Microprocessors and Microsystems 38 (2014) 137–151

Contents lists available at ScienceDirect

Microprocessors and Microsystems

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

Address independent estimation of the boundaries of cache performance

Diego Andrade *, Basilio B. Fraguela, Ramón Doallo

Dept. of Electronics and Systems, University of A Coruña, Spain

ARTICLE INFO

Article history:
Available online 14 January 2014

Keywords:
Real-time systems
BCET
WCET
Cache memory

ABSTRACT

Worst-case (WCET) and best-case (BCET) execution times must be estimated in real-time systems. Worst-case memory performance (WCMP) and best-case memory performance (BCMP) components are essential to estimate them. These components are difficult to calculate in the presence of data caches, since the data cache performance depends largely on the sequence of memory addresses accessed. These addresses may be unknown because the base address of a data structure is unavailable for the analysis or it may change between different executions. This paper introduces a model that provides fast and tight valid estimations of the BCMP, despite ignoring the base address of the data structures. The model presented here, in conjunction with an existing model that estimates the WCMP, can provide base-address independent estimations of the BCMP and WCMP. The experimental results show that the base addresses of the data structures have a large influence in the cache performance, and that the model estimations of the boundaries of the memory performance are valid for any base addresses of the data structures.

1. Introduction

Execution time must be bounded [1] in real-time systems. Namely, the worst-case execution time (WCET) estimates an upper limit of the execution time of a task and it may be used to perform any schedulability analysis, to ensure meeting deadlines and to assess resource needs for real-time systems. The best-case execution time (BCET) estimates the lower limit and it may be used to assess code quality and resource needs for non- or soft real-time systems, and to ensure that live lines and minimum sampling intervals are met. The presence of data caches [2] complicates the estimation of the memory performance components of the WCET and the BCET, which are the worst-case (WCMP) and best-case memory performance (BCMP), respectively. The reason is that the memory performance in the presence of caches depends largely on the exact sequence of memory addresses accessed by the program, and these addresses determine the placement of each piece of data in the cache. This sequence may be unavailable at compile-time due to the lack of the base address information of the data structures and/or the presence of irregular access patterns. The base addresses may also not be obtainable because of the usage of program modules or libraries compiled separately, stack variables or dynamically allocated memory. Furthermore, these addresses may change between different executions of the program.

The model presented in [3] is, to our knowledge, the only one to tackle a base address-independent prediction of the WCMP in the presence of data caches. It is based on the probabilistic miss equations (PME) model [4] and it can estimate rapidly and precisely the WCMP of codes with strided accesses (regular codes). This paper complements [3] with the ability to provide base address independent predictions of the BCMP. This would turn the model, if integrated with a CPU model, into a powerful tool to perform statically a thorough timing analysis.

The rest of this paper is organized as follows. Section 2 describes the basics of the PME model and its scope of application. The following two sections explain the changes required to estimate the BCMP. Namely, Sections 3 and 4 contain the core contribution of the paper as they are devoted to describe the method to calculate the BCMP. Section 5 highlights the main differences between the BCMP approach presented in this paper, and the WCMP approach presented in [3]. Then, Section 6 shows some experimental results, Section 7 is devoted to the related work and Section 8 concludes.

2. The probabilistic miss equations model

The probabilistic miss equation (PME) model [4] predicts the behavior of set-associative caches following a Least Recently Used (LRU) replacement policy. Before the model is introduced, let us start with some notions on cache memories. Caches are associative memories [5], located in the top levels of the memory hierarchy, that contain a subset of the data present in the lower levels and that are searched by the memory address provided by the processor. Caches are divided into cache lines, a line corresponding to the minimum amount of information that can be placed in the cache.

* Corresponding author. Tel.: +34 981 167 000x1298; fax: +34 981 167 160.
E-mail addresses: diego.andrade@udc.es (D. Andrade), basilio.fraguela@udc.es (B.B. Fraguela), doallo@udc.es (R. Doallo).

0141-9331/$ - see front matter © 2014 Elsevier B.V. All rights reserved.
http://dx.doi.org/10.1016/j.micpro.2014.01.001

Cache lines are grouped in cache sets, each set having the same number of lines, which is called the cache associativity. Under a CPU request only the lines in a set are searched. Also, a line can only be placed in a predetermined set, although any of the set lines is eligible for its placement.

The basic operation of a cache memory starts with the processor emitting a memory address, which may need to be translated into a physical address before accessing the cache if we consider a system with virtual memory and a physically-indexed cache. A part of this memory address, called index, is used to select the cache set where the cache line containing the requested address should be located. Another part of the address, called tag, is used to find out if the line is in the cache set. If it is present, the access turns into a cache hit. Then the cache retrieves the data associated to the address requested inside the line using another part of the address called displacement and it sends this data item to the processor. If the line is not present, the access turns into a miss and the data must be brought from the lower levels of the memory hierarchy to the corresponding cache set. Then, the access is treated as a hit.

When a new line is loaded into its corresponding cache set, the cache set may be full. In that case, one of the lines of the cache set must be replaced by the new line. The selection of which line is replaced is taken by the replacement policy. The most popular replacement policy is the Least Recently Used (LRU) one, that selects the line that has not been accessed for a longer time.

The PME model estimates the number of cache misses generated by the execution of a code. The model processes the static references of the analyzed code one by one. For each reference and each nesting level containing it, a separated probabilistic miss equation (PME) is generated. Each static reference generates several dynamic accesses. Each access affects one data item which is located in a given memory line. This access can result in a cache hit if the line is already loaded in the cache or a miss otherwise. Cache misses take place compulsorily the first time a memory line is accessed. The remaining accesses to memory lines are reuse attempts, given that a preceding access already loaded the line of interest in the cache. A reuse attempt on a memory line results in a miss if the lines brought to the cache during the reuse distance have evicted the line. This eviction happens with a given probability, called miss probability, which depends on the reuse distance. A PME formula classifies the accesses generated by the reference it models within the considered loop according to their reuse distance and computes the miss probability associated to each one of the reuse distances found. Then, this PME estimates the number of misses generated by the reference in the loop by adding the number of accesses with each given reuse distance weighted by their associated miss probability.

$$F_{Ri}(\mathrm{RD}_{\mathrm{input}}) = NAcc(\mathrm{RD}_{\mathrm{input}}) \times MissP(\mathrm{RD}_{\mathrm{input}}) + \sum_{i=1}^{NRD} NAcc(\mathrm{RD}_i) \times MissP(\mathrm{RD}_i) \quad (1)$$

Eq. 1 represents the general form of the PME associated to reference R and nesting level i. NRD is the number of different reuse distances found. $NAcc(\mathrm{RD})$ is the number of accesses generated by reference R whose reuse distance is RD and $MissP(\mathrm{RD})$ is the miss probability associated to that reuse distance. The PME also considers the first-time accesses of R to lines during one execution of the loop. While these accesses cannot exploit a RD within the loop, they may enjoy reuse with respect to accesses to the same line which took place in previous iterations of outer or preceding loop nests. Since such RDs cannot be found in the analysis of loop i, every PME F_{Ri} has an input. The number of misses generated by these accesses is accounted by the first term of the formula.

Example 2.1. Let us consider a simple code like the following

```
for (ind = 0; ind < 16; ind++) {
    a[ind] = b[ind];
}
```

and a direct-mapped cache that can store 16 elements of any of both arrays which is divided into 4 lines that can store 4 elements of the arrays each. Fig. 1 contains a control flow of the accesses generated by the example code. Let us assume that scalar accesses are usually mapped to processor registers, even if it is not the case, that is a reasonable best-case assumption. Thus, only the accesses to arrays `a` and `b` have an impact on the cache.

Let us see how Eq. 1 applies to the modeling of reference $R \equiv a[ind]$ in the scope of this loop at nesting level $i = 0$. Fig. 2 represents the positions of `a` accessed through this reference. Accesses to positions located in the same cache line are grouped and, first-time accesses, reuse attempts and reuse distances are identified. The first position of the array `a[0]` is located at the beginning of one line, thus, the loop accesses exactly 16/4 = 4 different cache lines. So, $NAcc(\mathrm{RD}_{\mathrm{input}}) = 4$ because 4 accesses visit a line for the first time in this loop and its associated reuse distance is $\mathrm{RD}_{\mathrm{input}}$, which is the RD for the first-time accesses of R in the loop. The remaining 12 iterations are reuse attempts of a line accessed in a previous iteration of the loop. In fact since the access is sequential, the reuse attempts always take place on the immediately previous iteration of the loop. So, all the reuse attempts are associated to a single reuse distance, thus, $NRD = 1$. This reuse distance RD_0 is concretely one iteration of the loop 0 we are considering, denoted as $\mathrm{Iter}_0(1)$. This way, $F_{R0}(\mathrm{RD}_{\mathrm{input}}) = 4 \times MissP(\mathrm{RD}_{\mathrm{input}}) + 12 \times MissP(\mathrm{RD}_0)$.

The example can be completed intuitively although the method to calculate the miss probability associated to a given reuse distance has not been explained yet. If the lines from array `a` have not been accessed in a previous loop, the miss probability associated to the $\mathrm{RD}_{\mathrm{input}}$ reuse distance is 1, which means that these $NAcc(\mathrm{RD}_{\mathrm{input}}) = 4$ accesses turn into misses. Regarding the 12 accesses with reuse distance $\mathrm{RD}_0 = \mathrm{Iter}_0(1)$, during one iteration of the loop one line from array `a` and one line from array `b` are accessed. The line from `a` is the reused line, so it does not interfere with itself. The line from `b` is going to be placed in one of the 4 sets of the cache, each one of them consisting of a single line. If it is placed in the same set as the reused line from `a`, it will eject it from the cache; otherwise there will be no interference. This way on

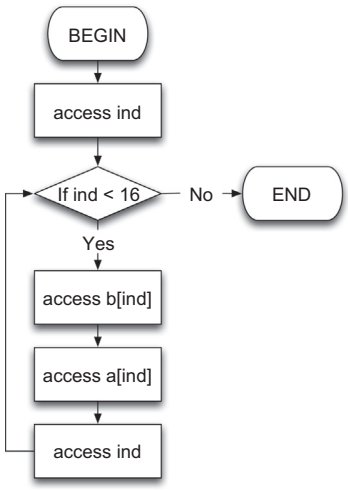

Fig. 1. Control flow of the accesses of Example 2.1.