Exact Analysis of the Cache Behavior of Nested Loops [2]

Simon Bihel simon.bihel@ens-rennes.fr

Monday 22nd January, 2018

University of Rennes I
École Normale Supérieure de Rennes

Introduction

- Predict cache behavior (i.e. count misses) for performance evaluation
- Exact prediction can be done with simulation but it is expensive
- Models permit simpler computations

Table of contents

Context

Memory Hierarchies

- 2 levels, one access at a time, no distinction between reads and writes
- Least Recently Used replacement policy

Memory Hierarchies

- 2 levels, one access at a time, no distinction between reads and writes
- Least Recently Used replacement policy

2 kinds of cache misses

Interior misses Independent to the initial cache state.

Boundary misses Depend on the initial cache state.

Polyhedral Model

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	$s_i = \mathcal{S}(b_i)$

- An iteration point is an array of indexes.

Presburger Arithmetic

Subset of first order logic

Constraints Equalities or inequalities

Operators $\neg$, $\wedge$ and $\vee$

Quantifiers $\forall$ and $\exists$

Used to describe cache structure and accesses

Cache Analysis Model

Valid Iteration Point

Express that every index is in the boundaries of its loop.

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

Ordering of Accesses

Having an access being done before another means that:

- the iteration point is preceding;

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

Ordering of Accesses

Having an access being done before another means that:

- the iteration point is preceding;
- or the reference is preceding.

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

Memory-Cache Mapping

Direct mapping.

- Express boundaries of memory address with cache set's boundaries addresses.
- Take into account memory block boundary alignments.

Boundary Misses

Initial state dependent misses.

- There is an access for a certain cache set s;

$$
\begin{aligned}
((R_u, i) \in \text{BoundMiss}(\mathbb{L}, \mathbb{C}_{in})) \stackrel{\text{def}}{=} i \in \mathcal{I} \wedge \\
\exists d, s : \text{Map}(\mathcal{L}_x(F_u(i)), d, s) \wedge \\
\neg(\exists e, j, v : (R_v, j) \lhd (R_u, i) \wedge \\
\text{Map}(\mathcal{L}_y(F_v(j)), e, s)) \wedge \\
\mathbb{C}_{in}(s) \neq \mathcal{B}(\mathcal{L}_x(F_u(i)))
\end{aligned} \tag{7}
$$

$\mathcal{L}$ is a layout function (offset of an array element). $\mathcal{B}$ is the block address function.

Boundary Misses

Initial state dependent misses.

- There is an access for a certain cache set s;
- it is the first;

$$
\begin{aligned}
((R_u, i) \in \text{BoundMiss}(\mathbb{L}, \mathbb{C}_{in})) \stackrel{\text{def}}{=} i \in \mathcal{I} \wedge \\
\exists d, s : \text{Map}(\mathcal{L}_x(F_u(i)), d, s) \wedge \\
\neg(\exists e, j, v : (R_v, j) \lhd (R_u, i) \wedge \\
\text{Map}(\mathcal{L}_y(F_v(j)), e, s)) \wedge \\
\mathbb{C}_{in}(s) \neq \mathcal{B}(\mathcal{L}_x(F_u(i)))
\end{aligned} \tag{7}
$$

$\mathcal{L}$ is a layout function (offset of an array element). $\mathcal{B}$ is the block address function.

Boundary Misses

Initial state dependent misses.

- There is an access for a certain cache set s;
- it is the first;
- the cache set does not correspond to the correct memory block.

$$\begin{aligned}((R_u, i) \in \text{BoundMiss}(\mathbb{L}, \mathbb{C}_{in})) \stackrel{\text{def}}{=} i \in \mathcal{I} \wedge \\ \exists d, s : \text{Map}(\mathcal{L}_x(F_u(i)), d, s) \wedge \\ \neg(\exists e, j, v : (R_v, j) \lhd (R_u, i) \wedge \\ \text{Map}(\mathcal{L}_y(F_v(j)), e, s)) \wedge \\ \mathbb{C}_{in}(s) \neq \mathcal{B}(\mathcal{L}_x(F_u(i)))\end{aligned} \tag{7}$$

$\mathcal{L}$ is a layout function (offset of an array element). $\mathcal{B}$ is the block address function.

Interior Misses

Two properties to have an interior miss for memory block b:

- there is an earlier access with a different memory block mapping to the same cache set;
- there is no access to b between the two access.

Extensions

Imperfect Loop Nests

Transformation [1] with guards on statements.

```
    do i = 0, n-1
      do j = 0, n-1
S0:     x = C[i,j];
        do k = 0, n-1
S1:       x = x + A[i,k]*B[k,j];
        end
S2:     C[i,j] = x;
      end
    end
```

```
    do i = 0, n-1
      do j = 0, n-1
        do k = 0, n-1
S0:       if (k == 0) x = C[i,j];
S1:       x = x + A[i,k]*B[k,j];
S2:       if (k == n-1) C[i,j] = x;
        end
      end
    end
```

Use `ifs` conditions to express *valid accesses*.

A-way Set-associativity

Complicated formula and not efficient.

"Will require more consideration."

$$
\begin{aligned}
((R_u, i) \in \text{IntMiss}) \stackrel{\text{def}}{=} i \in \mathcal{I} \wedge \\
\exists d, s : \text{Map}(\mathcal{L}_x(F_u(i)), d, s) \wedge \\
\exists e_0, j_0, v_0 : (R_{v_0}, j_0) \lhd (R_u, i) \wedge \\
\text{Map}(\mathcal{L}_{y_0}(F_{v_0}(j_0)), e_0, s) \wedge \\
(\exists e_1, \ldots, e_{A-1} : \\
\bigwedge_{a=1}^{A-1} (\exists j_a, v_a : (R_{v_0}, j_0) \lhd (R_{v_a}, j_a) \lhd (R_u, i) \wedge \\
\text{Map}(\mathcal{L}_{y_a}(F_{v_a}(j_a)), e_a, s)) \wedge \\
d \neq e_0 \neq \cdots \neq e_{A-1}) \wedge \\
\neg(\exists k, w : (R_{v_0}, j_0) \lhd (R_w, k) \lhd (R_u, i) \wedge \\
\text{Map}(\mathcal{L}_z(F_w(k)), d, s)) \qquad (9)
\end{aligned}
$$

Non-linear Data Layouts

Express the data layout's binary manipulations in a Presburger formula.

Evaluation

Evaluation Method

- Experimental evaluation
- Comparison with (specially written) cache simulator
- Multiple programs used to evaluate every extension
- All loop permutations tested
- Start with empty cache

General Results

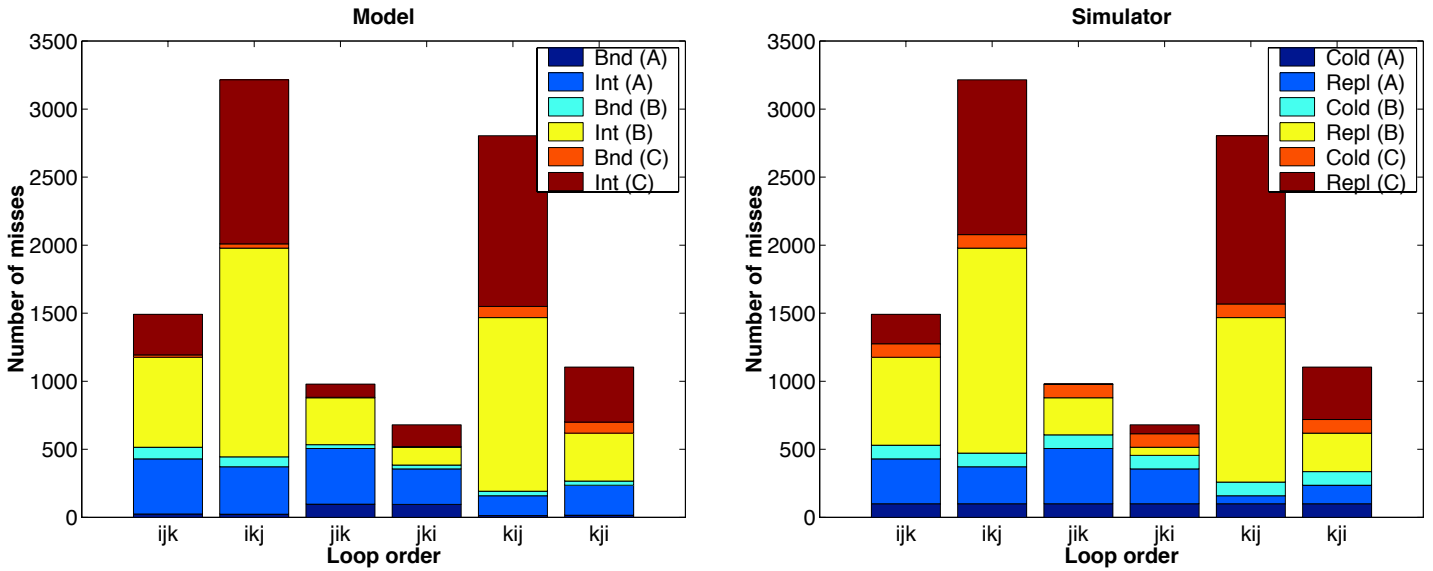

Figure 2: Miss counts for Problem 1, scenario 1. (a) Our approach. (b) Cache simulation.

- Same number of misses
- Different classification (*cold* and *replacement* misses)

Conclusion

Conclusion

- Mathematical description for more flexibility in computations
- Exploits the regularities of loop nests
- Future work: mix simulation and model-based computation to leap-frog loop nests

Critical View

- Assuming one memory access at a time and no distinction between reads and writes seems reasonable

Critical View

- Assuming one memory access at a time and no distinction between reads and writes seems reasonable
- The evaluation convinced me even if I first expected a more formal approach

Critical View

- Assuming one memory access at a time and no distinction between reads and writes seems reasonable
- The evaluation convinced me even if I first expected a more formal approach
- Code non-available

Critical View

- Assuming one memory access at a time and no distinction between reads and writes seems reasonable
- The evaluation convinced me even if I first expected a more formal approach
- Code non-available
- Still computationally expensive

Critical View

- Assuming one memory access at a time and no distinction between reads and writes seems reasonable
- The evaluation convinced me even if I first expected a more formal approach
- Code non-available
- Still computationally expensive
- I believe other policies than LRU could be implemented but the complexity would explode with an A-way set-associativity

Critical View

- Assuming one memory access at a time and no distinction between reads and writes seems reasonable
- The evaluation convinced me even if I first expected a more formal approach
- Code non-available
- Still computationally expensive
- I believe other policies than LRU could be implemented but the complexity would explode with an A-way set-associativity
- Same problem of complexity with more levels of hierarchy

Critical View

- Assuming one memory access at a time and no distinction between reads and writes seems reasonable
- The evaluation convinced me even if I first expected a more formal approach
- Code non-available
- Still computationally expensive
- I believe other policies than LRU could be implemented but the complexity would explode with an A-way set-associativity
- Same problem of complexity with more levels of hierarchy
- Sometimes confusing variables names between sections

References i

N. Ahmed, N. Mateev, and K. Pingali.
Synthesizing transformations for locality enhancement of imperfectly-nested loop nests.
International Journal of Parallel Programming, 29(5):493–544, 2001.

S. Chatterjee, E. Parker, P. J. Hanlon, and A. R. Lebeck.
Exact analysis of the cache behavior of nested loops.
ACM SIGPLAN Notices, 36(5):286–297, 2001.