Embedded Tutorial

Compilation Techniques for CGRAs: Exploring All Parallelization Approaches

Organizers:

Tom Vander Aa
Imec, Leuven, Belgium
vanderaa@imec.be

Praveen Raghavan
Imec, Leuven, Belgium
ragha@imec.be

Speakers:

Scott Mahlke,
University of Michigan

Bjorn De Sutter
Ghent University

Aviral Shrivastava,
Arizona State University

Frank Hannig,
University of Erlangen-Nuremberg

OVERVIEW

Coarse-Grained Reconfigurable Array (CGRA) processors accelerate inner loops of applications by exploiting instruction-level parallelism (ILP) and in some cases also data-level and task-level parallelism (DLP & TLP). The aim of this tutorial is to give insight in CGRA architectures and their compilation techniques to exploit parallelism. These topics will be covered:

- Polymorphic pipeline arrays, expanding coarse-grained arrays beyond innermost loops (Scott Mahlke, University of Michigan)
- Code-generation for coarse-grained arrays: flexibility and programmer productivity (Bjorn De Sutter, Ghent University)
- Memory-aware compilation techniques for CGRAs (Aviral Shrivastava, Arizona State University)
- Retargetable Mapping of Loop Programs on Coarse-grained Reconfigurable Arrays (Frank Hannig, University of Erlangen-Nuremberg)

Categories and Subject Descriptors

C.1.4 [**Computer Systems Organization**]: Processor Architectures – *parallel architectures.*

General Terms: Design, Performance

Polymorphic pipeline arrays, expanding coarse-grained arrays beyond innermost loops, *Scott Mahlke*

Mobile computing in the form of smart phones, netbooks, and personal digital assistants has become an integral part of our everyday lives. High definition audio and video as well as 3D graphics provide richer interfaces and compelling capabilities that will differentiate future products. However, multimedia algorithms are complex featuring substantial control flow and variable computational requirements where execution time is not dominated by innermost vector loops. The challenge is creating efficient computing systems that can simultaneously achieve high performance, energy efficiency, and programmability to support a wide array of mobile media algorithms.

In this talk, we will first examine the coarse-grain reconfigurable architecture, or CGRA, as an effective substrate for mobile media computing. CGRAs are a two dimensional grid of simple processing elements that are managed exclusively by software. CGRAs were initially designed to accelerate innermost loops, but this can be too restrictive for applications not dominated by a single kernel. To expand the scope beyond innermost loops, we introduce a generalization of the CGRA called a polymorphic pipeline array or PPA.

The PPA is designed with flexibility and programmability as first-order requirements to enable the hardware to be dynamically customizable to the application. PPAs exploit pipeline parallelism found in streaming applications to create a coarse-grain hardware pipeline to execute streaming media applications. PPA resources are allocated to each stage depending on its size and ability to exploit fine-grain parallelism.

Code-generation for coarse-grained arrays: flexibility and programmer Productivity, *Bjorn De Sutter*

High performance, low power consumption, high flexibility and high developer productivity are three important requirements of computing devices. This talk focuses on flexibility and programmer productivity of CGRA architectures. We present the state of the art in backend compiler technology for CGRA architectures, focusing on architecture modeling and on ILP code generation technology. We also discuss open challenges relating to retargetability, performance portability, and design space exploration.

Memory-aware compilation techniques for CGRAs, *Aviral Shrivastava*

Coarse-Grained Reconfigurable Arrays (CGRAs) are a very promising platform, providing up to 10-100 MOps/mW of power

Copyright is held by the author/owner(s).
CODES/ISSS'10, October 24–29, 2010, Scottsdale, Arizona, USA.
ACM 978-1-60558-905-3/10/10.

efficiency and still retain software programmability. However, this cardinal promise of CGRAs critically hinges on the effectiveness of application mapping onto CGRA platforms. Compilation for CGRAs has traditionally focused on two issues: i) placing operations (such as arithmetic/logic, multiplication, and load/store) of a loop kernel onto the PE array, and ii) guaranteeing the data flow, or communication, between operations using the existing interconnection resources. Another important dimension that critically affects the goodness of this mapping is where to place data, typically array variables, in the local memory. Most existing compilers assume a single unified memory architecture, however that is rarely the case in real CGRA architectures, and CGRA data memory typically comprises of several banks, each bank connected to only one row of PEs.

The focus of this talk is on demonstrating the importance of data placement in such architectures, and discuss some techniques on memory architecture cognizant data mapping techniques in the compiler.

Retargetable Mapping of Loop Programs on Coarse-grained Reconfigurable Arrays, *Frank Hannig*

In this work we consider tightly-coupled processing on multiple levels of parallelism: Array level parallelism (loop level), instruction level parallelism, sub-word parallelism, functional and software pipelining in CGRAs. We present a methodology that can be used to parallelize and map given loop programs on a highly parameterizable architecture template in order to achieve throughput optimized designs. That is, the architectural parameters (e.g., number of processing elements, communication channels, memory) have to be matched with the program transformation and scheduling parameters of the mapping method.

The considered architectures and the mapping methodology are designed to keep control overhead at a minimum. Moreover, we will present first ideas for symbolic mapping of loop programs. That is, the number of available computing resources is not fixed at compile-time but is only known at run-time. By this self-adaption, more flexibility and significantly reduced times compared with program re-compilation can be achieved, which is a step towards handling the complexity of future architectures.