Microprocessors and Microsystems 38 (2014) 1012–1024

Contents lists available at ScienceDirect

Microprocessors and Microsystems

journal homepage: www.elsevier.com/locate/micpro

Automatic custom instruction identification for application-specific instruction set processors

Chenglong Xiao [a,*], Emmanuel Casseau [b], Shanshan Wang [a], Wanjun Liu [a]

[a] Liaoning Technical University, China
[b] University of Rennes I, IRISA, INRIA, France

ARTICLE INFO

Article history:
Available online 16 September 2014

Keywords:
Extensible processors
ASIPs
DFG
Custom instruction
Custom instruction enumeration
Custom instruction selection

ABSTRACT

The application-specific instruction set processors (ASIPs) have received more and more attention in recent years. ASIPs make trade-offs between flexibility and performance by extending the base instruction set of a general-purpose processor with custom functional units (CFUs). Custom instructions, executed on CFUs, make it possible to improve performance and achieve flexibility for extensible processors. The custom instruction synthesis flow involves two essential issues: custom instruction enumeration (subgraph enumeration) and custom instruction selection (subgraph selection). However, both enumerating all possible custom instructions of a given data-flow graph and selecting the most profitable custom instructions from the enumerated custom instructions are computationally difficult problems. In this paper, we propose efficient algorithms for custom instruction enumeration and custom instruction selection. Compared with previously proposed well-known enumeration algorithms, our approach can achieve a significant speedup while generating the identical set of all possible custom instructions or only connected custom instructions. Experimental results also show that a code size reduction rate up to 76% can be achieved for a set of computational intensive programs, and the speed-up achieved is up to 8.2×.

1. Introduction

Various electronic devices implemented with application-specific instruction set processors (ASIPs) can be found in the market. Many commercial ASIPs have been proposed by several vendors. Tensilica Xtensa [1], ARC [2], STMicroelectronic ST200 [3] and NIOS [4] are only some of the existing ASIPs. Instruction-set extensible processors that consist of an existing processor core extended with application-specific custom functional units (CFUs) make trade-offs between flexibility and efficiency.

A custom instruction is a complex instruction that encapsulates several basic instructions. A CFU is the hardwired implementation of a custom instruction. Generally, application programs from the same domain have similar structure. Using the same set of custom instructions across different application programs from the same domain is the flexibility provided by ASIPs [5]. As the basic operators in a CFU are parallelized or chained together, embedded processors may achieve considerable performance/energy efficiency by executing the custom instructions on CFUs [6]. In image processing, for example, hardwired CFUs such as dot product and sum of absolute differences are utilized to speedup the computations.

The key steps involved in a custom instruction synthesis flow for ASIPs are custom instruction enumeration and custom instruction selection. However, it is really time-consuming to enumerate custom instructions and select custom instruction from a given program's data-flow graph (DFG). Hence, a fully automated custom instruction synthesis flow is necessary.

Fig. 1 presents the overview of the proposed design flow. The design flow accepts application program written in C or C++ as input. The application program is firstly translated to a control data-flow graph (CDFG) using the compiler GECOS [7]. A CDFG is a graph that represents the control dependencies among a number of basic blocks and the data dependencies inside the basic blocks. Then, all the subgraphs (custom instructions are represented as subgraphs) satisfying the architectural constraints are enumerated. Based on the enumerated subgraphs, only a subset of subgraphs are selected as custom instructions according to different strategies (minimum run time, minimum number of custom instructions, minimum power consumption, etc.). After selecting the most profitable custom instructions, the source code is transformed to a new code by incorporating the selected custom

* Corresponding author.
E-mail addresses: chenglong.xiao@gmail.com (C. Xiao), emmanuel.casseau@irisa.fr (E. Casseau), celine.shanshan.wang@gmail.com (S. Wang), liuwanjun39@163.com (W. Liu).

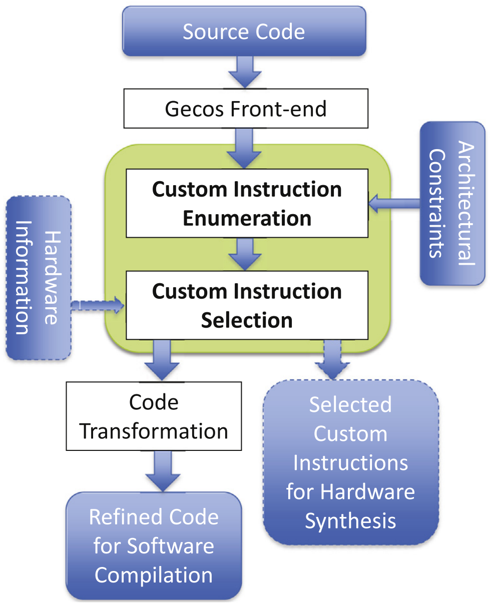
Source Code
Gecos Front-end
Custom Instruction Enumeration
Custom Instruction Selection
Architectural Constraints
Hardware Information
Code Transformation
Selected Custom Instructions for Hardware Synthesis
Refined Code for Software Compilation

Fig. 1. The design flow for automatic custom instruction identification.

instructions and this new code is provided to the compilation process. Finally, the set of selected custom instructions is provided to the hardware synthesis process to produce the corresponding hardware (CFU) that will be added to the ASIPs.

Both custom instruction enumeration and custom instruction selection are tackled in this paper, where the main contributions are:

- a very efficient custom instruction enumeration algorithm that can be tuned to enumerate all possible custom instructions or only connected custom instructions is proposed. Experiments with real world benchmarks show that our algorithm can achieve orders of magnitude speedup over the state-of-the-art algorithms;
- a heuristic algorithm and a genetic algorithm targeting a minimal number of custom instructions selection strategy are presented. As exact methods like branch-and-bound algorithm usually fail to give a solution in a reasonable time for most of benchmarks, the genetic algorithm that can give near-optimal solutions is proposed to evaluate the heuristic algorithm. The quality of the results obtained using the proposed heuristic algorithm is very close to the quality of the results produced by the genetic algorithm;
- results for a set of real world benchmarks showing that significant code size reduction (from 21% up to 76%) and performance improvement (from 1.3× up to 8.2×) can be achieved.

The remainder of the paper is organized as follows. In Section 2, the state-of-the-art is introduced. The problem formulation is given in Section 3. Section 4 presents the proposed algorithm for custom instruction enumeration. Custom instruction selection algorithms that aim at selecting minimal number of custom instructions are depicted in Section 5. Section 6 presents and evaluates the results of our proposed algorithms. Finally, conclusions are presented in Section 7.

2. Related work

A lot of related researches on custom instruction enumeration and custom instruction selection have been done in recent years. In this section, we start our review from custom instruction enumeration and then discuss custom instruction selection.

2.1. Custom instruction enumeration

Many previous researches focus on identifying custom instructions (subgraphs) under I/O ports constraint and convexity constraint. The authors [8] are the first ones that prove the exhaustive enumeration of subgraphs is inherently polynomial. A polynomial time subgraph enumeration algorithm is proposed in [9]. Several techniques have been proposed for an exhaustive enumeration of subgraphs, such as dynamic programming [10] and constraint programming [11,12]. However, these methods are not efficient when the application graphs become large. Yu et al. [13] build only connected feasible subgraphs by enumerating upward cones and downward cones. However, in this algorithm, a subgraph could be considered more than once. Authors of [14] first present an exhaustive algorithm based on a binary decision tree under convexity and I/O constraints. Pozzi et al. [15] further improved this algorithm by adding a pruning criterion based on the number of permanent inputs. Another algorithm is proposed in [8]. The algorithm enumerates both connected feasible subgraphs and disjoint feasible subgraphs. A more efficient algorithm that takes advantage of topological property of data flow graph is proposed in [16,17].

With the use of inner state registers and shadow registers, the maximal convex subgraphs (MCSs) can also be supported. Pothineni et al. [18] were the first ones to propose an algorithm for MCS enumeration. The proposed algorithm is based on an incompatibility graph. However, the algorithm only generates connected MCSs. In [19], the MCS enumeration problem is reformulated as a maximal clique enumeration problem after grouping equivalence nodes and building cluster graphs. Atasu et al. [20] proved that the number of MCSs is bounded by $2^{|V_I|}$, where V_I is the set of invalid nodes in the DFG. A top-down manner algorithm proposed in [21] solves the MCS enumeration problem efficiently by a division operation on the DFG. To further speedup the enumeration process, a more efficient algorithm is presented in [22]. The algorithm enumerates all MCSs using a sandwich manner that combines the advantage of the bottom-up manner and the top-down manner.

2.2. Custom instruction selection

According to the user's constraints, the custom instruction candidates are selected either due to the high frequency of occurrences in the application or due to their high performance or due to the number of nodes inside compared to other custom instructions. Therefore, developing a good custom instruction selection method is quite vital for highest gain in performance or code size reduction for the application. Many works were proposed for selecting custom instructions under some of the above constraints with different strategies (minimal number of custom instructions, minimal execution time, etc.). Selecting bigger subgraphs such as MCSs may result in better performance improvement [18–21]. In general, bigger subgraphs have less reoccurrences in an application graph. Hence, in practice, small subgraphs seem to be more interesting if we take into account CFU reuse (area cost).

An exact algorithm [23] targeting to cover each node with minimal number of custom instructions converts the selection problem to a unate covering. A novel method was presented in [11], the authors try to solve the problem by constraint programming. The selection of custom instructions is carried out with two respective scheduling strategies: time-constrained scheduling or resource-constrained scheduling. This method assumes that all nodes are not able to be covered by more than one subgraph. Yu et al. [24] proposed an optimal method based on ILP (integer linear