Chapter 12
The Role of Programming Models on Reconfigurable Computing Fabrics

João M. P. Cardoso
University of Porto, Portugal

João Bispo
Technical University of Lisbon, Portugal

Adriano K. Sanches
Technical University of Lisbon, Portugal

ABSTRACT
Reconfigurable computing architectures are becoming increasingly important in many computing domains (e.g., embedded and high-performance systems). These architectures promise comparable characteristics to specific hardware solutions with the flexibility and programmability of microprocessor solutions. This chapter gives a comprehensible overview of reconfigurable computing concepts and programming paradigms for the current and future generation of reconfigurable computing architectures. Two paramount aspects are highlighted: understanding how the programming model can help the mapping of computations to these architectures, and understanding also the way new programming models can be used to develop applications to these architectures. We include a set of simple examples to show different aspects of the use of the reconfigurable computing synergies, driven by the initial programming model used.

INTRODUCTION
Reconfigurable computing architectures are playing a very important role in specific computing domains (Hauck & DeHon, 2008). In the arena of high-performance computing (HPC), Field-Programmable Gate-Arrays (FPGAs) have exhibited in many cases outstanding performance gains over traditional von-Neumann based computer architectures (El-Ghazawi et al., 2008). In the context of embedded systems, FPGAs are common-place for early prototyping, and more recently even for deployment, given such characteristics as the substantial increase of resources in the high-end FPGAs, the ability to “zero-cost” update of hardware in early timing windows where modifications might have...
The Role of Programming Models on Reconfigurable Computing Fabrics

Figure 1. An example of a possible reconfigurable computing fabric which includes general purpose processors (GPPs)

Reconfigurable computing fabrics (with FPGAs being the most notable examples) mainly consist of aggregations of a large number of elements, namely: processing elements (PEs), memory elements (MEs), interconnection resources (IRs), and I/O buffers (IOBs). There are reconfigurable substrates which include microprocessor hardcores (i.e., fabricated on-chip processors side-by-side with reconfigurable logic). Examples of this are the Xilinx FPGAs with IBM PowerPC cores. Figure 1 shows a possible block diagram of a reconfigurable computing fabric consisting of GPPs (General Purpose Processors), reconfigurable resources, and memory blocks. The reconfigurable fabrics distinguish themselves according to the granularity of the PEs and IRs. There are fabrics using fine-grained hardware structures, i.e., configurable blocks with small bit-widths (e.g., 4 bits), fabrics using coarse-grained hardware structures, i.e., configurable blocks with large bit-widths (e.g., 16, 24, 32 bits), and fabrics with a mix of fine- and coarse-grained hardware structures.

The granularity of the fabric constrains the type of computing engines we can implement with its resources (see Figure 2). Fine-grained reconfigurable fabrics implement computing engines using gate-level circuitry descriptions (e.g., AND, OR gates), while coarse-grained reconfigurable fabrics implement computing engines at the word or ALU level. In coarse-grained reconfigurable fabrics we have to recall to a single or a set of pre-defined computing models while in fine-grained reconfigurable fabrics we are able to implement virtually any type of computing model. In the latter case, we can implement static or dynamic (tagged-token) Dataflow Machines, Khan Processor Networks, Petri Nets, Cellular Automata, VLSI (Very Large Scale Integration) and Systolic arrays, von-Neumann processors, SPMD (single-program, multiple-data), SIMD (single-instruction, multiple-data), and MIMD (multiple-instruction, multiple-data) processing engines, ASIPs (Application-Specific Instruction-Set Processors), application specific architectures, etc. This huge flexibility comes with costs: programming is more difficult, takes