Predicting Negative Cache Interference with Composable Application-Centric Models

Xiaoya Xiang, Bin Bao, Tongxin Bai, Chen Ding, Trishul Chilimbi
Outline

• Introduction
• Background
• Approximate all-window footprint
• Cache interference prediction
• Evaluation
• Summary
Introduction

- Applications are increasingly run in shared cache
- **Asymmetrical effect** on performance due to cache sharing
  - equake(<20%) vs vpr(82%)
- Traditional metrics cannot easily explain the asymmetry
- **Footprint** may help
What is footprint?

Given an execution window in a trace, the footprint is the number of distinct elements accessed in the window.

Example:

\[ k \text{ m m n n n} \]

Compared to reuse distance:

- The number of distinct data elements accessed between this and the previous access to the same data.
Background

• What is footprint?

  • Given an execution window in a trace, the footprint is the number of distinct elements accessed in the window.

• example

  \[
  \begin{array}{c}
  k \ b \ m \ m \ n \ n \ n \\
  \end{array}
  \]

  window size= 2       footprint=2

• compared to reuse distance

  • the number of distinct data elements accessed between this and the previous access to the same data.
Background

• What is footprint?
  • Given an execution window in a trace, the footprint is the number of distinct elements accessed in the window.
  • example

```plaintext
  k m m n n n
```

  window size= 3  footprint=2

  compared to reuse distance
  • the number of distinct data elements accessed between this and the previous access to the same data
What is footprint?

Given an execution window in a trace, the footprint is the number of distinct elements accessed in the window.

Example:

\[ k \, m \, m \, n \, n \, n \]

Window size = 4   Footprint = 2

Compared to reuse distance:

- The number of distinct data elements accessed between this and the previous access to the same data.
Locality on shared cache

\[ M(A) = P(A\text{'s reuse distance} \geq \text{cache size}) \]
\[ M(A|B) = P(A\text{'s reuse distance} + B\text{'s footprint} \geq \text{cache size}) \]
All-window footprint

- Given an execution of \( N \) run-time data accesses, calculate footprint of all possible windows

- There are \( \frac{N(N+1)}{2} \) different non-empty windows

- Intuitive way
  - traverse the data access trace
  - for each data access, compute the footprint of all windows ending at current access

- \( O(N^2) \)
• Observation

• footprint only changes, when moving left from the endpoint, at the last access of a given element before or up to the window endpoint (in blue)

• NM algorithm: counting footprints instead of counting windows

• only store the last access of each data (M is the number of distinct data)(Bennett&Kruskal, 1975)

• fix a footprint, measure the number of windows of that footprint in one step.

• $O(NM)$
Further improve by approximation

• $N \log M$ algorithm (Ding and Chilimbi, 2008)
  • Do not care about the exact value of big footprint
  • $1,000,000$ vs $1000,001$
  • For a relative precision, e.g. 99%, two footprints differ only if their difference is greater than 1% of the smaller one.
• store only $O(\log M)$ data to represent $M$ distinct data
• $O(N \log M)$
Further approximation?

- CKlogM algorithm by trace compression (my solution)
  - set a threshold \( C \), e.g. 3. Do not measure footprints smaller than \( C \)
  - acceptable since small footprints have little effect on cache sharing
  - divide a trace into a series of intervals called footprint intervals.
  - footprint only changes, when moving right from the startpoint, at the first access of a given element after the window startpoint (in blue)

\[
\text{footprint} \quad 2 \quad 3 \quad 3 \quad 4 \quad 4 \quad 4 \quad \text{end} = 7...12 \text{ (window ending point)}
\]

\[
\text{start} = 6 \quad \text{a footprint interval of size 3}
\]
CK\log M Algorithm

• at most $C$ first accesses of different data within a footprint interval of size $C$.

• $K$ is the number of footprint intervals in the trace.

• Reduce the asymptotic complexity from $O(N\log M)$ to $O(CK\log M)$

• Define $N/CK$ as the speedup factor
# Speedup for all tests

<table>
<thead>
<tr>
<th>prog.</th>
<th>$N$</th>
<th>$M$</th>
<th>NlogM time [sec]</th>
<th>CKlogM C=128 time (speedup)</th>
<th>CKlogM C=256 time (speedup)</th>
</tr>
</thead>
<tbody>
<tr>
<td>gzip</td>
<td>804M</td>
<td>9K</td>
<td>12K</td>
<td>328(35)</td>
<td>246(47)</td>
</tr>
<tr>
<td>vpr</td>
<td>298M</td>
<td>5K</td>
<td>4K</td>
<td>84(49)</td>
<td>41(90)</td>
</tr>
<tr>
<td>gcc</td>
<td>255M</td>
<td>15K</td>
<td>4K</td>
<td>37(102)</td>
<td>19(198)</td>
</tr>
<tr>
<td>mesa</td>
<td>173M</td>
<td>25K</td>
<td>2.6K</td>
<td>10(259)</td>
<td>9(288)</td>
</tr>
<tr>
<td>art</td>
<td>1.0B</td>
<td>5K</td>
<td>12K</td>
<td>119(104)</td>
<td>108(114)</td>
</tr>
<tr>
<td>mcf</td>
<td>40M</td>
<td>5K</td>
<td>414</td>
<td>52(7.8)</td>
<td>41(10)</td>
</tr>
<tr>
<td>equake</td>
<td>342M</td>
<td>40K</td>
<td>5K</td>
<td>42(126)</td>
<td>33(161)</td>
</tr>
<tr>
<td>crafty</td>
<td>935M</td>
<td>8K</td>
<td>15K</td>
<td>739(20)</td>
<td>187(78)</td>
</tr>
<tr>
<td>ammp</td>
<td>818M</td>
<td>51K</td>
<td>13K</td>
<td>1129(12)</td>
<td>1036(13)</td>
</tr>
<tr>
<td>parser</td>
<td>929M</td>
<td>24K</td>
<td>14K</td>
<td>142(101)</td>
<td>98(147)</td>
</tr>
<tr>
<td>gap</td>
<td>277M</td>
<td>147K</td>
<td>5K</td>
<td>30(168)</td>
<td>20(252)</td>
</tr>
<tr>
<td>vortex</td>
<td>2087M</td>
<td>65K</td>
<td>–</td>
<td>537(N/A)</td>
<td>283(N/A)</td>
</tr>
<tr>
<td>bzip2</td>
<td>3029M</td>
<td>60K</td>
<td>–</td>
<td>660(N/A)</td>
<td>565(N/A)</td>
</tr>
<tr>
<td>twolf</td>
<td>76M</td>
<td>309</td>
<td>631</td>
<td>3(210)</td>
<td>2(316)</td>
</tr>
<tr>
<td>median</td>
<td>320M</td>
<td>12K</td>
<td>5178</td>
<td>47(101)</td>
<td>41(131)</td>
</tr>
<tr>
<td>mean</td>
<td>497M</td>
<td>28K</td>
<td>7343</td>
<td>201(100)</td>
<td>153(142)</td>
</tr>
</tbody>
</table>
### SPEC2K benchmark statistics

<table>
<thead>
<tr>
<th>prog.</th>
<th>$N$ (10^9)</th>
<th>$M$ (10^3)</th>
<th>$K$ (10^6)</th>
<th>$N/K$ (10^3)</th>
<th>CKlogM Refs/sec (10^6)</th>
</tr>
</thead>
<tbody>
<tr>
<td>gZip</td>
<td>24</td>
<td>232</td>
<td>7.2</td>
<td>3.4</td>
<td>1.9</td>
</tr>
<tr>
<td>vpr</td>
<td>41</td>
<td>159</td>
<td>6.9</td>
<td>5.9</td>
<td>2.9</td>
</tr>
<tr>
<td>gcc</td>
<td>16</td>
<td>360</td>
<td>2.3</td>
<td>7.3</td>
<td>3.5</td>
</tr>
<tr>
<td>mesa</td>
<td>31</td>
<td>28</td>
<td>1.8</td>
<td>17.5</td>
<td>8.4</td>
</tr>
<tr>
<td>art</td>
<td>30</td>
<td>11</td>
<td>12</td>
<td>2.4</td>
<td>1.7</td>
</tr>
<tr>
<td>mcf</td>
<td>14</td>
<td>315</td>
<td>27</td>
<td>0.50</td>
<td>0.3</td>
</tr>
<tr>
<td>equake</td>
<td>108</td>
<td>167</td>
<td>7.6</td>
<td>14.2</td>
<td>6.5</td>
</tr>
<tr>
<td>crafty</td>
<td>41</td>
<td>7.5</td>
<td>25</td>
<td>1.7</td>
<td>1.3</td>
</tr>
<tr>
<td>ammp</td>
<td>4.7</td>
<td>51</td>
<td>2.6</td>
<td>1.8</td>
<td>1.1</td>
</tr>
<tr>
<td>parser</td>
<td>78</td>
<td>102</td>
<td>21</td>
<td>3.7</td>
<td>1.9</td>
</tr>
<tr>
<td>gap</td>
<td>68</td>
<td>787</td>
<td>3.6</td>
<td>19.0</td>
<td>7.3</td>
</tr>
<tr>
<td>vortex</td>
<td>31</td>
<td>196</td>
<td>3.0</td>
<td>10.4</td>
<td>4.9</td>
</tr>
<tr>
<td>bzip2</td>
<td>42</td>
<td>530</td>
<td>7.8</td>
<td>5.4</td>
<td>2.4</td>
</tr>
<tr>
<td>twolf</td>
<td>106</td>
<td>12</td>
<td>48</td>
<td>2.2</td>
<td>1.8</td>
</tr>
<tr>
<td>median</td>
<td>36</td>
<td>162</td>
<td>7.4</td>
<td>4.6</td>
<td>2.1</td>
</tr>
<tr>
<td>mean</td>
<td>45</td>
<td>211</td>
<td>12.5</td>
<td>6.8</td>
<td>3.3</td>
</tr>
</tbody>
</table>

C=128 relative precision = 90%
SPEC2K/Gzip(ref input)

Distribution of all-window footprint

1) the y-axle show the value of footprint times cache size (64) since we view each cache line as basic data unit
2) the graph shows statistics of footprints from $10^{20}$ different windows
3) both axles are in log scale
Cache interference prediction

• Shared-cache is a dynamic system

• Circular effect:
  • when two programs A and B are run together, memory access by A affects the performance of B
  • The change in B affects its memory access
  • The change of B’s memory access in turn affects the performance of A

• Execution dilation
  • defined as: \[
  \frac{\text{Execution time of } A \text{ when sharing cache with } B}{\text{Execution time of } A \text{ when running alone}}
  \]
Construct dilation model step by step

- time model

- dilation definition (i=1, 2)

- cache model in shared-memory system

- combine all to get the iterative model

\[
\frac{\delta_1}{\delta_2} = F\left(\frac{\delta_1}{\delta_2}\right)
\]
Construct dilation model step by step

- **time model**
  \[ T = T^n n + T^p m^p + T^s m^s \]

- **dilation definition** \((i=1, 2)\)

- **cache model in shared-memory system**

- **combine all to get the iterative model**
  \[ \frac{\delta_1}{\delta_2} = F\left(\frac{\delta_1}{\delta_2}\right) \]
Construct dilation model step by step

• time model

\[ T = T^n n + T^p m^p + T^s m^s \]

• dilation definition (i=1, 2)

\[ \frac{T^n n_i + T^p m^p_i + T^s m^s_i x_i}{T^n n_i + T^p m^p_i + T^s m^s_i} = \delta_i \]

• cache model in shared-memory system

• combine all to get the iterative model

\[ \frac{\delta_1}{\delta_2} = F\left( \frac{\delta_1}{\delta_2} \right) \]
Construct dilation model step by step

- time model
  \[ T = T^n n + T^p m^p + T^s m^s \]

- dilation definition (i=1, 2)
  \[ \frac{T^n n_i + T^p m^p_i + T^s m^s_i x_i}{T^n n_i + T^p m^p_i + T^s m^s_i} = \delta_i \]

- cache model in shared-memory system
  \[ x_1 = \frac{P \left[ d_1 + f_2 \left( t(d_1) \frac{cpi_1}{cpi_2} \delta_1 \right) \right] \geq C}{P \left[ d_1 \geq C \right]} \]

- combine all to get the iterative model
  \[ \frac{\delta_1}{\delta_2} = F\left( \frac{\delta_1}{\delta_2} \right) \]

---

<p>| | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>( T_n )</td>
<td>average cost of each instruction</td>
</tr>
<tr>
<td>( n )</td>
<td># instructions</td>
</tr>
<tr>
<td>( T_p )</td>
<td>average cost of private-cache misses</td>
</tr>
<tr>
<td>( m_p )</td>
<td>#private cache misses</td>
</tr>
<tr>
<td>( T_s )</td>
<td>average cost of shared-cache misses</td>
</tr>
<tr>
<td>( m_s )</td>
<td>#shared-cache misses</td>
</tr>
</tbody>
</table>

\( \delta_i \): relative increase in the number of capacity misses in shared cache

\( d_1 \): reuse distance of program 1

\( f_2 \): footprint of program 2

\( C \): cache size

\( t(d_1) \): a function returning the corresponding window size of reuse distance \( d_1 \)
Evaluation

- test set of 15 SPEC2K programs on a dual-core machine (Intel Xeon CPU @ 2.66GHz, 4MB shared cache)
- PAW profile each of the 15 programs in a sequential run to collect reuse distance and footprint information for each program
- Predict dilations of each possible pair (105 in total) and rank it from least performance interference to heaviest
- Alternative ranking methods
  - random ranking: run the standard 15-choose-2 method
  - miss-rate based ranking: based on total miss ratio in sequential run
  - measured ranking: based on results from exhaustive testing of all co-run choices and gives the best possible result.
comparing different interference ranking

Y-axis shows the average slowdown for the first $x$ pairs
Summary

• a novel all-window footprint analysis algorithm
  • combines single-window relative-precision approximation and all-window constant-precision approximation to have an asymptotic cost of $O(CK\log M)$.
  • $CK\log M$ algorithm is 100 times faster than $N\log M$ algorithm on average over 14 SPEC2K benchmarks.

• an iterative algorithm to compute the non-linear, asymmetrical effect of cache sharing.
  • a tool for ranking program co-run choices without parallel testing
  • ranking result is close to that from exhaustive parallel testing
• Thanks

• Q&A