An Automated Development Framework for a RISC Processor with Reconfigurable Instruction Set Extensions

Nikolaos Vassiliadis, George Theodoridis and Spiridon Nikolaidis
Section of Electronics and Computers, Department of Physics,
Aristotle University of Thessaloniki, 54124 Thessaloniki, Greece
nivas@skiathos.physics.auth.gr

Abstract

By coupling a reconfigurable hardware to a standard processor, high levels of flexibility and adaptability are achieved. However, this approach requires modifications to the compiler of the processor to take into account reconfigurable aspects. In this paper, a development framework for a RISC processor with reconfigurable instruction set extensions is presented. The framework is fully automated, hiding all reconfigurable related issues from the user and can be used for both program and fine-tune the architecture at design time. We demonstrate the above issues using a set of benchmarks. Experimental results show an x2.9 average speedup in addition to potential energy reduction.

1. Introduction

To amortize cost over high production volumes, embedded systems must exhibit high levels of flexibility and adaptability. An appealing option –broadly referred to as reconfigurable computing– is to couple a standard processor with reconfigurable hardware (RH). While the processor can serve as the bulk of the flexibility that can be used to implement any algorithm, the incorporation of the RH offers the adaptation of the system to the target application.

To program such hybrid architecture, the traditional software design flow of compiling for a target processor must be appropriately extended taking into account the presence of the RH [1]. Transparent incorporation of this feature is a must in order to preserve the time-to-market close to that of the traditional software design flow and continue to target software-oriented groups of users.

In this paper, we present an automated development framework for a dynamic Reconfigurable Instruction Set Processor (RISP). The target RISP architecture [2] consists of a RISC processor extended with a coarse-grain Reconfigurable Functional Unit (RFU). The framework was developed to be fully automated in the sense that hides all RH related issues requiring no interaction with the user other than that of a traditional software design flow. Furthermore, by allowing different values for various architectural parameters, the framework can be retargeted to different instances of the architecture. Thus, fine-tuning of the architecture during design time is possible, while support by the framework is maintained. To demonstrate the framework's usage, a number of applications, derived from various domains and benchmarking suites, were considered. Experimental results present an exploration to derive and evaluate an instance of the architecture using the proposed framework.

2. Related Work

The overwhelming majority of the proposed reconfigurable systems fall into two main categories based on the coupling between the processor and the RH: 1) the reconfigurable hardware is a co-processor of the main processor and 2) the reconfigurable hardware is a functional unit of the processor pipeline (we will state this category as RFU from now on).

The first category includes among others, the Garp, NAPA, Molen, REMARC, and PipeRench [3-7]. In this case, the coupling between the processor and the RH is loosely and communication is performed explicitly using special instructions. The available performance gain is significant but only parts of the code weakly interacting with the rest of the code, can be mapped to the RH and exploit this gain. These parts of the code must be identified and replaced with the appropriate special instructions. Garp and Molen features automation of this process but only for loop bodies and complete functions, respectively. For NAPA and PipeRench this process is performed manually.

Examples of the second category are systems such as PRISC, Chimaera, and XiRisc [8-10]. Communication is performed implicitly and the coupling is tight. The RH is treated as another functional unit of the processor. This

1-4244-0054-6/06/$20.00

makes control logic simple, while the communication overhead is eliminated but an opcode space explosion is likely. In XiRisc the identification of the extracted computational kernel must be performed manually, while PRISC and Chimaera feature no selection process for the identified instructions.

3. Target Architecture

The target architecture is a RISP processor [2] based on a standard 32-bit, single-issue, five-stage pipeline RISC architecture that has been extended with the following features:

- Extended ISA to support three types of operations performed by the RFU: 1) computations (comp), 2) addressing modes (mem), and 3) control transfer (cti).
- An interface supporting the tight couple of an RFU to the processor pipeline using the ISA extensions.
- An RFU array of Processing Elements (PEs).

On each execution cycle an instruction is fetched from the Instruction Memory. If the instruction is identified as reconfigurable its opcode is forwarded to the RFU. In addition, the opcode is decoded and produces the necessary control signals to drive the interface. At the same time the RFU is appropriately configured by downloading the necessary configuration bits from a local configuration memory with no cycle penalty.

The RFU consists of a 1-Dimension array of PEs. The array features an interconnection network that allows full connectivity. The granularity of PEs is 32-bit allowing the execution of the same word-level operations with the processor datapath. Each PE can be configured to provide unregistered or registered result. In the first case, spatial computation is exploited by executing chains of operations in the same clock cycle. When the delay of a chain exceeds the clock cycle, the register output is used to exploit temporal computation by providing the value to the next pipeline stage. In this way, the PEs can be seen as "floating" in the processor's pipeline.

We have designed and synthesized a complete hardware description model of the architecture using STM 0.13um technology [2]. Results indicate reasonable area overhead from the incorporation of the RFU. Also, the extensions do not introduce any overhead in the critical path of the processor.

4. Automated Development Framework

Our approach of compiling for reconfigurable computing involves primarily the transparent to the user incorporation of compiler extensions to support the reconfigurable instruction set extensions. Under this demand, we developed an automated development framework for the target RISP architecture, which organization is depicted in Figure 1. The complete flow is divided in five distinct stages, presented in detail below.

Front-End: MachSUIF [11] is used to generate the Control and Data Flow Graph (CDFG) of the application in SUIFvm Intermediate Representation (IR). In addition, a number of machine independent optimizations are performed in the CDFG.

Profiling: A MachSUIF pass has been developed that instruments the CDFG with profiling annotations, which mark the entrances and exits of basic blocks (we will state DFGs as basic blocks from now on). Another pass (m2c) translates the CDFG to equivalent C code and annotations regarding the basic blocks are converted to program counters. Compiling and executing the generated code, profiling information for the execution frequency of the basic blocks is collected.

Instruction Generation: This stage is divided in two steps. The goal of the first step (pattern generation) is the identification of complex patterns of primitive operations that can be merged into one reconfigurable instruction. The patterns generation engine is based on the MaxMISO (maximal multiple-input single-output) algorithm [12]. An enhanced version of the algorithm implemented in [13] is used. Enhancements consist of user defined parameters controlling: 1) the maximum number of inputs of the pattern, 2) the type of operations in the pattern and 3) the maximum number of operation in the patterns. Exploiting these features the user can configure the architecture at design time to fine-tune it towards an application domain.

In the second step, the mapping of the previously identified patterns in the RFU is performed and the actual reconfigurable instruction set extensions are generated. A mapper for the target RFU has been developed for this reason. Since any resource constraint has been resolved in the pattern generation step and the 1-D array of the RFU offers full connectivity, the implementation of the mapper is significantly simplified. The steps performed by the mapper are:

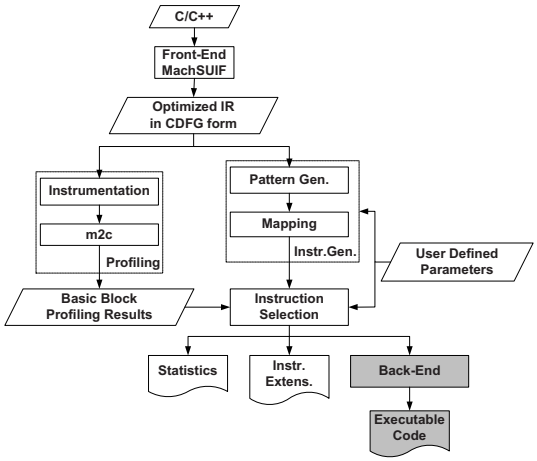

Figure 1. Automated development framework

1.Calculate the delay of each pattern. The delay is calculated using user parameters defining the delay of the modules of the RFU (PEs, interconnection etc.).
2. Place each operation of a pattern in a PE and appropriately configure its functionality.
3. Put the PE for execution in the appropriate pipeline stage based on the calculated delay and the type of the pattern (e.g. computation, addressing etc.). This is performed by selecting the registered or unregistered output of the PE.
4. Configure the multiplexers of the 1-D array for appropriate interconnection of the PEs.
5. Report the reconfigurable instruction set semantics (e.g. latency, type, resources etc.).

Instruction Selection: In this stage, the final instruction set extensions are selected. The only constraint for this selection is the total number of instructions that directly affects the storage size required for the configuration bits of the RFU. This constraint can be again defined externally to fine-tune the architecture during design time. Therefore, the only metric for the selection of an instruction is the offered speed-up.

Firstly, the static speed-up of each instruction is calculated by considering the software (processor) versus the hardware (RFU) execution cycles of the instruction. The static speed-ups are multiplied by the execution frequency of the basic block and the dynamic speed-ups are calculated. Finally, pair-wise graph isomorphism is performed. The instructions are ranked based on their dynamic speed-ups and the best are selected. The output of this stage is the reconfigurable instruction set extensions in addition to statistics (speed-up, number of instructions etc.).

Back-End: The back-end of the framework flow is the only stage that has not yet been fully implemented. However, since reconfigurable instructions do not require any special manipulation for the communication and synchronization between processor and RFU, the back-end is much like any traditional processor back-end.

5. Experimental Results

For the experiments we have consider a set of benchmarks derived from various suites and application fields. The complexity of the benchmarks varies from simple kernels (bitcount, fir, dct, quant, vlc) to implementation of algorithms in full applications (dijkstra-reed solomon encoder). Also a complex application (mpeg4 shape encoder) that uses several kernels is present. In the coming up experimental results, the possibilities of using the framework for fine-tune critical architectural parameters and to program/evaluate a specific instance of the architecture are demonstrated.

A parameter that can be used to drive the instruction generation stage is the number of operations in the patterns. This number is actually equal to the maximum numbers of PEs in the RFU. By appropriate selecting this number, the designer can accomplish the perfect balance between the required performance and area utilization. Figure 2 indicates that while an average speedup of x3 is possible with the maximum number of PEs (that is 13 PEs), the 97% of this speedup is feasible with only 8 PEs resulting to area reduction.

Another parameter that can be defined in the instruction generation stage is the number of maximum number of inputs for each pattern. This number is equal to the available read-ports of the register file. Even though our intention not to alter the instruction format and size of the base RISC processor have restricted us not to exceed the four inputs, Figure 3 gives an estimate of the unutilized performance. Thus, while unconstraint instruction generation can produce an average speedup of x3.2, the four inputs restriction utilizes the 94% and 91% of this speedup with no and eight PEs constraint in the number of PEs, respectively. Again the 8 PEs-4 Input instance seems like a very good trade-off between performance and area utilization.

Figure 4 shows the speedup of each application regarding the maximum number of available reconfigurable instructions. It must be pointed out that we assume all available instructions fit in the local configuration memory. This memory can provide on each cycle the appropriate configuration to the RFU without introducing any extra overhead. In general, the figure indicates that an average speedup of x2.9 over all considered benchmarks is possible.

The execution of complex patterns of operations as reconfigurable instruction in the RFU results to reduction of the program code size and instruction fetches. Figure 5 presents an average code size and instruction memory accesses reduction of 38% and 62%, respectively. Since a major source of energy consumption of embedded processors is the instruction memory accesses significant energy savings can be produced.

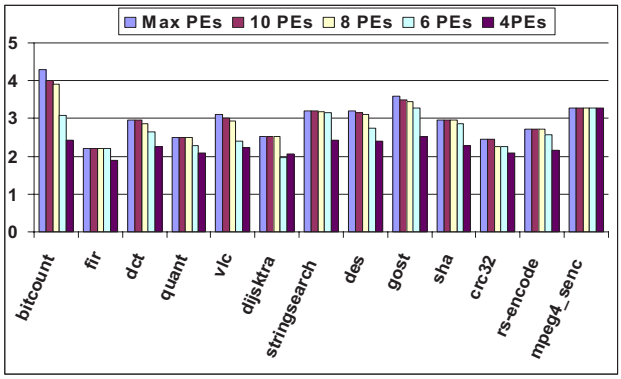

Figure 2. Speedup for different number of PEs

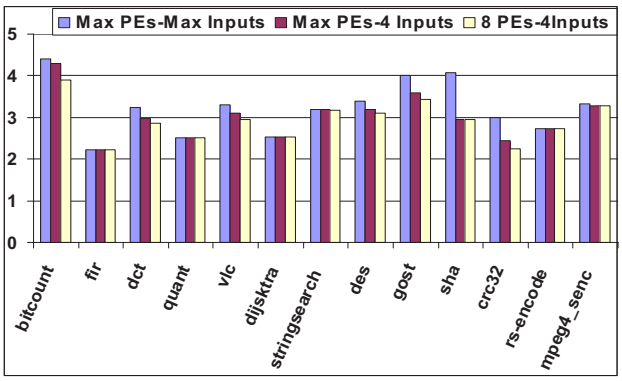

Figure 3. Speedup for various PEs/inputs

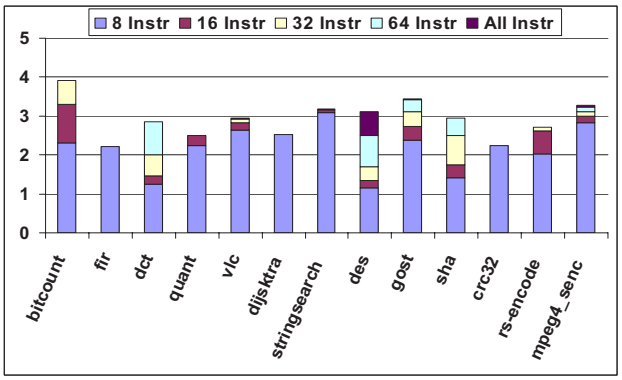

Figure 4. Speedup regarding the maximum number of reconfigurable instructions

6. Conclusions

We presented an automated development framework for a RISP that can be used both for fine-tune an instance of the RISP at design time but also to program it after fabrication. The framework accomplishes to hide all reconfigurable hardware related issues from the user. Thus, the target RISP architecture can be used by any software-oriented user with no knowledge of hardware design. Finally, the same framework can be used to explore different parameters of the architecture, derive the most suited for the targeted application, and support the new architecture instance without any modification.

Acknowledgement

This work was supported by the General Secretariat of Research and Technology of Greece and the European Union.

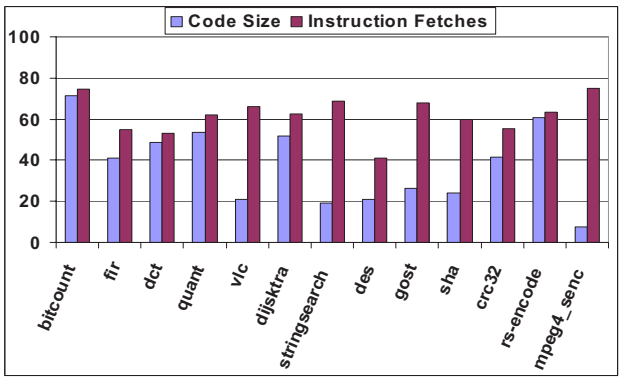

Figure 5. Code size and instruction fetches reduction

References

[1] F. Barat, R. Lauwereins, and G. Deconinck, "Reconfigurable Instruction Set Processors from a Hardware/Software Perspective," in *IEEE Trans. Softw. Eng. 28, 9*, pp. 847-862, 2002.

[2] N. Vassiliadis, N. Kavvadias, G. Theodoridis, and S. Nikolaidis "A RISC Architecture Extended by an Efficient Tightly Coupled Reconfigurable Unit", in ARC, pp. 41-49, 2005.

[3] T. J. Callahan, J. R. Hauser, and J. Wawrzynek, "The Garp Architecture and C Compiler," in *IEEE Computer*, vol. 33, no. 4, pp. 62-69, April, 2000.

[4] M. B. Gokhale and J. M. Stone, "NAPA C: Compiling for a Hybrid RISC/FPGA Architecture," in FCCM, pp. 126, 1998.

[5] S. Vassiliadis, S. Wong, G. Gaydadjiev, K. Bertels, G. Kuzmanov, and E. Moscu Panainte, "The MOLEN Polymorphic Processor," in *IEEE Transactions on Computers*, vol. 53, no. 11, pp. 1363-1375, November, 2004.

[6] T. Miyamori, and K. Olukotun, "REMARC: Reconfigurable Multimedia Array Co-Processor," in *IEICE Trans. Information Systems*, vol. E82-D, no. 2, pp. 389-397, February, 1999.

[7] S. C. Goldstein, H. Schmit, M. Moe, M. Budiu, S. Cadambi, R. Taylor, and R. Laufer, "Piperench: A Coprocessor for Streaming Multimedia Acceleration," in *26th Annual Int. Symposium on Computer Architecture*, pp. 28-39, 1999.

[8] R. Razdan and M. D.Smith, "A High-Performance Microarchitecture with Hardware-Programmable Functional Units," in MICRO 27, ACM Press, pp. 172-180, 1994.

[9] Z. A. Ye, N. Shenoy, and P. Baneijee, "A C compiler for a Processor with a Reconfigurable Functional Unit," in FPGA, ACM Press, pp.95-100, 2000.

[10] A. La Rosa, L. Lavagno, and C. Passerone, "Software Development for High-Performance, Reconfigurable, Embedded Multimedia Systems," in *IEEE Design and Test of Computers*, vol. 22, no. 1, pp. 28-38, 2005.

[11] Machine-SUIF research compiler.

[12] C. Alippi, W. Fornaciari, L. Pozzi, and M. Sami, "A DAG Based Design Approach for Reconfigurable VLIW Processors," in *DATE*, pp. 778-779, 1999.

[13] N. Kavvadias and S. Nikolaidis, "Automated Instruction-Set Extension of Embedded Processors with Application to MPEG-4 Video Encoding," in ASAP'05, pp. 140-145, 2005.