stream oriented applications. Therefore, this work proposes the Dynamic Instruction Merging (DIM), a Binary Translation mechanism responsible for transforming sequences of instructions into a coarse-grained array configuration at run-time, in a totally transparent process, with support to speculative execution. The proposed system does not impose any kind of modification to the source or binary codes, so full binary compatibility is maintained. Moreover, it can optimize any application, even those that do not present specific kernels for optimization. DIM presents, on average, 2.7 times of performance gains and 2.35 times of energy savings over a MIPS processor, and a higher IPC than an out-of-order superscalar processor, running the MIBench benchmark set.

1. Introduction

The increasing complexity of processors was driven by a constant reduction in the size of transistors, and has been pushing performance to higher levels. However, because of physical limits of silicon [1,2], Moore's Law, as known today, will no longer exist in the near future. Moreover, according to the ITRS roadmap [3], alternative technologies that will completely or partially replace silicon, either have a higher density, but are slower than traditional scaled CMOS; or they will be faster, but with a huge area and power overhead. Additionally, high performance architectures, such as the superscalar processors, are reaching some well-known limits of the ILP [4,7], and hence one faces serious problems when it comes to increasing the IPC rate [5,6]. Therefore, the stagnation in frequency scaling, excessive power dissipation and higher hardware costs to ILP exploitation, together with the foreseen technologies, are new architectural challenges to be dealt with.

In this scenario, reconfigurable architectures appear to be an attractive solution. By using the combinational logic available in reconfigurable systems to execute sequences of code more efficiently, it is possible to obtain huge performance gains with energy savings [33], at the price of extra area – exactly the only resource available nowadays and in future technologies. Besides exploring the ILP of the applications, reconfigurable systems can also speed up sequences of data dependent instructions, which is their main advantage when one compares to traditional processor architectures. Moreover, they are highly based on regular circuits, which beneficially affects the yield and diminishes costs, considering that as the more transistor size shrinks, the harder it will be to print the geometries employed today [8].

However, reconfigurable systems have two main drawbacks. The first one is that they are designed to handle very data intensive or streaming workloads. This means that the main design strategy is to consider the target applications as having very few computationally intensive kernels to be optimized, which narrows their field of application. Therefore, they are not able to fulfill the requirements of current desktop or embedded systems, since they must execute a large number of applications concurrently, with different behaviors and needs.

The second problem with reconfigurable computing is that the process of mapping chunks of code to reconfigurable logic usually involves some kind of transformation, which can be manual or automatic (by using special languages or tool chains). These

* Corresponding author. Address: Universidade Federal do Rio Grande do Sul, Instituto de Informática, Av. Bento Gonçalves, 9500, Campus do Vale, Bloco IV, Caixa Postal 15064, Bairro Agronomia, Porto Alegre 91501-970, RS, Brazil. Tel.: +55 (51) 3308 6804.
E-mail address: caco@inf.ufrgs.br (A.C.S. Beck).

http://dx.doi.org/10.1016/j.micpro.2014.03.004
0141-9331/

transformations modify the source or binary code. However, as Intel and ARM have been showing with their families of ISAs (Instruction Set Architecture), binary compatibility between generations of processors is mandatory, so it is possible to reuse legacy code and to maintain traditional programming paradigms.

Based on the aforementioned discussion, this work proposes a new reconfigurable system, composed of a reconfigurable unit and a special Binary Translation (BT) mechanism implemented in hardware. The BT method is called Dynamic Instruction Merging (DIM). It is used to detect and transform sequences of instructions to be executed on the reconfigurable unit at run time, in a totally transparent process. Therefore, there is no need for changing the code at all, which ensures binary compatibility, making possible the use of the reconfigurable hardware without requiring any tools or special compilers for the hardware/software partitioning.

The reconfigurable unit is a coarse-grained array, composed of simple functional units and multiplexers. The coarse grained nature facilitates the translation process by the DIM and saves memory space, in opposite to the complexity found in fine-grained configurations. Therefore, the proposed system is capable of adapt itself to speed up and reduce energy consumption even of control-flow oriented software, which includes applications different from DSP like or loop centered ones, which do not present *hot spots* (a few number of kernels with high execution rates).

The next section reviews the existing reconfigurable architectures and dynamic optimization techniques. Section 3 shows the experiments we have performed to figure out what are the main challenges when it comes to proposing a reconfigurable architecture capable of optimizing programs with different behaviors. The proposed approach, described in details in Sections 4-6, can be applied to embedded and general-purpose domains, and in this work both these application fields are analyzed. Section 7 presents the simulation environment, results and a critical analysis, and compares our architecture with others.

It will be shown that, by the cost of extra area and power consumption, 2.7 times of performance gains and 2.35 times of energy savings are achieved, on average, compared to a single issue MIPS executing the MIBench benchmark set. The proposed technique also shows higher IPC than a 4-issue out of order superscalar processor based on the MIPS ISA. Section 8 demonstrates our contribution considering the whole context of reconfigurable systems. Finally, the last section draws conclusions and introduces future work.

2. Related work

2.1. Traditional reconfigurable systems and HLS synthesis

A reconfigurable architecture has the ability to adapt according to the needs of a given task at hand, performing several and different hardware computations. Usually, it comprises a reconfigurable logic, a context memory to store configurations, a General Purpose Processor (GPP), and a special component to control and reconfigure the logic (the controller may be also responsible for the communication among the previously cited components). Chunks of code are executed on the reconfigurable fabric, while the remaining instructions are executed on the GPP. The main challenge is to find the best tradeoff between which chunks of code should be executed on the reconfigurable fabric and the extra area and memory footprint necessary for that.

In accordance to the classification studies presented in [9–11], in this section we discuss only the most relevant work in the field. Concise [12] and Chimaera [13] use a tightly coupled reconfigurable unit. It works as another ordinary functional unit and limited to combinational logic only. The GARP machine [14] comprises a MIPS compatible processor with a loosely coupled and fine-grained reconfigurable unit (bit level operations). REMARC [15] is also a fine-grained architecture, and works as a loosely coupled coprocessor. RaPiD [16] and Piperench [17] are examples of coarse-grained reconfigurable architectures (word level operations). The main novelty of the Piperench architecture is the "pipelined reconfiguration": a given kernel is broken into pieces, and these pieces can be reconfigured and executed on demand. Afterwards, in a process called virtualization, they are multiplexed in time and space to be executed in the reconfigurable logic. The Molen [18] microcoded reconfigurable unit is a fine-grained, loosely coupled, and works together with a PowerPC processor.

DISC [19], OneChip [20], PRISM-II [21] are other reconfigurable architectures that employ standard fine-grained FPGA resources. In the group of coarse-grained reconfigurable systems, one could also include: Pact-XPP [22], Morphosys [23], Pleiades [24] and ADRES [25]. Furthermore, there are also reconfigurable architectures that are very similar to dataflow machines. For instance, TRIPS is based on a hybrid von-Neumann/dataflow architecture that combines an instance of coarse-grained, polymorphous grid processor core with an adaptive on-chip memory system [26]. TRIPS uses three different execution modes, focusing on instruction-, data- or thread-level parallelism. Wavescalar [27] is another example, and its implementation is very similar to the structure found in TRIPS.

Furthermore, there are several approaches that generate synthesized code from high level programming languages by the use of automatic converters. In these cases, the most computationally intensive code parts of the applications are translated from high level languages, such as C, to hardware description languages, such as VHDL or Verilog. Those parts will be executed in dedicated FPGA hardware or ASIC instead of on the GPP. Usually these converters are limited to a sub-set of the C language (for instance, pointers, unbounded loops and compiler libraries cannot be translated); and the generated hard code cannot be easily modified in cases the application changes or if designs constraints change.

In [61], it is proposed a computing machine that uses a single, serial instruction representation of an application. The parallel portions of the application are converted into a spatial representation to be accelerated by the reconfigurable fabric. The runtime conversion to spatial representation is facilitated by the use of an instruction set architecture based on an operand queue (rather than based on a stack or register file), to express the dependencies between the operations. To better identify portions of code for hardware compilation, instructions (*loopbegin* and *loopend*) are inserted in the source code to indicate the boundaries of parallelizable loops. Mittal et al. [60] proposes automatic translation of software binaries (compiled to the C6000, a Texas Instruments DSP) to a Register Transfer Level (RTL) language – VHDL or Verilog, which is afterwards mapped to a Xilinx Virtex II FPGA. Many optimizations are applied in the code by the proposed translation tool (the FREEDOM compiler), such as data dependency analysis, procedure extraction, induction variable analysis, and memory optimizations. Good speedups over the DSP processor are shown (between 3 and 20 times). Molen [59], ROCCC [54], DRESC [55], PARLGRAN [56] and the approaches presented in [57,58] are representative examples that exploit the parallelism of loops by using techniques such as loop unrolling and software pipelining to boost the performance of applications.

However, besides not being capable of accelerating applications with heterogeneous behaviors, which narrows their field of application to few domains (such as streaming or multimedia), none of these architectures present the desired characteristics to maintain binary compatibility. They heavily rely on compiler driven resource allocation, and modify the binary or source codes before its execution, which increases the design cycle. Such issues can be solved only through the use of dynamic optimization: the system's ability to adapt itself during execution.