A Cache Capacity Sharing Model (Work in Progress)

Vlastimil Babka

DEPARTMENT OF DISTRIBUTED AND DEPENDABLE SYSTEMS
FACULTY OF MATHEMATICS AND PHYSICS
CHARLES UNIVERSITY IN PRAGUE

http://d3s.mff.cuni.cz
Motivation

- Predict performance of workloads running in parallel on a multicore processor

![Diagram showing competition for cache capacity between two workloads on a multicore processor.]
Processor Cache Essentials: Associativity

- Cache = a number of fixed size (64B) blocks (lines) holding least recently accessed data
- Typically set-associative, with (pseudo-)LRU replacement policy

Example: 4-way cache

Rows = cache sets (index determined by part of memory address) (Ex: ...11001100101101)

Set

<table>
<thead>
<tr>
<th>000</th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>001</td>
<td></td>
<td></td>
</tr>
<tr>
<td>010</td>
<td></td>
<td></td>
</tr>
<tr>
<td>011</td>
<td></td>
<td></td>
</tr>
<tr>
<td>100</td>
<td></td>
<td></td>
</tr>
<tr>
<td>101</td>
<td></td>
<td></td>
</tr>
<tr>
<td>110</td>
<td></td>
<td></td>
</tr>
<tr>
<td>111</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Columns = ways
Processor Cache Essentials: LRU Policy

- Example: 4-way cache with LRU
- A, B, C, D, E – memory blocks that map to the same cache set

Columns = ways in a single cache set
(logically ordered by LRU)

Access to D (hit)

Access to E (miss)

RAM
Stack distance profiles

- Stack distance of an access in a cache set
  - = number of distinct cache lines accessed since previous access to the same line
  - = the position of the line in the LRU stack when this line is accessed

- Stack distance profile (histogram) of a cache set
  - = counts (frequencies) of accesses depending on their stack distance
Stack distance profiles

- Miss ratio = last column in stack profile
- Hit ratio = sum of all other columns

Columns = ways in a single cache set (logically ordered by LRU)

<table>
<thead>
<tr>
<th>Time</th>
<th>Access to D (hit)</th>
<th>Access to E (miss)</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

RAM

Access frequency

0.00 0.20

Stack distance

1 2 3 4 5
The proposed cache sharing model

- Scenario: (Two) workloads sharing an L2 cache
- Goal: predict the increase of cache misses due to limited shared cache capacity, and the slowdown of the workloads
- Input: for each workload
  - Stack distance profile (averaged over all cache sets)
  - Accesses per instruction (API)
  - Instructions per cycle (IPC) when run in isolation
  - L2 cache miss penalty (in processor cycles)
- Output: modified stack distance profile and IPC, due to cache capacity sharing
The model: High-level overview

- Consider for each stack distance $1 \leq d \leq A$
  - Average time between two accesses to a cache line, where the second access has a stack distance $d$
    - Called *reuse time* of stack distance $d$, $r_d$
    - During this time, the cache line is “vulnerable” to accesses from the other workload
  - The number of distinct cache lines accessed by the other workload during the time $r_d$
  - The effective stack distance will be increased by the accesses of the other workload
Deriving reuse time $r_d$

- Average time between two accesses to a line, where the second access has a stack distance $d$
- Observation: to access a line $L$ with stack distance $d$, the following has to happen:
  - Previous access to $L$ puts it on the most recently used position
  - Accessing $d - 1$ distinct cache lines shifts line $L$ to stack distance $d$
  - Accessing $L$ second time, which means accessing $d$th distinct cache line
  - Therefore, $r_d = \text{the time to achieve occupation of } d \text{ distinct cache lines in an initially empty cache set}$
Deriving mean time to achieve occupation

- $r_d$ = mean time to achieve occupation of $d$ cache lines in an initially empty cache set
- Model the cache set as a Markov chain
  - states 0 ... $d$ representing the number of occupied lines
  - discreet steps = cache accesses, which may hit or miss, depending on current occupation
- Transition matrix $P_d$ is defined as follows
  - $p_{0,1} = 1$, $p_{d,d} = 1$ (absorbing state)
  - $p_{i,i} = \text{hit}(i)$, $1 \leq i < d$
  - $p_{i,i+1} = 1 - \text{hit}(i)$, $1 \leq i < d$
Deriving mean time to achieve occupation

\[ \text{hit}(i) = \sum_{j=1}^{i} sdp(j) \]

Ex.: \( i=3 \)

- Mean number of accesses to achieve occupation of \( d \) cache lines = mean time (steps) to absorption in the Markov chain, starting from state 0
  - \( t_d \), can be derived from the transition matrix

- Converting \( t_d \) to processor cycles:
  \[ r_d = \frac{t_d}{API_1 \cdot IPC_1} \]
The model: High-level overview

- Consider for each stack distance $1 \leq d \leq A$
  - Average time between two accesses to a cache line, where the second access has a stack distance $d$
    - Called *reuse time* of stack distance $d$, $r_d$
    - During this time, the cache line is “vulnerable” to accesses from the other workload
  - The number of distinct cache lines accessed by the other workload during the time $r_d$
  - The effective stack distance will be increased by the accesses of the other workload
Interference from the other workload

- Average number of accesses from the other workload during the time $r_d$:
  $$a_d = r_d \cdot API_2 \cdot IPC_2$$

- How many distinct cache lines will be accessed?
  - Again, equal to the cache set occupation the other workload would achieve in this number of accesses
  - Can be derived using the same kind of Markov chain, but constructed for workload 2, with states 0 ... A
  - We can then derive a probability vector $D(n)$, where $D_m(n) = P$ (after n accesses, occupation = m)
    $$D(n) = u Q^n, \text{ where } u = \{1, 0, ..., 0\}$$
  - $n = a_d$, non-integers approx. by linear interpolation
The model: High-level overview

• Consider for each stack distance $1 \leq d \leq A$
  
  ✔  ▪ Average time between two accesses to a cache line, where the second access has a stack distance $d$
    • Called *reuse time* of stack distance $d$, $r_d$
    • During this time, the cache line is “vulnerable” to accesses from the other workload
  
  ✔  ▪ The number of distinct cache lines accessed by the other workload during the time $r_d$
  
  ➡  ▪ The effective stack distance will be increased by the accesses of the other workload
Deriving partial stack distance profile

• $D(a_d)$ gives us distribution for the number of distinct lines accessed by the second workload between the first workload's reuses of lines with distance $d$
  
  ▪ We take the accesses with distance $d$ in the original profile, and distribute them between distances $d, d+1 \ldots d+A$, proportionally to $D(a_d)$

• (resulting distances larger than $A+1$ are trimmed to $A+1$)

Example: $D(ad) = (0.2, 0.3, 0.2, 0.2, 0.1)$
Combining partial distance profiles

- For each stack distance $1 \leq d \leq A$, derive a partial profile for accesses with original distance $d$
- Add up (distance-wise) the partial profiles
  - Misses in the original profile are added unchanged
Using the new stack distance profile

- We determine the difference in MPA due to sharing
- Determine the difference in CPI
  \[ \Delta CPI = \Delta MPA \times API \times miss\_penalty \]
- Derive IPC under sharing
  \[ IPC' = 1/(1/IPC + \Delta CPI) \]
Putting it together

• We determine the new IPC or first workload due to interference from second workload, and vice versa

• Problem: the IPC values are both input and output of the model!
  - IPC determines the frequency of cache accesses
  - Recall: $r_d = \frac{t_d}{API_1 \cdot IPC_1}$, $a_d = r_d \cdot API_2 \cdot IPC_2$

• Solution: iterative
  - Start with isolated IPC values
  - In each step, determine new IPC values from previous
  - Terminate when difference against previous step < $\varepsilon$
Selected example result

FFT, IPC=1.06

LBM, IPC=0.19

IPC’_pred=0.48
IPC’_meas=0.49

IPC’_pred=0.16
IPC’_meas=0.15
Obtaining model inputs

- Stack distance profiles difficult to obtain
  - Simulation or instrumentation is expensive
- Metrics as IPC or miss penalty are most accurately obtained from running on target system
- A method for obtaining profiles and other metrics from runtime recently proposed
  - Xu et. al, ISPASS'10
    - Based on artificial workload *stressmark* and cache accesses/misses read via performance counters
  - We are trying to adopt it
    - Implementation in RIP and RPG frameworks by T. Martinec
Obtaining stack profiles with stressmark

- Idea: a workload running in parallel with measured workload, frequently accessing $k$ ways in each cache set to keep them in cache
  - The measured workload has effectively only $A - k$ ways for its own data, its $MPA_k$ will increase with $k$
  - Perf. counters used to obtain MPA values
  - Difference in measured workload's MPA between $k$ and $k+1$ ways occupied by interfering workload = the frequency of accesses with distance $(A - k)$
Stack profiles via stressmark: limitations

- Caveat: Accessing $k$ ways by the stressmark does not ensure all $k$ ways are occupied
  - The measured workload may evict some data
- Solution: measure also MPA of the stressmark to determine how many ways are effectively kept
  - The stack distance profile of stressmark must be known
  - Uniform (“flat”) profile is achieved by selecting ways to access randomly

Example: $k=4$
MPA = 0.25
→ $k' = 4 * 0.25 = 3$ ways that are effectively occupied
Stack profiles via stressmark: limitations

- The number of ways actually occupied by stressmark is \( k' \leq k \)
  - The measured workload occupies \((A - k')\) ways
  - Even with \( A \) ways (whole cache) accessed by stressmark, the measured workload might occupy more than 1 way
  - We might therefore not be able to distinguish accesses with distance lower than some \( L > 1 \)
  - We assume the profile for distances \( \leq L \) is flat
Obtaining the rest of model inputs

• IPC and API of measured workload
  ▪ Performance counters in isolated execution
• Miss penalty of measured workload
  ▪ Using the cache miss counts and execution times (in processor clocks) obtained when executing with the stressmark
    ▪ Considering difference ($\Delta$) against isolated execution
    ▪ We get a set of pairs ($\Delta$misses, $\Delta$clocks) for varying $k$
      ▪ Linear regression:
        $\Delta$clocks = $\Delta$misses $\times$ miss_penalty
Related models

- Chandra et. al, HPCA'2005
  - Concerned only with misses, not timing penalties
  - More complex input than a stack distance profile
  - No feedback for the relative access frequencies

- Xu et. al, ISPASS'10
  - Same inputs, different approach, also involves iteration
  - Outputs: effective cache sizes, IPC
    - We get effective stack distance profiles, potential for model composability
  - Some details unclear (e.g. related to prefetching)
  - Comparing the results would be prudent
More (preliminary) results
Potential sources of error

- Stressmark not achieving full cache occupation
  - Trying better access patterns based on RIB/RIP experience, also better randomization (T.Martinec)
- Too much “averaging” (e.g. over all cache sets) and assumptions (independent accesses...)
- Capacity is not the only sharing factor
  - Request sharing capacity
  - Prefetching (for now, disabled)
  - Shared memory
  - Cache miss penalty not constant
Thank you...

• Questions ?