Journal of Systems Architecture 60 (2014) 179–193

Contents lists available at ScienceDirect

Journal of Systems Architecture

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

Studying the code compression design space – A synthesis approach

Sreejith K. Menon *,[1]

Mathworks, 3 Apple Hill Drive, Natick, MA 01760, United States

ARTICLE INFO

Article history:
Available online 14 November 2013

Keywords:
Code compression
High level synthesis
Logic synthesis
Power estimation
Embedded systems

ABSTRACT

Embedded domain has witnessed the application of different code compression methodologies on different architectures to bridge the gap between ever-increasing application size and scarce memory resources. Selection of a code compression technique for a target architecture requires a detailed study and analysis of the code compression design space. There are multiple design parameters affecting the space, time, cost and power dimensions. Standard approaches of exploring the code compression design space are tedious, time consuming, and almost impractical with the increasing number of proposed compression algorithms. This is one of the biggest challenges faced by an architect trying to adopt a code compression methodology for a target architecture. We propose a novel synthesis based tool-chain for fast and effective exploration of the code compression design space and for evaluation of the tradeoffs. The tool-chain consists of a frontend framework that works with different compression/decompression schemes and a backend with high-level-synthesis, logic-synthesis, and power estimation tools to output the critical design parameters. We use the tool-chain to effectively analyze different code compression/decompression schemes of varying complexities.

1. Introduction

The size and complexity of embedded applications have been increasing consistently over the past few decades to meet the manifold demands of end users. With the availability of new tools and frameworks assisting the users to design, develop and verify complex applications, code volume has dramatically increased from a few thousands of lines of code to tens of millions of lines of code. This trend is believed to continue in the future with the use of embedded processors in more advanced and emerging fields and domains [27]. But this persistent growth in embedded software is constrained by the requirement for smaller memory size which helps in reducing the die-size, power consumption, and the overall cost. Code compression has been suggested as a solution to reconcile these contradictory requirements [2]. A general-purpose code compression scheme tries to improve the code density by replacing an instruction stream with a more compact compressed stream of instructions, with decompression being performed dynamically during program execution. Object code compression schemes can be broadly classified into (i) statistical and (ii) dictionary schemes. Statistical compression schemes, which include Huffman and arithmetic coding schemes, offer better compression results at the cost of increased decompression overhead. On the other hand, dictionary based schemes employ comparatively simple decompression steps, which make them suitable candidates for applications where program execution time is paramount. A general purpose code compression algorithm should have the ability to achieve fast address resolution at decompression time. Jump, branch, and call statements that alter the flow of control should be directed to the new address location in the compressed instruction stream. One way to achieve this is by compressing small blocks of instructions and by using a map table, generally called the Line Address Table (LAT), to hold the address mapping between the compressed block of instructions and the corresponding uncompressed block [2].

In the past few decades, a large number of compression algorithms have been proposed and studied on a wide variety of architectures resulting in a considerable reduction in the embedded application size. But our recent studies [23,26] have concluded that the efficacy of different algorithms varies with the type of architecture. i.e. a compression algorithm that gives impressive results on a particular architecture may not perform as well on a totally different architecture. Therefore an architect needs to experiment with different sets of compression/decompression algorithms for a target architecture. There are several dimensions to be considered while evaluating the code compression design space, the code size reduction being the most important one. Number of cycles consumed for decompressing a compressed block of instructions is

* Tel.:+1 508 647 8514.
E-mail address: sreejith.menon@mathworks.com
[1] This research was done when the author was working in the Implementation Group of Synopsys India Pvt Ltd.

1383-7621/$ - see front matter © 2013 Elsevier B.V. All rights reserved.
http://dx.doi.org/10.1016/j.sysarc.2013.11.001

also as critical because of its direct effect on the program execution time. Decompression hardware used for fast decoding of compressed block of instructions adds additional dimensions of cost and power. Hardware cost and power expended by the additional hardware should be evaluated to estimate the overall impact of a code compression scheme. Power consumption is considered to be an important constraint in the present day embedded devices. With the increased significance of power reduction in embedded systems, early evaluation of power consumption is critical to bound the overall power budget.

Exploring the code compression design space becomes difficult due to the unavailability of flexible tools or frameworks in the field of code compression that assist the architect to make informed decisions based on the tradeoffs between different dimensions. In our previous work [26], we had developed an advisory tool capable of computing compression ratios and power reduction parameters using execution traces. The advisory tool worked with cycle accurate simulators to determine the performance degradation due to on-the-fly decompression. But the tool was not capable of evaluating the number of cycles needed for decompression operation and it had to be provided as input. We take a totally different synthesis based approach in this paper to study the code compression design space with the design of an efficient tool-chain capable of analyzing different code compression schemes and evaluating the tradeoffs. The tool-chain is divided into two parts – a frontend framework that accepts different compression/decompression algorithms, and a backend with high-level synthesis, logic synthesis, and power estimation tools. Along with the automated synthesis of decompression hardware, the tool-chain outputs relevant design parameters at different stages. We have implemented the frontend framework and integrated different compression/decompression schemes of varying complexities and coupled the framework with the synthesis tools in the backend for effective synthesis of the decompression hardware. The commercially available Synopsys' Synphony C Compiler [38], Design Compiler [37], and Power Compiler [39] have been used in our experiments for high level synthesis, logic synthesis, and power estimation of ASIC designs. The different code compression schemes used in the tool-chain resulted in compression ratios in the range of 67–95% in VLIW and RISC architectures using a set of Mediabench [32] programs. The decompression overheads varied between 8 and 119 cycles for a compressed block of eight 32 bit instructions and the synthesized hardware ranged from 2849 μm^2 to 33775 μm^2 cell area using TSMC 65 nm standard cell library and 250 MHz clock frequency. The power estimation results of the decompression hardware extended from 0.71 mW to 4.09 mW.

The rest of the paper is organized as follows. Section 2 describes the high level approach of the tool-chain; Specific details of the frontend and backend of the tool-chain are discussed in Sections 3 and 4, respectively. Experiments and results are outlined in Section 5. Section 6 presents related work and comparison with existing work in the field. Section 7 contains the conclusion and possibilities for future work.

2. The tool chain

2.1. The problem

In industry and academia, there are different instances where a detailed and fast evaluation of tradeoffs of different design parameters of code compression is inevitable. Let us consider two different scenarios selected from the industry and academic perspectives.

- *Scenario 1:* Suppose a code compression scheme has to be selected for a specific architecture to improve the overall code density. The target architecture, set of applications, and desired range of design parameters are provided with a final goal of selecting one or more code compression schemes and their variants satisfying the given constraints.
- *Scenario 2:* Suppose a new code compression algorithm is proposed after a detailed research in the field. There is a requirement to compare the new algorithm with existing compression schemes and to report all the design parameters of the new scheme to evaluate its benefits.

In both scenarios, one or more compression schemes have to be developed, corresponding decompression hardware have to be designed and implemented, all design parameters of interest have to be analyzed, targeted optimizations have to be performed, and tradeoffs have to be studied. In the first scenario, a team of hardware and software experts along with analysts of design parameters will be formed to solve these complex issues. In the second scenario, a small research team may have to accomplish the same complex task. Space, time, cost, and power dimensions of the code compression design space need to be explored thoroughly in both the cases to meet the final goals. Some of the code compression schemes [20,22] are based on the Instruction Set Architecture (ISA) whereas others work independent of the ISA. In some other cases, based on the cost and complexity of the decompression hardware, parameters of the compression algorithm need to be varied. There have been prior works [4,6] evaluating the effects of achieving the decompression in software, but the need for fast decompression operation favors the hardware approach. The critical design parameters of code compression influencing the space–time-cost-power tradeoffs are

- Compression ratio – compressed code size/original code size.
- Decompression overhead per compressed block – number of cycles consumed for decompressing a block of instructions.
- Hardware cost – total cell area of the decompression hardware.
- Power usage – power expended by the decompression hardware.

The conventional way of estimating the above mentioned parameters requires extensive software implementation for evaluating the compression scheme along with the design and implementation of the decompression hardware using some Hardware Description Language (HDL). Analyzing the code size reduction of different schemes involves implementation and comparison of multiple code compression schemes and their variants. Decompression hardware also offers multiple options for memory ports, decompression engine types (serial vs. parallel, speculative vs. non-speculative), frequencies etc. Above all, an optimization targeted for a dimension of the design space may have catastrophic effects over other dimensions of the same space. All these factors and their interactions contribute to the complexity of the problem.

2.2. Our approach

Our current studies reveal that the complexity can be reduced to a great extent by abstracting out and reusing the different modules of code compression schemes and by using high-level synthesis, logic synthesis, and power estimation tools for automated synthesis of hardware from a higher level of abstraction and for gate level power estimation. The tool-chain shown in Fig. 1, an extension of our earlier work [31], tries to achieve this and outputs all the critical design parameters of code compression. By adding power evaluation to the work in [31], we ensure that the tool-chain covers all the design parameters of code compression. The frame-