Analysis of Inner-Loop Mapping onto Coarse-Grained Reconfigurable Architectures Using Hybrid Particle Swarm Optimization

Rani Gnanaolivu, Memorial University of Newfoundland, Canada
Theodore S. Norvell, Memorial University of Newfoundland, Canada
Ramachandran Venkatesan, Memorial University of Newfoundland, Canada

ABSTRACT

Coarse-Grained Reconfigurable Architectures (CGRAs) have gained currency in recent years due to their abundant parallelism and flexibility. To utilize the parallelism found in CGRAs, this paper proposes a fast and efficient Modulo-Constrained Hybrid Particle Swarm Optimization (MCHPSO) scheduling algorithm to exploit loop-level parallelism in applications. This paper shows that Particle Swarm Optimization (PSO) is capable of software pipelining loops by overlapping placement, scheduling and routing of successive loop iterations and executing them in parallel. The proposed algorithm has been experimentally validated on various DSP benchmarks under two different architecture configurations. These experiments indicate that the proposed MCHPSO algorithm can find schedules with small initiation intervals within a reasonable amount of time. The MCHPSO scheduling algorithm was analyzed with different topologies and Functional Unit (FU) configurations. The authors have tested the parallelizability of the algorithm and found that it exhibits a nearly linear speedup on a multi-core CPU.

Keywords: Coarse-Grained Reconfigurable Architectures, Inner Loop, Loop-Level Parallelism, Modulo Scheduling, Mutation Operator, Particle Swarm Optimization

INTRODUCTION

Reconfigurable systems (Abielmona, 2005) have drawn increasing attention from both academic and commercial researchers in the past few years because they combine flexibility with efficiency and upgradability (Todman, Constantinides, Wilton, Mencer, Luk, & Cheung, 2005). Among reconfigurable architectures, many Coarse-Grained Reconfigurable Architectures (GGRAs) have been proposed as an alternative
to FPGA-based systems (Mei, Vernalde, Verkest, Man, & Lauwereins, 2003). CGRAs consist of programmable coarse-grained Functional Units (FUs) which support a predefined set of word-level operations; a programmable interconnection network; a configuration memory; and a controller (Vassiliadis & Soudris, 2007). Unfortunately the available parallelism has been exploited by a few automated design and compilation tools (Mei, Vernalde, Verkest, Man, & Lauwereins, 2003).

The massive amounts of parallelism found in CGRAs can be used to speed up time critical loops of an application. This can be achieved by modulo scheduling (Hatanaka & Bagherzadeh, 2007), which is a software pipelining technique that overlaps several iterations of a loop by generating a schedule for an iteration of the loop. Modulo scheduling uses the same schedule for subsequent iterations. Iterations are started at a constant interval called the initiation interval (II). The time taken to complete a loop of $n$ iterations is roughly proportional to $II$, thus the main goal of modulo scheduling is to find a schedule with as low an $II$ as possible.

Several heuristic techniques have been tried by researchers in solving the modulo scheduling problem. In this paper, we propose a modulo scheduling algorithm based on particle swarm optimization (PSO). We call this the modulo-constrained hybrid particle swarm optimization (MCHPSO) algorithm. PSO provides near optimal solutions with fast convergence and low execution time for various combinatorial and multidimensional optimization problems (Abdel-Kader, 2008). We have used a hybrid PSO with a mutation operator to decide the placement and scheduling decisions in CGRAs. The MCHPSO algorithm has been tested on the benchmarks (Texas Instruments Inc., 1995; Park, 2005; VLSI Design Laboratory, 2002). The benchmarks are derived from applications written in the C programming language.

Motivational Example

Figure 1 illustrates the compilation flow with a motivational example. Consider the architecture configuration shown in Figure 1 (a), and the DFG represented in Figure 1 (c). The architecture components in Figure 1 (a) are input ports (I), functional units (FU), write ports (WP), read ports (RP), register files and Register File (RF). Figure 1 (b) shows an RRG created by the given target applications in reasonable time, with efficient utilization of resources.

The rest of this paper is organized as follows: An overview of compilation and background is given first. Modulo scheduling and PSO related work are discussed next. Our proposed PSO-based modulo scheduling algorithm (MCHPSO) is explained and the experiments conducted are discussed. Finally we present the conclusion and future work.

BACKGROUND

In this paper, we propose an algorithm for modulo scheduling of loops to be mapped onto CGRAs. At the same time as it schedules, the algorithm places — assigns operations to FU — and routes — finds paths through space and time for data.

Each source program is converted from an imperative program to a data flow graph (DFG). The given target architecture (TA) is represented by a graph containing all the necessary information such as the number of resources, capacity and interconnections as well as other specific information for each resource. The generic TA graph representation was designed to allow a wide range of architectures. The TA is replicated for many time cycles to form the Routing Resource Graph (RRG), an internal time-space graph representation.

The mapping algorithm MCHPSO maps each node of the DFG to a node of the RRG and each edge of the DFG to a path in the RRG. The generated scheduled code of the loop exhibits a high degree of instruction level parallelism (ILP).

Motivational Example

Figure 1 illustrates the compilation flow with a motivational example. Consider the architecture configuration shown in Figure 1 (a), and the DFG represented in Figure 1 (c). The architecture components in Figure 1 (a) are input ports (I), functional units (FU), write ports (WP), read ports (RP), register files and Register File (RF). Figure 1 (b) shows an RRG created by
Chaos-Enhanced Firefly Algorithm with Automatic Parameter Tuning

[www.igi-global.com/article/chaos-enhanced-firefly-algorithm-automatic/62643?camid=4v1a](www.igi-global.com/article/chaos-enhanced-firefly-algorithm-automatic/62643?camid=4v1a)