A Class of Code Compression Schemes for Reducing Power Consumption in Embedded Microprocessor Systems

Luca Benini, Francesco Menichelli, and Mauro Olivieri, Member, IEEE Computer Society

Abstract—Compression of executable code in embedded microprocessor systems, used in the past mainly to reduce the memory footprint of embedded software, is gaining interest for the potential reduction in memory bus traffic and power consumption. We propose three new schemes for code compression, based on the concepts of static (using the static representation of the executable) and dynamic (using program execution traces) entropy and compare them with a state-of-the-art compression scheme, IBM’s CodePack. The proposed schemes are competitive with CodePack for static footprint compression and achieve superior results for bus traffic and energy reduction. Another interesting outcome of our work is that static compression is not directly related to bus traffic reduction, yet there is a trade off between static compression and dynamic compression, i.e., traffic reduction.

Index Terms—Microprocessor/microcomputer applications, low-power design, code compression.

1 INTRODUCTION

Processor cores find widespread usage in the vast majority of current system-on-chip (SoC) architectures. State-of-the-art embedded processors are based on high-performance RISC architectures, with on-chip caches and memory management unit, peripheral and input-output controllers. These processors and their software development tools are sometimes developed internally, but they are also often purchased as “intellectual-property components” by system integrators from specialized core developers.

RISC instruction sets are generally very regular, with fixed-length instructions: The most successful RISC cores on the market today have 32-bit instructions. Even though the adoption of a fixed-length instruction set eases compiler development and code portability, it hinders code compaction. Achieving sufficient instruction fetch bandwidth is therefore one of the most challenging issues in the design of a memory system for a RISC processor. Methods for incrementing effective memory bandwidth have been proposed [1], but on-chip instruction caches are probably the most commonly adopted technique for eliminating instruction fetch bottlenecks. Caches exploit the well-known principle of locality of memory accesses, but they do not address the inefficiencies in encoding the information contained in the instruction stream.

In recent years, several techniques have addressed the scarce “information density” of RISC ISAs through various forms of executable code compression (refer to [2] for a good overview). The basic idea is to store programs in a compressed form and decompress them on-the-fly at execution time. Code compression is clearly beneficial for memory size reduction because it shrinks the static memory footprint of executable code. However, memory becomes exponentially cheaper as technology improves and static memory footprint reduction may not be a concern for many applications. On the other hand, code compression is extremely beneficial for power as well because it reduces the energy consumed in reading instructions from memory and transferring them to the execution engine [3], [4], [5], [6], [7]. In contrast with memory cost, memory and memory transfer power is rapidly becoming dominant as technology progresses. Hence, code compression is desirable even when memory size is not a critical issue.

A large body of knowledge is available on lossless compression [8] and hardware for low-power and high-performance compression and decompression has been proposed [9]; however, code compression is characterized by a few distinctive requirements. First, it must be possible to decompress a program during execution starting from several points inside the program (i.e., branch destinations). Second, compression and decompression algorithms can be highly asymmetric because compression can be performed once for all (offline) when the executable is generated, while decompression is performed during program execution; thus, it should be fast and power efficient because its hardware cost must be fully amortized by the corresponding savings in memory size and power, without compromising performance.

A simple code compression approach, implemented in several commercial core processors, defines a “dense” instruction set, with a limited number of short instructions (e.g., ARM Thumb [10], MIPS16 [11], Tensilica Xtensa [12] instruction sets). The main shortcomings of this approach are: 1) performance penalty caused by the lack of several instructions in the dense instruction set; 2) supporting the