Microprocessors and Microsystems 45 (2016) 176–186

Contents lists available at ScienceDirect

Microprocessors and Microsystems

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

A comparison of heuristic algorithms for custom instruction selection

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

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

A R T I C L E I N F O

Article history:
Received 16 July 2015
Revised 21 January 2016
Accepted 2 May 2016
Available online 10 May 2016

Keywords:
Heuristic algorithms
Data-flow graph
Extensible processors
Custom instructions
Custom function units

A B S T R A C T

Extensible processors with custom function units (CFU) that implement parts of the application code can make good trade-off between performance and flexibility. In general, deciding profitable parts of the application source code that run on CFU involves two crucial steps: subgraph enumeration and subgraph selection. In this paper, we focus on the subgraph selection problem, which has been widely recognized as a computationally difficult problem. We have formally proved that the upper bound of the number of feasible solutions for the subgraph selection problem is $3^{n/3}$, where n is the number of subgraph candidates. We have adapted and compared five popular heuristic algorithms: simulated annealing (SA), tabu search (TS), genetic algorithm (GA), particle swarm optimization (PSO) and ant colony optimization (ACO), for the subgraph selection problem with the objective of minimising execution time under non-overlapping constraint and acyclicity constraint. The results show that the standard SA algorithm can produce the best results while taking the least amount of time among the five standard heuristics. In addition, we have introduced an adaptive local optimum searching strategy in ACO and PSO to further improve the quality of results.

1. Introduction

Due to a good trade-offs between performance and flexibility, extensible processors have been used more and more in embedded systems. Such examples include Altera NIOS processors, Xilinx MicroBlaze and ARC processors. In extensible processors, a base processor is extended with custom function units that execute custom instructions. Custom function units running the computation-intensive parts of application can be implemented with application-specific integrated-circuit or field-programmable gate-array technology. A custom instruction is composed of a cluster of primitive instructions. In general, the critical parts of an application are selected to execute on CFUs and the rest parts are run on the base processor. With the base processor, the flexibility can be guaranteed. As selected custom instructions usually impose high data-level parallelism [1], executing these custom instructions on CFUs may significantly improve the performance.

Deciding which segments of code to be executed in CFUs is a time-consuming work. Due to time-to-market pressure and requirement for lower design cost of extensible processors, automated custom instruction generation is necessary. Fig 1. shows the design flow for automatic custom instruction generation. Starting with provided C/C++ code, a front-end compiler is called to produce the corresponding control data-flow graph (CDFG). Then, the subgraph enumeration step enumerates all subgraphs (graphic representation of custom instructions) from DFGs inside CDFG that satisfy the micro-architecture constraints and user-defined constraints [2–5]. Next, in order to improve the performance, the subgraph selection step selects a subset of most profitable subgraphs from the set of enumerated subgraphs, while satisfying some constraints (e.g. non-overlapping constraint and area constraint). Based on the selected subgraphs, the subgraphs with equivalent structure and function are grouped together. The behavioral descriptions of the selected custom instructions along with the code incorporating the selected custom instructions are finally produced. The crucial problems involved in custom instruction generation are: subgraph enumeration and subgraph selection. In this paper, we focus on the subgraph selection problem. The main contributions of this paper are:

• formulating the subgraph selection problem as a maximum cliques problem;

• an upper bound on the number of feasible solutions is $3^{n/3}$, where n is the number of subgraph candidates;

• adaptation of five popular heuristic algorithms for the subgraph selection problem;

• detailed comparison of these five algorithms in terms of search time and quality of the solutions. Furthermore, we extend

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

http://dx.doi.org/10.1016/j.micpro.2016.05.001
0141-9331/© 2016 Elsevier B.V. All rights reserved.

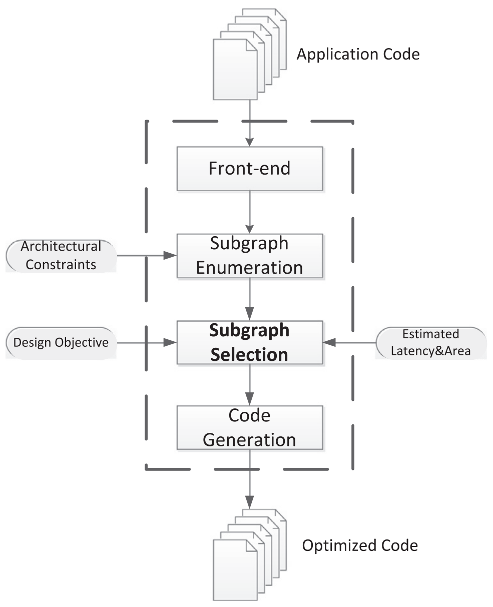

Fig. 1. The design flow for custom instruction generation.

PSO and ACO algorithms to include an adaptive local optimum searching strategy. Results show that the quality of the solutions can be further improved.

The rest of the paper is organized as follows. Section 2 reviews the related work on the subgraph selection problem, while section 3 formally formulates the subgraph selection problem as a weighted maximum problem and presents an upper bound on the number of feasible solutions. Section 4 discusses the proposed SA, TS, GA, PSO and ACO algorithms for the subgraph selection problem. In section 5, the experiments with practical benchmarks are performed to evaluate the algorithms in terms of search time and quality of the solutions. Some discussion on the future work is given in section 6. Section 7 concludes this paper.

2. Related work

Prior to review the related work on custom instruction selection (or subgraph selection), we start by discussing algorithms for custom instruction enumeration.

Plenty of previous research on custom instruction enumeration have been presented in recent years. The custom instruction enumeration tries to enumerate a set of subgraphs from the application graph with respect to the micro-architecture constraints or user-defined constraints. The custom instruction enumeration process is usually very time-consuming. For example, it was proven in [6] that the number of valid subgraphs satisfying the convexity and I/O constraints is n^{I+O}, where n is the number of nodes in the application graph, and I and O are the maximum number of inputs and outputs respectively. A set of algorithms with $O(n^{I+O})$ time complexity can be found in existing literature [3,5,7]. Some other algorithms targeted to only enumerate maximal convex subgraphs by relaxing the I/O constraints have also been proposed [8,9]. However, the number of maximal convex subgraphs can still be exponential. In [10], the number of maximal convex subgraphs was formally proven to be $2^{|F|}$, where F is the set of forbidden nodes in the application graph. It can be observed from previous experiments that the subgraph enumeration step may produce tens of thousands of subgraphs for both enumeration of convex subgraphs under I/O constraints and enumeration of maximal convex subgraphs.

Subgraph selection is the process of selecting a subset of profitable subgraphs from the set of enumerated subgraphs as custom instructions that will be implemented in custom function units. As the number of candidate subgraphs can be very large, the subgraph selection problem is generally considered as a computationally difficult problem [1]. Previous work on subgraph selection problem can be grouped into two categories:

Optimal algorithms: Some of previous work try to find exact solution for the subgraph selection problem. For example, the authors of [11] propose a branch-and-bound algorithm for selecting the optimal subset of custom instructions. Cong et al. solve this problem by using dynamic programming [12]. Some other researchers address the subgraph selection problem using integer linear programming (ILP) or linear programming (LP) [13–15]. These methods treat the selection problem as maximizing or minimizing a linear objective function, while each subgraph is represented as a variable, which has a boolean value. The area constraint or non-overlapping constraint is expressed as linear inequality. These formulations are provided to an ILP or LP solver as input. Then, the solver may produce a solution. Similar to LP method, Martin et al. try to deal with the selection problem by using constraint programming method [16,17]. However, due to the complexity of the subgraph selection problem, these methods may fail to produce a solution when the size of the application graph becomes large.

Although the subgraph selection problem is widely considered as a computationally difficult problem, it still lacks of an exact upper bound on the number of feasible solutions. In this article, we first formulate the subgraph selection problem under non-overlapping constraint as a maximum clique problem. Then, with this formulation, the upper bound on the number of feasible solutions is formally proved to be $3^{n/3}$, where n is the number of subgraphs.

Heuristic algorithms: Since optimal algorithms are usually intractable when the problem size is large, it is necessary to solve the problem with heuristic algorithms. Kastner et al. heuristically solve the problem by contracting the most frequently occurring edges [18]. Wolinski and Kunchcinski propose a method that selects candidates based on the occurrence of specific nodes [19]. A method attempting to preferentially select the subgraphs along the longest path of a given application graph has been proposed by Clark et al [20].

In recent years, due to good scalability and trade-offs between search time and quality of the solutions, many meta-heuristic algorithms have been proposed to solve the subgraph selection problem. As an example, a genetic algorithm (GA) is introduced to overcome the intractability of the subgraph selection problem [3]. The experiments show that the genetic algorithm may produce near-optimal solutions for problems with different sizes. In [21], a tabu search algorithm (TS) is adapted to solve the subgraph selection problem. The authors report that the tabu search algorithm can provide optimal solutions for medium-sized problems. It can still produce solutions when optimal algorithms fail to produce solutions for large-sized problems. Other meta-heuristic algorithms like particle swarm optimization (PSO) [22], ant colony optimization (ACO) [23,24] and simulated annealing algorithms (SA) [25] have been also introduced or adapted to address the subgraph selection problem or similar problems. However, the comparisons between these popular meta-heuristic algorithms are still missing in the existing literature. Thus, the main objective of this paper is to adapt the SA, TS, GA, PSO and ACO algorithms to solve the subgraph selection problem, and compare these algorithms in terms of runtime and quality of the results.

3. Problem formulation

The definition of the subgraph selection problem discussed in this paper is based on the following notations.