RapidMRC: Approximating L2 Miss Rate Curves on Commodity Systems for Online Optimizations

David Tam, Reza Azimi, Livio Soares, Michael Stumm

University of Toronto

{tamda, azimi, livio, stumm}@eecg.toronto.edu

March 10, 2009

ASPLOS
Motivation

- In shared cache (L2/L3):
  - Applications compete for space
  - LRU replacement policy used
  - Interference: applications can evict each other's content
Cache Partitioning

- Eliminates cache space interference
- More flexible than private caches
- Up to 50% improvement in IPC
OS-Based Partitioning

- Apply page-coloring technique to implement set-based partitioning
- Guide physical page allocation to control cache line usage
- Works on existing processors [WIOSCA'07]
OS-Based Partitioning

- Apply page-coloring technique to implement set-based partitioning
- Guide physical page allocation to control cache line usage
- Works on existing processors [WIOSCA'07]
Determining L2 Size

- **Use Miss Rate Curve (MRC)**

  ![Graph showing Miss Rate Curve](image)

  - Shows trade-off spectrum
    - Impact of allocating more/less cache space
  - Also used for online sizing of main memory
    - e.g. [Patterson95], [Zhou04], [Soundararajan08], etc...
Our Approach: RapidMRC

- Online approximation of L2 MRC
- Software-based + hardware performance counters
  - Requires no change to application binary or source
  - Runs on commodity hardware
- Rapid: 230 ms latency
RapidMRC Steps

(1) Track memory accesses

(2) Feed accesses into Mattson's stack algorithm (1970)
   - Emulates LRU aging of cache lines
   - Maintains histogram of stack distances
   - Generates MRC using histogram
Tracking Accesses Online

- **Hardware technique**
  - e.g. [Berg04], [Suh04], [Qureshi06], etc...
  - Target future processors

- **Software technique**
  - Track accesses by instrumenting program code
  - High tracking cost: large volume of data

- **Hybrid technique**
  - Track accesses with hardware performance counters
  - Lower tracking cost: smaller volume of data
Tracking Accesses in IBM POWER5

Hardware Performance Counter Configuration

- Upon every L2 access:
  1. Update *sampling* register with data address
  2. Trigger interrupt to copy register to trace log in main memory

- May miss some L2 cache accesses
  - Caused by multiple in-flight L2 accesses
  - Results show negligible impact
Mattson's Stack Algorithm

- For each L2 access in trace log:
  - Find element, record stack distance, move element to top
  - Update histogram with stack distance

- Stack size
  - One element per L2 cache line

- Optimizations
  - Hashing: eliminates linear traversal
  - Coarse-grained stack distance: reduces update operations
Experimental Setup

- 1.5 GHz dual-core POWER5
  - 1.875 MB shared L2 cache
    - 128-byte line size
    - 10-way set-associative
    - 16 partitions (colors) possible

- Linux 2.6.x
  - Added RapidMRC mechanism
  - Added cache partitioning mechanism

- Trace log length
  - 10 x number of cache lines

- 30 applications
  - SPECjbb2000, SPECcpu2000, SPECcpu2006
Obtaining Real L2 MRCs

- **Offline method**: run application 16 times
  - Once for each cache partition size
  - Measure L2 cache miss rate
Obtaining Real L2 MRCs

- **Offline method:** run application 16 times
  - Once for each cache partition size
  - Measure L2 cache miss rate
Obtaining Real L2 MRCs

- **Offline method:** run application 16 times
  - Once for each cache partition size
  - Measure L2 cache miss rate

![Graph showing L2 Miss Rate vs Instructions Completed](chart)

**Real L2 MRC**
Obtaining Real L2 MRCs

- **Offline method**: run application 16 times
  - Once for each cache partition size
  - Measure L2 cache miss rate
Obtaining Real L2 MRCs

- **Offline method:** run application 16 times
  - Once for each cache partition size
  - Measure L2 cache miss rate
Obtaining Real L2 MRCs

- **Offline method**: run application 16 times
  - Once for each cache partition size
  - Measure L2 cache miss rate
Obtaining Real L2 MRCs

- **Offline method**: run application 16 times
  - Once for each cache partition size
  - Measure L2 cache miss rate

![Graph showing L2 Miss Rate vs. Instructions Completed](image)

- **Real L2 MRC**
Obtaining Real L2 MRCs

- **Offline method:** run application 16 times
  - Once for each cache partition size
  - Measure L2 cache miss rate
Obtaining Real L2 MRCs

- **Offline method:** run application 16 times
  - Once for each cache partition size
  - Measure L2 cache miss rate

![Graph showing L2 Miss Rate vs Instructions Completed and L2 Cache Size](image)
Obtaining Real L2 MRCs

- **Offline method**: run application 16 times
  - Once for each cache partition size
  - Measure L2 cache miss rate

![Graph showing L2 Miss Rate (MPKI) vs Instructions Completed for different cache partition sizes.](image)
Obtaining Real L2 MRCs

- Offline method: run application 16 times
  - Once for each cache partition size
  - Measure L2 cache miss rate

![Graph showing L2 cache miss rate over instructions completed for different cache partition sizes.](image)
Obtaining Real L2 MRCs

- **Offline method:** run application 16 times
  - Once for each cache partition size
  - Measure L2 cache miss rate
Obtaining Real L2 MRCs

- **Offline method:** run application 16 times
  - Once for each cache partition size
  - Measure L2 cache miss rate
Obtaining Real L2 MRCs

- **Offline method:** run application 16 times
  - Once for each cache partition size
  - Measure L2 cache miss rate
Obtaining Real L2 MRCs

- **Offline method**: run application 16 times
  - Once for each cache partition size
  - Measure L2 cache miss rate
Obtaining Real L2 MRCs

- **Offline method:** run application 16 times
  - Once for each cache partition size
  - Measure L2 cache miss rate
Obtaining Real L2 MRCs

- **Offline method:** run application 16 times
  - Once for each cache partition size
  - Measure L2 cache miss rate
RapidMRC vs Real MRC

- **Accuracy:** execution slice at 10 billion instrs

![Graphs showing comparison between RapidMRC and Real MRC for various workloads.](image-url)
Latency of RapidMRC

- **Invoke RapidMRC**: 147 ms
- **Trace**: 83 ms
- **Mattson's Alg**: 230 ms
- **4X app slowdown**: 230 ms
- **App is paused**: 230 ms

RapidMRC

Time

---

RapidMRC
Latency of RapidMRC

- **Invoke RapidMRC**: 147 ms
- **Trace**: 83 ms
- **Mattson’s Alg**: 230 ms
- **App is paused**: Obtained RapidMRC
- **4X app slowdown**: Time
- **Phase Change**: Invoke RapidMRC

RapidMRC
Latency of RapidMRC

- **Invoke RapidMRC**: 147 ms
- **Trace**: 83 ms
- **Mattson’s Alg**: 230 ms
- **Obtained RapidMRC**: 4x app slowdown
- **Amortize costs**
- **App is paused**
- **Phase Change**

**Time**
Latency of RapidMRC

- Invoke RapidMRC
- 4X app slowdown
- 147 ms Trace
- 83 ms Mattson’s Alg
- 230 ms RapidMRC
- App is paused

Phase length: 5 mins median

Amortize costs

Phase Change

Invoke RapidMRC
Latency of RapidMRC

Phase change detection
- Abrupt change in IPC, miss rate
- Detectable online with low cost using hardware performance counters

Phase length: 5 mins median

Invoke RapidMRC
147 ms Trace
83 ms Mattson’s Alg
230 ms RapidMRC

Amortize costs

Invoke RapidMRC

Instructions Completed (Billions)

L2 Miss Rate (MPKI)
Latency of RapidMRC

Phase length: 5 mins median

Invoke RapidMRC

Obtained RapidMRC

Invoke RapidMRC

Phase Change

Time

147 ms
Trace

83 ms
Mattson’s Alg

App is paused

4X app slowdown

230 ms RapidMRC

Amortize costs

Phase change detection
- Abrupt change in IPC, miss rate
- Detectable online with low cost using hardware performance counters
RapidMRC for Sizing Partitions

- For *equake + twolf*
  - Has one long stable phase

- Feed MRCs into utility function
  - e.g. Minimize total miss rate

---

**equake**

- Miss Rate (MPKI)

  ![equake Miss Rate Graph]

**twolf**

- Miss Rate (MPKI)

  ![twolf Miss Rate Graph]
RapidMRC for Sizing Partitions

- For *equake + twolf*
  - Has one long stable phase
- Feed MRCs into utility function
  - e.g. Minimize total miss rate

![Graphs showing Miss Rate vs Cache Size for equake and twolf](image)

- **equake**
  - RapidMRC vs Real
  - Miss Rate (MPKI) vs Cache Size (# colors)
- **twolf**
  - RapidMRC vs Real
  - Miss Rate (MPKI) vs Cache Size (# colors)
RapidMRC for Sizing Partitions

- For *equake* + *twolf*
  - Has one long stable phase
- Feed MRCs into utility function
  - e.g. Minimize total miss rate

---

**equake**

<table>
<thead>
<tr>
<th>Cache Size (# colors)</th>
<th>Miss Rate (MPKI)</th>
</tr>
</thead>
<tbody>
<tr>
<td>RapidMRC</td>
<td>Real</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>2</td>
<td>1</td>
</tr>
<tr>
<td>3</td>
<td>1</td>
</tr>
<tr>
<td>4</td>
<td>1</td>
</tr>
<tr>
<td>5</td>
<td>1</td>
</tr>
</tbody>
</table>

**twolf**

<table>
<thead>
<tr>
<th>Cache Size (# colors)</th>
<th>Miss Rate (MPKI)</th>
</tr>
</thead>
<tbody>
<tr>
<td>RapidMRC</td>
<td>Real</td>
</tr>
<tr>
<td>1</td>
<td>25</td>
</tr>
<tr>
<td>2</td>
<td>20</td>
</tr>
<tr>
<td>3</td>
<td>15</td>
</tr>
<tr>
<td>4</td>
<td>10</td>
</tr>
<tr>
<td>5</td>
<td>5</td>
</tr>
</tbody>
</table>

---

RapidMRC
RapidMRC for Sizing Partitions

- For *equake + twolf*
  - Has one long stable phase
- Feed MRCs into utility function
  - e.g. Minimize total miss rate

![Equake and Twolf Graphs](image)
Performance Impact of Sizes

- twolf: 27% IPC improvement
- equake: unaffected

Performance of uncontrolled sharing
Conclusion

• **RapidMRC**
  - A tracing mechanism can be built with hardware performance counters
  - Accurately approximates L2 MRCs online in software
  - 230 ms latency, invocable upon phase change

• **Application of RapidMRC**
  - Enables online sizing of L2 cache partitions
    - Up to 27% performance improvement
Future Work

• Explore online optimizations made possible by:
  • RapidMRC
    • Reducing energy
    • Guiding co-scheduling
  • Tracing mechanism

• Extend model
  • Account for non-uniform miss penalties