CHOP: Integrating DRAM Caches for CMP Server Platforms

Uliana Navrotska
766785
What I will talk about?

**Problem:** in the era of many-core architecture at the heart of the trouble is the so-called memory wall.
- the growing disparity between how fast a CPU can operate on data and how fast it can get the data it needs
- the number of cores per processor is increasing, the number of connections from the chip to the rest of the computer is not.

**Basic idea:** CHOP - Caching Hot Pages
- hot means frequently used

**Result:** allocating only hot pages we avoid the memory bandwidth problem because we don’t waste bandwidth on pages with low spatial locality
To be sure that idea with hot pages will be relevant engineers analyzed four workloads to obtain their DRAM cache access information and calculate how many hot pages each has.

Result of the statistics you can see below:

<table>
<thead>
<tr>
<th>Benchmark</th>
<th>Hot-page percent</th>
<th>Min. number of accesses</th>
</tr>
</thead>
<tbody>
<tr>
<td>Standart Application (SAP)</td>
<td>24.8</td>
<td>95</td>
</tr>
<tr>
<td>SPEC Java Application Server 2004 (Sjas)</td>
<td>38.4</td>
<td>65</td>
</tr>
<tr>
<td>SPEC Java Server 2005 (Sjbb)</td>
<td>30.6</td>
<td>93</td>
</tr>
<tr>
<td>TPC-C</td>
<td>7.2</td>
<td>64</td>
</tr>
<tr>
<td>Average</td>
<td>25.2</td>
<td>79</td>
</tr>
</tbody>
</table>

**Conclusion:**
- average number of hot-page is 25%
- average number of access is 79 times
If we capture such hot pages and allocate them into the DRAM:
• we can reduce memory traffic by 75%
• get 80% of DRAM cache hits

The question is:

**How we can identify these hot pages?**

The answer is:

**We will use filter-based DRAM caching.**

In next minutes I will introduce you:
- Three different filter cache schemas:
  - Filter cache (CHOP-FC)
  - Memory-based filter cache (CHOP-MFC)
  - Adaptive filter cache (CHOP-AFC and CHOP-AMFC)
- Make evaluation of these three schemas
- Compare performance of these schemas
Filter cache (CHOP-FC)

Basic ideas:
• filter cache is placed on die (built onto CPU chip)
• both filter cache and DRAM cache use page-size lines
• each filter cache entry consists of:
  ▪ tag (to identify the line)
  ▪ Least recently used (LRU) bits
  ▪ counter for how often line is touched

How filter works?

➢ first time filter allocates the line into its tag array when LLC (Last-Level Cache) miss occurs
➢ counter is set to zero and increases incrementally for subsequent misses on that line
➢ when counter reaches threshold -> filter considers line to be hot and put it into DRAM
But, to put new line into DRAM cache we should find a “victim” first, the line that we will remove....

**How it happens?**

- DRAM cache also has counter similar to that in the filter
- victim is chosen according to the values of this counter (victim line is a line with the smallest value of the counter)
- when victim is chosen DRAM cache returns the tag of victim line to the filter cache as cold
- filter cache set counter of this page to half of it’s current value
- victim has better chance to become hotter than other new lines

**Problem of such filter cache:**

✓ The counter history is completely lost when line is deleted from filter cache.

To resolve this problem next type of filter was proposed....
Memory-based filter cache (CHOP-MFC)

Basic ideas:
- to save a counter into memory when a line is deleted from filter cache
- restore the value of counter when the line is fetched back again

As a result:
- we have more accurate hot-page identification
- also we can safely reduce the filter cache size

Where to save counters?
- in the main memory - > requires off-chip memory access to look up the counters
- in the DRAM cache
- in the page table entry of each page
Adaptive filter cache schemes

- **CHOP-AFC** – adaptive filters based on CHOP-FC
- **CHOP-AMFC** – adaptive filters based on CHOP-MFC

**Basic Idea:** processor turns the filter cache on and off dynamically on the basis of the memory utilization status

**Implementation:**
- add monitors to track memory traffic
- when memory utilization exceeds a certain threshold, the processor turns on the filter cache so that only hot pages are fetched into the DRAM cache

**Expectation:**
- more performance benefits by using the entire available memory bandwidth
CHOP-FC evaluation for 4-Kbyte cache line

We compare:

- regular 128-Mbyte DRAM cache
- RAND: caching a random subset of LLC misses
- CHOP-FC with 32 threshold for “hot-paging”

Conclusion:
- CHOP-FC schema outperforms DRAM and RAND
- average speedup of 17.7%
- although hot pages were only 25.24%, they contributed to 80% of the entire LLC misses
- as a result caching only hot pages significantly reduces memory bandwidth utilization
CHOP-MFC evaluation for 4-Kbyte cache line

We compared:

- regular 128-Mbyte DRAM cache
- CHOP-MFC with various counter thresholds (ex. MFC-32 for threshold 32)

Conclusion:
- CHOP-MFC outperforms DRAM in most cases
- MFC-128 performed best with average speedup of 18.9%
- CHOP-MFC achieves higher hot-page identification accuracy
- The performance improvements are due to the significant reduction in memory bandwidth utilization
CHOP-AFC evaluation for 4-Kbyte cache line

Basic ideas:

- memory utilization threshold varies from 0 to 1
- we show only CHOP-AFC because CHOP-AMFC exhibited similar performance
- For thresholds less than 0.3, the speedups achieved remained the same
- For threshold values greater than 0.3, filter cache could be turned off. As a result more pages go to the DRAM and so number of hits increases

More we increase memory utilization threshold queuing delay began to dominate, that’s why we have performance decreasing at the end
Comparison of the three schemas

Conclusion:
- all filter-based techniques improve performance by caching only hot pages
- CHOP-MFC achieved on average speedup of 18.6%
- CHOP-FC achieved 17.2% speedup
- CHOP-AFC and CHOP-AMFC outperform CHOP-FC and CHOP-MFC
- when memory bandwidth is enough adaptive filters enable more blocks to be cached (more hits as a result)

THE END