Microprocessors and Microsystems 38 (2014) 681–691

Contents lists available at ScienceDirect

Microprocessors and Microsystems

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

Implementation-aware selection of the custom instruction set for extensible processors

Amir Yazdanbakhsh [a,1], Mehdi Kamal [a], Sied Mehdi Fakhraie [a], Ali Afzali-Kusha [a,*], Saeed Safari [a], Massoud Pedram [b]

[a] School of Electrical and Computer Engineering, University of Tehran, Iran
[b] EE Department, University of Southern California, USA

ARTICLE INFO

Article history:
Available online 6 June 2014

Keywords:
Extensible processor
Design space exploration
Hardware/software codesign
Application specific instruction set processors
Microarchitecture

ABSTRACT

This paper presents an approach for incorporating the effect of various logic synthesis options and logic level implementations into the custom instruction (CI) selection for extensible processors. This effect translates into the availability of a piecewise continuous spectrum of delay versus area choices for each CI, which in turn influences the selection of the CI set that maximizes the speedup per area cost (SPA) metric. The effectiveness of the proposed approach is evaluated by applying it to several benchmarks and comparing the results with those of a conventional technique. We also apply the methodology to the existing serialization algorithms aimed at relaxing register file constraints in multi-cycle custom instruction design. The comparison shows considerable improvements in the speedup per area compared to the custom instruction selection algorithms under the same area-budget constraint.

© 2014 Elsevier B.V. All rights reserved.

1. Introduction

Increased rate of embedded applications calls for high performance, low power consumption, flexibility, and cost efficiency of systems that realize such applications. Extensible processors have emerged in the field of embedded computing as a promising approach to remedy many shortcomings of ASICs and general-purpose processors [1]. This approach exploits a simple general-purpose processor and extends its instruction set architecture with beneficial custom instructions (CIs) to provide flexibility and high performance [2]. In designing these processors, the runtime behavior of applications in the target domain are analyzed to determine the critical code segments of the applications. Based on this information, the base processor is augmented with a number of special instructions (custom instructions) for the computationally intensive parts of the code.

Various algorithms have been developed to identify and select the CIs in order to minimize the execution time of the underlying applications in the target domain [2–7]. In the CI identification phase, a pool of feasible CIs is determined subject to meeting pre-defined constraints (i.e., I/O constraint) whereas in the CI selection phase, a subset of identified CIs is chosen based on the specified objective function(s) and subject to constraint(s) on the layout area.

During the high-level synthesis process, different implementations of each primitive operation are explored in order to generate a area-delay Pareto optimal curve for that operation [8]. Previous works on CI selection for extensible processors has considered only a single point in the design space and ignored the role of subsequent logic synthesis and optimizations on the physical implementation of the design when identifying the most effective CIs.

In this paper, we propose a framework to improve the design of extended processors by considering different implementations of a primitive for a selected custom instruction. This is achieved by considering the Pareto optimal curve (delay versus area) of the CIs. Using this method, we are able to determine the best implementation where the area usage of the CI is minimum while the propagation delay of the CI does not violate the propagation delay constraint. This exploration which enables us to employ more CIs in a predefined area budget and also, more speed gain, is performed before the CI selection phase of the design flow. The results show that applying the technique gives rise to a considerable improvement in the speed enhancement of the extended processors compared to the case of the conventional design approach in which fixed delay and area is considered for each primitive operation. To the best of our knowledge, this investigation has not

* Corresponding author. Tel.: +98 2182084920; fax: +98 2188778690.
E-mail addresses: a.yazdanbakhsh@gatech.edu (A. Yazdanbakhsh), mehdikamal@ut.ac.ir (M. Kamal), fakhraie@ut.ac.ir (S.M. Fakhraie), afzali@ut.ac.ir (A. Afzali-Kusha), saeed@ut.ac.ir (S. Safari), pedram@usc.edu (M. Pedram).
[1] Present address: College of Computing, Georgia Institute of Technology, USA.

http://dx.doi.org/10.1016/j.micpro.2014.05.007
0141-9331/© 2014 Elsevier B.V. All rights reserved.

reported for ASIPs in the literature. Additionally, to have a reasonably low runtime, we had to use an efficiently fast heuristic technique, which is called WALK, to find (near) optimal implementation of each CI. This problem can be reduced to Knapsack problem, which is an NP-hard algorithm.

The remainder of this paper is organized as follows. In Section 2, related works are briefly reviewed while motivation of the work along with the problem definition and formulation are presented in Section 3. Section 4 describes the proposed approach and algorithm. Experimental setup and results are discussed in Section 5. Finally, Section 6 concludes the paper.

2. Related works

There are several works in the literature focusing on CI identification and selection algorithm (see, e.g., [2,10–18]). In [2], an enumeration algorithm aimed at generating all valid CIs considering just convexity and I/O constraints was presented. The concept of binary decision tree was utilized to search among all valid sub-graphs. Each internal node represented a potential sub-graph, which was analyzed based on the specified constraints to identify the validity of the enumerated CIs. The inclusion or exclusion of a node in a sub-graph was distinguished by the movement toward the right or the left branch, respectively. An approach similar to [2] and an exact algorithm to enumerate the entire feasible CIs in a reasonable time were provided in [10].

Different approaches to reduce the computation time in the CI identification phase have been proposed in [11–15]. While the architectural space is comprehensively explored in these works, the selection of the CIs is accomplished based on a constant area-delay table derived from the synthesis tool regardless of logic-level implementation of the primitive cells. Different algorithms for high-level CI identification for extensible processors have been proposed in [16–18]. In [19], the arithmetic operations of selected custom instructions are optimized. In this work, to improve the speedup, the normal hardware blocks were replaced by the delay-optimized ones.

In [20], a design flow for reconfigurable ASIPs (rASIPs) has been proposed. In the proposed design flow, where the processor was described by using LISA language, the custom instructions were extracted for mapping them to a coarse grained reconfigurable architecture. In this work, the CI extraction method of [21], which did not consider the area usage of the CIs during the selection phase, was used. A framework for performance and area trade-off evaluation in the CI extraction has been proposed in [23]. The information about how the area usage of each identified CI has been extracted was not presented in detail. Clearly, no area usage optimization of the identified CIs before the selection phase has been utilized. In [24], a reconfigurable transparent accelerator based on the look-up table was proposed. Designing this accelerator, called Programmable Carry Function Unit (PCFU), was the main focus of the paper. In [25], similar to [24], a transparent accelerator, named Configurable Compute Accelerator (CCA), has been proposed.

An ILP-based CI identification framework, which extracts CIs from the critical code segment, has been proposed in [27]. The extraction was performed using the available data bandwidth and transfer latencies between custom logic and a baseline processor. In [28], a method to expedite CI generation from the high-level application descriptions was discussed. An approach for increasing the number of I/O ports of the CFU to access the General Purpose Registers (GPRs) has been suggested in [29]. It was based on the existing idea of register clustering in VLIW processors without a significant increase in the size of the GPR files. In [30], an automated ISE synthesis, which consider both the user-specified and processor-specific constraints have been proposed. The authors did not provide the details of the CI area usage estimation and the way that the usage is considered in the selection phase. In [6], a framework for estimating the area utilization and latencies of custom instructions on lookup-table based commercial FPGAs was proposed. In [21], multiple implementations per each special (custom) instruction were added to the extensible processor. A run-time system was proposed to dynamically select the appropriate variation of the special instruction based on the available hardware resources. This work introduced a novel run-time adaptive extensible processor to increase the hardware usage efficiency.

While the aforementioned works have introduced novel algorithms and architectures to increase the flexibility and reconfigurability of custom instruction selection and implementation, they have not considered different synthesis constraints (in terms of area and delay) for each custom instruction during the CI selection phase in the ASIP design flow. We should mention that the idea of using different area-delay implementation of primitives has been used in other fields of digital circuit design such as high level synthesis [8] and reconfigurable architecture design [20]. In these fields, the objective functions and as well the granularity of the exploration are different than those used in the ASIP design flow considered in this work.

3. Motivation and problem formulation

In this section, first, we describe the motivation of the work by an example. Next, key concepts in the field of custom instruction are formally presented. Also, the problem of finding a subset of CIs that maximizes the total speedup per area while satisfying an area budget constraint is formulated.

3.1. Motivation

The CI selection algorithms select a subset of instructions from the generated CIs in the CI identification phase such that the *speedup per area* (SPA) metric is maximized while an area budget constraint is not violated [2,6]. Previous works consider a constant value for the delay and area of the primitive operations. Whereas, we consider different logic implementations (different delays and areas) of the primitive operations in the algorithm to maximize the SPA parameter. Note that the SPA merit function is utilized in the CI selection phase which is after the CI identification phase. The proposed technique is applied in the stage between identification and selection phases and its efficiency is independent from the merit function used in the selection phase. Hence, in this work, without loss of generality, we use the SPA metric for the CI selection under a predefined area budget.

Fig. 1 shows two custom instructions generated from a data flow graph example under micro-architectural constraints. The CI selection algorithm should make a decision between these two CIs and select the one which is optimum in terms of the SPA metric. In the case of the first CI, we assume that the area and delay are fixed, and hence, SPA is fixed too. In the case of the second CI, the (32-bit) adder primitive is synthesized under different delay constraints assuming fixed areas and delays for the other primitives. Different delay constraints are synthesized with different area values due to different logic implementations. The area versus delay characteristic for this primitive is depicted in Fig. 2. The characteristic is an area-delay Pareto optimal curve. As shown in this figure, the area of the synthesized 32-bit adder decreases as the delay constraint increases. The discontinuity in the characteristic originates from the use of two micro-architecture implementations of the adder (carry look-ahead adder on the left and ripple-carry adder on the right.)

Download English Version:

https://daneshyari.com/en/article/462579

Download Persian Version:

https://daneshyari.com/article/462579

Daneshyari.com