Scheduling, Binding and Routing System for a Run-Time Reconfigurable Operator Based Multimedia Architecture

Erwan Raffin, Caps-enterprise, France
Christophe Wolinski, IRISA/University of Rennes 1, France
François Charot, INRIA Rennes–Bretagne Atlantique, France
Emmanuel Casseau, IRISA/University of Rennes 1, France
Antoine Floc’h, INRIA Rennes–Bretagne Atlantique, France
Krzysztof Kuchcinski, Lund University, Sweden
Stéphane Chevobbe, CEA–LIST, France
Stéphane Guyetant, CEA–LIST, France

ABSTRACT

This article presents an integrated environment for application scheduling, binding and routing used for the run-time reconfigurable, operator based, ROMA multimedia architecture. The environment is very flexible and after a minor modification can support other reconfigurable architectures. Currently, it supports the architecture model composed of a bank of single (double) port memories, two communication networks (with different topologies) and a set of run-time functionally reconfigurable non-pipelined and pipelined operators. The main novelty of this work is simultaneous solving of the scheduling, binding and routing tasks. This frequently generates optimal results, which has been shown by extensive experiments using the constraint programming paradigm. In order to show flexibility of our environment, we have used it in this article for optimization of application scheduling, binding and routing (the case of the non-pipelined execution model) and for design space exploration (case of the pipelined execution model).

Keywords: Binding, Compilation for Reconfigurable Architecture, Constraint Programming, Embedded Platform for Multimedia, Reconfigurable Data-Path, Routing, Scheduling

DOI: 10.4018/jertcs.2012010101
INTRODUCTION

Nowadays, multimedia applications play an important role in our daily lives because of their presence in many consumer electronics embedded systems. These applications are characterized by data-intensive and complex computation requirements leading designers to define application-specific architectures. But, due to a strong competition between actors on the consumer electronics market and constantly renewing and adjusting worldwide standards, their lifetime tends to shorten the time-to-market. Solutions combining flexibility and efficiency, in terms of computing power, silicon area and energy consumption are mandatory.

For about ten years, they have been containing, more and more frequently, run-time reconfigurable data-paths (Baron, 2004; Cervero, 2009) tailored to selected classes of applications. The interest for reconfigurable architectures and systems with hardware elements that can be reconfigurable, possibly even dynamically, on-the-fly and on demand, is growing. Their flexibility and their potential performance in a wide range of performance metrics (power consumption, execution time) have allowed them to become a vehicle of choice in many computing domains. Despite this potential, reconfigurable architectures are extremely hard to program. The lack of programming tools and effective methodologies generally result in long and error-prone mapping processes. This could result in a bad exploitation of their capabilities.

The challenge is to have software tools able to deploy application programs, written in established high-level languages, on these target architectures. The role of the compiler is to automatically generate high-performance executables, which can include both the instruction stream for the RISC microprocessor responsible for controlling the reconfigurable hardware and the configuration context for the reconfigurable hardware. The compilation determines the quality of executables, which impacts the performance of applications running on a reconfigurable processor.

The overall compiler flow of a traditional compiler for a coarse-grained reconfigurable architecture (CGRA) is as follows (Guo et al., 2005). Initially, the given application is written in a high level programming language. Then the source code is analyzed, transformed and optimized to obtain the desired intermediate representation (IR) in the form of data flow graphs (DFG). During IR steps, a possible parallelism is extracted. This process is beneficial because the more parallelism we can extract from the source code, the better the performance of the compilation will be. After the DFG is available, it is analyzed during the clustering phase to extract a set of clusters with respect to the target architecture. Then these clusters are mapped to the target architecture during the mapping phase. The scheduling phase determines which cluster will be executed in which clock cycles. During this phase, the available resources and memory are utilized properly to achieve maximum benefit. During the routing phase, operand values are routed using the available interconnection network. After this phase, the desired code is generated and converted to binary form for execution.

Compiling applications to CGRAs, after the source code of the target application has been transformed and optimized to a suitable intermediate representation, is a combination of three tasks: scheduling, placement or binding, and routing. Scheduling assigns time cycles to the operations for execution. Placement (binding) places these scheduled operation executions on specific processing elements. Routing plans the movement of data from producer processing elements to consumer processing elements using the interconnect structure of the target architecture. Scheduling, binding and routing are not trivial tasks. To tackle this problem, we developed a generic framework based on the constraint programming paradigm (CP) that enables simultaneous application scheduling, binding and routing.

In this article, this framework targets our run-time reconfigurable, operator based, multimedia architecture called ROMA. ROMA (Menard et al., 2009) is a reconfigurable processor.
Increasing Throughput of MIMO LTE-Advanced using Carrier Aggregation Feature


A Two Step Multi-Carrier Proportional Fair Scheduling Scheme for Cloud Radio Access Networks