Compiling for the Molen Programming Paradigm

Elena Moscu Panainte, Koen Bertels, Stamatis Vassiliadis
Computer Engineering Lab
Faculty of Electrical Engineering, Mathematics and Computer Science
TU Delft, The Netherlands

1 Introduction

In the last decade, several approaches have been proposed for coupling an FPGA to a GPP. For a classification of these approaches the interested reader is referred to [4]. There are four shortcomings of current approaches, namely:

1. **Opcode space explosion**: a common approach (e.g. [2], [3]) is to introduce a new instruction for each part of application mapped into the FPGA. The consequence is the limitation of the number of operations implemented into the FPGA, due to the limitation of the opcode space.
2. **Limitation of the number of parameters**: In a number of approaches, the operations mapped on an FPGA can only have a small number of input and output parameters ([1], [6]), due to the encoding limits.
3. No support for **parallel execution** on the FPGA of sequential operations: an important and powerful feature of FPGA's can be the parallel execution of sequential operations when they have no data dependency. Many architectures [4] do not take into account this issue and their mechanism for FPGA integration cannot be extended to support parallelism.
4. No **modularity**: each approach has a specific definition and implementation bounded for a specific reconfigurable technology and design. Consequently, the applications cannot be (easily) ported to a new reconfigurable platform. Further there are no mechanisms allowing reconfigurable implementation to be developed separately and ported transparently.

A general approach is required that eliminates these shortcomings. In this paper, a programming paradigm for reconfigurable architectures [5], called the Molen Programming Paradigm and a compiler are described that offer alternatives and a solution to the above presented limitations.

2 The Programming Paradigm

The Molen programming paradigm[5] is a sequential consistency paradigm for programming CCMs possibly including a general purpose computational engine(s). The paradigm allows for parallel and concurrent hardware execution and it is intended (currently) for single program execution. It requires only a one time architectural extension of few instructions to provide a large user reconfigurable operation space. The added instructions (currently supported by the compiler) include:

- Two instructions[1] for controlling the reconfigurable hardware, namely:
  - SET < *address* >: at a particular location the hardware configuration logic is defined
  - EXECUTE < *address* >: for controlling the executions of the operations on the reconfigurable hardware
- Two move instructions for passing values to and from the GPP register file (GPR) and XRs. hardware.

Code fragments constituted of contiguous statements (as they are represented in high-level programming languages) can be isolated as generally implementable functions (that is code with multiple identifiable input/output values). The parameters stored in registers are passed to special reconfigurable hardware registers denoted as Exchange Registers(XRs).

3 Compiler Extensions

The compiler system relies on the Stanford SUIF2 (Stanford University Intermediate Format) Compiler Infrastructure for the front-end, while the back-end is built over the framework offered by the Harvard Machine SUIF. The last component has been designed

[1] Actually, five if partial reconfiguration, pre-loading of reconfiguration and executing microcode are explicitly assumed [5].

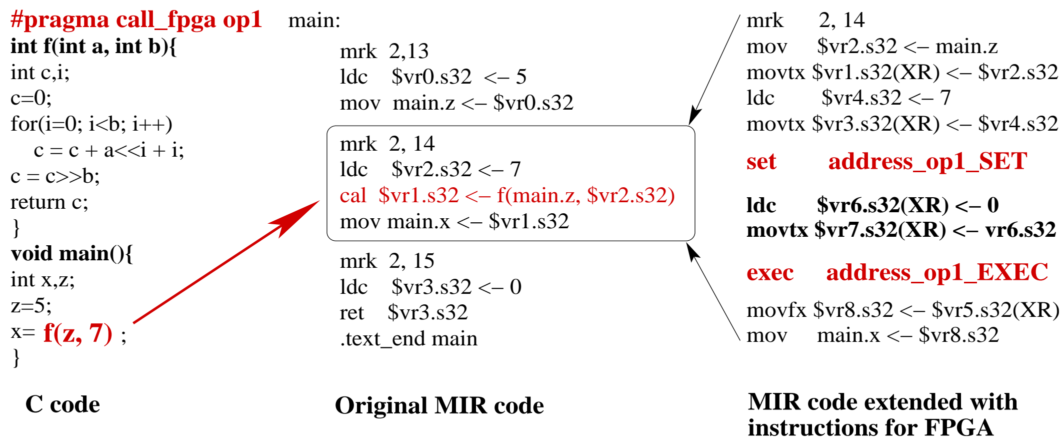

Figure 1. Code Generation at MIR level

with retargetability in mind. It provides a set of backends for GPPs, powerful optimizations, transformations and analysis passes. These are essential features for a compiler targeting a CCM. We have currently implemented the following extensions:

- Code identification: The identification is based on code annotation with special pragma directives (similar to [2]). In this pass, all the calls of the recognized functions (executed on the reconfigurable hardware) are marked for further modification.
- Instruction Set extension: the Instruction Set has been extended with SET/ EXECUTE instructions at both Medium an Low Intermediate Representation level.
- Register file extension: the Register File Set has been extended with the XRs. The register allocation algorithm allocates the XRs in a distinct pass applied before the GPR allocation.
- Code generation: code generation for the reconfigurable hardware is performed when translating SUIF to Machine SUIF IR, and affects the function calls marked in the front-end.

An example of the code generated by the extended compiler for the Molen programming paradigm is presented in Figure 1. In the first part, the C program is given. The function implemented in reconfigurable hardware is annotated with a pragma directive named *call_fpga*. It has incorporated the operation name, *op1* as specified in the description file. In the central part of the picture,the code generated by the original compiler for the C program is depicted. The pragma annotation is ignored and a normal function call is included. The last part of the picture presents the code generated by the compiler extended for the Molen programming paradigm; the function call is replaced with the appropriate instructions for sending parameters to the reconfigurable hardware in XRs, hardware reconfiguration, preparing the fix XR for the microcode of the EXECUTE instruction, execution of the operation and the transfer of the result back to the GPP. The presented code is at MIR level and the register allocation pass has not been applied. The compiler extracts from a description file the information about the target architecture such as microcode address of SET and EXECUTE instructions for each operation implemented in the reconfigurable hardware, the number of XRs, the fix XR associated with each operation.

4 A Case Study

In order to evaluate the performance improvements provided by the Molen processor, we used two well-known multimedia benchmarks, namely *mpeg2enc* and *ijpeg* for which we perform a pure software analysis. The input bitstreams are (i) for mpeg2enc: the frames included in the benchmark and (ii) for ijpeg: specmun, 1024 * 688 The operations that are candidates for the hardware implementation are the well-known time-consuming multimedia operations[5]: SAD DCT , IDCT and VLC . In order to study the performance improvements, we use the *Halt* library available in Machine SUIF and which we modified to suit our purpose. This library is an instrumentation package that allows the compiler to change the code of the program being compiled in order to collect information about the program own behavior (at run-time).

For the above considered applications, we measured for their pure software implementation on the GPP the exact types and numbers of instructions - generated by the compiler- which are executed in the whole application and in each chosen function for hardware implementation plus their exact number of calls and the number of cycles. Based on these data, the following information can be computed for the Molen reconfigurable processor: (i)the code reduction as a result of implementation of parts of the application in reconfigurable hardware and (ii) an approximation of the maximum performance improvement of processor cycles for the whole application and for a particular implemen-

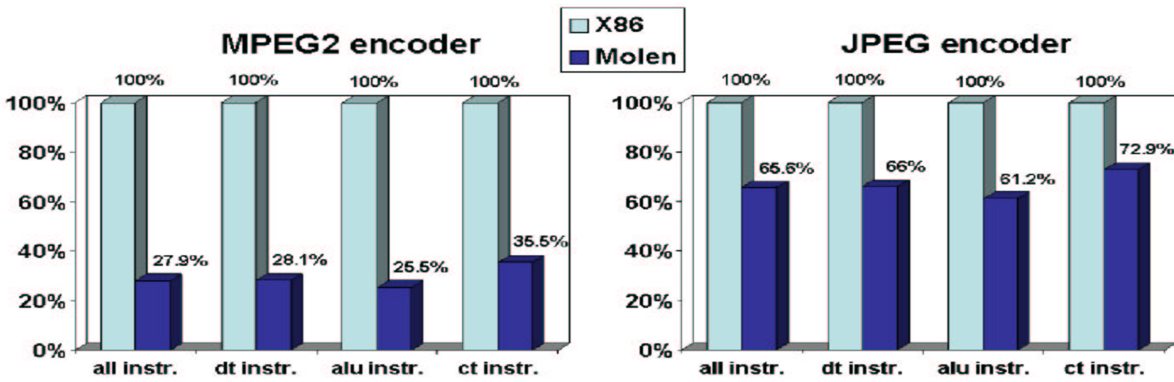

Figure 2. mpeg2enc and ijpeg encoder instruction results

Fct	Cycles	% Total
SAD	149.947.461	55.2 %
DCT	42.529.647	15.7 %
VLC	3.946.954	1.4 %
IDCT	3.693.986	1.36 %
mpeg2enc Application	271.616.655	100 %

Table 1. The mpeg2enc (cycles) results

Fct	Cycles	%Total
DCT	40.206.773	12.5 %
VLC	36.571.622	10.5 %
ijpeg enc Application	341.316.466	100 %

Table 2. The jpeg encoder (cycles) results

tation of one operation.

The measured data for the GPP alone and the computed data for the Molen processor are compared for mpeg2enc and ijpeg in Figure 2. The most important categories of instructions have been considered, namely data transfer (dt) instructions, arithmetic and logical (alu) instructions and control transfer(ct) instructions. From these pictures, a substantial reduction of the number of instructions is achieved by the Molen reconfigurable processor compared to the GPP: 72.1% for mpeg2enc and 34.4 % for ijpeg encoder. Also it is obvious that in both cases the alu instructions are the most reduced category of instructions, while the ct instructions are the least reduced instructions. In Table 1 and 2, the cycle measurements are reported. From these results, we can identify those functions that potentially give the highest performance improvement, given an efficient hardware implementation. The numbers suggest that the SAD function is the most promising candidate for hardware implementation, while the rest of the functions can provide at best a moderate performance improvement.

5 Conclusions

The compiler extensions allow to generate code where the operations implemented on the reconfigurable hardware are automatically (rather then manually) substituted by the appropriate SET-EXECUTE instructions. It has been shown through experimentation that the compiler can be used as an important tool to support the design process focusing on the identification of good candidates for the reconfigurable hardware implementation. The presented results show a substantial reduction of the executed number of instructions and potential reduction of processor cycles for two multimedia benchmarks for their execution on the Molen reconfigurable processor compared to their pure software implementation on the GPP.

References

[1] F. Campi, R. Canegallo, and R. Guerrieri. IP-Reusable 32-Bit VLIW Risc Core. In *Proc. of the 27th European Solid-State Circuits Conference*, pages 456–459, Villah, Austria, Sep 2001.

[2] M. Gokhale and J. Stone. Napa C: Compiling for a Hybrid RISC/FPGA Architecture. In *Proc. IEEE Symp. on Field-Programmable Custom Computing Machines*, pages 126–137, Napa, California, April 1998.

[3] A. L. Rosa, L. Lavagno, and C. Passerone. Hardware/Software Design Space Exploration for a Reconfigurable Processor. In *Proc. of the DATE 2003*, pages 570–575, 2003.

[4] M. Sima, S. Vassiliadis, S.Cotofana, J. van Eijndhoven, and K. Vissers. Field-Programmable Custom Computing Machines - A Taxonomy. In *FPL*, pages 79–88, Montpellier, France, Sep 2002.

[5] S. Vassiliadis, S. Wong, and S. Cotofana. The MOLEN $\rho\mu$-Coded Processor. In *FPL, Springer-Verlag LNCS*, volume 2147, pages 275–285, Belfast, UK, Aug 2001.

[6] Z. Ye, N. Shenoy, and P. Banerjee. A C Compiler for a Processor with a Reconfigurable Functional Unit. In *ACM/SIGDA Symposium on FPGAs*, pages 95–100, Montery, California, USA, 2000.