Combining Rewriting-Logic, Architecture Generation, and Simulation to Exploit Coarse-Grained Reconfigurable Architectures*

Carlos Morra[1], João Bispo[2], João M. P. Cardoso[2], and Jürgen Becker[1]

[1] Institut für Technik der Informationsverarbeitung (ITIV)
Universität Karlsruhe (TH), Karlsruhe, Germany
{morra,becker}@itiv.uni-karlsruhe.de

[2] UTL/IST, Department of Computer Science and Engineering
INESC-ID, 1000-029, Lisbon, Portugal
joaobispo@gmail.com, jmpc@acm.org

1. Introduction

Coarse-grained reconfigurable arrays (CGRAs) based on the VLIW concept can be interesting solutions to speed-up hotspots of certain applications. They rely on a 1D array of Processing Elements (PEs), as illustrated in the example in Fig. 1. The PEs can typically be programmed to execute ALU operations, multiplications, load/store operations, etc. It is common to use a global register file for storing scalar variables and intermediate results. Although the VLIW concept has limited forms of spatial computing, using register files to communicate globally data eases the mapping process, since the interconnects between computing structures are simplified to register assignments and subsequent moves. However, compared to architectures using interconnects between PEs, in VLIW-based CGRAs data communication between PEs is slower and the cost increases with the number of ports for the register files. CGRAs may differ by the number and complexity of their PEs. For instance, PEs may support directly 2-, 3- and 4-operand instructions. To better customizing a CGRA template, suitable for one or more kernels, we need approaches to evaluate those main differences. Our work addresses a global and unique methodology to exploit CGRAs-1D, such as the one in Fig. 1.

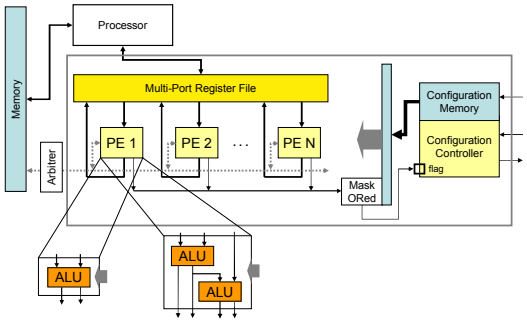

Fig. 1. Typical 1-D array currently exploited.

Our methodology uses term-rewriting logic [2] for the compiler to be aware of the templates of instructions directly supported by each PE, when mapping an imperative programming model to the target architecture. For early evaluation, the methodology is now enriched by a high-level clock cycle simulator and an architecture generator. The architecture generator outputs, from a textual structural description, VHDL code ready for logic synthesis.

Recent advances to our approach, previously presented in [1], allow us to analyze the effect of the number and different types of PEs on performance, FPGA resource utilization, and maximum clock speed.

* This work was partially supported by a DAAD/CRUP cooperation project entitled ACER. Cardoso and Bispo were also partially supported by project COBAYA, funded by the Portuguese Foundation for Science and Technology (FCT).

2. Architecture Exploration using Rewriting-Logic

Our proposed environment is presented in Fig. 2. The input is an extended, three-address, Static Single Assignment (SSA), intermediate form, generated by the Nau compiler [3] from the Java Bytecodes of a given class method.

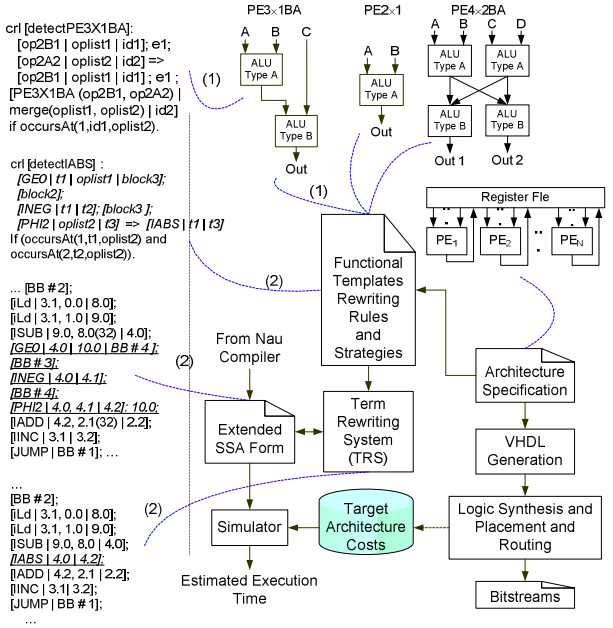

Fig. 2. Proposed environment.

The different types of PEs are described using *term rewriting* rules which are functional templates used by the Rewriting-Logic environment [4] to identify sets of SSA instructions that can be executed in each PE. The mapping process is managed using logic strategies. Different strategies can be defined and selected in order to find the effect of the number and PE types on the performance of the application being mapped. An example of a PE definition is given in Fig. 2, path (1). In this example, a two level PE with an

978-0-7695-3307-0/08 $25.00 © 2008 IEEE
DOI 10.1109/FCCM.2008.37

ALU of type B and an ALU of type A is described. This rule is equivalent to: "if there is an operation of Type B whose output is used as the first input of an operation of Type A, then rewrite these operations into a PE of type PE3×1BA". An example of mapping a set of SSA instructions into an absolute value operator is given in Fig. 2, path (2).

We have developed a simulator and an architecture generator to exploit CGRAs-1D. The simulator is able to execute and dynamically schedule to the 1-D array the instructions in the SSA intermediate representation output by the compiler or after the TRS step. Our tool accepts a textual description of a CGRA and generates the VHDL code ready for logic synthesis.

3. Experiments

For validation purposes, we show here some experiments using typical signal and image processing, computationally intensive kernels, namely, fdct, fft, fir, fwt2D, hamming, and smooth. We consider 32-bit CGRAs. Fig. 3 (top) shows the number of FPGA (a Xilinx Virtex-4, xc4vfx140ff1517-11, is used) resources used when considering CGRAs with PEs of type PE2×1 (2/1 I/Os) and PE4×2BA (4/2 I/Os and 2 levels of FUs with both levels supporting logic and arithmetic operations, but with multiplications only in the first level). We use register files with 128 registers and with a number of I/O ports dependent on the number of PEs and their I/O ports. The configuration memory consists of 128 words, being the width of each word specific to each array. When varying the number of PEs, the number of FPGA resources increases almost linearly, being the exception the number of BRAMs which increases quadratically (note that a register file with M-input and N-output ports requires N×M memory blocks [5]). The maximum clock frequency (*MaxFreq*) for a given architecture varies slightly with the number of PEs. However, the *MaxFreq* is on average 1.7× higher for CGRAs with PE2×1 (80 to 83 MHz) than with PE4×2BA (47 to 50 MHz). CGRAs with PE4×2BA need on average almost 4× more FF and LUT resources, and 2× more DSP and RAM blocks than the PE2×1 CGRA-1D.

Fig. 3 (bottom) shows the average speed-ups achieved for the examples with the two different architectures, when varying the number of PEs from 1 to 13. The speed-ups are related to the execution of the examples in a single PE2×1 architecture. We include the speed-ups, identified as "optimistic", that would have been obtained by the PE4×2BA CGRAs if they could achieve the same *MaxFreq* of PE×1 CGRAs. For all the experiments, we consider a single-port memory with single clock cycle load and store operations.

For PE2×1 CGRAs-1D, the maximum speed-ups are obtained with 8 PEs. From 7 to 8 PEs a small improvement of 2.87% is achieved. In the case of PE4×2BA arrays, the maximum speed-ups are achieved with 12 PEs, with improvements between 8 to 12 PEs and between 7 to 12 PEs of about 7.6 and 10%, respectively. Considering the examples used, the preliminary results show that the ILP degree currently exposed is mostly fulfilled with 8 PEs. Although we use non-pipelined PEs, the results also indicate that the gains achieved by complex PEs (e.g., with 4/2 I/Os) over simple PEs (e.g., with 2/1 I/Os) are minimal (they are even worse when considering maximum clock frequencies).

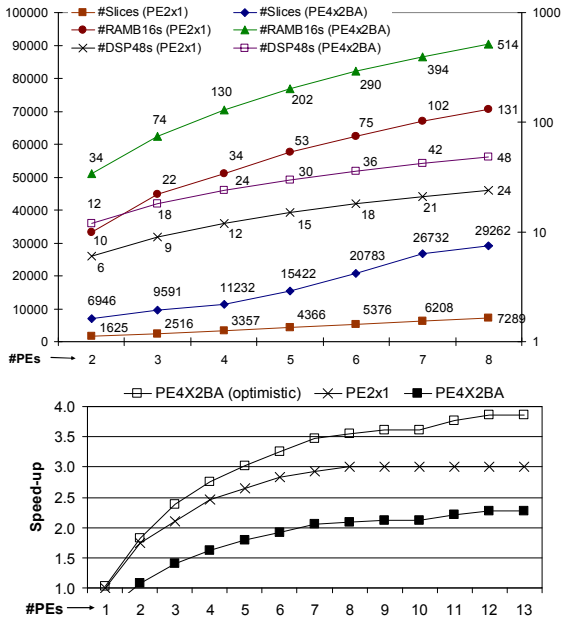

Fig. 3. FPGA resources (top) and average speed-ups (bottom).

Ongoing work focuses on analytical strategies to automatically explore the design space. We plan to evaluate more aggressive forms of ILP for acquiring the potential of large CGRAs to improve performance.

4. References

[1] C. Morra, J. M. P. Cardoso, and J. Becker, "Using Rewriting Logic to Match Patterns of Instructions from a Compiler Intermediate Form to Coarse-Grained Processing Elements," in *14th Reconfigurable Architectures Workshop (RAW'07)*, Long Beach, CA, USA, March 27-28, 2007.

[2] F. Baader, and T. Nipkow, *Term Rewriting and All That*, Cambridge University Press, 1998.

[3] R. Rodrigues, J. M. P. Cardoso, and P. C. Diniz, "A Data-Driven Approach for Pipelining Sequences of Data-Dependent Loops," in *15th IEEE Symposium on Field Programmable Custom Computing Machines (FCCM'07)*, Napa Valley, CA, USA, April 23-25, 2007, pp. 219-228.

[4] M. Clavel, et al., "The Maude 2.0 System," in *Rewriting Techniques and Applications (RTA 2003)*, LNCS 2706, Springer-Verlag, June 2003, pp. 76-87.

[5] M. A. R. Saghir, and R. Naous, "A Configurable Multi-Ported Register File Architecture for Soft Processor Cores," in *Int'l Workshop on Applied Reconfigurable Computing (ARC'07)*, Springer-Verlag LNCS 4419, March 27-29, 2007, pp. 14-25.