Memory Hierarchy for Web Search

HPCA, 2018

Grant Ayers  
Stanford University

Jung Ho Ahn  
Seoul National University

Christos Kozyrakis  
Stanford University

Parthasarathy Ranganathan  
Google

Presented by Pascal Störzbach
Executive Summary

• **Motivation:** Online data-intensive services (OLDI) comprise a significant and growing portion of datacenter-scale workloads

• **Problem:** Complexity of OLDI services (such as web search) has precluded detailed architectural evaluations and optimizations of processor design trade-offs

• **Goal:** Provide in-depth study of the microarchitecture and memory system behavior of commercial web search

• **Observations and Ideas:**
  – Memory hierarchy an bottleneck
  – Significant reuse of data not captured by current cache hierarchies
  – Adding an latency-optimized L4 cache using eDRAM

• **Result:** Cache hierarchy optimized for web search without using more transistors on the die
  – 27 % overall performance improvement
  – Die size equal to original die size (18-core with 2.5 MiB/core to 23-core with 1 MiB/core design)
Outline

• Characterizing Search in the Wild
• Characterizing Memory Hierarchy for Search
• Optimized Memory Hierarchy
• Discussion
CHARACTERIZING SEARCH IN THE WILD
Background

3 Components:
- Crawling
- Indexing
- Serving
Search attributes:

1. Index stored on multiple machines (index divided into *shards*)
2. Query processing requires billions of instructions
3. Search has request-level *parallelism*
4. Search is *latency-critical*
Methodology

- Measurements on 2 platforms (PLT1, PLT2)
- 4 different metrics:
  - IPC
  - Misses per Kilo-Instructions (MPKI) (L2 and L3 cache)
  - Branch MPKI

<table>
<thead>
<tr>
<th></th>
<th>PLT1 [16]</th>
<th>PLT2 [53]</th>
</tr>
</thead>
<tbody>
<tr>
<td>Microarchitecture</td>
<td>Intel Haswell</td>
<td>IBM POWER8</td>
</tr>
<tr>
<td>Number of sockets</td>
<td>2</td>
<td>2</td>
</tr>
<tr>
<td>Cores</td>
<td>18 per socket</td>
<td>12 per socket</td>
</tr>
<tr>
<td>SMT</td>
<td>2</td>
<td>8</td>
</tr>
<tr>
<td>Cache block size</td>
<td>64 B</td>
<td>128 B</td>
</tr>
<tr>
<td>L1-I$ (per core)</td>
<td>32 KiB</td>
<td>32 KiB</td>
</tr>
<tr>
<td>L1-D$ (per core)</td>
<td>32 KiB</td>
<td>64 KiB</td>
</tr>
<tr>
<td>Private L2$ (per core)</td>
<td>256 KiB</td>
<td>512 KiB</td>
</tr>
<tr>
<td>Shared L3$ (per socket)</td>
<td>45 MiB</td>
<td>96 MiB</td>
</tr>
</tbody>
</table>

Table II: Key attributes of PLT1 and PLT2 platforms.
Hardware Optimizations

• Web search **benefits** significantly from features like
  – High core counts
  – Simultaneous Multithreading (SMT)
  – Large pages
  – Prefetching
Key Search Characteristics

- IPC is reasonably **high**
- L2 MPKI for instruction accesses is **high**
- L3 MPKI for data is **significant**
- Branch MPKI is uniformly **high**
- Only 32% of slots are used for retirements
  - Hence, memory hierarchy is a big opportunity for improvement
CHARACTERIZING MEMORY HIERARCHY FOR SEARCH
Challenges and Methodology

Challenges
1. No known timing simulator can run search for non-trivial amount of virtual time
2. Performance counters are limited and often broken

Methodology
• Validated measurements from real machines
• Trace-driven functional cache simulation (modelling a PLT1-like system)
• Analytical model based on curve-fitting data from the fleet
Footprint and Working Set Scaling

**Footprint observations:**
- Heap dominates non-shard memory footprint
- Heap size grows slower than linear as there are several shared data structures

**Working set observations:**
- Shard footprint is constant (100’s GiB) but it’s working set grows
- Heap working set significantly smaller than footprint
Cache Effectiveness

- L1 and L2 caches experience significant misses of all types.
- L3 cache virtually eliminates code misses but is insufficient for heap and shard.

(a) Cache misses across the memory hierarchy classified by type.
L3 cache scaling

- 16 MiB L3 cache is sufficient to remove code misses
- L3 cache is ineffective with shard accesses
- Large (1 GiB) shared caches are effective for heap accesses
Type of Misses

- Conflict misses not significant
- Default associativity: a good design point
- Limited benefit of larger cache lines
OPTIMIZED MEMORY HIERARCHY FOR WEB SEARCH
Key Insights

• Good thread-level parallelism
• Memory hierarchy is a significant bottleneck
• Some cache hierarchy decisions effective others ineffective
Optimization Strategy

• Repurpose expensive on-chip transistors in the L3 cache for cores
• Exploit the available locality in the heap with cheaper and higher-capacity DRAM incorporated into a latency-optimized L4 cache
Cache vs. Cores Trade-off

Measurements for Intel Haswell architecture

- Core area cost is 4 MiB L3 cache
Cache vs. Cores Trade-off

- Some L3 transistors could be better used for cores
  - (9c|2.5MiB/core worse than 11c|1.23MiB/core)
- Core count is not all that matters
  - (All 18c with < 1MiB/core are bad)

Figure 9: Search performance (QPS) vs. L3-equivalent area for various core count and cache size combinations.
Cache vs. Cores Trade-off

What’s the right cache per core balance?

Use linear model incorporated from data of previous measurements
- Performance linear to core count
- 2 measurements per each cache ratio

Result: 1 MiB/core allows 5 extra cores and 14% performance improvement

Figure 10: Search performance when trading cache capacity for cores.
Latency-optimized L4 Cache

- Target the locality in the fixed 1 GiB heap
- Use of eDRAM (Embedded DRAM) instead of on-chip SRAM
  - eDRAM cheaper with competitive latencies
  - More energy efficient
  - Often considered as L4 cache
  - Requires refreshes
- Less than 1% die area overhead (L4 controller)
- Latency optimized
  - Memory accessed in parallel
  - Direct-mapped organization
L4 Cache Evaluation

- Baseline is 23-core design with 1MiB/core L3 cache (iso-area to 18-core)
- 1 GiB cache size achieves most of the benefits for the heap

(a) L4 hit rate vs. size
L4 Cache Evaluation

- 27% overall performance improvement
- 22% pessimistic
- 38% future (+10% latency & misses)
Executive Summary

• **Motivation:** Online data-intensive services (OLDI) comprise a significant and growing portion of datacenter-scale workloads

• **Problem:** Complexity of OLDI services (such as web search) has precluded detailed architectural evaluations and optimizations of processor design trade-offs

• **Goal:** Provide in-depth study of the microarchitecture and memory system behavior of commercial web search

• **Observations and Ideas:**
  – Memory hierarchy an bottleneck
  – Significant reuse of data not captured by current cache hierarchies
  – Adding an latency-optimized L4 cache using eDRAM

• **Result:** Cache hierarchy optimized for web search without using more transistors on the die
  – 27% overall performance improvement
  – Die size equal to original die size (18-core with 2.5 MiB/core to 23-core with 1 MiB/core design)
DISCUSSION
Strengths

• Most of important aspects are evaluated
• Uses production application for performance analysis
• Tries to predict future improvements
• Well written
Weaknesses

• Considers only one architecture (though they analyse PowerPC)
• Only applicable to Google Search
Follow Up Work

• Code Layout Optimization for Near-Ideal Instruction Cache
Open Discussion

• Should this kind of analysis be done also for other kind of software?
• Can there be other benefits of a L4 cache?
• Should software be able to control cache behavior (i.e. evicting strategy)?
• Online Discussion on Moodle
Further Readings

- **HPCA**: Technology Comparison for Large Last-level Caches (L3cS): Low-leakage SRAM, Low Write-energy STT-RAM, and Refresh-optimized eDRAM