T-SPaCS – A Two-Level Single-Pass Cache Simulation Methodology

Wei Zang and Ann Gordon-Ross+
University of Florida
Department of Electrical and Computer Engineering

+ Also Affiliated with NSF Center for High-Performance Reconfigurable Computing
Introduction

- Power hungry caches are a good candidate for optimizations
- Different applications have vastly different cache requirements
  - Configure cache parameters: size, line size, associativity
  - Cache parameters that do not match an application’s behavior can waste over 60% of energy (Gordon-Ross 05)

- **Cache tuning**
  - Determine appropriate cache parameters (*cache configuration*) to meet optimization goals (e.g., lowest energy)
  - Difficult to determine the *best cache configuration* given very large design spaces for highly configurable caches
Simulation-Based Cache Tuning

- Cache tuning at design time via simulation
  - Performed by the designer
  - Typically iterative simulation using exhaustive or heuristic methods

\[ C_1, C_2, C_3, ..., C_n \] are the \( n \) cache configurations in design space
Single-Pass Cache Tuning

- Simultaneously evaluate multiple cache configurations during one execution
  - Trace-driven cache simulation
    - Use memory reference trace

Embedded Application → Generate trace file through single functional simulation → Single-pass trace-driven cache simulation

- Miss rate with \( c_1 \)
- Miss rate with \( c_2 \)
- Lowest energy \( c_3 \)
- Miss rate with \( c_n \)

Speedup simulation time
Previous Work in Single-Pass Simulation

- Stack-based algorithm
  - Stack data structure stores access trace
  - State-of-the-art: 14X speedup over iterative (Viana 08)

- Tree data structure-based algorithm
  - Decreased simulation time
  - Complex data structures, more storage requirements

- Limitation
Contributions

- **Two-level Single-Pass trace-driven Cache Simulation methodology – T-SPaCS**
- Use a stack-based algorithm to simulate both the level one and level two caches simultaneously
- Accurately determine the optimal energy cache configuration with low storage and simulation time complexity
Single-level Cache Simulation

• Stack-based single-pass trace-driven cache simulation for single-level cache

One cache configuration in design space: block size = 4 \( (2^2) \), number of cache sets = 8 \( (2^3) \)
Single-level Cache Simulation

- Stack-based single-pass trace-driven cache simulation for single-level cache

One cache configuration in design space:
block size = 4 ($2^2$), number of cache sets = 8 ($2^3$)

### Trace addresses
```
010 111 10
110 101 00
111 111 01
101 010 00
010 111 10
001 111 10
```

### Processing address
(010) 111 10

### Search stack
No previous access in stack

### Compulsory miss

### Stack update
(010) 111 10
Single-level Cache Simulation

- Stack-based single-pass trace-driven cache simulation for single-level cache

One cache configuration in design space:
- block size = 4 \( (2^2) \),
- number of cache sets = 8 \( (2^3) \)

Trace addresses:
- 010 111 10
- 110 101 00
- 111 111 01
- 101 010 00
- 010 111 10
- 001 111 10

Processing address:
- (101) 010 00

Search stack:
- No previous access in stack

Compulsory miss:
- Stack update

Stack:
- (101) 010 00
- (001) 111 10
-
Single-level Cache Simulation

- Stack-based single-pass trace-driven cache simulation for single-level cache

One cache configuration in design space:
block size = 4 \( (2^2) \), number of cache sets = 8 \( (2^3) \)

Trace addresses

<table>
<thead>
<tr>
<th>010</th>
<th>111</th>
<th>10</th>
</tr>
</thead>
<tbody>
<tr>
<td>110</td>
<td>101</td>
<td>00</td>
</tr>
<tr>
<td>111</td>
<td>111</td>
<td>01</td>
</tr>
<tr>
<td>101</td>
<td>010</td>
<td>00</td>
</tr>
<tr>
<td>010</td>
<td>111</td>
<td>10</td>
</tr>
<tr>
<td>001</td>
<td>111</td>
<td>10</td>
</tr>
</tbody>
</table>

Processing address
(111) 111 01

Search stack
No previous access in stack

Compulsory miss

Stack update

Stack

(111) 111 01
(010) 111 10
(001) 111 10
Single-level Cache Simulation

• Stack-based single-pass trace-driven cache simulation for single-level cache

One cache configuration in design space:
block size = 4 ($2^2$), number of cache sets = 8 ($2^3$)

Trace addresses

<table>
<thead>
<tr>
<th>Tag Index</th>
<th>Block Offset</th>
</tr>
</thead>
<tbody>
<tr>
<td>010</td>
<td>111 10</td>
</tr>
<tr>
<td>110</td>
<td>101 00</td>
</tr>
<tr>
<td>111</td>
<td>111 01</td>
</tr>
<tr>
<td>101</td>
<td>010 00</td>
</tr>
<tr>
<td>010</td>
<td>111 10</td>
</tr>
<tr>
<td>001</td>
<td>111 10</td>
</tr>
</tbody>
</table>

Processing address

(110) 101 00

Search stack

No previous access in stack

Compulsory miss

Stack update

<table>
<thead>
<tr>
<th>Stack</th>
</tr>
</thead>
<tbody>
<tr>
<td>(110) 101 00</td>
</tr>
<tr>
<td>(101) 010 00</td>
</tr>
<tr>
<td>(010) 111 10</td>
</tr>
<tr>
<td>(001) 111 10</td>
</tr>
</tbody>
</table>
Single-level Cache Simulation

- Stack-based single-pass trace-driven cache simulation for single-level cache

One cache configuration in design space:
block size = 4 ($2^2$), number of cache sets = 8 ($2^3$)

Trace addresses

<table>
<thead>
<tr>
<th></th>
<th>010</th>
<th>111</th>
<th>10</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>110</td>
<td>101</td>
<td>00</td>
</tr>
<tr>
<td></td>
<td>111</td>
<td>111</td>
<td>01</td>
</tr>
<tr>
<td></td>
<td>101</td>
<td>010</td>
<td>00</td>
</tr>
<tr>
<td></td>
<td>010</td>
<td>111</td>
<td>10</td>
</tr>
<tr>
<td></td>
<td>001</td>
<td>111</td>
<td>10</td>
</tr>
</tbody>
</table>

Processing address

(010) 111 10

Search stack

Conflicts: blocks that map to the same cache set as processed address

Conflicts # = 1
cache associativity >= 2, hit

Stack update

Same block

Stack

<p>| | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>(110) 101 00</td>
<td></td>
</tr>
<tr>
<td>(111)</td>
<td>111 01</td>
<td></td>
</tr>
<tr>
<td></td>
<td>(101) 010 00</td>
<td></td>
</tr>
<tr>
<td>(010)</td>
<td>111 10</td>
<td></td>
</tr>
<tr>
<td></td>
<td>(001) 111 10</td>
<td></td>
</tr>
</tbody>
</table>

12
Two-Level Cache Simulation

- Stack-based single-level cache simulation maintains one stack to record L1 access trace
- Naïve adaption of stack-based single-level cache simulation to two-level caches requires multiple stacks
  - Assumes inclusive cache hierarchy
  - L1 access trace: one stack based on memory reference trace
  - L2 access trace: depends on L1 miss
  - Requires $n$ stacks for $n$ L1 configurations
  - Disadvantage: large storage space and lengthy simulation time
- To reduce storage space and simulation time

**Exclusive cache hierarchy!**
**Inclusive vs. Exclusive Hierarchy**

**Inclusive Operation (L1/L2 LRU)**

<table>
<thead>
<tr>
<th>Trace</th>
<th>L1 (2-way)</th>
<th>L2 (2-way)</th>
<th>Hit/miss</th>
</tr>
</thead>
<tbody>
<tr>
<td>A</td>
<td>A</td>
<td>A</td>
<td>L1/L2 miss</td>
</tr>
<tr>
<td>B</td>
<td>B A</td>
<td>B A</td>
<td>L1/L2 miss</td>
</tr>
<tr>
<td>A</td>
<td>A B</td>
<td>B A</td>
<td>L1 hit</td>
</tr>
<tr>
<td>C</td>
<td>C A</td>
<td>C B</td>
<td>L1/L2 miss</td>
</tr>
<tr>
<td>B</td>
<td>B C</td>
<td>B C</td>
<td>L2 hit</td>
</tr>
</tbody>
</table>

**Exclusive Operation (L1 LRU, L2 FIFO-like)**

<table>
<thead>
<tr>
<th>Trace</th>
<th>L1 (2-way)</th>
<th>L2 (2-way)</th>
<th>Hit/miss</th>
</tr>
</thead>
<tbody>
<tr>
<td>A</td>
<td>A</td>
<td></td>
<td>L1/L2 miss</td>
</tr>
<tr>
<td>B</td>
<td>B A</td>
<td></td>
<td>L1/L2 miss</td>
</tr>
<tr>
<td>A</td>
<td>A B</td>
<td></td>
<td>L1 hit</td>
</tr>
<tr>
<td>C</td>
<td>C A</td>
<td>B</td>
<td>L1/L2 miss</td>
</tr>
<tr>
<td>B</td>
<td>B C</td>
<td>A</td>
<td>L2 hit</td>
</tr>
</tbody>
</table>

*Separate L1 and L2? L2 access is decided by L1*

**Combined cache**

Simulate L1 & combined cache and derive L2 cache

*One Stack! Reduced storage space and simulation time*
T-SPaCS Overview

Execute application

Access trace file

Cache config. in design space

Stack

Stack processing for conflicts for each $B$ and $S'_1$, $S'_2$

$T[t]$ is L1 hit/miss

$T[t]$ is L2 hit/miss

L1 analysis based on conflicts # for all $W'_1$

L1 miss

L2 analysis

Stack update

Accumulated L1 & L2 misses for all cache config.

$B$: block size
$S'_1$: number of sets in L1
$S'_2$: number of sets in L2
$W'_1$: number of associativities in L1
$W'_2$: number of associativities in L2

T-SPaCS

T[N]

::

T[t]

::

T[3]

T[2]

T[1]
L2 Analysis

- Stack processing for combined cache
  - Conflict evaluation (same as single-level cache)
- **Compare-exclude** operation to derive L2 conflicts
  - Conflicts for combined cache still contain some conflicts stored in L1
  - Isolate the exclusive L2 conflicts
  - Based on three different inclusion relationships; consider as three scenarios

Scenario 1: $S' = S^2$

Scenario 2: $S' < S^2$

Scenario 3: $S' > S^2$

$L1$ conflicts $L2$ conflicts

Conflicts for combined cache

$L1$ conflicts $L2$ conflicts

$L1$ conflicts $L2$ conflicts

$S'$: number of sets in L1
$S^2$: number of sets in L2
Scenario 1: $S^1 = S^2$

$L1$ set (2 ways) → $L2$ set (2 ways)

Access X1

Conflicts: X4, X3, X2

L1 miss when $W^1 = 2$

Blocks in L1

L2 conflicts

L2 conflicts # = 1, L2 hit when $W^2 \geq 2$

$S^1$: number of sets in L1

$S^2$: number of sets in L2
Scenario 2: \( S^1 < S^2 \)

\[ \begin{align*}
&X1 \quad Y4 \quad X3 \quad X2 \quad Y1 \quad X1 \\
&\text{Trace} \\
&\text{L1 set (2 ways)} \\
&\text{L2 set (2 ways)} \\
&\text{L2 set (2 ways)} \\
&\text{Access X1} \\
&\text{L1 conflicts: } Y4 \quad X3 \quad X2 \quad Y1 \\
&\text{L1 miss when } W^1 = 2 \\
&\text{Conflicts for combined cache: } X3 \quad X2 \\
&\text{L2 conflicts } # = 1, \text{ L2 hit when } W^2 \geq 2
\end{align*} \]

\( S^1 \): number of sets in \( L^1 \)
\( S^2 \): number of sets in \( L^2 \)
Special Case in Scenario 2

Access X2
Hit in L2

Access X5

- From cache: miss in L1/L2
- From compare-exclude operation:

Blocks in L1: X2, Y2

Conflicts for combined cache: X2, X1, X3, X4

L2 conflicts \# = 3 < 4, L2 hit!

Inaccurate! L2 conflicts should count BLK after X4

Solution: occupied blank labeling

- Bit-array to label BLK, ‘set’ bit: an BLK follows labeled address.
- In processing X2, label BLK with the \( W^2 \)-th L2 conflict(X4).
- In processing X5, detected BLK in the bit-array of X4. (i.e., X4 is the last block in L2). X5 is L2 miss.

\( S_1 \): number of sets in L1
\( S_2 \): number of sets in L2

Occupied blank (fetching X2 evicted Y1 that maps to different L2 set)
Scenario 3: $S^1 > S^2$

$L1$ set (2 ways)

$L2$ set (4 ways)

Access $X1$

L1 Conflicts: $X4$, $X3$, $X2$

L1 miss when $W^1=2$

Blocks in L1

Conflicts for combined cache: $X4$, $Y3$, $X3$, $Y2$, $X2$, $Y1$

Conflicts for complimentary set: $Y3$, $Y2$, $Y1$

Blocks in complimentary set

L2 conflicts $# = 2$, L2 hit when $W^2>=3$

$S^1$: number of sets in L1

$S^2$: number of sets in L2
Accelerate Stack Processing

- Stack processing: *very time consuming!*
- Conflicts for one L1 configuration repeatedly compared with conflicts for all L2 configurations
- Save conflicts in a tree structure for later reference

Processed address (1001)10110110

<table>
<thead>
<tr>
<th>$S_1$</th>
<th>$S_2$</th>
<th>$S_3$</th>
<th>$S_4$</th>
<th>$S_5$</th>
<th>$S_6$</th>
<th>$S_7$</th>
</tr>
</thead>
<tbody>
<tr>
<td>4</td>
<td>8</td>
<td>16</td>
<td>32</td>
<td>64</td>
<td>128</td>
<td>256</td>
</tr>
</tbody>
</table>

- $S_1$: number of sets in L1
- $S_2$: number of sets in L2

$S_{\text{min}}$: Store conflicts with “10” index

Stack address

Conflict Evaluation

Store in tree node

Next stack address

Complementary sets
Experiment Setup

- **Design space**
  - L1: cache size (2k\(\rightarrow\)8k bytes); block size (16B\(\rightarrow\)64B); associativity (direct-mapped\(\rightarrow\)4-way)
  - L2: cache size (16k\(\rightarrow\)64k bytes); block size (16B\(\rightarrow\)64B); associativity (direct-mapped\(\rightarrow\)4-way)
  - 243 configurations
    - Exclusive cache requires L1 and L2 to have the same block size

- 24 benchmarks from EEMBC, Powerstone, and MediaBench
- Modify ‘sim-fast’ to generate access traces
- Modify ‘sim-cache’ to simulate exclusive hierarchy cache to produce the *exact* miss rates for comparison
- Build energy model to determine optimal cache configuration with minimum energy consumption (Gordon-Ross 09)
Results – Miss Rate Accuracy

• L1 miss rate
  – 100% accurate for all benchmarks

• L2 miss rate
  – Accurate for 240 configurations (99% of the design space)
  – Across all benchmarks

<table>
<thead>
<tr>
<th>Max. average miss rate err.</th>
<th>Max. standard deviation</th>
<th>Max. absolute miss rate err.</th>
</tr>
</thead>
<tbody>
<tr>
<td>1.16%</td>
<td>0.64%</td>
<td>1.55%</td>
</tr>
</tbody>
</table>

• Inaccuracy comes from Scenario 3: $S^1 > S^2$
  – Reason
    • Multiple L1 sets evict blocks in the same L2 set
    • Eviction order is not consistent to access order
  – Introduced error is small

• Tuning accuracy: accurately determined energy optimal cache!

$S^1$: number of sets in L1
$S^2$: number of sets in L2
Simplified-T-SPaCS

• Omit occupied blank labeling to reduce complexity and simulation time

• Tradeoff – additional miss rate error
  – L2 miss rate errors for additional 228 configurations where $S' < S^2$ (95% of the design space)
  – Across all benchmarks

<table>
<thead>
<tr>
<th>Max. average miss rate err.</th>
<th>Max. standard deviation</th>
<th>Max. absolute miss rate err.</th>
</tr>
</thead>
<tbody>
<tr>
<td>0.71%</td>
<td>0.90%</td>
<td>3.35%</td>
</tr>
</tbody>
</table>

• Tuning accuracy: accurately determined energy optimal cache!

$S'$: number of sets in L1
$S^2$: number of sets in L2
Simulation Time Efficiency

Max 24.7X
Max 18X
Avg 15.5X
Avg 8X
Conclusions

• T-SPaCS simulates instruction cache with exclusive hierarchy in a single-pass
• T-SPaCS reduces the storage and time complexity
  – T-SPaCS is 8X faster than iterative simulation on average
  – Simplified-T-SPaCS increases average simulation speedup to 15X at the expense of inaccurate miss rates for 95% of the design space
  – Both T-SPaCS and simplified-T-SPaCS can determine accurate optimal energy configurations
• Our ongoing work extends T-SPaCS to simulate data and unified cache, and implement in hardware for dynamic cache tuning