Memory Hierarchy for Web Search

Grant Ayers[*▯], Jung Ho Ahn[†▯], Christos Kozyrakis[*], Partha Ranganathan[‡]

Stanford University[*]
Seoul National University[†]
Google[‡]

▯*Work performed while authors were at Google*

The world is headed toward cloud-based services

...and we’re still optimizing for SPEC

Research Objective

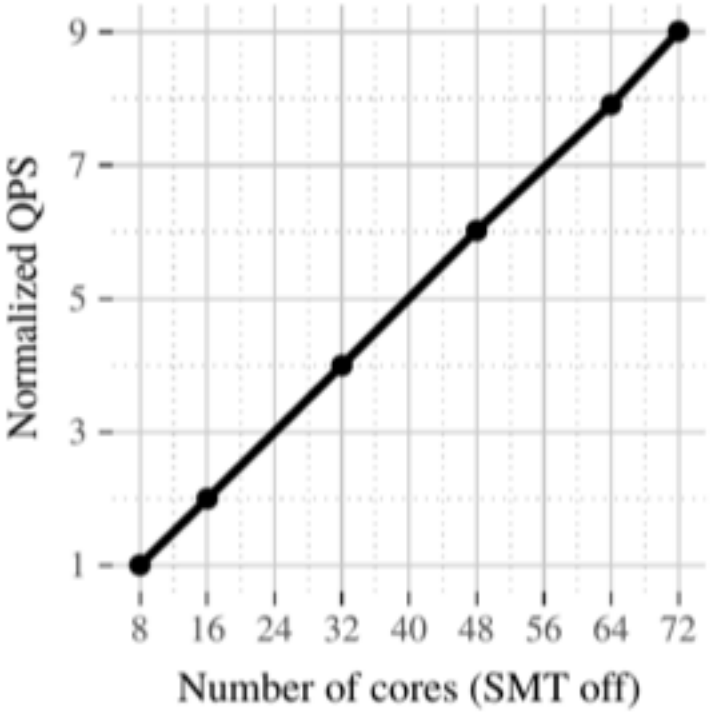
Design tomorrow's CPU architectures for OLDI workloads like web search

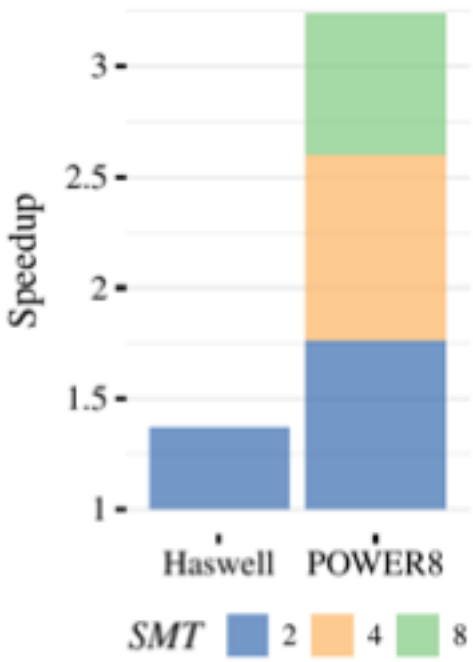
1. Provide the first public in-depth study of the microarchitecture and memory system behavior of commercial web search
2. Propose new performance optimizations with a focus on the memory hierarchy

Results show 27% performance improvement, and 38% with future devices.

Understanding Web Search on Current Architectures

Google’s web search is scalable

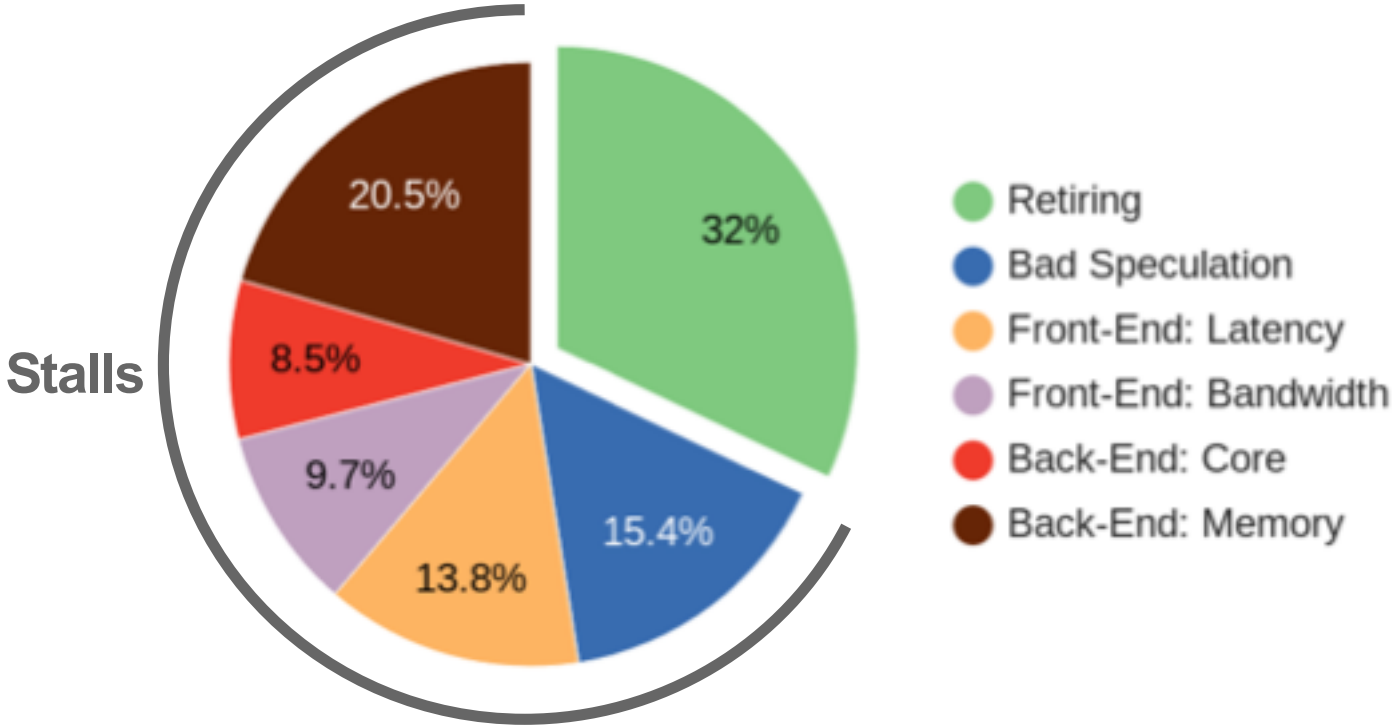




Scalability / Hardware Optimizations

- Linear core scaling
- Not bandwidth or I/O bound
- SMT (+37%), huge pages (+11%), hardware prefetching (+5%)
- Architects can assume excellent software scaling

Google web search performance on Intel Haswell



Web search leaf node CPU utilization

Memory Hierarchy Characterization

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

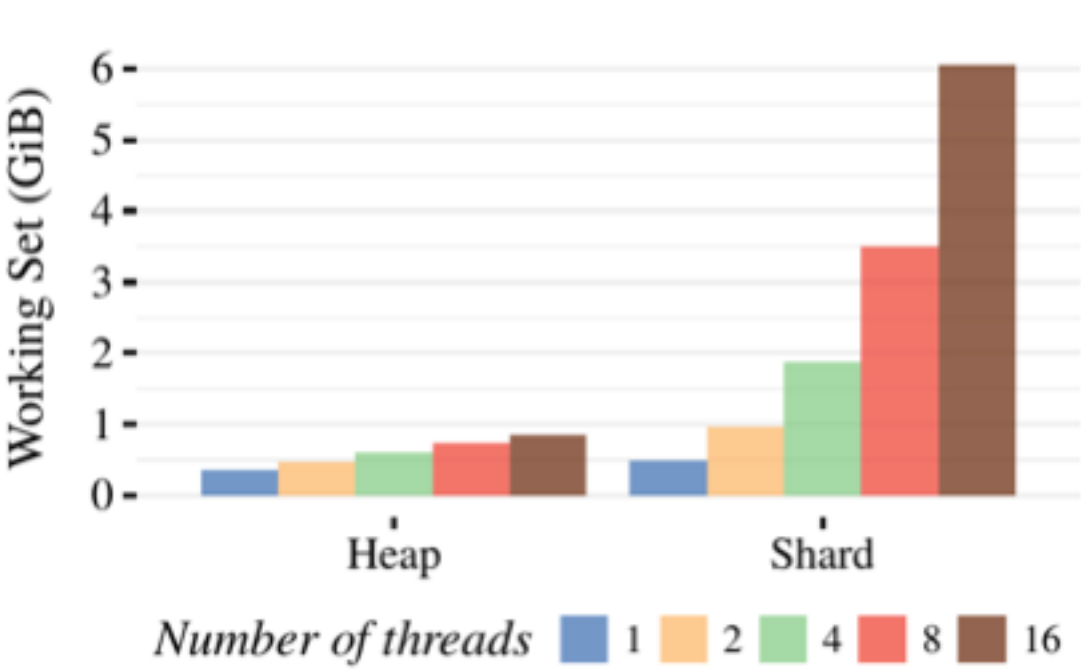
Methodology

- Measurements from real machines
- Trace-driven functional cache simulation (Intel Pin, 135 billion instructions)
- Analytical performance modeling

Working set scaling

Memory accessed in steady-state

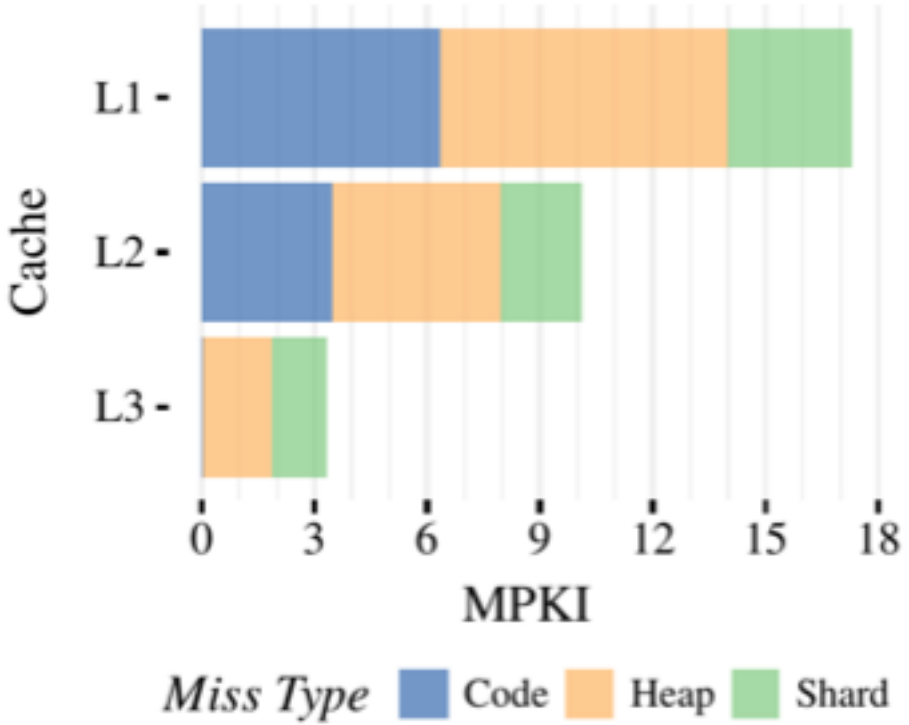
- Shard footprint is constant, but touched footprint grows with cores and time (little data locality in the shard)
- Heap working set converges around 1 GiB, suggests sharing and cold structures



Overall cache effectiveness

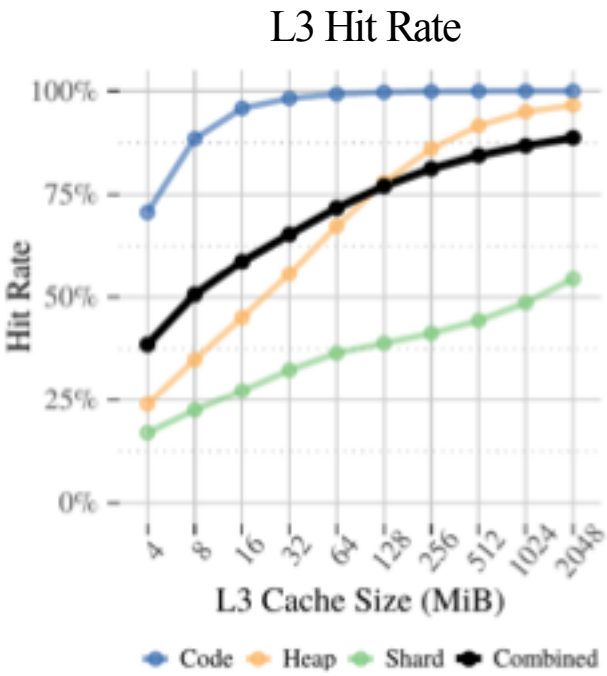
- L1 and L2 caches experience significant misses of all types
- L3 cache virtually eliminates code misses but is insufficient for heap and shard

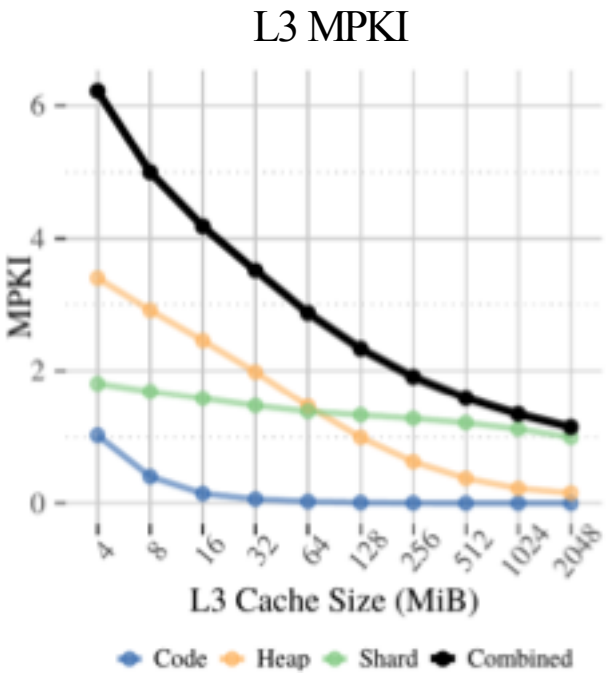
What's the ideal L3 size?



L3 cache scaling

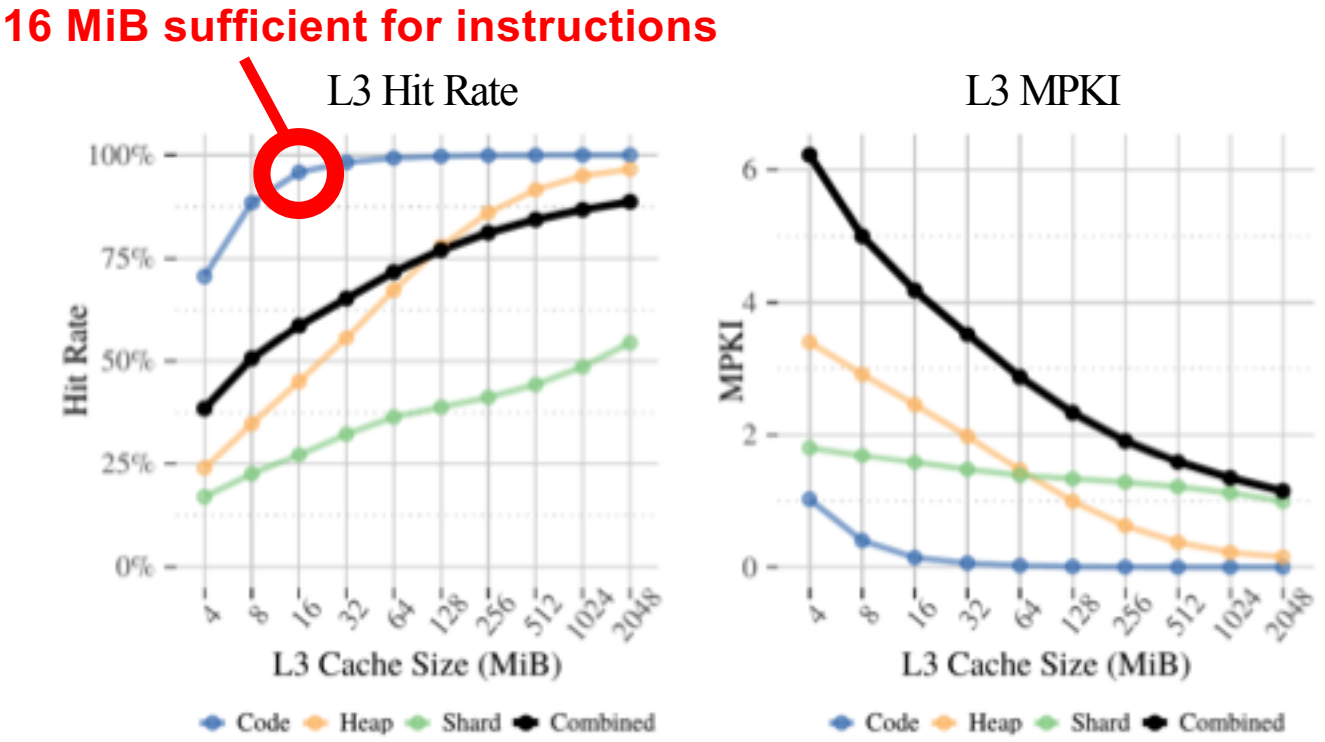
- 16 MiB sufficiently removes code misses
- Not even 2 GiB captures the shard
- 1 GiB would capture the heap





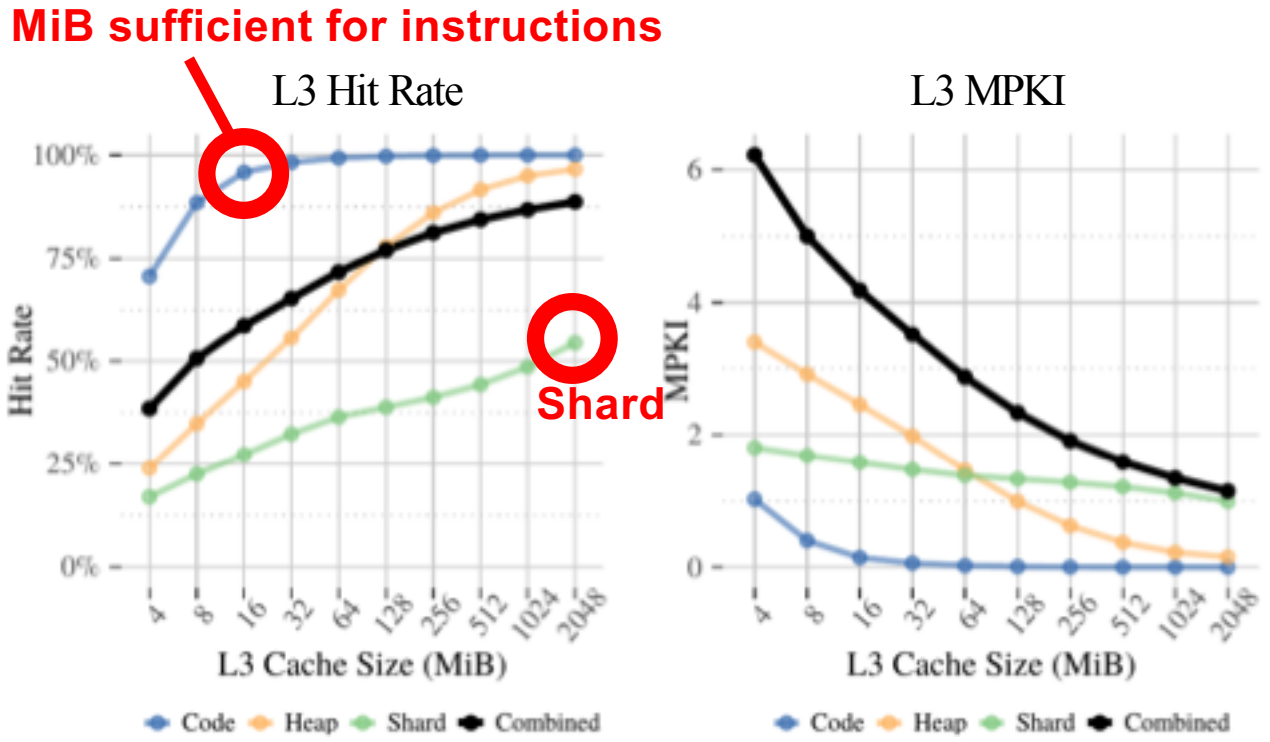
L3 cache scaling

- 16 MiB sufficiently removes code misses
- Not even 2 GiB captures the shard
- 1 GiB would capture the heap



L3 cache scaling

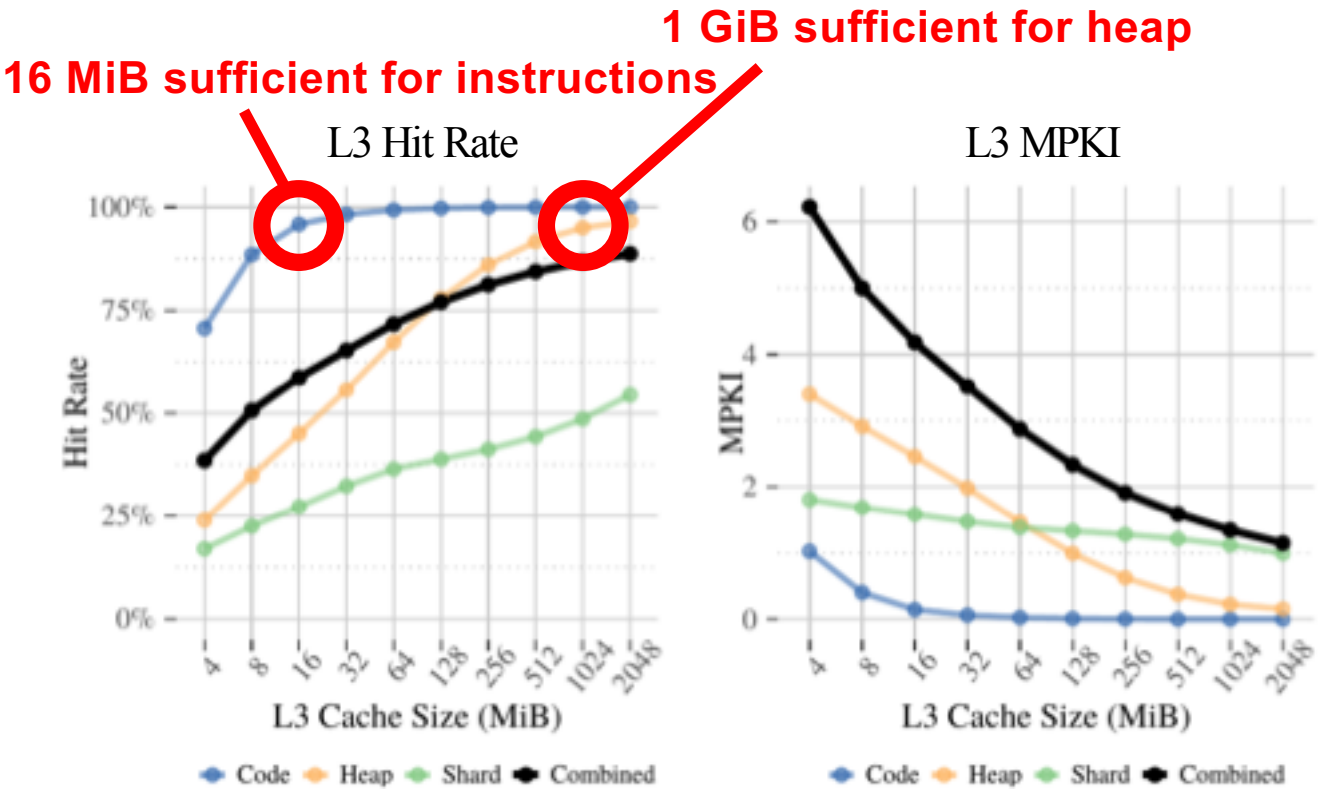
- 16 MiB sufficiently removes code misses
- Not even 2 GiB captures the shard
- 1 GiB would capture the heap



L3 cache scaling

- 16 MiB sufficiently removes code misses
- Not even 2 GiB captures the shard
- 1 GiB would capture the heap

Large shared caches are highly effective for heap accesses.

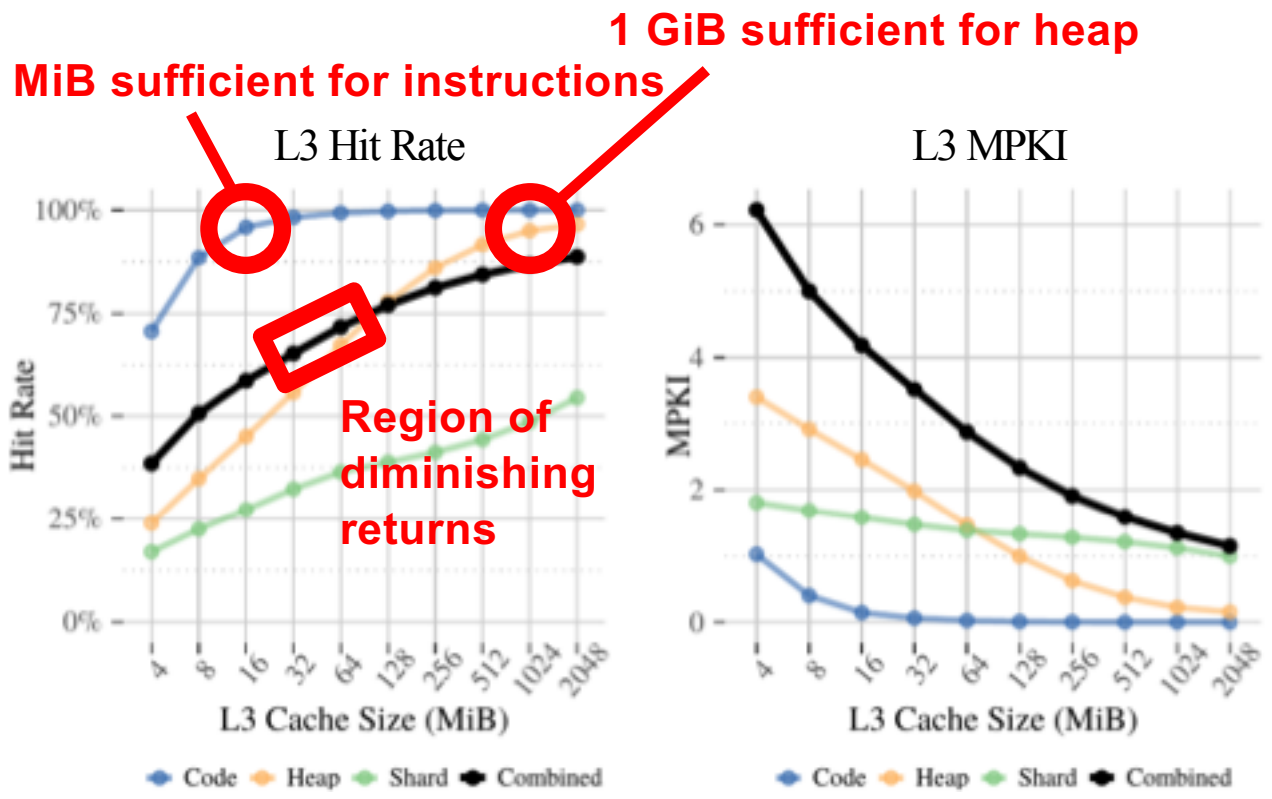


L3 cache scaling

- 16 MiB sufficiently removes code misses
- Not even 2 GiB captures the shard
- 1 GiB would capture the heap

Large shared caches are highly effective for heap accesses.

The L3 cache is in a region of diminishing returns



Memory Hierarchy for Hyper-scale SoCs

Optimization strategy

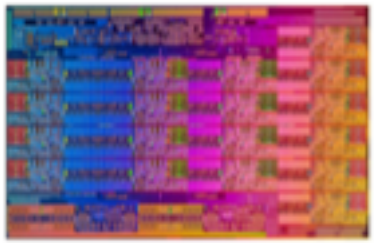
Analysis indicates diminishing returns for L3 caches, but potential for larger caches, thus two contrasting optimizations:

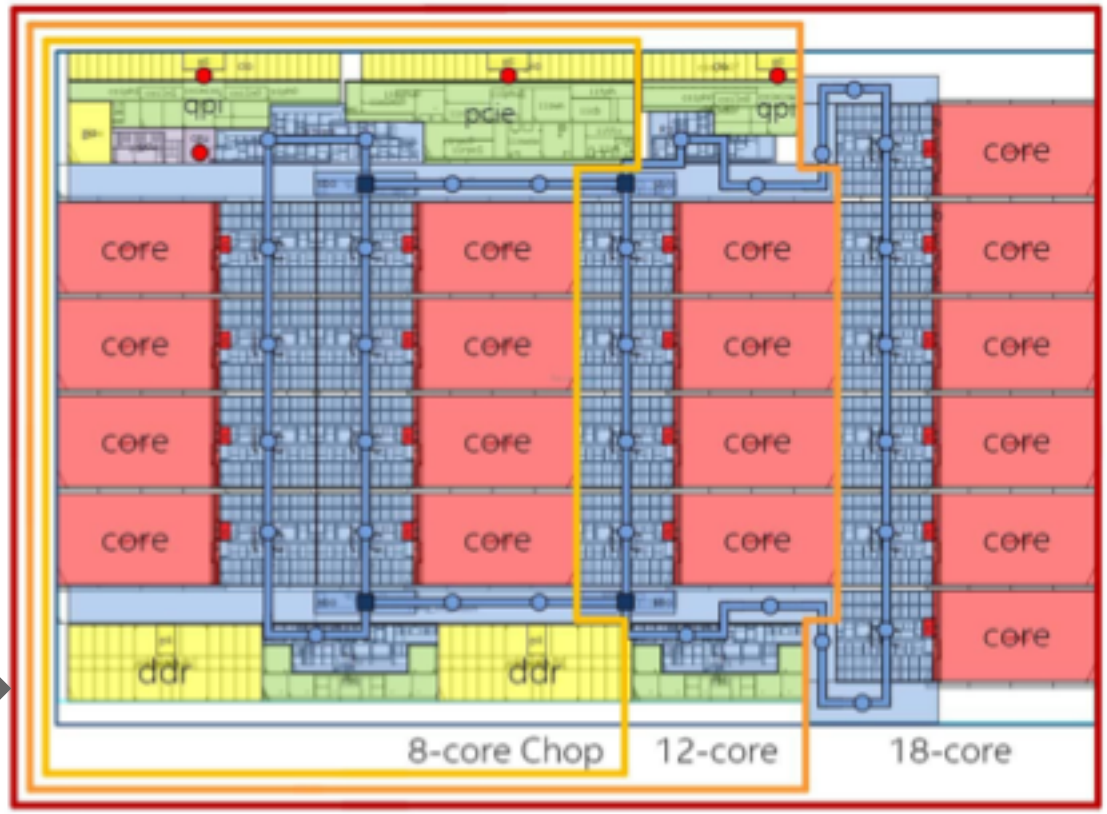
1. Repurpose expensive on-chip transistors in the L3 cache for cores
2. Exploit the locality in the heap with cheaper, higher-capacity DRAM incorporated into a *latency-optimized* L4 cache

Cache vs. Cores Trade-off

Intel Haswell[1]

- 18 cores
- 2.5 MiB L3 per core
- Core area cost is 4 MiB L3





[1] "The Xeon Processor E5-2600 v3: A 22nm 18-core product family" (ISSCC '15)

Trading cache for cores

Sweep core count and L3 capacity in terms of chip area used

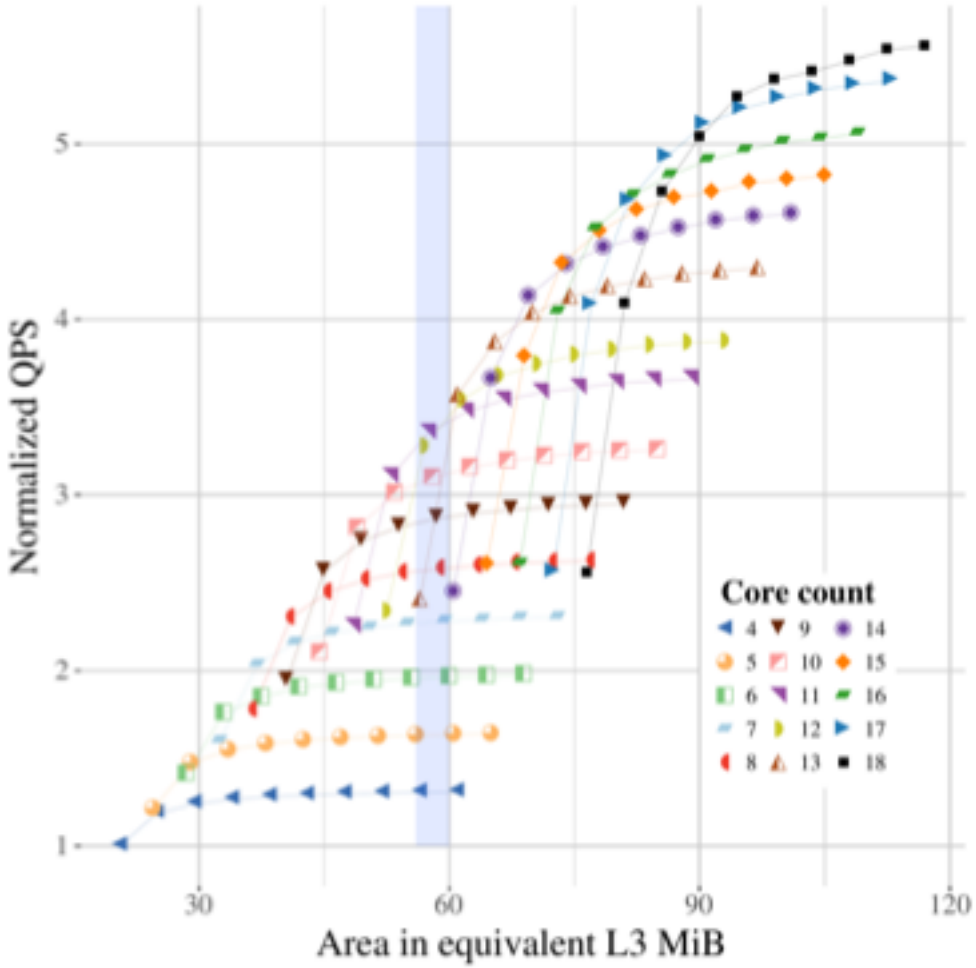
- Each core is 4 MiB of L3
- Use CAT to vary L3 from 4.5 to 45 MiB

Some L3 transistors could be better used for cores

(9c/2.5 MiB/core worse than 11c/1.23 MiB/core)

Core count is not all that matters!

(All 18c with < 1 MiB/core are bad)



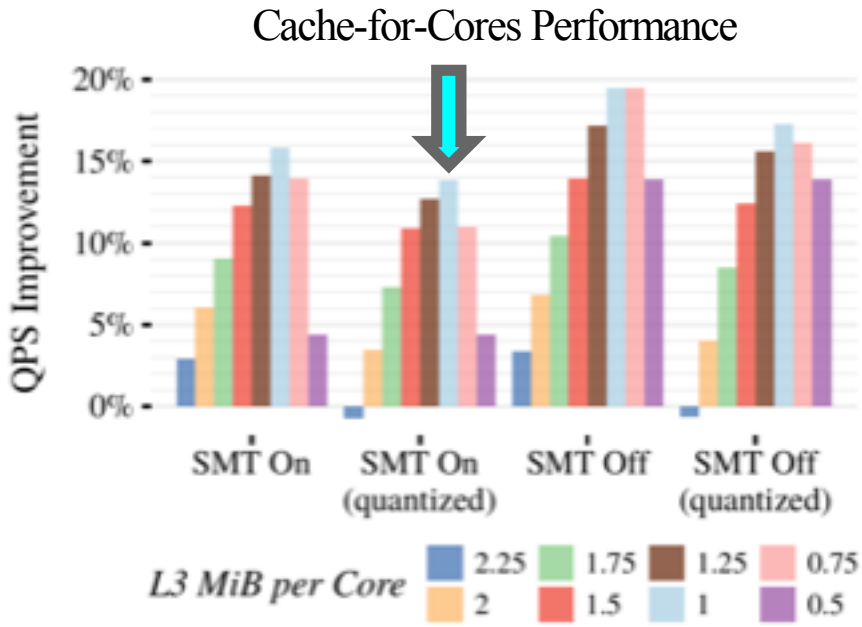
Trading cache for cores

What's the right cache per core balance?

Incorporate the sweep data into a linear model

- Performance is linear with respect to core count
- We have two measurements for each cache ratio

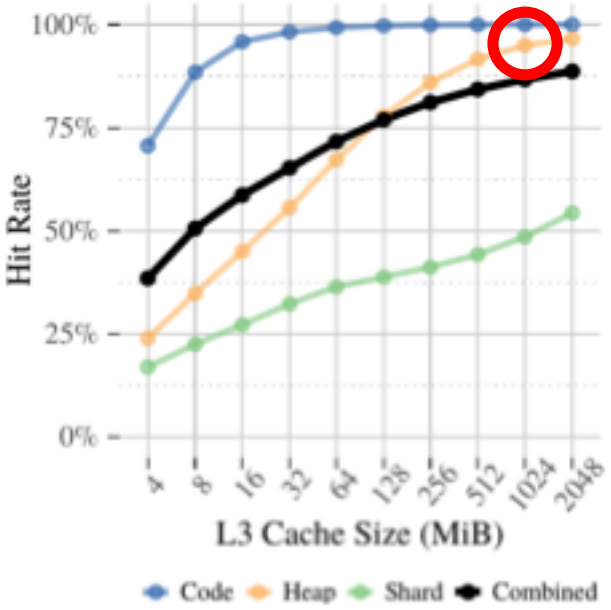
1 MiB/core of L3 cache allows 5 extra cores and 14% performance improvement



Latency-optimized L4 cache

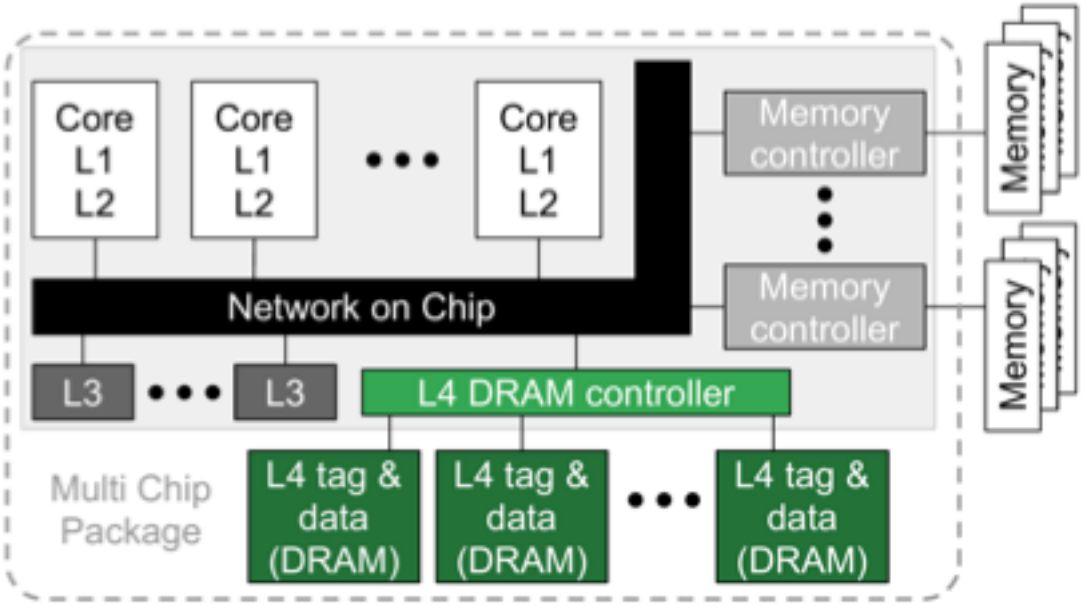
Target the available locality in the fixed 1 GiB heap

- Not feasible with an on-chip SRAM cache
- Need an off-chip, on-package eDRAM cache
  - eDRAM provides lower latency
  - Multi-chip package allows for existing 128 MiB dies
- Less than 1% die area overhead
- Use an existing high-bandwidth interface such as Intel's OPIO



Latency-optimized L4 cache

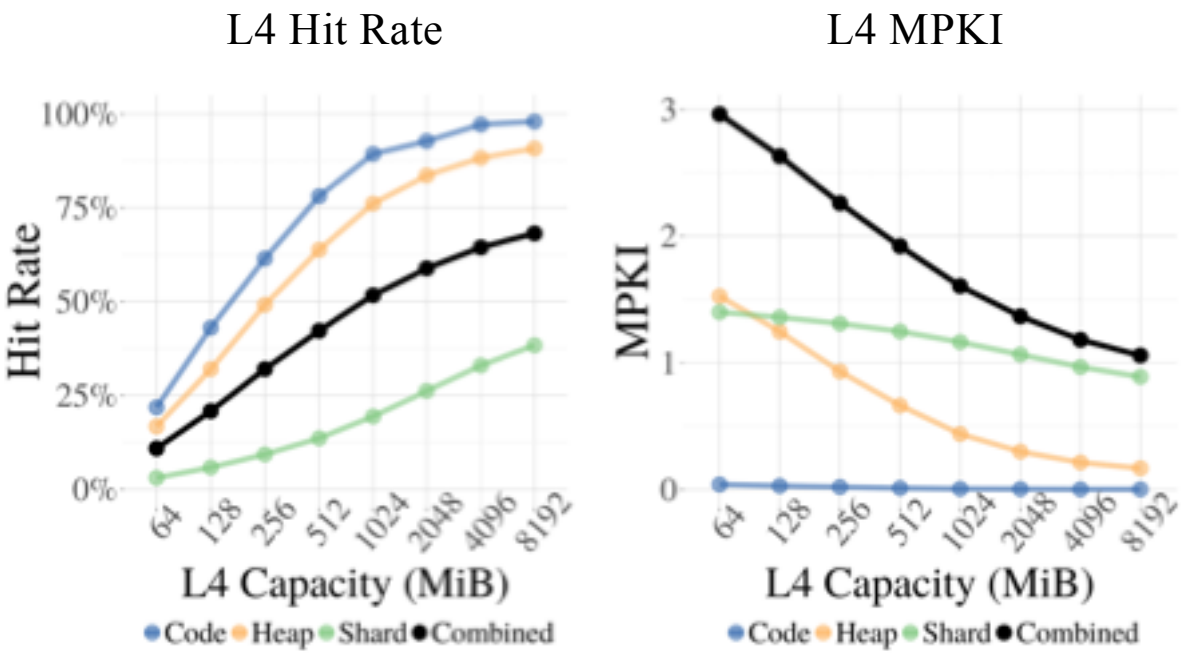
- 1 GiB on-package eDRAM
- 40-60 ns hit latency
- Based on Alloy cache
- Parallel lookups with memory
- Direct-mapped
- No coherence



Proposed L4 Cache based on eDRAM

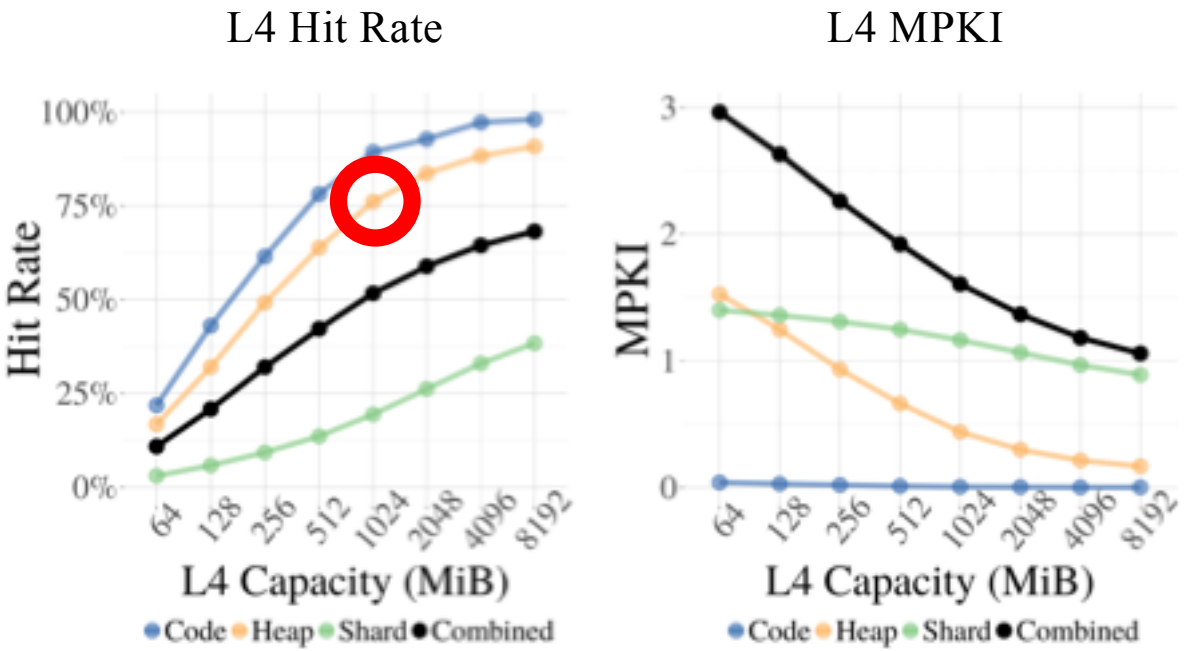
L4 cache miss profile

Baseline is optimized 23-core design with 1 MiB L3 cache per core (iso-area to 18-core)

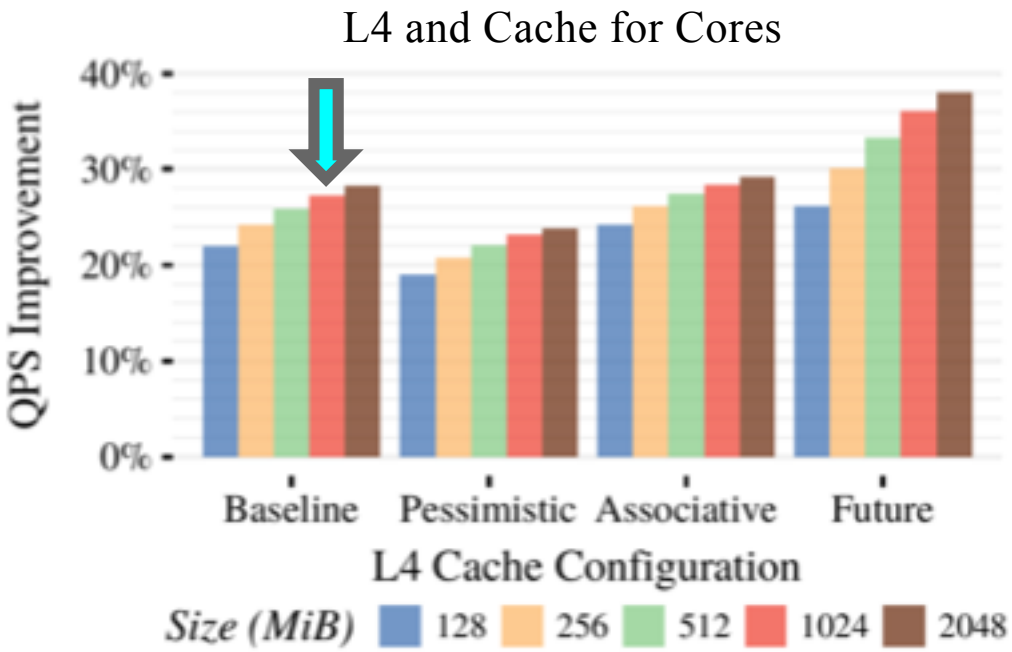


L4 cache miss profile

Baseline is optimized 23-core design with 1 MiB L3 cache per core (iso-area to 18-core)



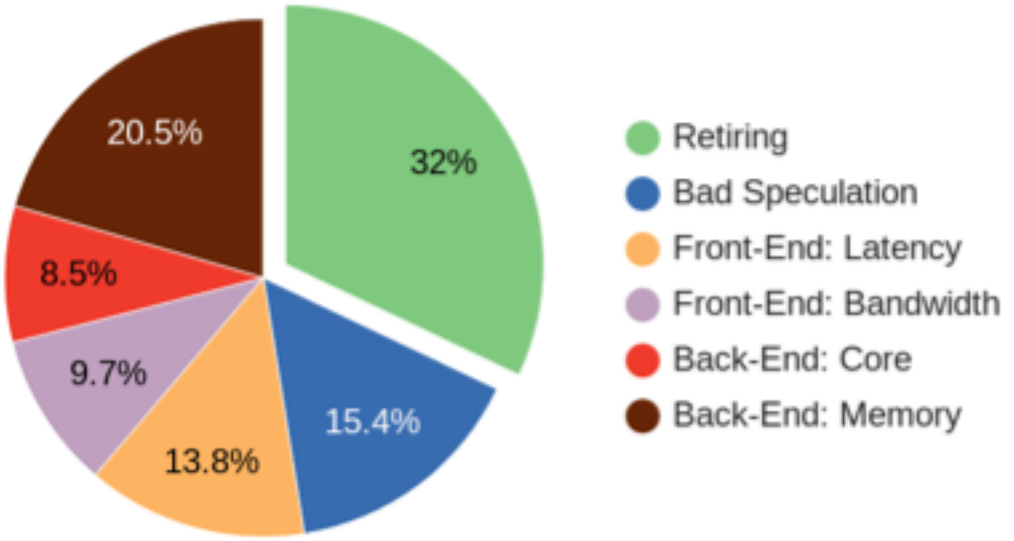
L4 cache + cache-for-cores performance



- **27% overall performance improvement**
- 22% “pessimistic” (60ns hit, 5ns additional miss penalty)
- 38% “future” (+10% latency & misses)

Ongoing work

1. Shard memory misses
2. Instruction misses
3. Branch stalls and BTB misses
4. New system balance ratios



Web search leaf node CPU utilization

Conclusions

1. OLDI is an important class of applications about which little data is available
2. Web search is a canary application for OLDI that is inefficient in hardware
3. Through a careful rebalancing of the memory hierarchy, we're able to improve Google's web search by 27% today, and 38% in the future
4. There is high potential for new SoCs specifically designed for OLDI workloads