Embedded System Programming

WCET Analysis (2)
(Module 39)

Yann-Hang Lee
Arizona State University
yhlee@asu.edu
(480) 727-7507

Summer 2014
Micro-architectural Modeling -- Cache

- Modify cost function (cache hit and miss have different costs)
- Add linear constraints to describe relationship between cache hits and misses
- **Basic idea**
  - Basic blocks assumed to be smaller than entire cache
  - Subdivide instruction counts \( (x_i) \) into counts of cache hits \( (x_i^{hit}) \) and misses \( (x_i^{miss}) \)
  - Line-block (or l-block) is a contiguous sequence of code within the same basic block that is mapped to the same cache line in the instruction cache
  - Either all hit or all miss in a l-block
Basic Blocks to Line Blocks (Direct-mapped)

Color Cache Set

<table>
<thead>
<tr>
<th>Set</th>
<th>0</th>
<th>1</th>
<th>2</th>
<th>3</th>
</tr>
</thead>
<tbody>
<tr>
<td>B1.1</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>B1.2</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>B1.3</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>B2.1</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>B2.2</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>B3.1</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>B3.2</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Cache Constraints:

No conflicting l-blocks: \( x^{miss}_{2.2} \leq 1 \)

(only the first execution has a miss)

Two nonconflicting l-blocks are mapped to same cache line

\( x^{miss}_{1.3} + x^{miss}_{2.1} \leq 1 \)

Conflicting blocks: affected by the sequence
Cache Conflict Graph

- For every cache set containing two or more conflicting l-blocks
  - start node, end node, and node $B_{k,l}$ for every l-block in the cache set
- Edge from $B_{k,l}$ to $B_{m,n}$: control can pass between them without passing through any other l-blocks of the same cache set.
  - $p_{(i,j,u,v)}$: the number of times that the control passes through that edge.
Cache Constraints Example (1)

```plaintext
B1.1 s = k;
B2.1 while (k < 10) {
  B3.1 if (ok)
    B4.1 j++;
  B5.1 j = 0;
  ok = true;
}
B6.1 k++;
B7.1 r = j;
```
Cache Constraints Example (2)

Total execution time =

\[ \sum_i N \sum_j (c_{i,j}^\text{hit} x_{i,j}^{\text{hit}} + c_{i,j}^\text{miss} x_{i,j}^{\text{miss}}) \]

\[ x_i = x_{i,j}^{\text{hit}} + x_{i,j}^{\text{miss}} \quad j = 1,2, \ldots, n_i \]

\[ x_i = \sum_{u,v} p(u,v,i,j) = \sum_{u,v} p(i,j,u,v) \]

\[ \sum_{u,v} p(s,u,v) = \sum_{u,v} p(u,v,e) = 1 \]

\[ p(i,j,i,j) \leq x_{i,j}^{\text{hit}} \leq p(s,i,j) + p(i,j,i,j) \]
Progress During the Past 10 Years

The explosion of penalties has been compensated by a reduction of uncertainties!

- 1995 (Lim et al.): 20-30% cache-miss penalty
- 2002 (Thesing et al.): 15% over-estimation
- 2005 (Souyris et al.): 25% over-estimation

Note: The graph shows a comparison of cache-miss penalties and over-estimations over the years 1995, 2002, and 2005, attributed to various researchers.
Open Problems

- **Architectures are getting much more complex.**
  - Can we create processor behavior models without the pain?
  - Can we change the architecture to make timing analysis easier?

- **Small changes to code and/or architecture require completely re-doing the WCET computation**
  - Use robust techniques that learn about processor/platform behavior

- **Need more reliable ways to measure execution time**

- **References:**
  - Li, Malik, and Wolfe, “Cache Modeling for Real-Time Software: Beyond Direct Mapped Instruction Caches”
Supplementary Slides