Lecture: Profiling a warehouse-scale computer

Adam Belay <abelay@mit.edu>

Logistics

- Lab 1 is now available on the course website. Feel free to ask us questions on Piazza. Discussion and collaboration is encouraged but write and submit your own answers
- If you're presenting a paper next week, meet with us during office hours (Friday 1-3PM) with a rough draft of your slides

Warehouse–scale computing

- Thousands of nodes involved in each service
- Emphasis on both performance and cost efficiency
- Theory: Is this infrastructure an entirely new class of computer that we can design for?

Credit: This and other pictures of a Google datacenter are available here.

Key findings

- Studies live production workloads on a warehouse-scale computer over three years with billions of users
- Optimizing for SPEC is not the same as optimizing for warehouse-scale computers
- No “silver bullet” application, significant diversity
- Hotspot functions comprise a “datacenter tax”
- High instruction and data cache miss rates stall processors (5-10% of time no instructions)

Workload characteristics

- Distributed, multi-tier services, each service exposes a narrow API
- Communication exclusively through RPCs
  - Data serialized using protocol buffers.
  - Latency (especially the tail) is the defining metric
- Fast release cycles: Weekly or even daily
- Single large repository for all code to encourage sharing
  - Binaries deployed through static linking, no dependencies
  - Results in large binaries (100's of MB), increasing instruction footprint

Google-Wide Profiling

- Randomly selects a fraction of servers to profile each day
- Uses `perf` to gather profiling data and symbols for a brief period on each server
- Aggregates samples into a database called Dremel for analysis
- Focus is on C++, but Java, Python, and Go used too

Performance counters

- Nearly all CPUs have architecture-specific event counters of various metrics (e.g., cache misses, branch misses, page walk times, etc.)
- These counters are provided by a performance management unit (PMU). Typically one per core.
- A counter is *architectural* if it works across different microarchitectures. A counter is *precise* if it is cycle accurate. Not all counters have these properties.
- Can't time-multiplex counters within a measurement
- How to read one on x86:

```
wrmsrl(MSR_P6_EVNTSEL0, sel);
rdmsrl(MSR_P6_PERFCTR0, val);
```

Metrics

- Common metric: Misses per kilo-instruction (MPKI)
  - Can be misleading about performance
  - Why? CPUs run instructions out-of-order to hide misses
- Instead: Use cycles, how many are stalled waiting for misses? (not performing work)
  - Captures wasted CPU resources

Workloads

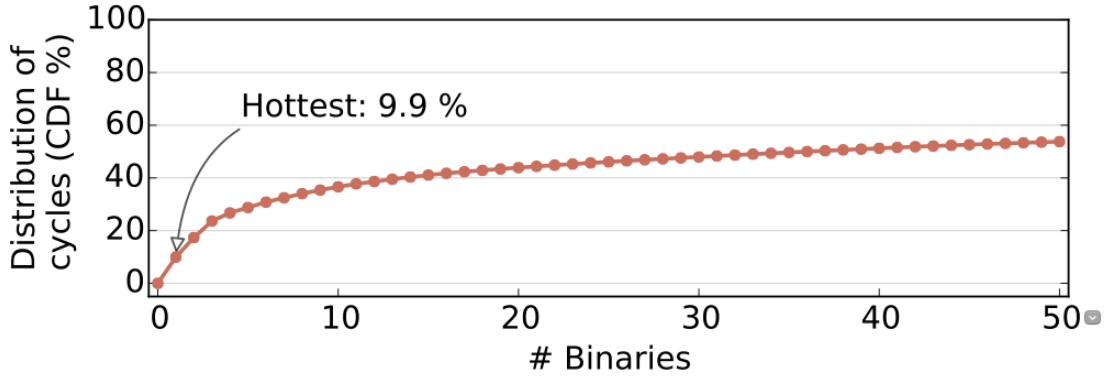


Figure 1: There is no "killer application" to optimize for. The top 50 hottest binaries only cover ≈60% of WSC cycles.

No killer bullet

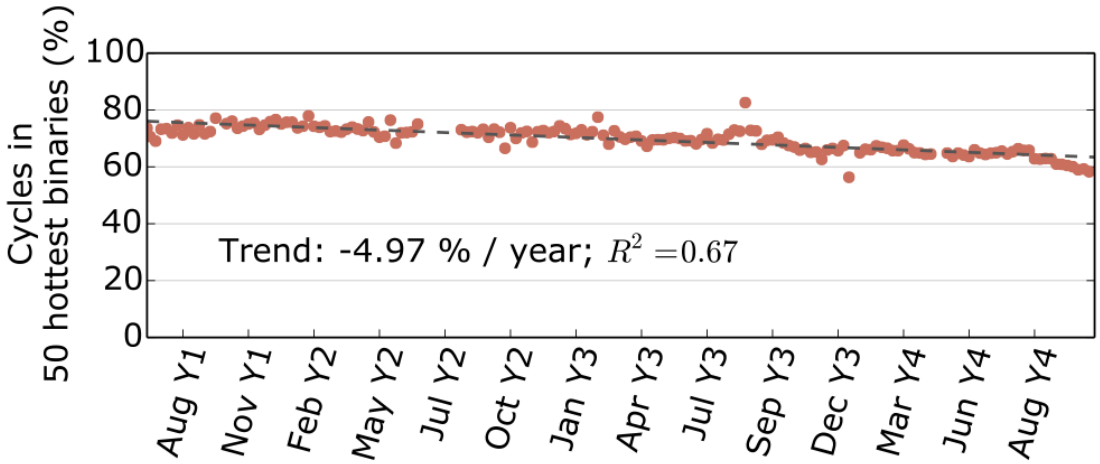


Figure 2: Workloads are getting more diverse. Fraction of cycles spent in top 50 hottest binaries is decreasing.

Diversity increasing over time

Workloads themselves are diverse

- 353 functions account for 80% of cycles
- Indicates hotspots have already been optimized away

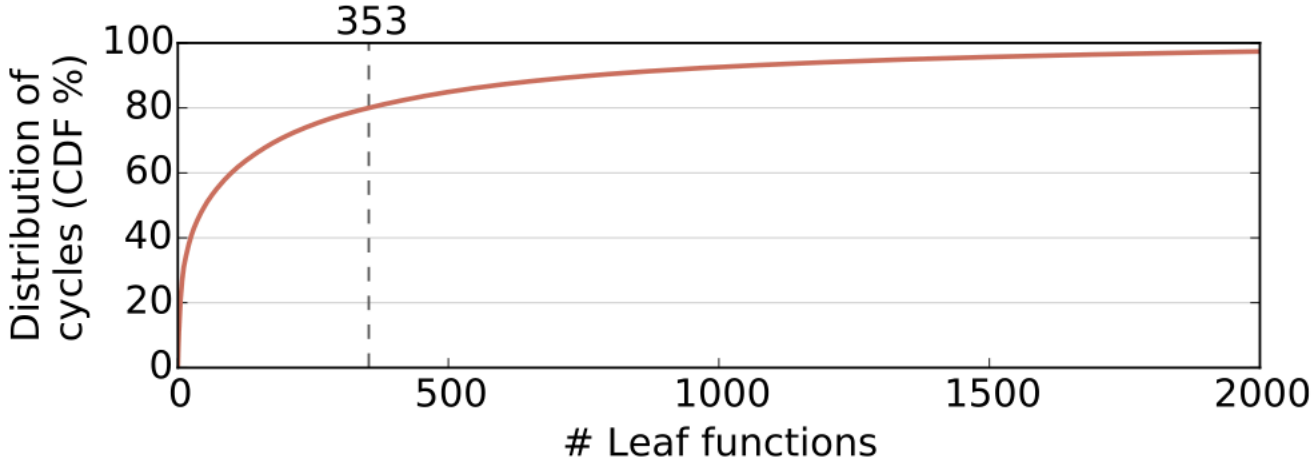
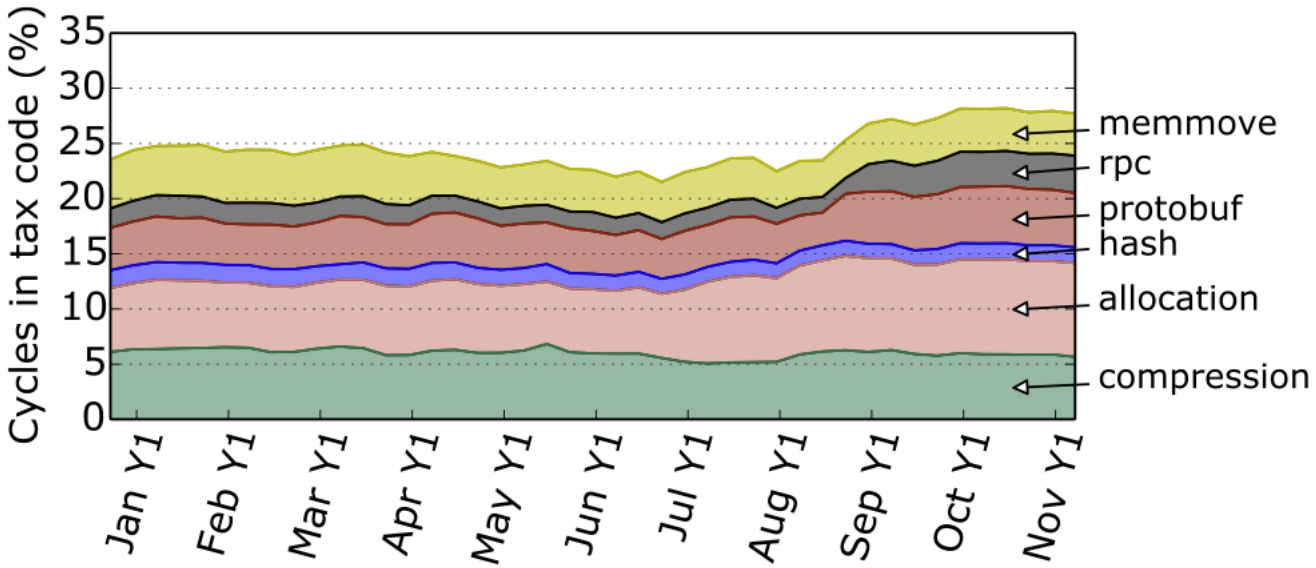


Figure 3: Individual binaries are already optimized. Example binary without hotspots, and with a very flat execution profile.

Q: Why does workload diversity make performance optimization challenging?

The datacenter tax

- Common building blocks comprise 22-27% of all cycles
  - Optimizing these could significantly improve efficiency
  - Protocol buffers, RPCs, hashing, compression, memory allocation, data movement



Datacenter tax breakdown

1. Protocol buffers
   - Cycles spent on serializing/deserializing data
   - Could it be hardware accelerated?
2. RPCs (e.g., gRPC)
   - Load balancing, encryption, failure detection (1/3 of cycles)
   - The rest is payload data movement
3. Data movement
   - `memcpy()` and `memmove()`. Inline copies not tracked
   - 4-5% of datacenter for just these two functions
4. Compression
   - Google built Snappy to improve speed at cost of ratio
   - If hardware accelerated, could target higher ratios
5. Memory allocation
   - Still expensive despite Google's highly optimized TCMalloc allocator
6. Hashing
   - Intel has recently started accelerating this in hardware

Q: How would you design hardware accelerators for the datacenter tax?

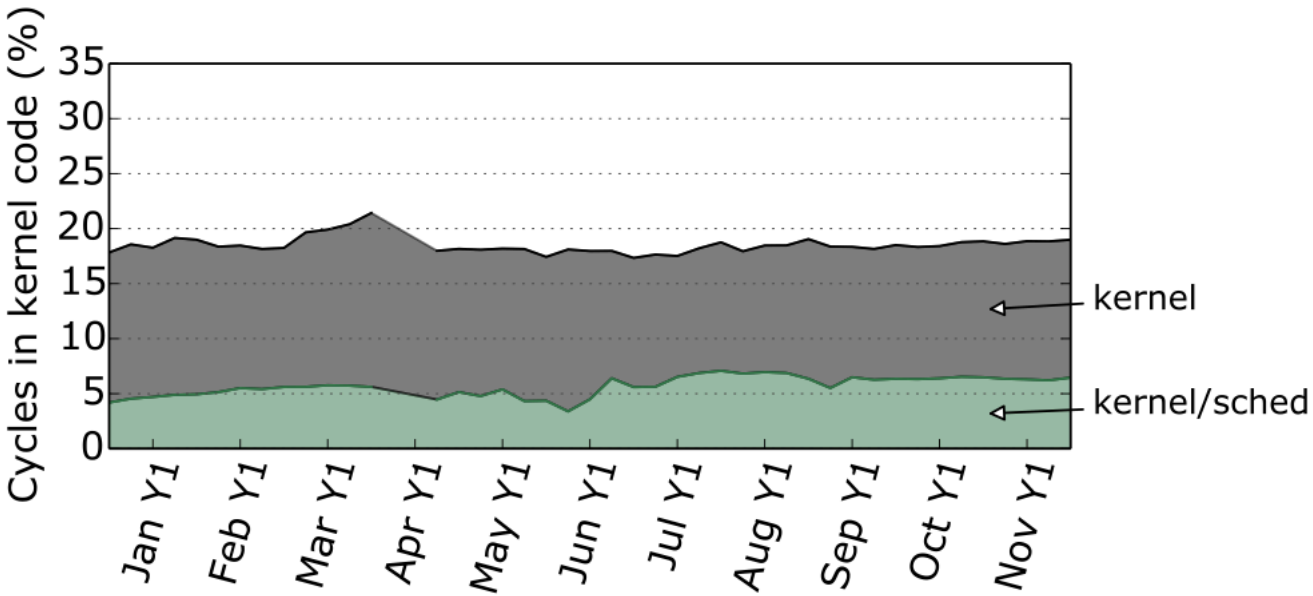
Q: How would you design hardware accelerators for the datacenter tax?

Challenge: Fine-grained and tightly coupled

Likely need direct integration with CPU cache

What about the kernel?

- 1/5th of all cycles spent in the kernel
- Not small, can't be accelerated with hardware
- Scheduler expensive, a 90th %tile machine runs 4500 threads concurrently



Just enough computer architecture

Front End: Fetching + Decoding

Back End: Number crunching

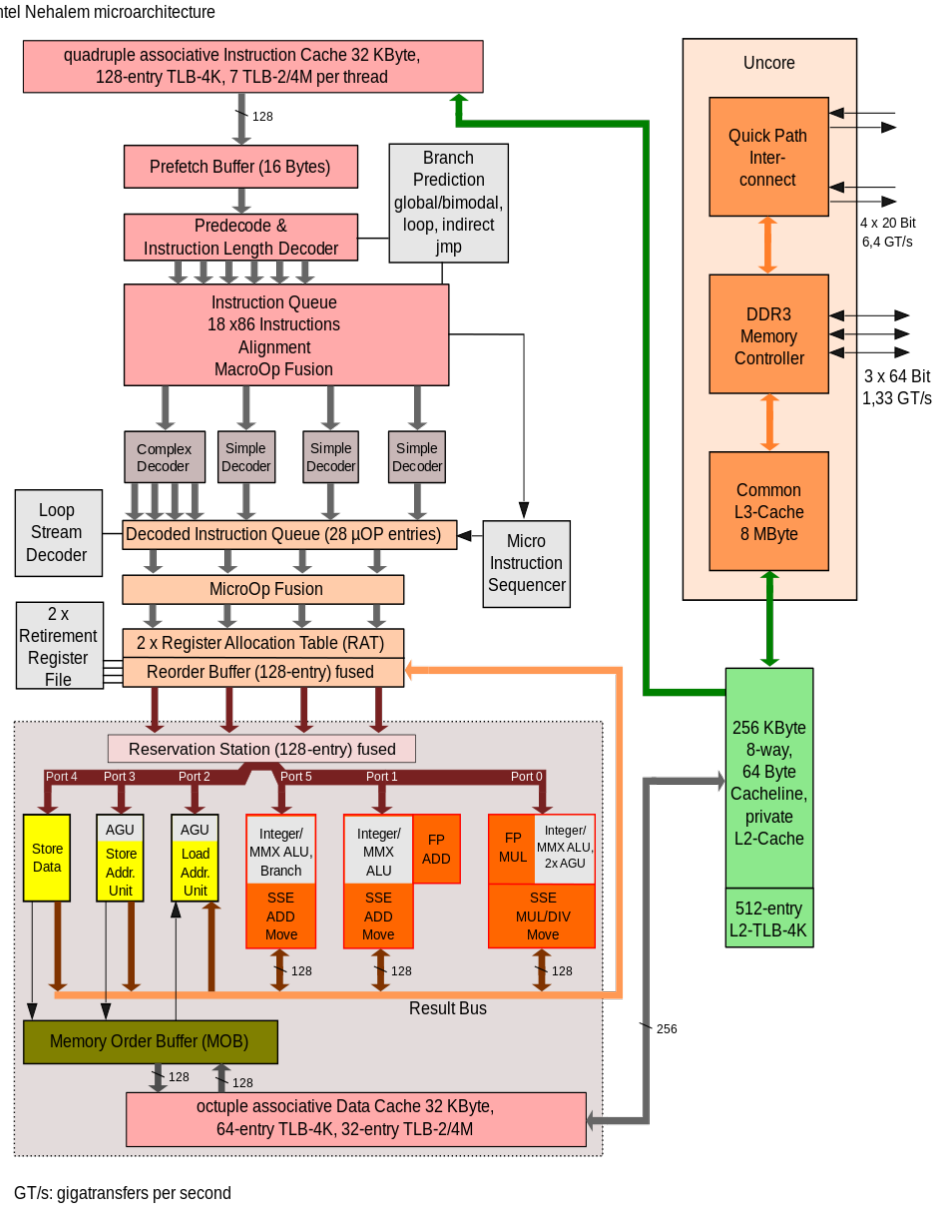
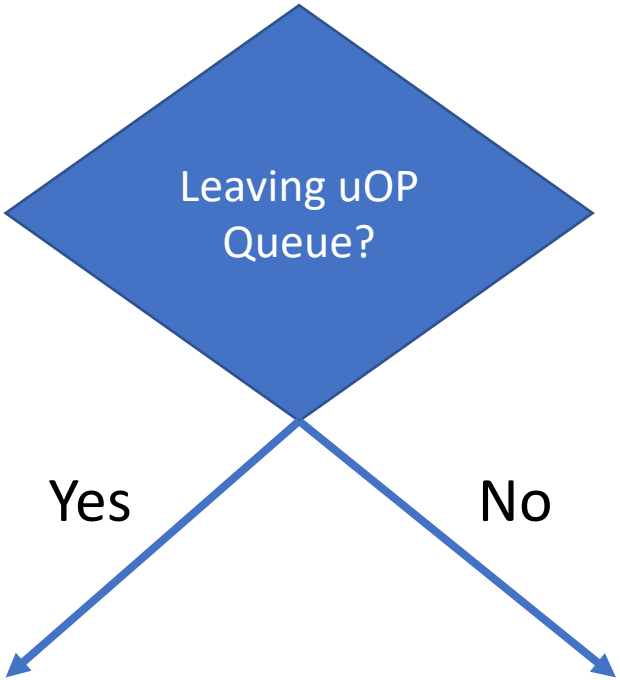


Diagram credit: [Wikipedia]

Pipeline slots

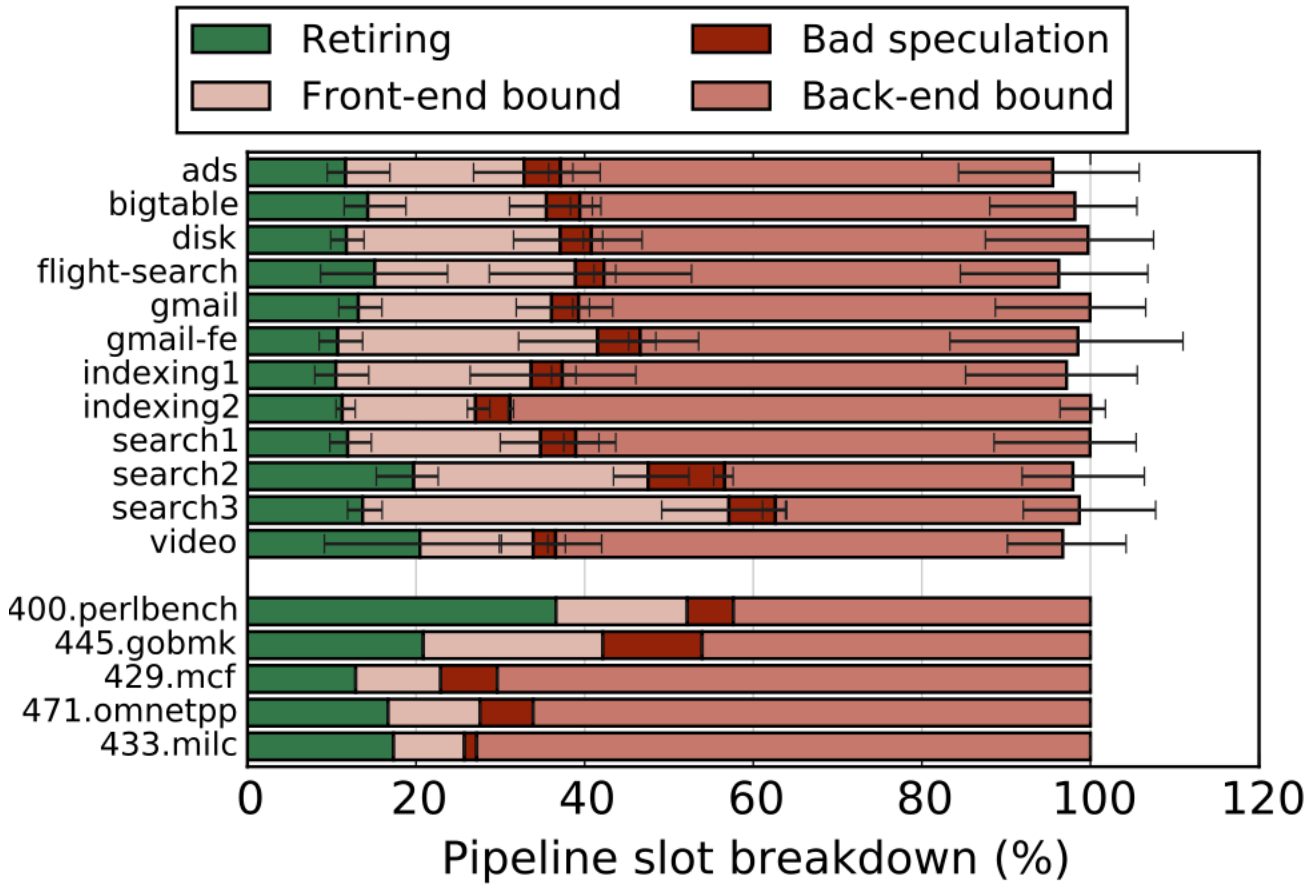


Retired if committed,
Otherwise bad speculation

Stalled by front-end or backend

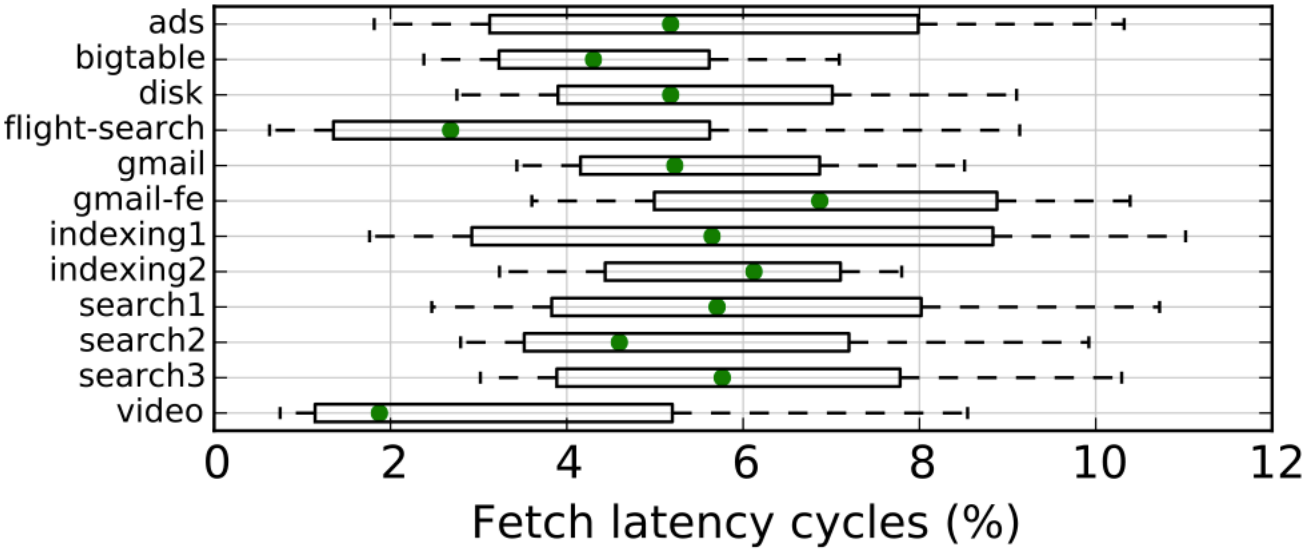
Microarchitectural analysis

- SPEC CPU is used by researchers to evaluate computer architectures
- WSC workloads have low retirement rates and are front-end bound



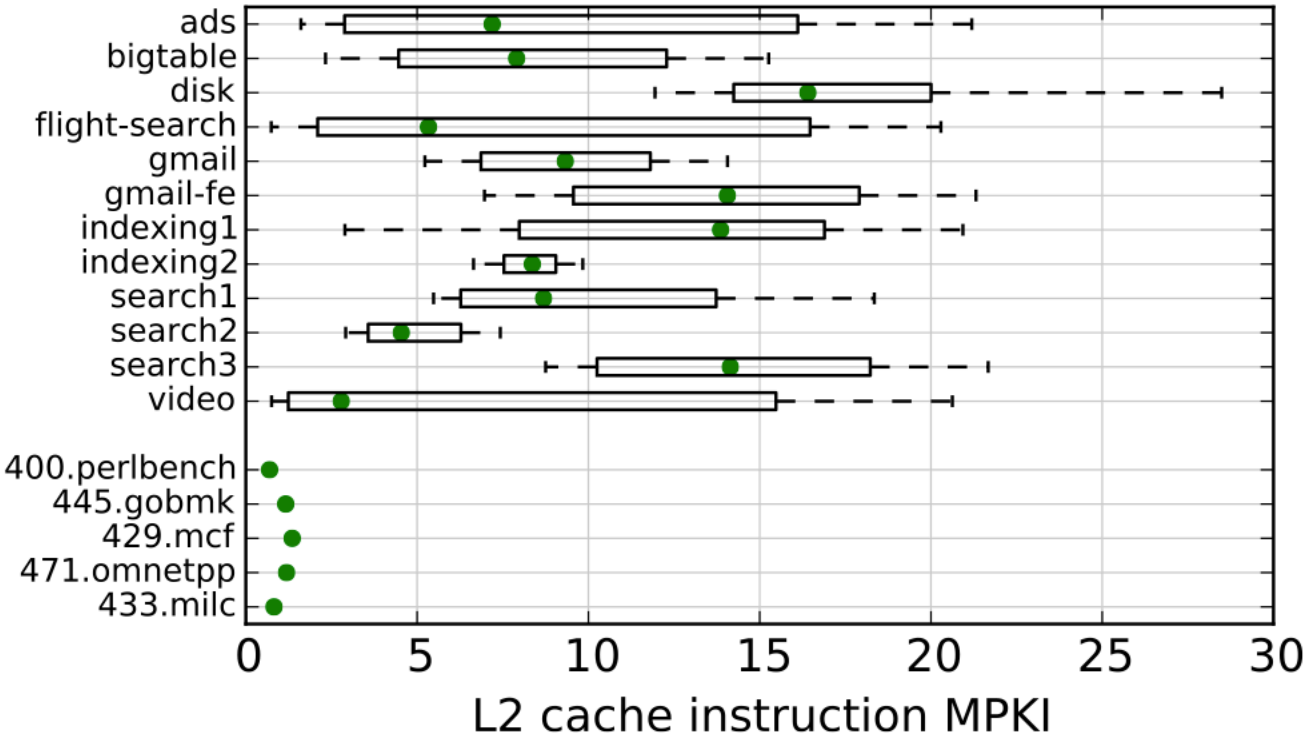
Why is the front-end a bottleneck?

~5% of cycles stalling waiting to fetch instructions



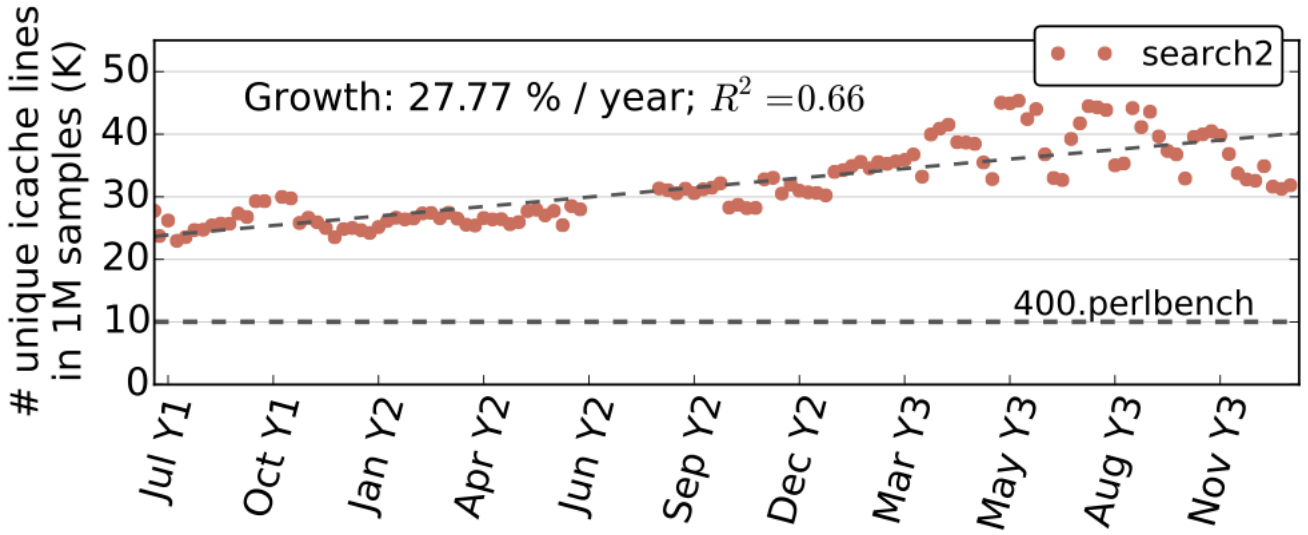
Why is the front-end a bottleneck?

- WSC workloads have large code footprints (100MB)
- Many lukewarm instructions, must go to L2 cache



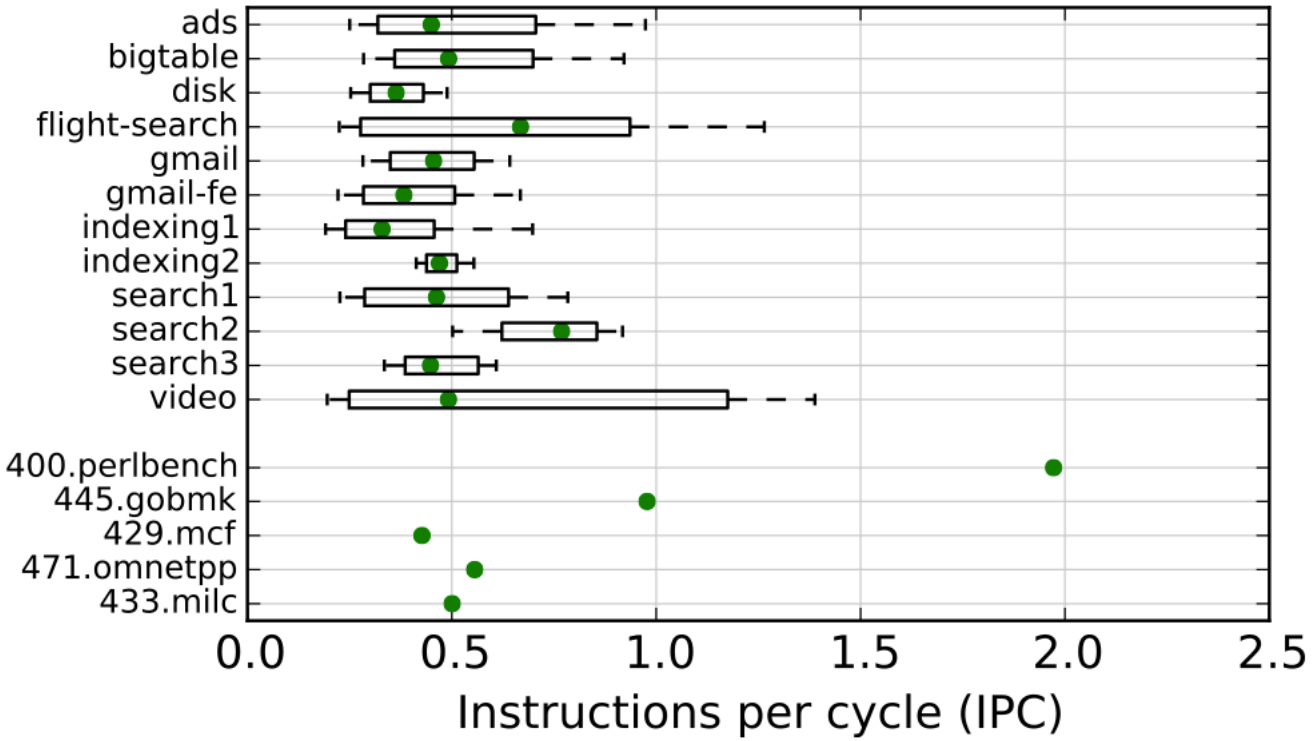
Instruction cache footprints can get worse over time

- Hypothesis: Projects with new features, and higher development velocity, increase instructions faster



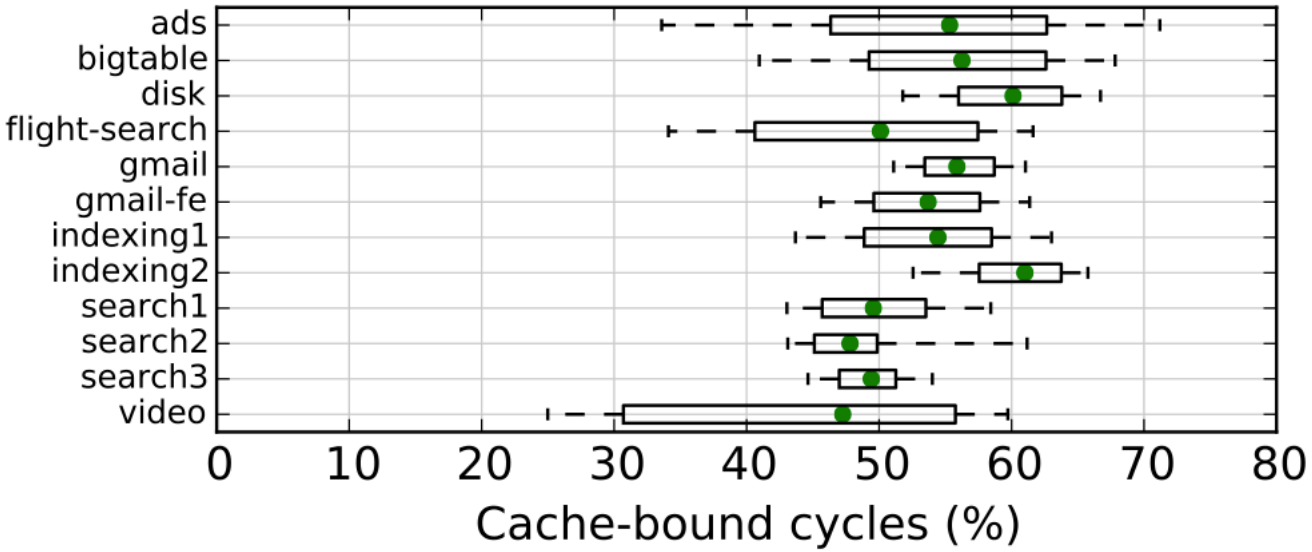
Why is the back-end a bottleneck?

- Time spent serving data cache requests
- Lack of instruction-level parallelism (ILP)



Why is the back-end a bottleneck?

- Over half of cycles are stalled on data cache misses
- But ILP can still be high, bursty computations



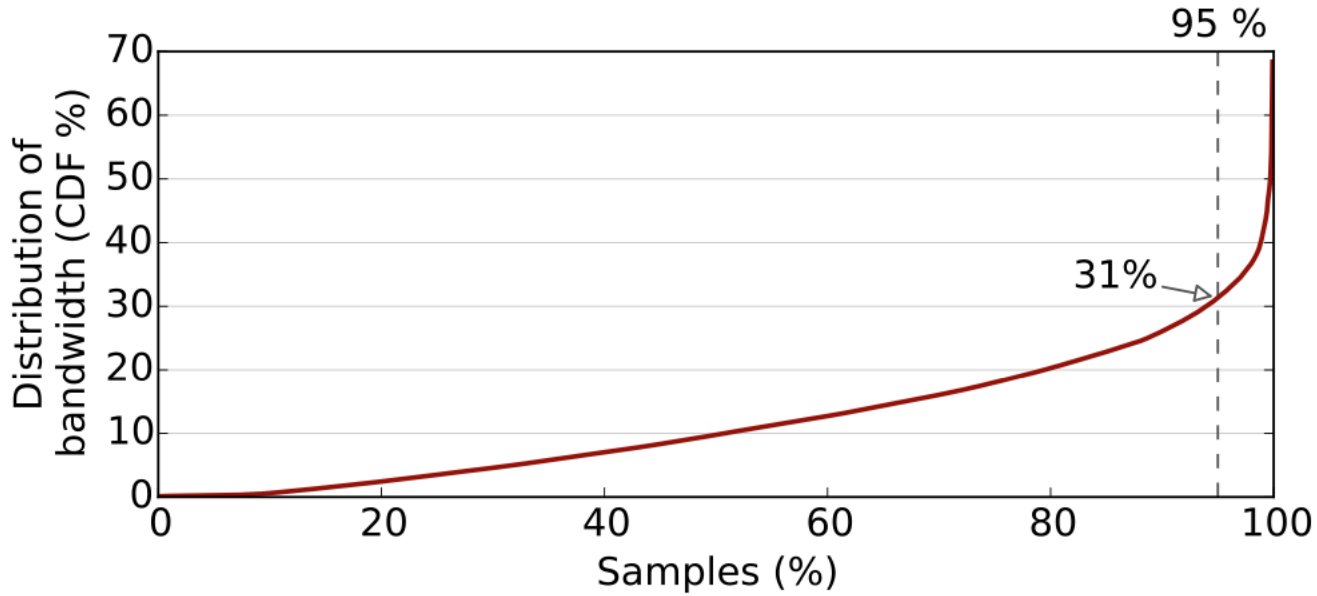
Q: If pipeline is stalled so often, why not use wimpier (slower) cores?

Q: If pipeline is stalled so often, why not use wimpier (slower) cores?

See: Brawny cores still beat wimpy cores, most of the time

Memory bandwidth

- DRAM bandwidth usage is universally low
  - Partly explained by low CPU utilization (30%-70%)
  - But dependent cache accesses limit bandwidth too
- Optimize for memory latency? Provision less bandwidth?



What about SMT?

- Allows two logical threads share the resources of a physical core
- In general, most efficient when workloads have different bottlenecks
- WSC applications have deficiencies in front-end and back-end, with fine-grained intense compute phases, making them well suited for SMT
- Challenge: SMT partitions resources and reduces cache and execution unit capacity
  - Decreases single-thread performance, harming latency, but improves efficiency overall

Q: Would wider SMT (more than two threads) help WSC workloads?

Conclusion

- WSC workloads are unique and deserve special optimizations
- All of these are great research challenges!

Finding	Investigation direction
`workload diversity`	Profiling across applications.
`flat profiles`	Optimize low-level system functions.
`datacenter tax`	Datacenter specific SoCs (protobuf, RPC, compression HW).
`large (growing) i-cache footprints`	I-prefetchers, i/d-cache partitioning.
`bimodal ILP`	Not too "wimpy" cores.
`low bandwidth utilization`	Trade off memory bandwidth for cores. Do not use SPECrate.
`latency-bound performance`	Wider SMT.