Regularities Considered Harmful: Forcing Randomness to Memory Accesses to Reduce Row Buffer Conflicts for Multi-Core, Multi-Bank Systems

Heekwon Park*, Seungjae Baek*, Jongmoo Choi, Donghee Lee, Sam H. Noh
1. Memory Organization - Bank

- Bank Size
  - $= 32K (\text{# of Entries}) \times 4\text{Kbytes (size of Row)}$
  - $= 128\text{Mbytes}$

- Row-buffer
  - (size: 4Kbyte)

- Rows (32K Entry)
1. Memory Organization – Rank, Channel
1. Memory Organization – Row-buffer

- Row buffer is a cache area used to exploit spatial locality.
- If row buffer cache missed, “precharge” and “activate” operation is performed sequentially.
- Precharge: row buffer > original row
- Activate: entire row > row buffer
- This whole scenario is called “Row Buffer Conflict”
- Precharge and activate operation’s overhead occurs.

Reduce Row Buffer Conflict!!!
Main Objective: Develop row-buffer conflict aware “memory allocator”
2. Row-buffer Conflict

Previous Approach
- Memory Request Scheduling
- Exploiting Data Access Locality
- Micro-pages => Collocation of chunks from different pages

Our Approach
1. Kernel-level approach
2. Apply randomness in the page frame allocation decision!
3. M-cube Memory Allocator

procedure ANALYZER_CORE(start_of_mem, end_of_mem)
  *CL_1 <- First cache line from start_of_mem
  *CL_2 <- Second cache line from start_of_mem
  CACHE_CONTROL(start_of_mem, end_of_mem, Uncached)
  repeat
    i <- 0
    Begin_time <- STOPWATCH()
    while i < 500 do
      *CL_1 <- *CL_2 + 1
      *CL_2 <- *CL_1 + 1
      i <- i + 1
    end while
    End_time <- STOPWATCH()
    RECORD_TIME(End_time - Begin_time)
  end repeat
  CL_2 <- Next cache line
  until CL_2 <= end_of_mem
End procedure

Pseudo code of analyze memory organization

Analyze result
3. M-cube Memory Allocator
3. M-cube Memory Allocator
3. M-cube Memory Allocator – Memory Container

- Purpose: Allocate pages to core from different bank as much as possible
Original Buddy Allocator
Assumption: Physical memory consists of 8 page frames
Scenario: Five requests, each wants a single page frame
### Randomized Algorithm

**Assumption:** Physical memory consists of 8 page frames  
**Scenario:** Five requests, each wants a single page frame

1. **Individual page frame management**  
2. **Downward search**

---

#### Diagram

<table>
<thead>
<tr>
<th>State 1: Initial State</th>
<th>State 2</th>
<th>State 3</th>
<th>State 4</th>
<th>State 5</th>
<th>State 6</th>
<th>State 7</th>
</tr>
</thead>
<tbody>
<tr>
<td>2(^{1}) -7</td>
<td>2(^{1})</td>
<td>2(^{1})</td>
<td>2(^{1})</td>
<td>2(^{1})</td>
<td>2(^{3})</td>
<td>2(^{3})</td>
</tr>
<tr>
<td>2(^{2}) -3</td>
<td>2(^{2})</td>
<td>2(^{2})</td>
<td>2(^{2})</td>
<td>2(^{2})</td>
<td>2(^{2})</td>
<td>2(^{2})</td>
</tr>
<tr>
<td>2(^{1}) -5</td>
<td>2(^{1})</td>
<td>2(^{1})</td>
<td>2(^{1})</td>
<td>2(^{1})</td>
<td>2(^{1})</td>
<td>2(^{1})</td>
</tr>
<tr>
<td>2(^{6}) -6</td>
<td>2(^{6})</td>
<td>2(^{6})</td>
<td>2(^{6})</td>
<td>2(^{6})</td>
<td>2(^{6})</td>
<td>2(^{6})</td>
</tr>
</tbody>
</table>

- \(2^{i}\) denotes page frame of size \(2^{i}\)

#### Table

<table>
<thead>
<tr>
<th>Freelist</th>
<th>State 1: Initial State</th>
<th>State 2</th>
<th>State 3</th>
<th>State 4</th>
<th>State 5</th>
<th>State 6</th>
<th>State 7</th>
</tr>
</thead>
<tbody>
<tr>
<td>2(^{1}) -7</td>
<td>2(^{1}) -3</td>
<td>2(^{1}) -5</td>
<td>2(^{1}) -6</td>
<td>2(^{3}) -2</td>
<td>2(^{3}) -1</td>
<td>2(^{3}) -5</td>
<td></td>
</tr>
<tr>
<td>2(^{2}) -3</td>
<td>2(^{2}) -3</td>
<td>2(^{2}) -5</td>
<td>2(^{2}) -6</td>
<td>2(^{2}) -2</td>
<td>2(^{2}) -1</td>
<td>2(^{2}) -5</td>
<td></td>
</tr>
<tr>
<td>2(^{6}) -6</td>
<td>2(^{6}) -6</td>
<td>2(^{6}) -6</td>
<td>2(^{6}) -6</td>
<td>2(^{6}) -2</td>
<td>2(^{6}) -1</td>
<td>2(^{6}) -5</td>
<td></td>
</tr>
</tbody>
</table>
4. Performance Evaluation

• Environment
  • 2 Intel Xeon x5570 quad core processors with hyperthreading
  • 32GB DDR3 main memory
  • 8 2.5 inch SAS disks of 450GB
  • Implement M-cube memory allocator on the Linux kernel version 2.6.32

• Benchmarks
  • Memory intensive benchmarks
    • Stream, Sysbench-memory, and Ramspeed
  • CPU or I/O intensive benchmarks
    • Kernel Compile, Dbench, and Unixbench
  • PARSEC benchmark
4. Performance Evaluation

- Memory intensive benchmark
  - Performance Improvement: 6.5% ~ 85.2%
  - Average: 26.6%, 50.4%, and 42.1% (RAMspeed, STREAM, and Sysbench-Memory)
- CPU or I/O intensive benchmark
  - Not much improved, since portion of memory reference is low
4. Performance Evaluation

- Canneal, facesim, fluidanimate, and streamcluster: improvement up to 21.7%, average of 10.4%
5. Conclusion

• Paper proposed M-cube, kernel-level memory allocator for multi-core, multi-bank systems.

• To dedicate multiple banks to a core as much as possible for maximize memory parallelism.
  • Memory container

• To reduce cases where multiple cores access the same bank at the same time.
  • Randomizing memory allocation algorithm

• For further research,
  • They will analyze the randomized algorithm more formally.
  • Investigate issues such as fragmentation and lock contention.
Any Questions?