Adapting Cache Partitioning Algorithms to Pseudo-LRU Replacement Policies

Kamil Kedzierski\textsuperscript{1,3}, Miquel Moreto\textsuperscript{1,3}, Francisco J. Cazorla\textsuperscript{2,3}, Mateo Valero\textsuperscript{1,3}

\textsuperscript{1} Technical University of Catalonia
\textsuperscript{2} Spanish National Research Council
\textsuperscript{3} Barcelona Supercomputing Center
Chip Multiprocessors (CMPs)

- CMPs are good representative of the transition from ILP to TLP

- Current CMPs share the Last Level Cache (LLC)
  - Pros: Better utilization than a private LLC, which translates into improved performance
  - Cons: LLC has been identified as a source of contention between threads
    - Cache competition may lead to performance degradation

- Cache Partitioning Algorithms (CPAs) control the interaction between threads
  - CPAs can deliver a flexible and easy-to-manage infrastructure to control threads’ behavior in shared caches
  - CPAs have become the central element of current QoS frameworks for CMPs
Cache Partitioning Algorithms

- We focus on dynamic CPAs
  - Execution divided into time intervals
  - At interval boundary we select a new cache partition based on the behavior in the previous interval(s)

- Cache partitioned at the way granularity
  - Each thread assigned a number of ways, between 1 and A – N
    - A – associativity
    - N – number of cores

- Main components of CPAs
  - Profiling logic
  - Partitioning logic
  - Enforcement logic
Motivation

- Limiting factors to implement CPAs in real processors
  - Size of the profiling logic (Auxiliary Tag Directory)
    - Its size can be similar to the size of the L1 cache
    - Received significant attention
      - Sampled profiling logic
      - No profiling (check all cases and select the best performing one)
    - We conclude the problem has been solved
  - Replacement scheme
    - So far solutions focus on LRU replacement scheme
    - LRU has high implementation cost
    - High associativity caches use pseudo-LRU schemes
    - It has not been shown how current CPAs work with pseudo-LRU → Problem not solved
Outline

- Replacement schemes
- Problem definition for pseudo-LRU schemes
- Profiling for pseudo-LRU
- Results
- Conclusions
Outline

- Replacement schemes
  - LRU: Least Recently Used
  - NRU: Not Recently Used (UltraSPARC) (pLRU)
  - BT: Binary Tree (IBM) (pLRU)
- Problem definition for pseudo-LRU schemes
- Profiling for pseudo-LRU
- Results
- Conclusions
Least Recently Used (LRU)

- **Hit**
  - Each line that is between the MRU line and the hit line increments its LRU bits
  - In the worst case positions of all the lines are updated
  - Hit line is promoted to the MRU position

- **Miss**
  - Search for value 3 in corresponding replacement bits
  - Promote the line to MRU position and set its bits to 0
  - Increase all the other bits
Not Recently Used (NRU)

- **Hit**
  
  - Set corresponding used bit to 1
  
  - If it causes all used bits to be 1, reset all the other bits

- **Miss**
  
  - Start looking for a victim at the position pointed by the replacement pointer
  
  - Search for used bit equal 0
  
  - Set corresponding used bit to 1
  
  - If it causes all used bits to be 1, reset all the other bits
  
  - Rotate the replacement pointer forward one way
Binary Tree (BT)

- **Hit**
  - B access
  - Update corresponding bits so that they point to MRU position

- **Miss**
  - E access
  - Update corresponding bits so that they point to MRU position
Summary

- LRU requires more replacement bits
- LRU requires more information to update
- Current processors available on the market use pseudo-LRU replacement policies
Outline

- Replacement schemes

- Problem definition for pseudo-LRU schemes
  - Cache Partitioning Algorithms
  - Profiling Logic

- Profiling for pseudo-LRU

- Results

- Conclusions
Cache Partitioning Algorithms

- Profiling Logic
  - Observe each thread behavior in L2 cache

- Partitioning Logic
  - Make the decision on how to partition the cache
  - We use way partitioning

- Enforcement Logic
  - Put the partitions into practice
Profiling Logic for LRU

- Auxiliary Tag Directory (ATD)
  - Separate copy of the tag directory with the same associativity
  - Simulates single-threaded behavior
  - On every cache access reports LRU stack position to SDH

- Stack Distance Histogram (SDH)
  - Gathers stack positions
  - Allows us to derive the miss curve of the thread as a function of the ways assigned to a thread
Profiling Background for LRU

- Building SDH, ATD content (1 set)

- Building miss curve

Replacement schemes
Problem definition for pseudo-LRU schemes
Profiling for pseudo-LRU
Results
Conclusions

IPDPS, April 2010
Kamil Kedzierski
kkedzier@ac.upc.edu
Profiling in pseudo-LRU?

Problem definition for pseudo-LRU schemes
Profiling for pseudo-LRU
Results
Conclusions

... but what is the stack position?

LRU

- B access

<table>
<thead>
<tr>
<th>3</th>
<th>A</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>B</td>
</tr>
<tr>
<td>0</td>
<td>C</td>
</tr>
<tr>
<td>2</td>
<td>D</td>
</tr>
</tbody>
</table>

MRU

<table>
<thead>
<tr>
<th>0</th>
<th>A</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>B</td>
</tr>
<tr>
<td>1</td>
<td>C</td>
</tr>
<tr>
<td>0</td>
<td>D</td>
</tr>
</tbody>
</table>

NRU

<table>
<thead>
<tr>
<th>0</th>
<th>A</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>B</td>
</tr>
<tr>
<td>1</td>
<td>C</td>
</tr>
<tr>
<td>0</td>
<td>D</td>
</tr>
</tbody>
</table>

BT

<table>
<thead>
<tr>
<th>0</th>
<th>A</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>B</td>
</tr>
<tr>
<td>1</td>
<td>C</td>
</tr>
<tr>
<td>0</td>
<td>D</td>
</tr>
</tbody>
</table>

p-LRU

<table>
<thead>
<tr>
<th>0</th>
<th>A</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>B</td>
</tr>
<tr>
<td>1</td>
<td>C</td>
</tr>
<tr>
<td>0</td>
<td>D</td>
</tr>
</tbody>
</table>

BUT: don't know

BUT: don't know
Outline

- Replacement schemes
- Problem definition for pseudo-LRU schemes
- Profiling for pseudo-LRU
  - NRU scheme
  - BT scheme
  - Limitations
- Results
- Conclusions
Profiling in NRU

- Used bits in a 4-way ATD using NRU for three consecutive accesses. The arrows point to the line of the last access with the estimated stack distance next to it.

  - Count number of used bits equal 1 (U)
    - If current used bit = 1, stack distance is between 1 and U
    - If current used bit = 0, stack distance is between U+1 and A

ATD for CDD accesses

ATD for ABC accesses
Profiling in BT

Estimated SDH profiling

Decoder for ID bits extraction from the way number

Replacement schemes
Problem definition for pseudo-LRU schemes
Profiling for pseudo-LRU
Results
Conclusions
Limitations

- Over- vs. under-estimation of the position in the pseudo-LRU stack
- Two stacks with the same BT bits affect profiling accuracy

We evaluate three scaling factors:

- 1.0 x used bits equal “1”
  - assume stack distance 4
- 0.75 x used bits equal “1”
  - assume stack distance 3
- 0.5 x used bits equal “1”
  - assume stack distance 2
Outline

- Replacement schemes
- Problem definition for pseudo-LRU schemes
- Profiling for pseudo-LRU
- Results
- Conclusions
Without Cache Partitioning

- Performance of LRU, NRU and BT. Analysis for 1, 2, 4 and 8 core CMPs using a 16-way 2MB L2 cache with 128 bytes lines

- Is it worth to develop complex, area expensive, power hungry LRU replacement for high associativity caches and win 2% - 5% in performance?
Cache partitioning

- Analysis done for a 16-way 2MB L2 cache with 128 bytes lines

- Counters (any \( K \) ways out of \( A \)) vs. Masks (specific \( K \) ways out of \( A \))

- Negligible difference for 1 million cycles sampling interval
Cache partitioning

- Analysis done for a 16-way 2MB L2 cache with 128 bytes lines

- We select 0.75 factor as a winner

- Random-like NRU replacement evicts not least recently used data
  - One replacement pointer for all the sets
  - Gets significant when the number of cores increases
Cache partitioning

- Analysis done for a 16-way 2MB L2 cache with 128 bytes lines

 Alternating nodes do not evict least recently used line

  - Misses not evenly distributed among partition
  - Gets significant when the number of cores increases
Outline

- Replacement schemes
- Problem definition for pseudo-LRU schemes
- Profiling for pseudo-LRU
- Results
- Conclusions
Conclusions

- We propose a complete partitioning design that targets two pseudo-LRU replacement policies.
  - Not Recently Used, implemented in the L2 cache in the market UltraSPARC T1/T2 processor
  - Binary Tree proposed by IBM
- We identify profiling logic as the main source of the so-far lack of CPA implementations
- The results show a negligible performance degradation with respect to the LRU-based CPA
  - For NRU our design loses as much as 0.3%, 3.6% and 7.3% throughput for 2, 4 and 8-core CMP architectures, respectively
  - For BT the proposal degrades throughput by 1.4%, 3.4% and 9.7%, respectively
Thank you

Q & A

Kamil Kedzierski$^{1,3}$, Miquel Moreto$^{1,3}$, Francisco J. Cazorla$^{2,3}$, Mateo Valero$^{1,3}$

$^1$ Technical University of Catalonia
$^2$ Spanish National Research Council
$^3$ Barcelona Supercomputing Center