Journal of Systems Architecture 77 (2017) 3–13

Contents lists available at ScienceDirect

Journal of Systems Architecture

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

Data re-allocation enabled cache locking for embedded systems

Chun Xue[a], Keni Qiu[a,b,*], Weigong Zhang[a,b], Jing Wang[a,b], Yuanchao Xu[b], Mengying Zhao[c]

[a] Beijing Advanced Innovation Center for Imaging Technology, Beijing, China
[b] Capital Normal University, Beijing, China
[c] Shan Dong University, Jinan, China

ARTICLE INFO

Article history:
Received 6 February 2016
Revised 5 November 2016
Accepted 13 December 2016
Available online 14 December 2016

Keywords:
Data cache locking
Data re-allocation
Data object similarity
Interference graph

ABSTRACT

Cache locking is a cache management technique to preclude the replacement of locked contents. Cache locking methods have been proposed to improve predictability and worst-case execution time (WCET) previously. Recently, instruction cache locking has also been applied to improve average-case execution time (ACET). However, we observe that the previous ACET-driven instruction cache locking technique shows very limited improvement on performance when applied in data cache. The underlying reason lies in that object similarity of data accesses in data memory blocks are relatively low. This paper presents a data re-allocation enabled cache locking framework where data objects are first re-allocated to enhance data object similarity in memory blocks and then a data cache locking is well motivated. In this way, locking efficiency for data cache can be enhanced and thus system performance can be improved. The experimental results show that the miss rate, memory access cycles and dynamic energy can obtain good improvements across a suite of benchmarks.

1. Introduction

Cache is a widely used component in embedded systems to bridge the widening processor-memory performance gap. Cache locking is a technique to further improve cache efficiency which is supported by many modern processors, such as MIPS32 series [1], Intel XScale series [2], ARM processor series [3] and MPC processors [4]. Cache locking is implemented by executing a special locking routine. It empowers a processor with the ability to prevent part or all of its instructions or data from being evicted unless it is unlocked. Once a memory block is locked in the cache, all the subsequent accesses to the locked contents are cache hits. Although locking contents in the cache avoids cache miss penalty, the available cache space is reduced because any locked cache resource is unavailable for caching other parts of the main memory. Hence, a benefit-cost tradeoff should be carefully considered when implementing cache locking.

Recent work has shown that instruction cache locking can be able to improve average execution time (ACET) of a program by eliminating the evictions resulting from conflict accesses [6]. However, the effectiveness is different in data cache locking. Data cache analysis is more complicated than instruction cache due to two major reasons. First, locality of a reference is harder to capture for data accesses. Second, single instruction may refer to multiple memory locations [7]. In other words, data cache has poorer locality than instruction cache. The prior work summarizes four different types of data reference reuses: self-temporal reuse, self-spatial reuse, group-temporal reuse and group-spatial reuse [8]. As a result, reuse analysis of *data object*, a key procedure in cache locking content determination, is very complicated for data cache. In this paper, a *data object* denotes a generic variable that holds an individual value. It is used to differentiate a single number from a vector or matrix array etc.. The size of a data object may be defined differently among different types such as integers, floats, chars, strings, Booleans and so on.

In this paper, we conduct a set of experiments and observe that the conventional locking heuristic technique applied in instruction cache to improve ACET performance exhibits poor efficiency in data cache for most benchmarks. By analyzing the relationship of data cache locking and data cache properties, we have two interesting observations. First, object similarity of data access behavior in a memory block is weaker than that in an instruction memory block. The term *object similarity* means how similar data access behavior in a memory block exhibits in execution flows. Second, the one-trace based analyzing approach is not suitable

* Corresponding author at: Beijing Advanced Innovation Center for Imaging Technology, Beijing, China.
E-mail addresses: qiukn@cnu.edu.cn, qiukeni2015@sohu.com (K. Qiu).

http://dx.doi.org/10.1016/j.sysarc.2016.12.002
1383-7621/

for data cache locking because accesses to data cache are more sensitive to input data sets than accesses to instruction cache. We consider this input-sensitivity cannot be negligible. Therefore, we are motivated to put forward a solution in which a data layout is conducted beforehand to improve data object similarity inside memory blocks and then cache locking is determined based on the entire program control flow graph. In this way, a good benefit is expected to be exploited from data cache locking.

This paper proposes a data re-allocation enabled data cache locking technique to improve ACET performance of programs based on global data flow analysis. The main concept is to reorganize data layout by placing data objects with the largest *interference frequency* into the same memory block and select the memory block with the highest *interference risk* for locking. The reduction of interferences leads to a reduction of possible cache miss, which in turn improves ACET performance. The proposed approach has double-fold effects. First, data re-allocation can eliminate a partial of potential conflict misses and meanwhile improve the similarity of data objects inside a memory block. Second, cache locking can exploit the highest efficiency in performance improvement based on the data re-allocation effort. The experimental results show that the proposed approach, data re-allocation followed by data cache locking, can be an effective strategy to improve system ACET performance. The contributions of this paper include:

- Observations based on a set of experiments are presented that the conventional instruction cache locking approach has little effect on data cache. Furthermore, data cache locking efficiency is sensitive to program inputs.
- The above observations result from the fact that data cache exhibits inherent poorer object similarity than instruction cache.
- A data re-allocation approach is proposed to reorganize data objects in the memory so as to minimize the memory block interference frequency and improve object similarity of inside data memory blocks.
- The techniques of building a data interference graph and a memory block interference graph are proposed to direct data re-allocation and data cache locking respectively to achieve performance improvement.

The rest of the paper is organized as follows. The related work is introduced in Section 2. The observations and discussion of data cache locking are illustrated in Section 3. In Section 4, a motivational example is presented and the data re-allocation enabled cache locking problem is addressed. Section 5 presents the detailed solution framework. A set of experiments is conducted to evaluate the proposed approach in Section 6. Finally, Section 7 concludes the paper.

2. Related work

In previous work, cache locking has been studied to have promises in improving predictability in hard real time systems. As the hits and misses of the references can be counted statically with cache locking, the memory access cycles can be determined accurately, leading to tight worst-case execution time (WCET) analysis. Asaduzzaman et al. [9] proposed an instruction locking scheme of locking the blocks that cause the most misses using offline analysis on the Heptane platform. Liu and Xue [10] formulated the WCET optimization problem under instruction cache locking as an ILP instance. They proposed a recursive algorithm to deliver an optimal WCET. Arnaud and Puaut [11] devised a dynamic instruction cache locking mechanism targeting on WCET improvement. They proposed an algorithm which partitions the task into a set of regions. Each region owns statically a locked cache content determined offline.

Instruction cache locking has been applied to improve average case execution time (ACET). Anand and Barua [12] presented a static instruction cache locking scheme which can be embedded inside a binary rewriter. A heuristic method based on a cost–benefit tradeoff model is applied to select the maximal net-benefit cache lines for locking by iterative simulations. Liang and Mitra [6] subsequently proposed an instruction cache locking approach by analyzing memory block reuses based on a temporal reuse profile. These two works both focus on a particular execution trace.

Thus far, a few works target data cache locking study. Vera et al. [7] proposed a worst-case memory performance estimation approach using cache partitions and dynamic data cache locking in multitasking systems. Data cache locking is applied to improve the prediction precision of a WCET.

Zheng and Wu [13] proposed a dynamic data cache locking to improve WCET. They proposed two dynamic cache locking approaches for a single task. The first approach formulates the problem as a global Integer Linear Programming (ILP) problem that simultaneously selects a near-optimal set of variables as the locked cache contents. The second one iteratively constructs a subgraph of the control flow graph (CFG) of the task where the lengths of all the paths are close to the longest path length, and uses an ILP formulation to select a near-optimal set of variables in the subgraph as the locked cache contents. For both of the locking approaches, they proposed a novel efficient data cache allocation algorithm. Their work differs from ours in two aspects. First, their locking contents selection is determined by iteratively reducing the longest path length since their work focuses on WCET. Second, they proposed a *k-Longest-Path-based* algorithm to allocate data cache with an objective to minimize the longest path of the interference data acyclic graph (DAG). Our work targets the ACET of programs under locking with a locking unit of a cache line. Therefore, we concern memory block interferences in terms of a cache set.

Most recently, Adegbija and Gordon-Ross [14] proposed a phase-based cache locking to improve data cache's performance and energy efficiency. In their work, a phase-based cache locking, which leverages an application's varying runtime characteristics, is applied to data cache.

Our work differs from that work by using static cache locking where the locking contents are determined before a program starts running. In our work, there is no overhead on locking content determination and locking implementation because they do not introduce delay at runtime. In that dynamic locking work [14], a phase classification module is embedded to classify the applications' phases and determine the phases' persistence. The scheme requires extra hardware supports to store key flags and locking content locations. What is more, since the locking is conducted at runtime, the locking content determination and locking implementation will introduce additional delay.

Actually, these two works are suitable for different kinds of applications. Our approach fits for small applications which have only one dominant execution phase in terms of data accesses. However, the dynamic locking method in [14] works well for large applications which are characteristic of multiple execution phases.

To the best of our knowledge, our work is the first to employ data cache locking integrated with data re-allocation to improve ACET performance of applications in embedded systems.

3. Insights to data cache locking

In this section, we first give some preliminaries of cache locking, and then analyze the properties of data cache locking by comparing to instruction cache locking based on a set of experiments.