Computer Languages, Systems & Structures 37 (2011) 168–177

Contents lists available at ScienceDirect

Computer Languages, Systems & Structures

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

# Reducing memory space consumption through dataflow analysis

Ozcan Ozturk *

*Computer Engineering Department, Bilkent University, 06800 Bilkent, Ankara, Turkey*

ARTICLE INFO

*Article history:*
Received 22 August 2010
Accepted 18 July 2011
Available online 29 July 2011

*Keywords:*
Memory
Dataflow analysis
CFG
Compiler

ABSTRACT

Memory is a key parameter in embedded systems since both code complexity of embedded applications and amount of data they process are increasing. While it is true that the memory capacity of embedded systems is continuously increasing, the increases in the application complexity and dataset sizes are far greater. As a consequence, the memory space demand of code and data should be kept minimum. To reduce the memory space consumption of embedded systems, this paper proposes a control flow graph (CFG) based technique. Specifically, it tracks the lifetime of instructions at the basic block level. Based on the CFG analysis, if a basic block is known to be not accessible in the rest of the program execution, the instruction memory space allocated to this basic block is reclaimed. On the other hand, if the memory allocated to this basic block cannot be reclaimed, we try to compress this basic block. This way, it is possible to effectively use the available on-chip memory, thereby satisfying most of instruction/data requests from the on-chip memory. Our experiments with this framework show that it outperforms the previously proposed CFG-based memory reduction approaches.



## 1. Introduction

Recent advances in embedded processor design techniques have led to the development of complex embedded systems. Memory is a key parameter in these embedded systems since both code complexity of embedded applications and amount of data they process are increasing. While it is true that the memory capacity of embedded systems is continuously increasing, the increases in the application complexity and dataset sizes are far greater. Moreover, as embedded systems become increasingly complex, there is a growing demand for executing multiple applications concurrently, thereby putting even higher pressure on memory system.

Scratch Pad Memories (SPMs) have received considerable attention as on-chip memory building blocks. This is especially true for embedded systems as SPMs consume less energy and exhibit a very good runtime data locality behavior. Unlike a conventional cache managed by hardware, SPM is controlled by a programmer or a compiler. Therefore, if supported by appropriate compiler analysis and optimizations, SPM can cut the number of off-chip data accesses dramatically. Moreover, when compared to a cache, SPM provides predictability and reproducibility of timings, which is crucial for time critical embedded systems, or other systems where precise timing is important. Prior studies have explored different approaches to exploit the use of SPMs as memory blocks for both instruction and data [2,3,7,9,11,24].

We focus on the efficient use of a two level SPM memory hierarchy shared by multiple applications executing at the same time. In such an SPM memory hierarchy, this paper proposes a control flow graph (CFG) based technique to reduce the memory space consumption of applications. Specifically, it tracks the lifetime of instructions at the basic block level.

* Tel.: +90 312 2903444.
*E-mail addresses:* ozturk@cs.bilkent.edu.tr, ozturk@nec-labs.com

1477-8424/$ - see front matter © 2011 Elsevier Ltd. All rights reserved.
doi:10.1016/j.cl.2011.07.001

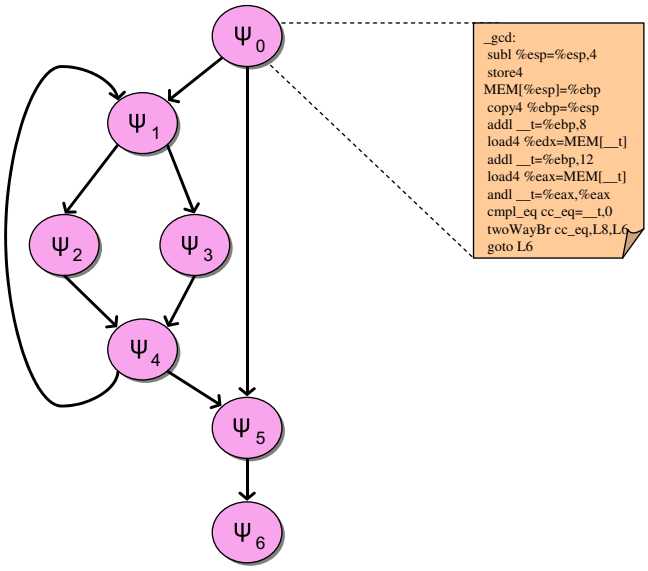


**Fig. 1.** An example CFG fragment is given for a sample GCD implementation.

Based on the CFG analysis, if a basic block is known to be not accessible in the rest of the program execution, the instruction memory space allocated to this basic block is reclaimed. On the other hand, if the memory allocated to this basic block cannot be reclaimed, our CFG based approach tries to compress the basic block if this basic block is rarely accessed. This way, it is possible to effectively use the available on-chip memory, thereby satisfying most of instruction/data requests from the on-chip memory. This is particularly important when available on-chip memory space is shared by multiple applications. In this paper, we make the following contributions:

- The first contribution of this paper is to reclaim the instruction memory that is no longer needed. This is achieved by generating a live set for each basic block in a given application. An example CFG fragment is given for a sample GCD implementation in Fig. 1. If execution reaches basic block $\Psi_5$, only live basic blocks are $\Psi_5$ and $\Psi_6$. Therefore, memory allocated to basic blocks $\Psi_0$, $\Psi_1$, $\Psi_2$, $\Psi_3$, and $\Psi_4$ can be deallocated.
- The second contribution of this paper is to extend memory space savings by compressing less frequently accessed basic blocks. Based on the profiling information, compression algorithm selects target basic blocks to compress. One needs to be careful in compressing a basic block as it can cause excessive execution overheads if not carefully selected. For example, if we consider the same CFG given in Fig. 1, one can see that $\Psi_0$ is followed either by $\Psi_1$ on the left branch or $\Psi_5$ on the right branch. If we know based on the profiling information that $\Psi_0$ is followed by $\Psi_5$ most of the time, compressing basic blocks $\Psi_1$, $\Psi_2$, $\Psi_3$, and $\Psi_4$ will reduce the memory space dramatically. As a result, this approach can be expected to be most successful in situations where there exist a few basic blocks with very high reuse.
- Third contribution of this paper is to evaluate the proposed CFG based memory reduction scheme using eleven benchmarks. It also compares our approach to a previously proposed basic block level garbage collection approach. Our experiments with several applications show that our approach can be very useful in increasing the benefits coming from an SPM.
- The last contribution of this paper is to show how saved memory space can be used to increase energy savings in banked memory architectures currently employed in some embedded systems.

The rest of this paper is organized as follows. Section 2 discusses related work, and Section 3 gives the details of the execution environment, dataflow analysis and proposed algorithms. Section 4 presents the results from our experimental evaluation and Section 5 concludes the paper.

## 2. Related work

Prior SPM studies primarily focused on the data access management. For instance, Panda et al. [16] present a static data partitioning scheme to eliminate the potential conflict misses due to limited associativity of on-chip cache. This approach benefits applications with a number of small (and highly reused) arrays that can fit in the SPM. In [8], authors propose a dynamic SPM management scheme for data accesses. Their framework uses both loop and data transformations to maximize the reuse of data elements stored in the SPM. Cooper et al. [6] show that using the register allocation's coloring paradigm can significantly reduce the amount of memory required for the program. Catthoor [5] discuss how accesses to a software-managed memory hierarchy can be optimized through code/data transformations. In [22], authors present