Profiling a Warehouse-scale Computer

Presented by Alain Kohli

S. Kanev, J. P. Darago, K. Hazelwood, P. Ranganathan, T. Moseley, G.-Y. Wei and D. Brooks,
“Profiling a Warehouse-scale Computer”, ISCA, 2015

Executive Summary

- **Problem:** Are current microarchitectures suited for WSC workloads?
- **Motivation:** No prior research investigated this issue before
- **Goal:** Identify parts of the microarchitecture that can be optimized for WSCs
- **Methodology:** Profile a live WSC to get insights into microarchitectural bottlenecks
- **Results:**
  - Diverse workloads
  - Unusually large i-cache stress
  - Large d-cache stress as expected
  - Low Instruction level parallelism
- **Conclusion:**
  - Better i-cache prefetcher
  - i/d-cache separation
  - Datacenter specific SoCs

Outline

- Background, Problem & Goal
- Methodology
- Results
  - Workload diversity
  - Microarchitectural breakdown
  - Front-end bottlenecks
  - Back-end bottlenecks
  - SMT
- Strengths & Weaknesses
- Ideas & Takeaways
- Questions & Discussion

Outline

- Background, Problem & Goal
- Methodology
- Results
  - Workload diversity
  - Microarchitectural breakdown
  - Front-end bottlenecks
  - Back-end bottlenecks
  - SMT
- Strengths & Weaknesses
- Ideas & Takeaways
- Questions & Discussion

Background, Problem & Goal

- Warehouse-Scale Computers (WSCs) become more important every year
- WSCs pose different challenges compared to traditional servers due to the scale
- Optimizing servers in a datacenter isn't going to help if the performance of the system as a whole isn't considered
- Latency is the defining performance metric
- There are many possible bottlenecks in a WSC
- Microarchitectural bottlenecks are not very well understood yet
- WSCs are very complex and theoretical analysis has its limitation

Background, Problem & Goal - Previous research

- Isolated benchmarks / unrealistic workloads
  [M. Ferdman et al., ASPLOS, 2012]
- Analyzed other aspects (software/system design)
  [C. Kozyrakis et al., IEEE Micro, 2010]
- Somewhat outdated
  [L. A. Barroso, IEEE Micro, 2003]

Background, Problem & Goal - Main question

What parts of the micro-architecture are the biggest bottlenecks in a real warehouse-scale computer?

Outline

- Background, Problem & Goal
- Methodology
- Results
  - Workload diversity
  - Microarchitectural breakdown
  - Front-end bottlenecks
  - Back-end bottlenecks
  - SMT
- Strengths & Weaknesses
- Ideas & Takeaways
- Questions & Discussion

Methodology

- Profiling a live Google datacenter over 1-3 years (~20,000 machines)

- Restricted to C++ and Intel Ivy Bridge
  - Get consistent and comparable data
  - Simplifies analyzing the callstack
  - C++ consumes the most amount of CPU cycles, even if it isn't the most popular

- Workloads for microarchitecture analysis
  - Standard Google workloads that are as diverse as possible
    - ads, bigtable, disk, flight-search, gmail, gmail-fe, indexing1/2, search1/2/3, video
  - Common benchmarks as a reference
    - 400.perlbench, 445.gobmk, 429.mcf, 471.omnetpp, 433.milc

Methodology - Google-Wide-Profiling (GWP)

- Google in-house profiling
- Non-intrusive performance sampling

- Procedure
  - Randomly select a small fraction of servers to profile each day
  - Trigger collection of 1s long profile samples (through `perf`)
  - Symbolize the collected sample's callstacks
  - Aggregate samples in a database for analysis

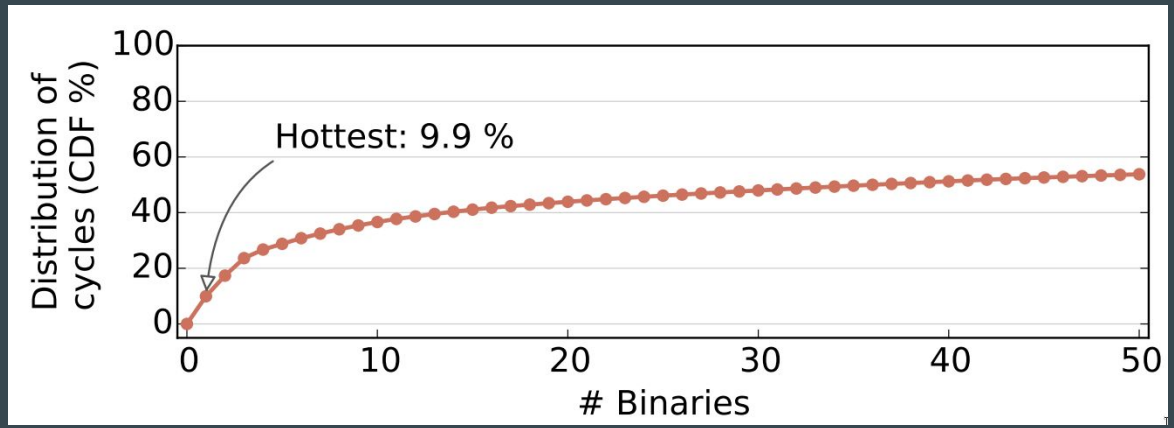
- Validations required by the Performance Monitoring Unit (PMU) are applied
  - Exact validations not further specified

Outline

- Background, Problem & Goal
- Methodology
- Results
  - Workload diversity
  - Microarchitectural breakdown
  - Front-end bottlenecks
  - Back-end bottlenecks
  - SMT
- Strengths & Weaknesses
- Ideas & Takeaways
- Questions & Discussion

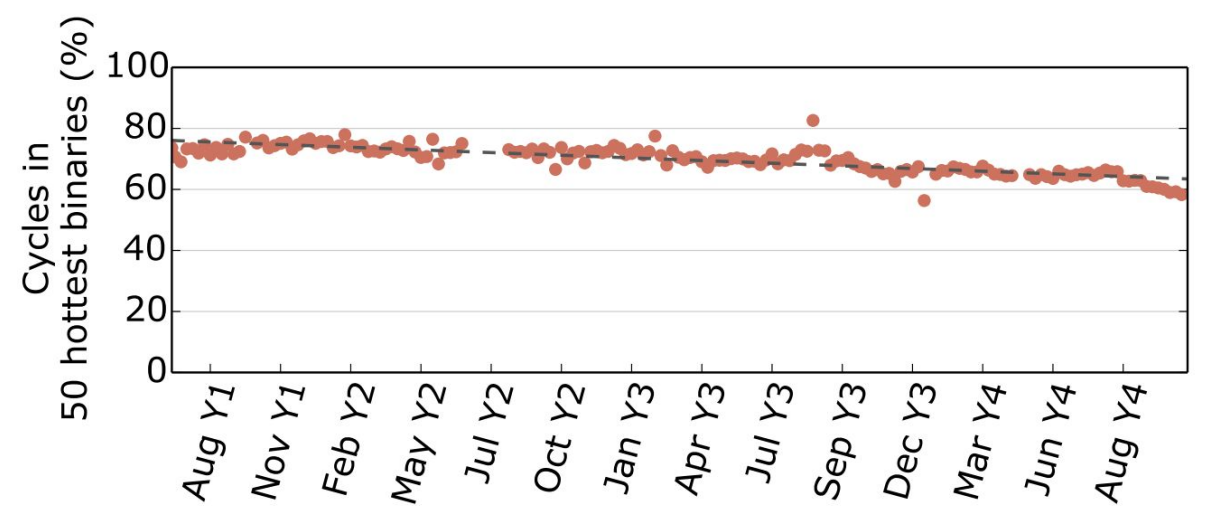
Results - Workload diversity

- We would like to see how important it is to analyze the datacenter as a whole
- Is there one application that uses up almost all resources?
- Percentage of total cycles consumed by the X most CPU intensive binaries
- The 50 hottest binaries consume less than 60% of all CPU cycles
- There is not one single “killer application” for performance



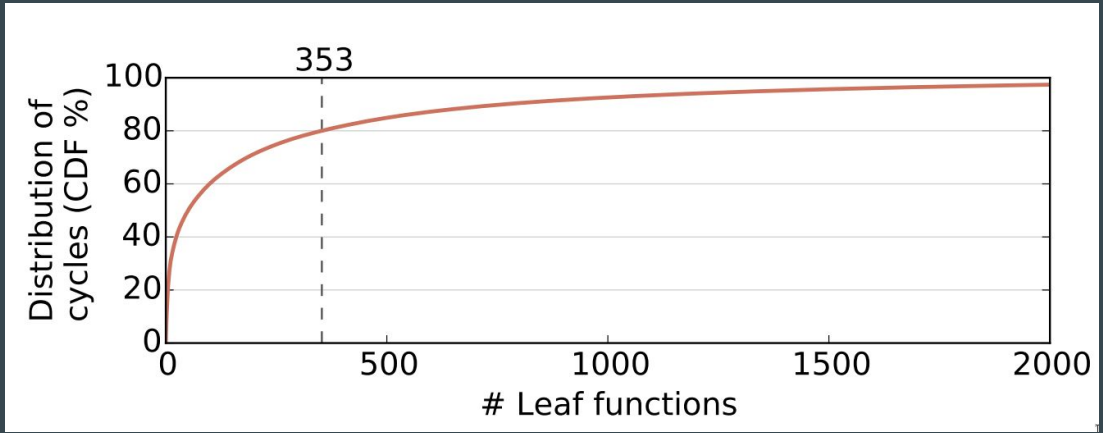
Results - Workload diversity

- There is a slight trend for less heavy applications over the years
- The 50 hottest binaries use a smaller percentage of CPU cycles every year



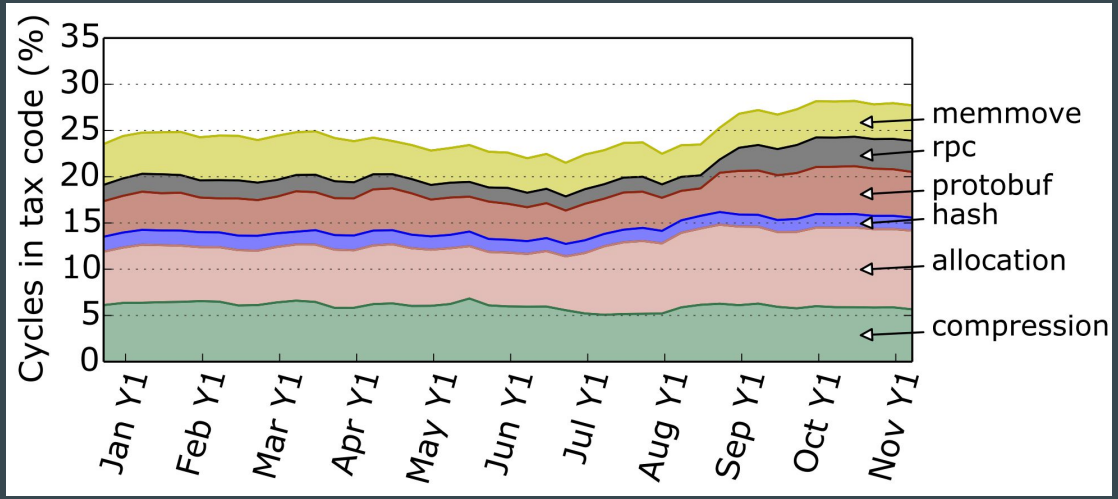
Results - Workload diversity

- Are there clear things to optimize on an application basis at least?
- Percentage of total cycles consumed by the X hottest leaf functions in `search3`
- There are no clear hotspots to optimize within the applications either



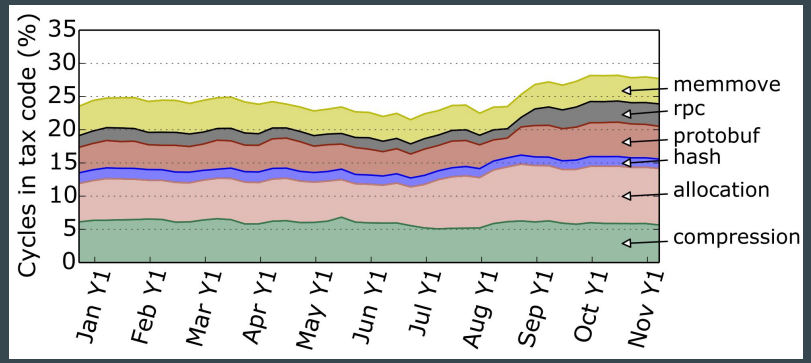
Results - Workload diversity (datacenter tax)

- Are there lower level common operations that cause a slowdown?
- Common components that could be prime candidates for hardware acceleration
- Datacenter tax has a trend to slightly increase



Results - Workload diversity (datacenter tax)

- **memmove:** `memcpy()` / `memmove()`, not counting other forms of copying
- **rpc:** Remote Procedure Calls (load balancing, encryption, data movement, etc.)
- **protobuf:** Protocol buffers, used for serialization
- **hash:** Hashing used in various parts, i.e. for communication
- **allocation:** All form of memory allocation
- **compression:** Compression using multiple algorithms

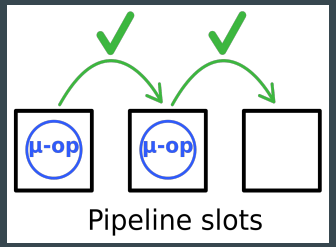


Results - Microarchitectural bottleneck breakdown

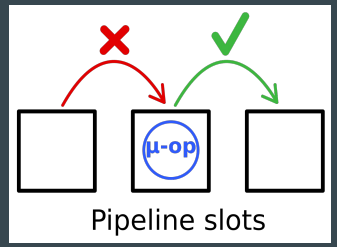
- What is preventing us from fully using the microarchitectural resources we have?
- We look at the state of the pipeline each cycle and determine, whether µ-ops successfully pass through it
- If µ-ops do not successfully pass through the pipeline, we want to know which part of the pipeline is stalling and preventing them from doing so
- Broad categorization first (retiring, bad speculation, frontend/backend bound)

Results - Microarchitectural bottleneck breakdown

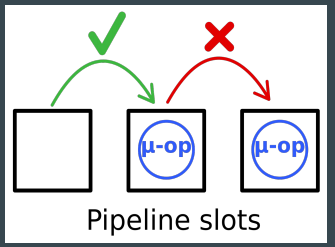
- Retiring
  Useful work
- Bad speculation
  Branch prediction fail
- Front-end bound
  Can’t fill pipeline fast enough
- Back-end bound
  Can’t empty pipeline fast enough



Retiring
Bad speculation



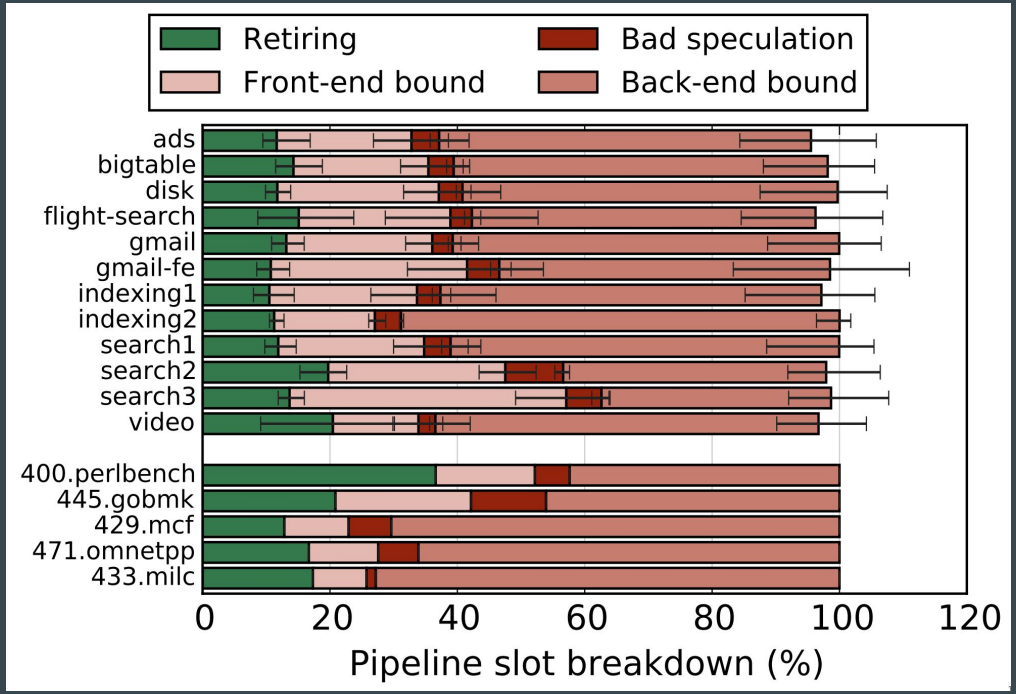
Front-end bound



Back-end bound

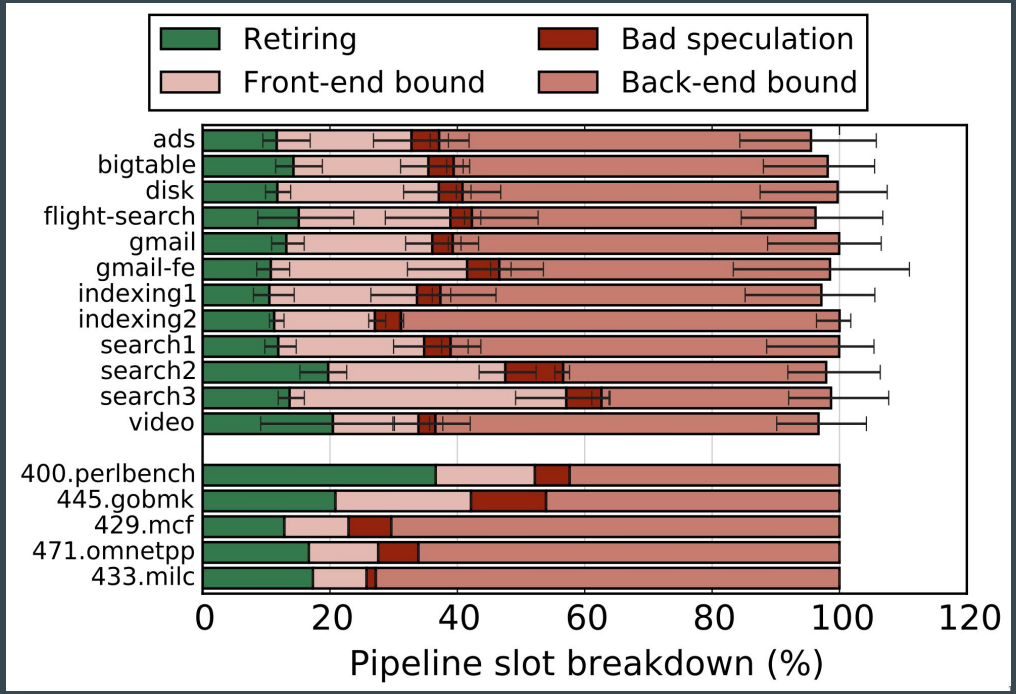
Results - Microarchitectural bottleneck breakdown

- Retiring
  Useful work
- Bad speculation
  Branch prediction fail
- Front-end bound
  Can't fill pipeline fast enough
- Back-end bound
  Can't empty pipeline fast enough



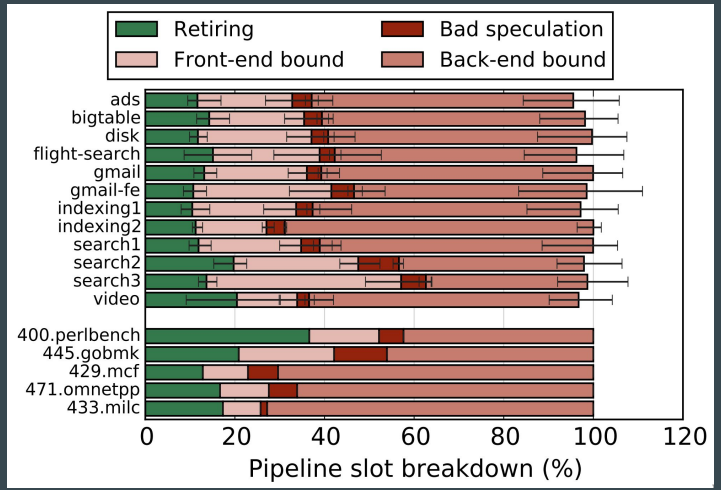
Results - Microarchitectural bottleneck breakdown

- 400.perlbench
  High IPC/i-cache stress
- 445.gobmk
  Hard to predict branches
- 429.mcf / 471.omnetpp
  Memory bound/latency
- 433.milc
  Memory bound/bandwidth



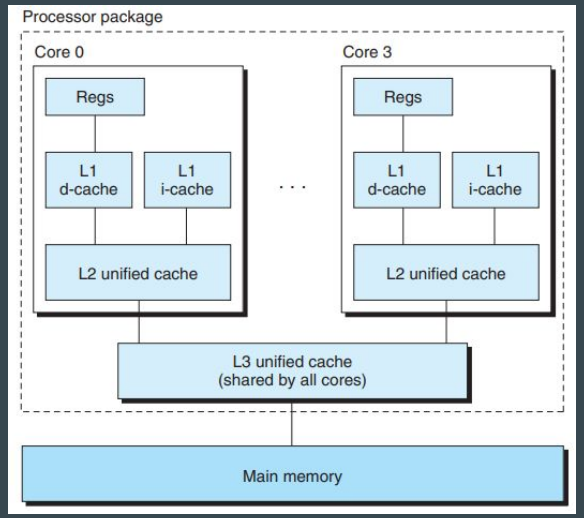
Results - Microarchitectural bottleneck breakdown

- Is this what we would expect?
- Bad speculation seems to be comparable to the benchmarks, no further analysis
- The backend bound part is also comparable
- Significantly larger frontend bound part than the reference
- Smaller retiring part than the reference



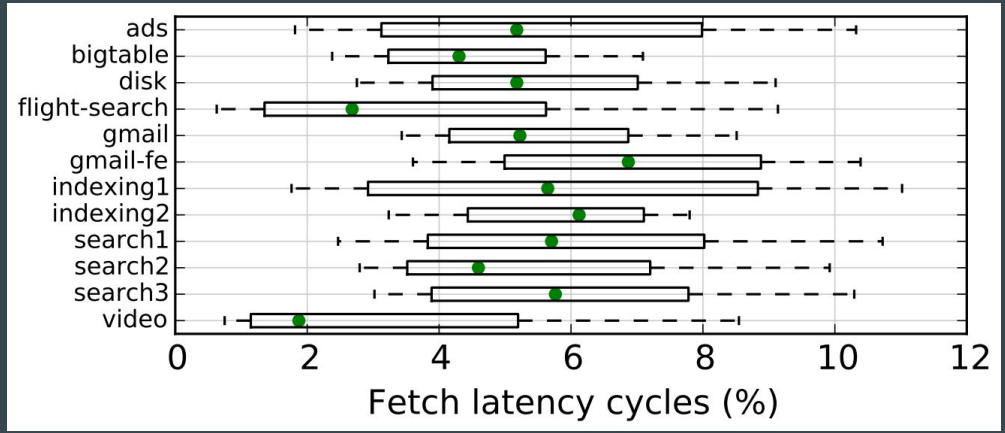
Memory hierarchy overview

- Instruction cache could bottleneck the frontend
- Data cache could bottleneck the backend
- Instruction and data caches are shared on L2



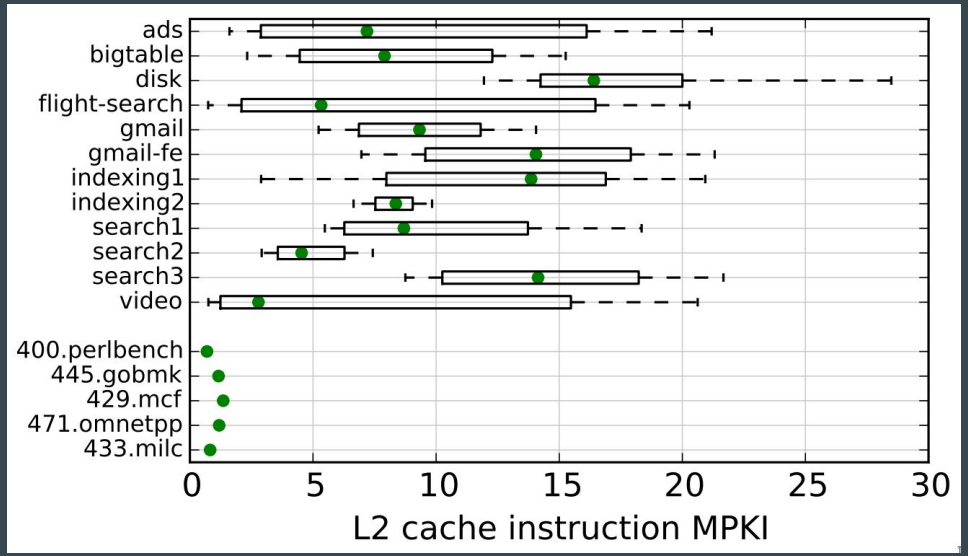
Results - Front-end bottlenecks (Instruction cache)

- We want to find the reasons for the front-end bottleneck
- Measure the amount of cycles no instruction enters the pipeline from the frontend
- For a significant part of instructions, the pipeline is completely starved



Results - Front-end bottlenecks (Instruction cache)

- Much higher L2 i-cache MPKI (Misses per kilo instruction)
- Starvation is most likely due to large i-cache stress



Results - Front-end bottlenecks (Instruction cache)

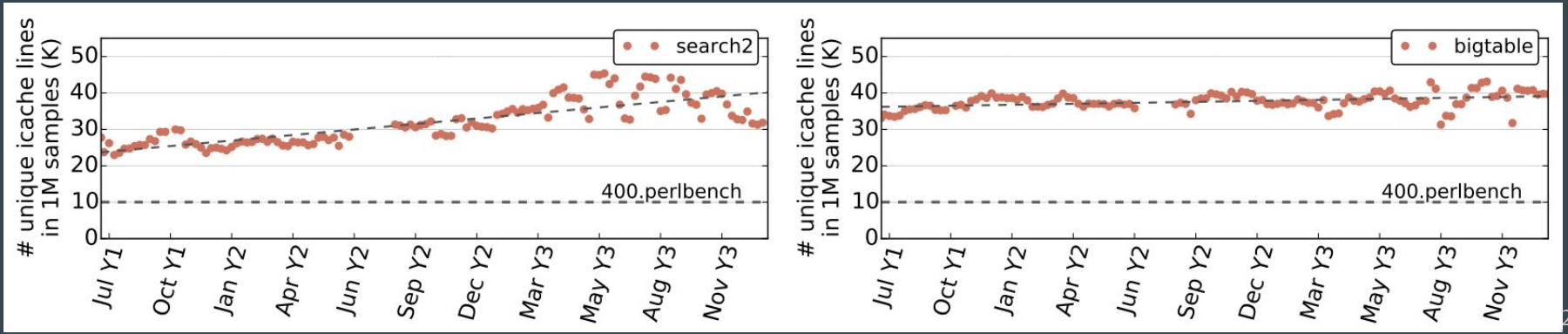
- Large (and growing) binaries without significant hotspots → More i-cache stress

Possible solutions:

- Larger i-caches
- Better prefetchers
- Separate i-cache and d-cache

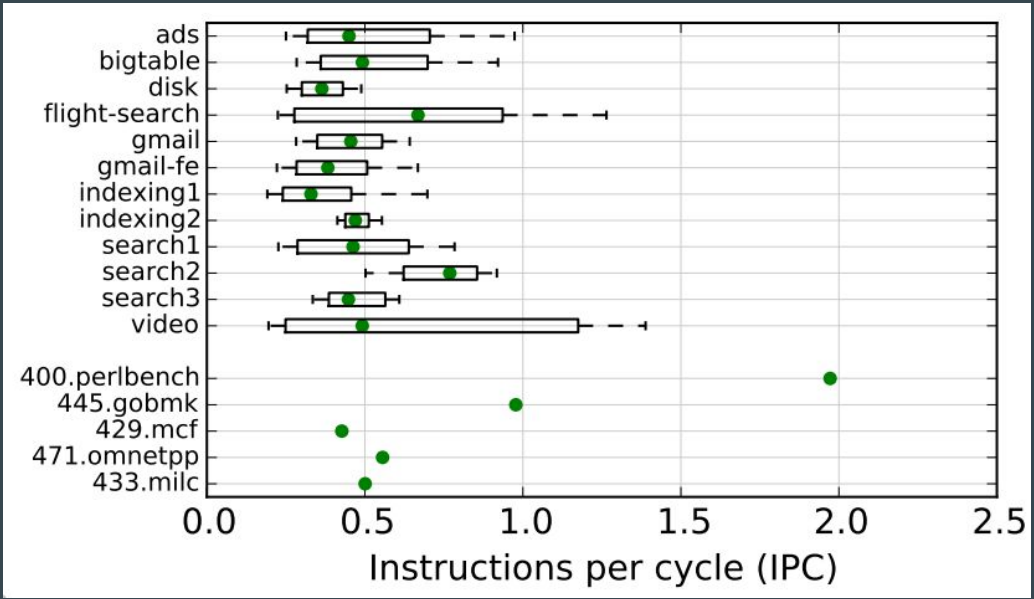
Results - Front-end bottlenecks (Instruction cache)

- How large would i-caches need to be to alleviate this problem?
- Usually this would be simulated, which is hard with such a complex system
- Measuring the amount of unique i-cache lines that would be required to cover 99% of the instruction pointer samples
- I-caches become more and more stressed over the years
- Much larger than current L2 caches, not even accounting for data (688KB or more)



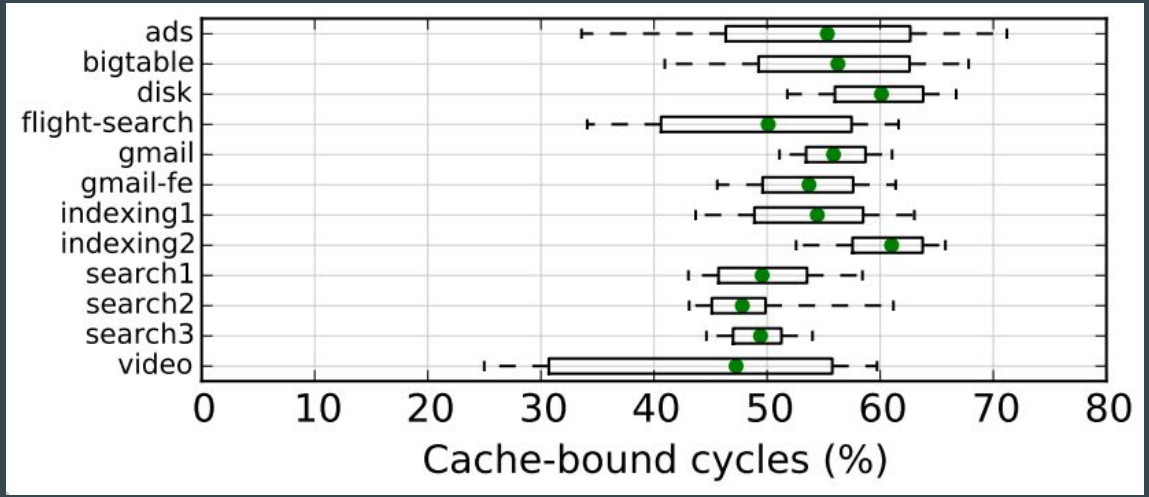
Results - Back-end bottlenecks

- Generally very low IPC due to d-cache stalls and/or low ILP



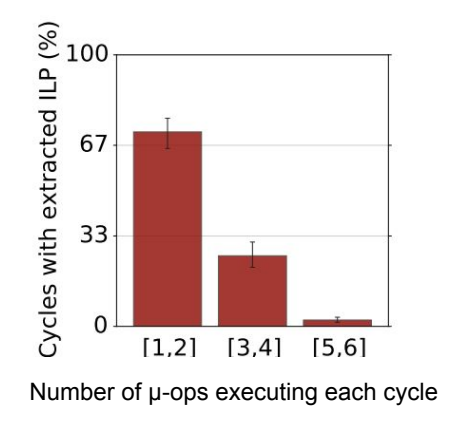
Results - Back-end bottlenecks (Data cache)

- Is the slowdown mostly d-cache driven?
- Data cache stalls are 50-60% of all cycles (~80% of backend bound cycles)
- WSC applications are very memory intensive



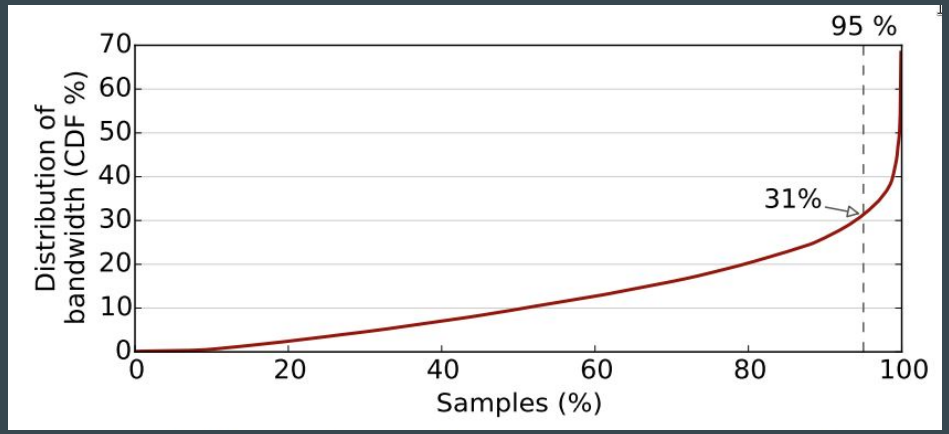
Results - Back-end bottlenecks (ILP)

- How well is Instruction Level Parallelism (ILP) used?
- Up to 6 µ-ops executing in parallel (6 execution ports)
- Generally low ILP in line with cache misses
- Possibly due to a fine-grained mix of dependent cache accesses and bursty computation



Results - Back-end bottlenecks (Memory bandwidth)

- Are the d-cache stalls bandwidth or latency limited?
- Very low median memory bandwidth utilization, lower than median CPU utilization (10% vs 40-70%)
- Memory latency is more important than bandwidth



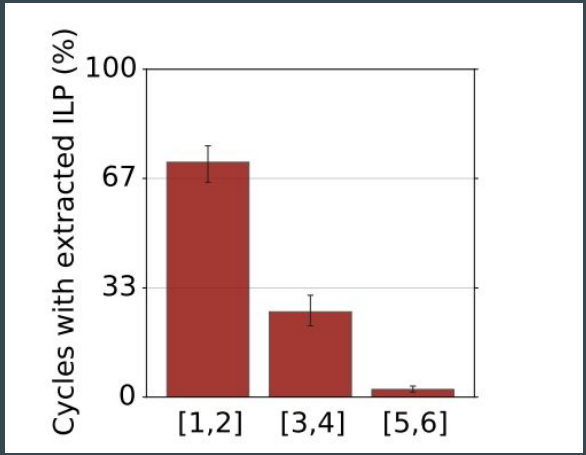
Results - SMT

- So far, Simultaneous Multi-Threading (SMT) was not accounted for
- Only an estimate, as SMT can’t be turned off in the live system measured
- The workload seems like a good candidate for SMT
- Measuring the core utilization increase from SMT
- Comparing per-thread and per-core metrics

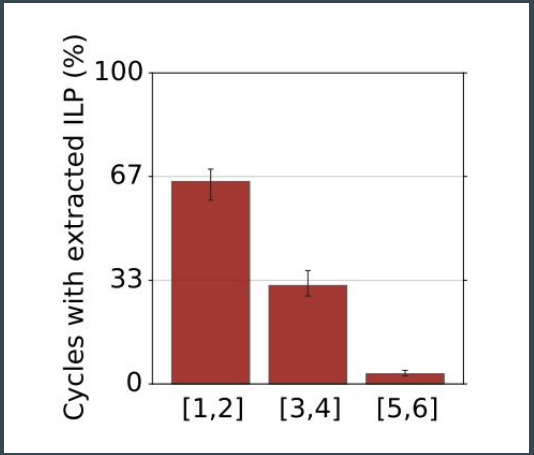
Results - SMT (Backend)

- On the backend, functional execution unit utilization increases as expected
- More execution ports tend to be utilized

Per hyperthread

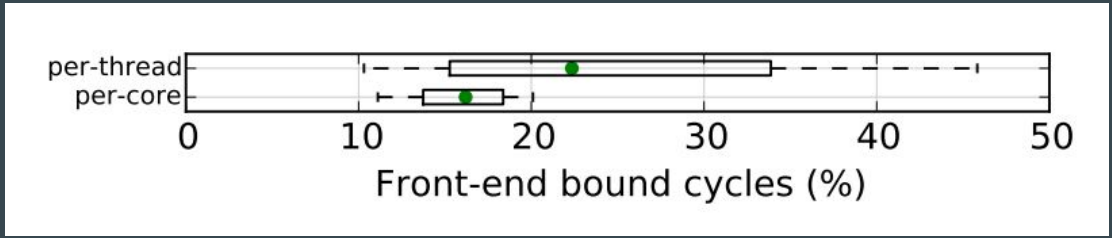


Per core



Results - SMT (Frontend)

- We might see either
  - Instruction cache pressure increases with SMT, increasing the frontend bottleneck
  - Long latency fetch bubbles can be absorbed by fetching from another hyperthread, reducing the frontend bottleneck
- The latter seems to dominate, as utilization increases



Results - Paper conclusion

- Workloads are diverse with few hotspots
  - Profiling across the whole WSC is important to get accurate measurements
- Some common operations use up a large percentage of cycles (datacenter tax)
  - Datacenter specific SoCs
- Instruction cache footprints are large and growing
  - Better prefetching, separate i/d-cache
- Memory bandwidth is usually not fully utilized
  - Make bandwidth tradeoffs in favor of e.g. more cores

Outline

- Background, Problem & Goal
- Methodology
- Results
  - Workload diversity
  - Microarchitectural breakdown
  - Front-end bottlenecks
  - Back-end bottlenecks
  - SMT
- Strengths & Weaknesses
- Ideas & Takeaways
- Questions & Discussion

Strengths

- Analyzes relevant, real-world workloads
- Gives plausible explanations for why certain slowdowns happen
- Gives concrete solution ideas for every problem seen
- Has a systematic top-down approach

Weaknesses

- Generalizes all problems as WSC problems, even if they might be Google specific
- The conclusion argues about brawny/wimpy cores without much analysis
- The Performance Monitoring Unit (PMU) limitations and validation for profiling are left very vague
- For SMT backend measurements, only ILP was considered and not d-cache stress, which is 80% of the backend-bottlenecks
- Most of the conclusion section was only presented through keywords, which weren't always very clear

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.

Summary of findings and suggestions for future investigation.

Outline

- Background, Problem & Goal
- Methodology
- Results
  - Workload diversity
  - Microarchitectural breakdown
  - Front-end bottlenecks
  - Back-end bottlenecks
  - SMT
- Strengths & Weaknesses
- Ideas & Takeaways
- Questions & Discussion

Ideas/Takeaways

- Compare specific aspects of WSC workloads to justify generalization
- Memory related architectural changes have great potential
- Reducing the instruction memory footprint of applications could have great benefits
- Improving i-cache stress seems harder compared to d-cache stress as you are more bound to the CPU
- Existing benchmarks don’t seem to be quite good enough to analyze WSC workload behavior

Questions & Discussion

Should software developers strive for smaller binary sizes? Is this even possible?

Could you think of any more radical solutions for improving i-cache stress, similar to PIM for d-cache stress?

Why aren’t WSC hardware accelerators used yet?

Do you think consumer software will start running into similar limitations soon?