Exact Analysis of the Cache Behavior of Nested Loops
Critical Synthesis

Simon Bihel

January 19, 2018

1 Introduction

This is a synthesis of [CPHL01].

In order to optimize a program you need to measure or predict its performances. A simple method is to execute it or simulate the execution of the program to measure the performances, but this has the drawback of being highly computational. The authors of the paper propose an exact model of the cache behavior of loop nests to make it possible to compute the exact performances of the cache as you could do with a simulation, but faster and with more insights on the behavior.

In Section 2 we explain the problem the authors aim to solve along with the tool they use. After that in Section 3 we see how they employ these tools to effectively model the cache behavior of loop nests. Section 4 presents the evaluation of the contribution. Section 5 sums up the contribution in the light of a more critical point-of-view.

2 Context

2.1 Memory Hierarchies

The memory hierarchy considered is a simplified one, with one memory access at a time, no distinction between reads and writes, composed of two levels, and with a Least Recently Used replacement policy.

The goal is to count cache hits and misses. The authors differentiate two kinds of misses.

Interior misses are misses that are independent of the initial cache state when the program fragment starts its execution.

Boundary misses are misses that depend on the initial cache state.

In other words there are three kinds of memory accesses: those that are guaranteed to hit, those that are guaranteed to miss, and those that could hit or miss depending on the cache initial state.

2.2 Polyhedral Model

Loop nests can be described mathematically. Figure 1 sums up the notations used for the polyhedral model. An iteration point is an array of indexes to express the position in each loop.

More details will be given when needed when describing the model in Section 3.

2.3 Presburger Arithmetic

Presburger arithmetic is a subset of first order logic consisting of constraints (equality or inequality) on integer variables. The logical operators available are $\neg$, $\wedge$ and $\vee$, and the quantifiers available are $\forall$ and $\exists$. With such formulas it is possible to describe cache structure and accesses, and thus compute misses.

Object	Mathematical Representation
An iteration point	ℓ
The ith array reference	$R_i = (Y^{(j)}, F_i, S_h)$
The access made by R_i at ℓ	(R_i, ℓ)
The array element accessed by R_i at ℓ	$e_i = Y^{(j)}[F_i(\ell)]$
The byte address of e_i	$m_i = \mu_j + \mathcal{L}_j(F_i(\ell)) \cdot \beta_j$
The block address of m_i	$b_i = \mathcal{B}(m_i)$
The cache set to which b_i maps	$s_i = \mathcal{S}(b_i)$

Figure 1: Table of notation

3 Cache Analysis Model

In this section only a few formulas are presented by a lack of space.

3.1 Cache Structure

3.1.1 Valid iteration point

The first basic formula (1) is to describe an iteration point. d is the number of levels of nesting and ℓ_i the current index for the i-th loop. That index has to be valid, i.e. in the index range (0 to n_i).

$$\ell \in \mathcal{I} \overset{\text{def}}{=} \bigwedge_{i=0}^{d-1} 0 \leqslant \ell_i < n_i \tag{1}$$

3.1.2 Lexicographical ordering of accesses

Then we need to express order in access. Formula (2) describes that for iteration points l and m, and for references R_u and R_v. It means that if l is an iteration point that precedes m then the references are obviously preceding, but if $l = m$ then we need to make sure R_u occurs before R_v.

$$\begin{aligned}(R_u, \ell) \lhd (R_v, m) \overset{\text{def}}{=} {} & \ell \in \mathcal{I} \wedge m \in \mathcal{I} \wedge \\ & (\bigvee_{i=0}^{d-1} (\ell_i < m_i \wedge \bigwedge_{j=0}^{i-1} \ell_j = m_j) \vee \\ & (\bigwedge_{j=0}^{d-1} \ell_j = m_j \wedge u < v))\end{aligned} \tag{2}$$

3.1.3 Mapping memory locations to cache sets

To express the memory location mapping to a cache set, they bound the memory value with cache sets boundaries while making sure it falls on a valid memory block boundary.

3.2 Cache Behavior

For now, a direct-mapped cache is considered.

3.2.1 Interior misses

To determine if an access to a memory block b results in an interior miss it is enough to now two things: (i) that there is an earlier access with a different memory block mapping to the same cache set; (ii) and that there is no access to b between this earlier access and current access to b. These properties can be expressed with formulas that we have seen.

3.2.2 Boundary misses

For boundary misses we are only interested in access that are the first to map to a cache set. We can write the formula with previous formulas, but we also have to use the initial state of the cache to express that the memory block is not in the cache already.

3.2.3 Cache state

To have the final state of the cache we need to keep the memory block that have not been replaced.

3.3 Extensions

3.3.1 Imperfect loop nests

Sometimes there are instructions outside the inner-most loop, which make loop nests *imperfect*. To handle these, they use a transformation [AMP01] to move the problematic instructions in the inner-most with guards (i.e. `if` condition). As the consequence they modified the formula of valid iteration point to include the guard's boolean condition.

3.3.2 Associativity

Defining interior misses with a A-way set-associative cache requires a longer formula as we can allow A different access before declaring a miss. The downside is that the complexity of the formula increases with A to a point that only modest values of A can be handled.

3.3.3 Non-linear data layouts

To handle array layouts based on bit interleaving, memory addresses are expressed in binary to express the layouts in Presburger formulas. Such fine grain grows the formula and thus has an impact in the overall complexity.

4 Evaluation

To assess the correctness of their method they run it on different examples and compare the results with a cache simulation. They run all possible loop permutations. Examples include basic loop nests, imperfect loop nest, and non-linear arrays.

They find that the number of misses match, but that the classification differs. This is because the cache simulation has a cold miss category which has no link with context as boundary misses do.

5 Conclusion

The authors have developed a model that can exploit the regularity of loop nests to express the problem of counting cache misses in common mathematical formulas. While it aims at replacing simulating, such model is still very much computationally expensive. The authors propose as future work to mix both simulation and model-based computation to avoid polyhedral loop nests.

Assuming one memory access at a time without distinction between reads and writes seems reasonable. Examples used in the evaluation were diverse enough to convince me of the correctness of the model. While they assume a Least Recently Used replacement policy, I believe more complex policies could be modelled but I think it could make the A-way set-associativity complexity even worse. In the same branch, they assume a two-level memory hierarchy, maybe the complexity would significantly increase with a third level.

References

[AMP01] Nawaaz Ahmed, Nikolay Mateev, and Keshav Pingali. Synthesizing transformations for locality enhancement of imperfectly-nested loop nests. *International Journal of Parallel Programming*, 29(5):493–544, 2001.

[CPHL01] Siddhartha Chatterjee, Erin Parker, Philip J Hanlon, and Alvin R Lebeck. Exact analysis of the cache behavior of nested loops. *ACM SIGPLAN Notices*, 36(5):286–297, 2001.