Warming Up a Cold Front-End with Ignite

David Schall, Andreas Sandberg, Boris Grot

October 30, 2023

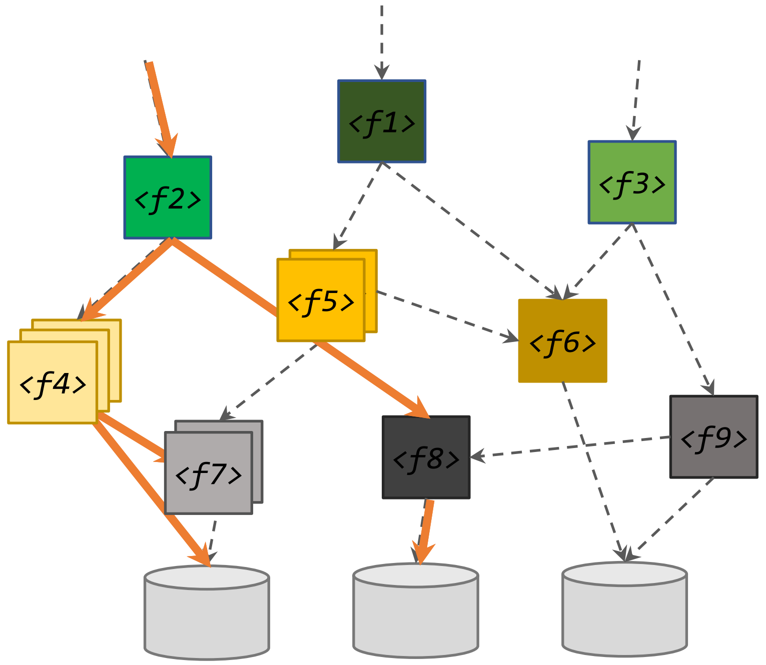
Serverless

Rapidly emerging cloud execution model

- More than 50% of all cloud customers of AWS, Google Cloud and Azure use serverless *[Datadog 2023]*

Applications are organized as a graph of tiny stateless functions

- Functions **run only on-demand**
  - Invoked via trigger event
- Functions are **stateless**
  - Facilitates **rapid** and on-demand **scaling down to zero**
- Developer: **pay per invocation** (CPU+memory), not idle time
- Cloud providers: high density and resource utilization



Serverless is a hot topic for research!

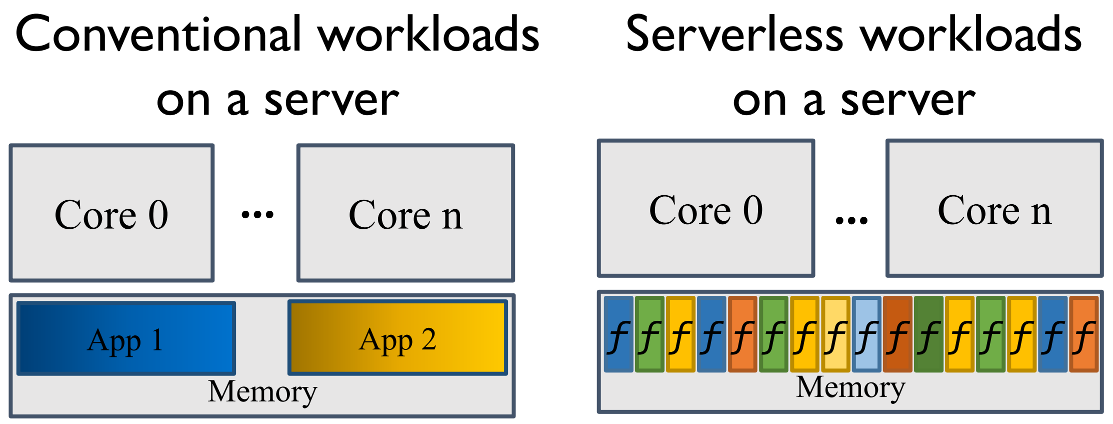
Serverless Characteristics on a Server

Serverless sharply contrasts with conventional workloads

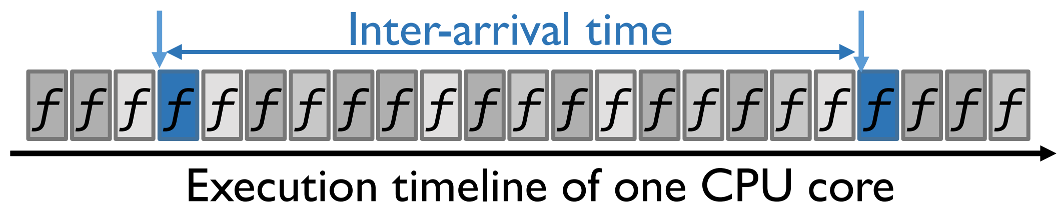
- **Conventional:**
  - Large VMs constantly occupy server resources
- **Serverless:**
  - Short function execution times: a few ms or less
  - Small memory footprint: tens of MB
  - Sporadically invoked (seconds or minutes)
    [Microsoft Azure @ATC20]

Implications:

- **Extreme multi-tenancy:**
  Thousands of functions resident on a server
- **Huge degree of interleaving** between two invocations of the same function



Conventional workloads on a server
Core 0
...
Core n
App 1
App 2
Memory
Serverless workloads on a server
Core 0
...
Core n
f
Memory



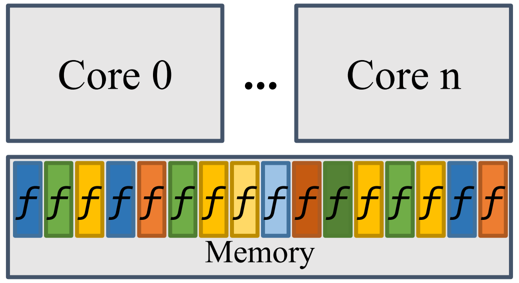
Microarchitecture is not designed for serverless!

Serverless on a Cloud Server

Prior work found *[Lukewarm Serverless Functions @ISCA'22]*:

- Huge degree of function interleaving causes severe performance degradation.
- Lukewarm execution:
  - **Functions reside warmly in memory**
  - **On-chip microarchitectural state is cold for every invocation**
    - Thrashed by other interleaved executed functions

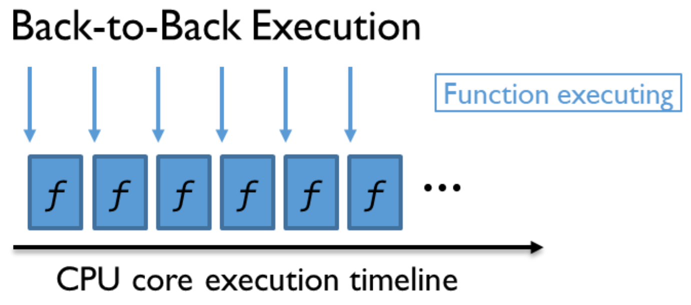
Serverless workloads on a server

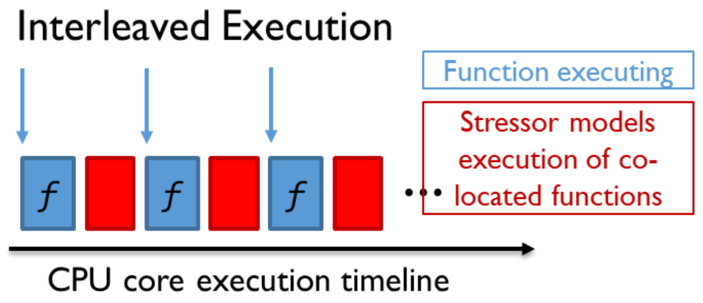


Understanding Lukewarm Execution

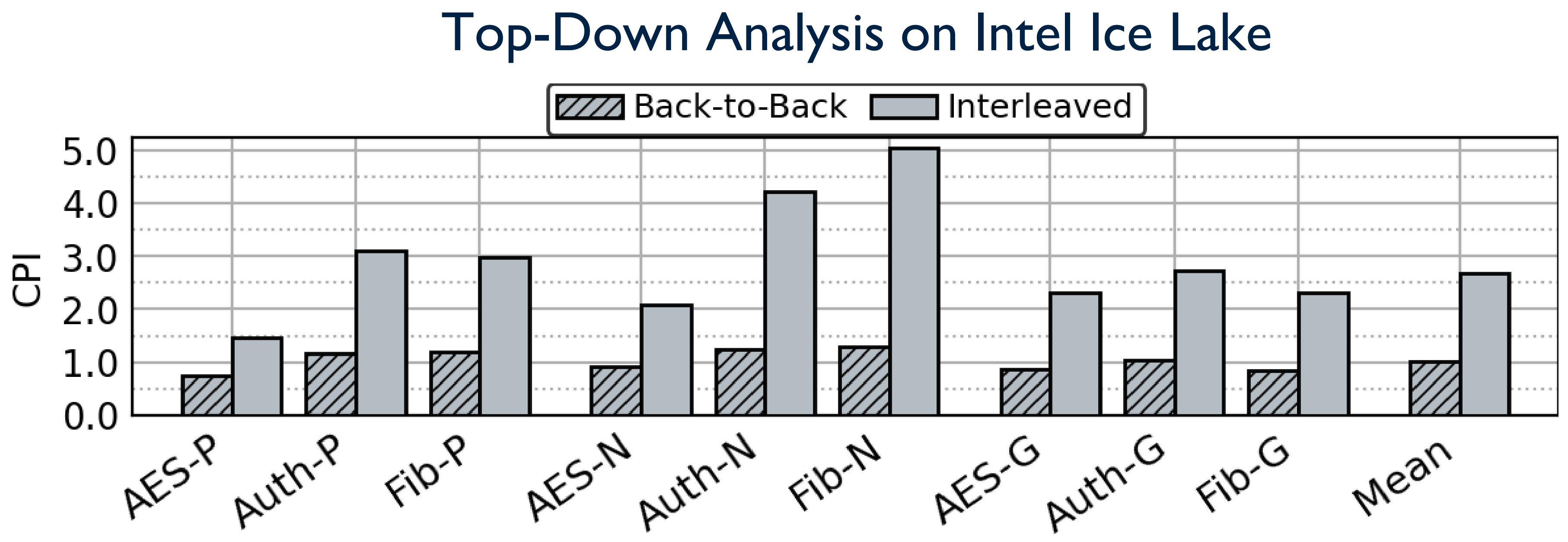
Use Top-down Analysis to study 20 serverless functions

- **Intel Ice Lake Server CPU**
  - 32 cores, two socket, SMT disabled, 32KB L1I, 48KB L1D, 1.25MB L2/core, 54MB LLC
- **Workloads:**
  - 20 diverse serverless functions from vSwarm
  - … on 3 different runtimes: Python, Golang, NodeJS
  - … in 3 main types of programming languages: compiled, interpreted, JIT'ed
- **Compare back-to-back to interleaved execution**
  - Back-to-back: Core repeatedly processes the same function
  - Interleaved: After each invocation of the function, a stressor is used to thrash the on-chip uarch state




Interleaved Execution
Function executing
Stressor models execution of co-located functions
f
…
CPU core execution timeline

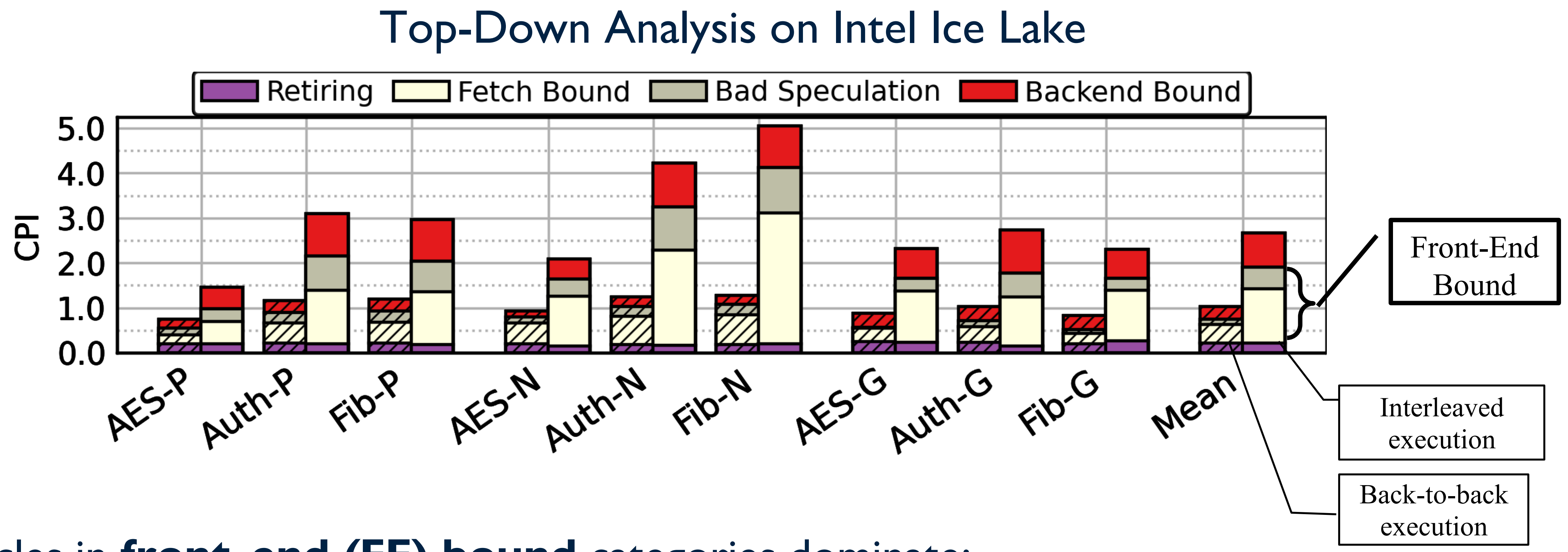
Lukewarm Executions Challenge Modern CPUs



Interleaving drastically increases CPI by 100-294%

- If the microarchitecture is warm, performance is high
  - Caches and the branch predictor leverage high commonality across invocations to boost performance
- The problem is the cold microarchitecture due to interleaving

Lukewarm Executions Challenge Modern CPUs



Stall cycles in **front-end (FE) bound** categories dominate:

- **FE**: **Fetch Bound** (I-cache, I-TLB) + **Bad Speculation** (BTB + branch predictor)
- FE bound stalls collectively increase by 130-490% (215% on average) due to interleaving

Does prior work help with the cold front-end?

Simulation Infrastructure

Full system simulation of a two-node setup in **gem5**:

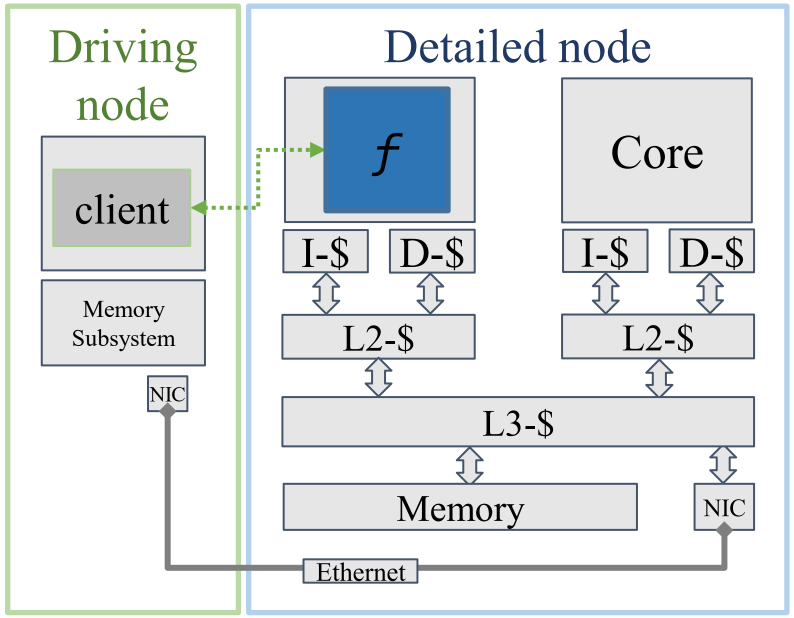
- Detailed **dual-core Ice Lake-like CPU** model
  - 2.4 GHz, 352 entry ROB, 5-way decode
  - Caches: L1-I/D: 32/48KB, : L2:1.25MB/core, L3: 8MB
  - BPU: 64kB LTAGE, 12k entry BTB (After Sapphire Rapids)
- Secondary node for driving invocations

Exact same software stack as on real hardware

- Full end-to-end simulations

Extend gem5 by an industry-standard front-end design

- **Fetch directed prefetch (FDP)**
  - Employed in IBM's z14/z15, Arm's Neoverse, Samsung's Exynos
- Released and in the process of upstreaming to gem5
  - https://github.com/dhschall/gem5-fdp/

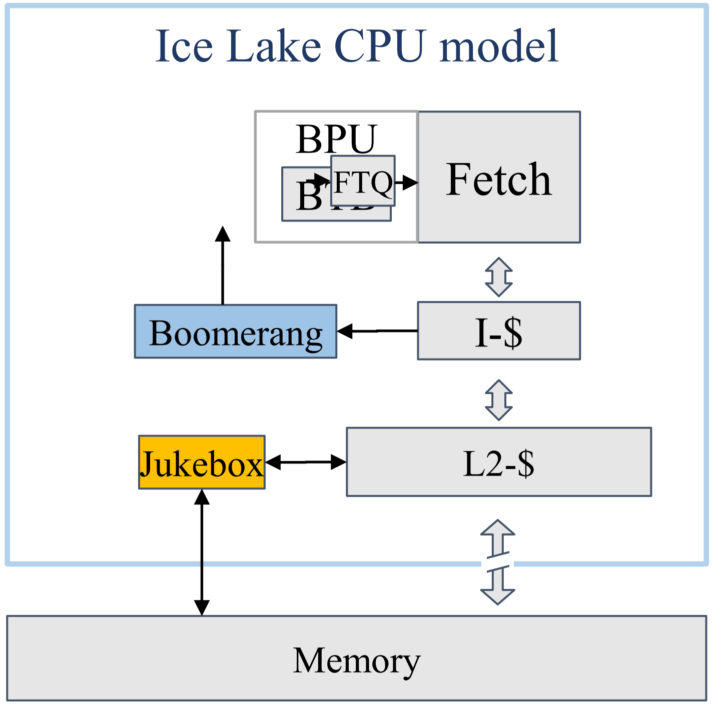


Prior Art in Front-end Mitigation

Baseline: Ice Lake-like CPU model with Next-line instruction prefetcher

Evaluate three different designs:

1. ***Jukebox***: Prior work on lukewarm execution
   - Focus on off-chip instruction misses
   - Record-and-Replay instruction prefetcher
   - Targets L2 cache
2. ***Boomerang***: FDP + BTB prefilling
   - FDP-based instruction prefetching
   - Targets L1 instruction cache
   - + proactive branch target pre-decoding
3. ***Ideal:*** Warm front-end
   - Perfect L1-I and BTB
   - Pre-trained conditional branch predictor



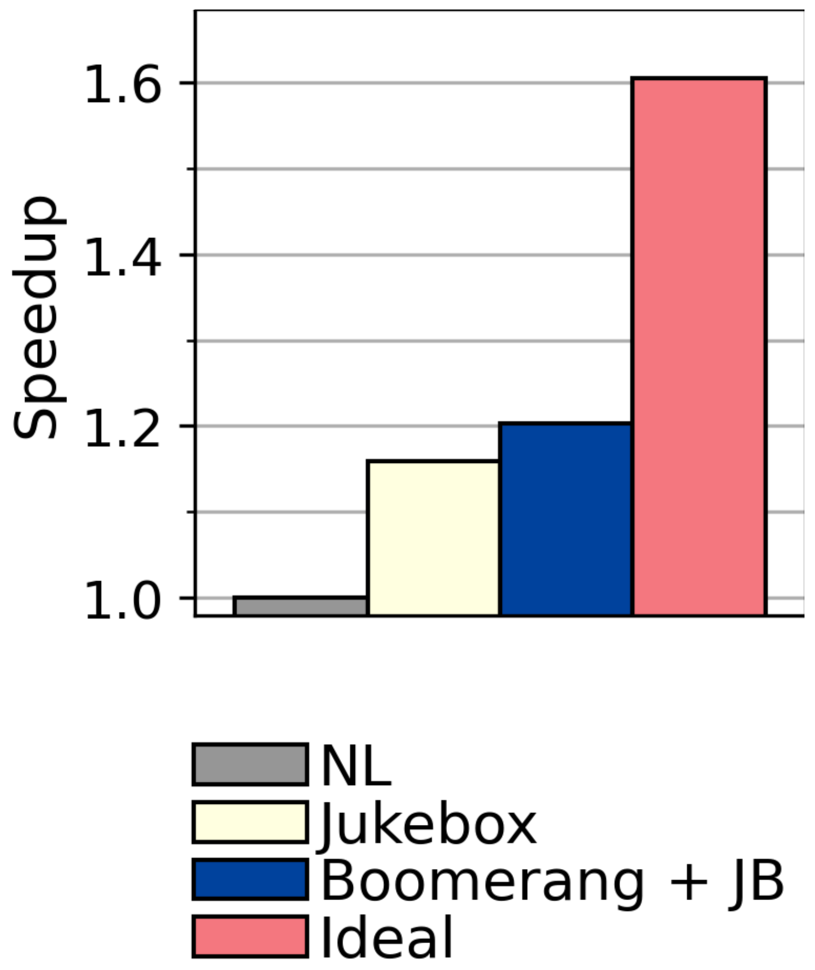
Prior Art in Front-end Mitigation

Prior works fail to address lukewarm executions

- Jukebox mitigates only off-chip instruction misses
- Jukebox + Boomerang is also ineffective. **Why?**

Reason:

- **Cold** microarchitectural state of the **Branch Predictor Unit (BPU)**
  - Short execution times of serverless functions impede BPU training
  - High BTB + CBP MPKI (> 30 MPKI!)



Why is a warm BPU so important?

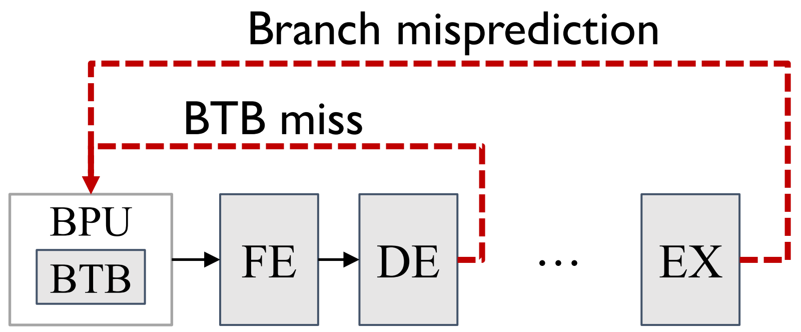
Effect of the Cold Branch Predictor

Front-end cannot stay on the correct path

- Boomerang prefetches instructions and branches from the wrong path
  → Poor miss coverage
- Boomerang has no notion of whether a branch is taken or not
  → **BTB prefetching is ineffective** or even counterproductive

Branch mispredictions are resolved late in the pipeline

→ **Pipeline flush**



The cold BPU limits the efficacy of prior work

Addressing the Cold Front-end

Objectives to address the problem of a cold front-end:

1. **Instructions on-chip**
   - To shield the CPU from long instruction miss latencies
2. **Warm BTB**
   - To allow the front-end detecting control flow
3. **Warm Branch Predictor**
   - To stay on the correct path and avoid pipeline flushes

Question:

Can we get it all with a practical, non-invasive design?

Addressing the Cold Front-end

Observations:

1. **High commonality across invocations**
   - Most instructions and branch executions are the same across invocations
   - → We can record and replay control flow
2. **The BTB working set is a compressed version of a program's control flow**
   - Contains all control flow discontinuities
   - → Can be used for instruction and BTB prefetching
3. **Significant fraction of compulsory branch mispredictions**
   - Large instruction footprint + short execution time → code gets streamed
   - Too few dynamic executions per branch to amortize compulsory miss
   - → We can focus on compulsory misprediction

Addressing the Cold Front-end

Observations:

1. **High commonality across invocations**

2. **The BTB working set is a compressed version of a program’s**

➔ BTB working set can be used to replay the entire control flow!

What about the cold branch predictor?

Objectives:
- ✓ Instructions on-chip
- ✓ Warm BTB
- ✓ Warm Branch Predictor

Insight:

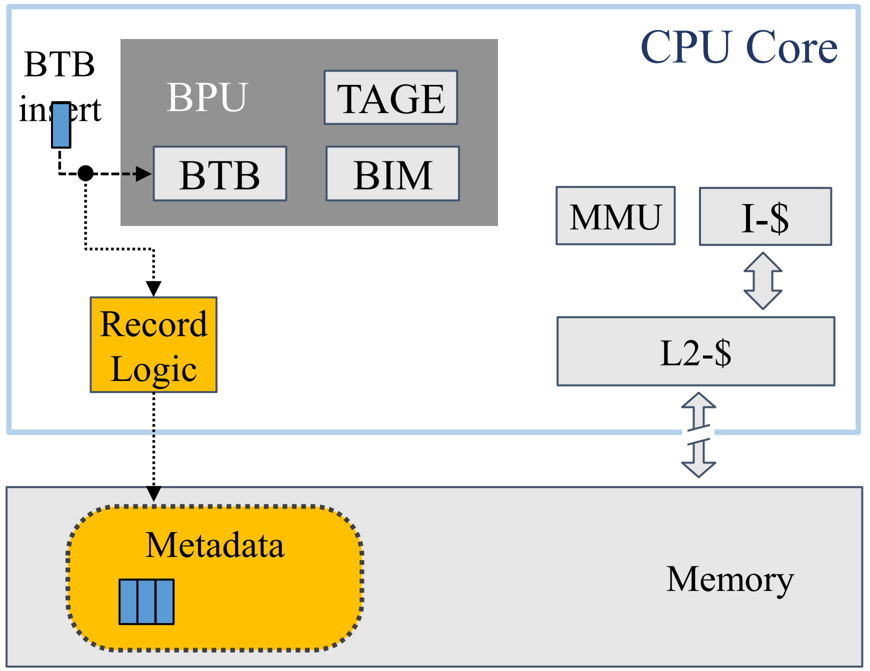
- **A BTB insertion happens only upon a compulsory misprediction**
  - A branch is taken the first time
    → Not present in BTB → gets installed

➔ Replay BTB insertions to restore instruction, branch targets, and branch directions

IGNITE

Comprehensive solution for **restoring** front-end **microarchitectural state**

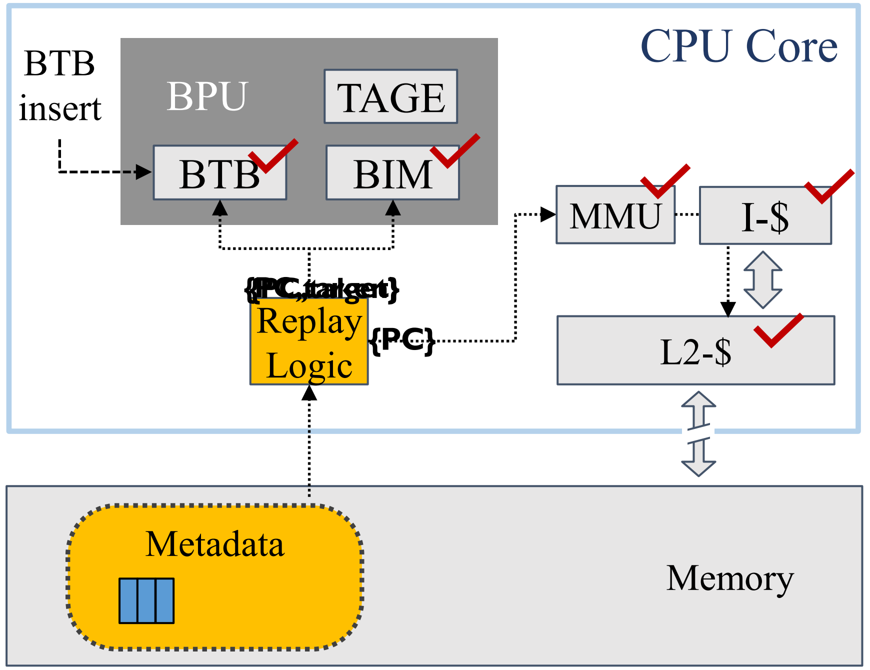
- **Records** *complete* and *non-redundant control flow* graph of one invocation
  - Monitors only BTB insertions
  - Stores trace of BTB insertions as metadata in memory



IGNITE

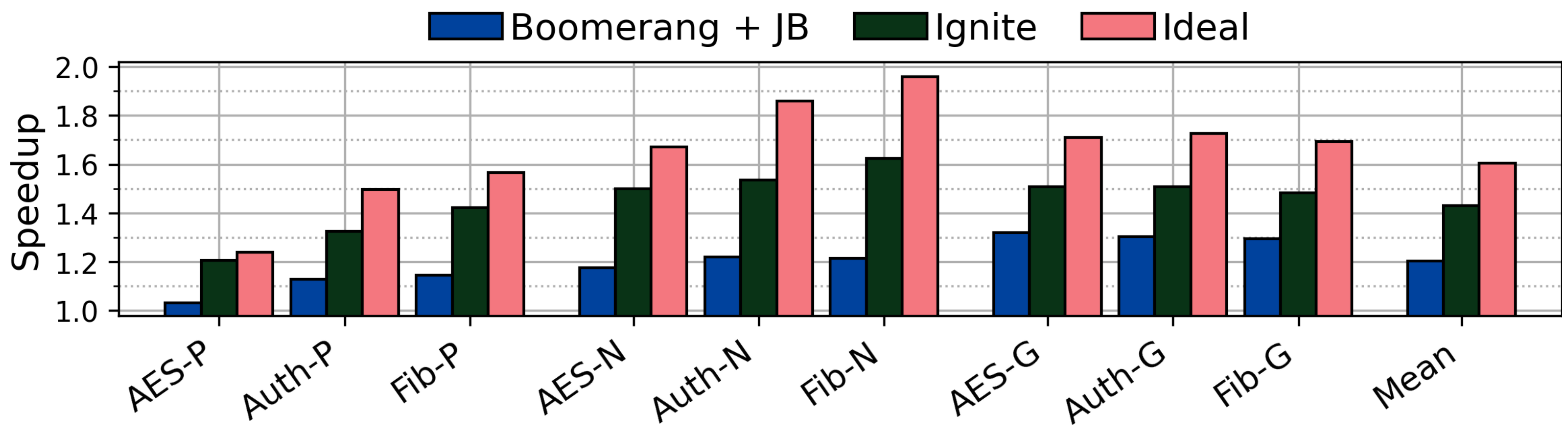
Comprehensive solution for **restoring** front-end **microarchitectural state**

- **Replays** u-arch state upon next invocation of the same function
  - Single, *unified* metadata is used to:
    1. Restore BTB entries
    2. Initializes the Bimodal predictor
    3. Prefetch instructions on-chip into L2
       - Populates the I-TLB as a by-product
- **Integrates with FDP**
  - The warmed-up BPU effectively prefetches instructions into L1
- **Fully decoupled** from the core
  - Triggered by function invocation



Simple, non-invasive design with unified metadata

IGNITE: Performance



Ignites comprehensive state restoration

- Covers > 80% BTB misses and > 50% L1I misses
- Reduces branch mispredictions by > 60%

  → This translates to a 43% speedup over next-line
- Captures the bulk of the opportunity at low design complexity

Ignite is a simple and effective

Takeaways

Serverless functions present new challenges for modern CPUs

- **Lukewarm execution:** cold u-arch state due to heavy function interleaving

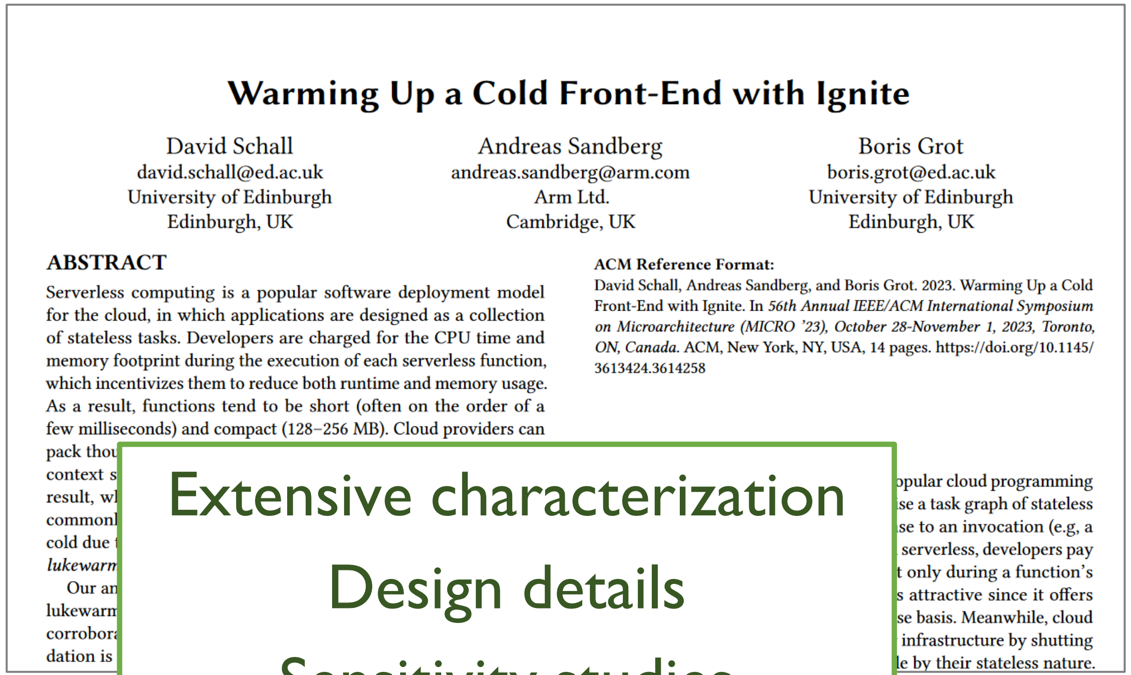
Analysis shows severe front-end bottlenecks due to lukewarm execution

- L1 instruction misses, BTB misses and branch mispredictions
- **Cold Branch Predictor Unit** limits the efficacy of prior work
- A solution for cold front-end must comprise:
  - **instructions, branch targets** and **branch directions**

Ignite: Comprehensive microarchitectural state restoration for the CPU front-end

- Single, unified metadata for instructions, branch targets and branch directions
- Simple and effective solution for the cold front-end
- Speeds up interleaved serverless function executions by 43%

Thank you!



Warming Up a Cold Front-End with Ignite

David Schall
david.schall@ed.ac.uk
University of Edinburgh
Edinburgh, UK

Andreas Sandberg
andreas.sandberg@arm.com
Arm Ltd.
Cambridge, UK

Boris Grot
boris.grot@ed.ac.uk
University of Edinburgh
Edinburgh, UK

ABSTRACT

Serverless computing is a popular software deployment model for the cloud, in which applications are designed as a collection of stateless tasks. Developers are charged for the CPU time and memory footprint during the execution of each serverless function, which incentivizes them to reduce both runtime and memory usage. As a result, functions tend to be short (often on the order of a few milliseconds) and compact (128–256 MB). Cloud providers can pack thou
context s
result, wh
commonl
cold due t
lukewarm
Our an
lukewarm
corrobora
dation is

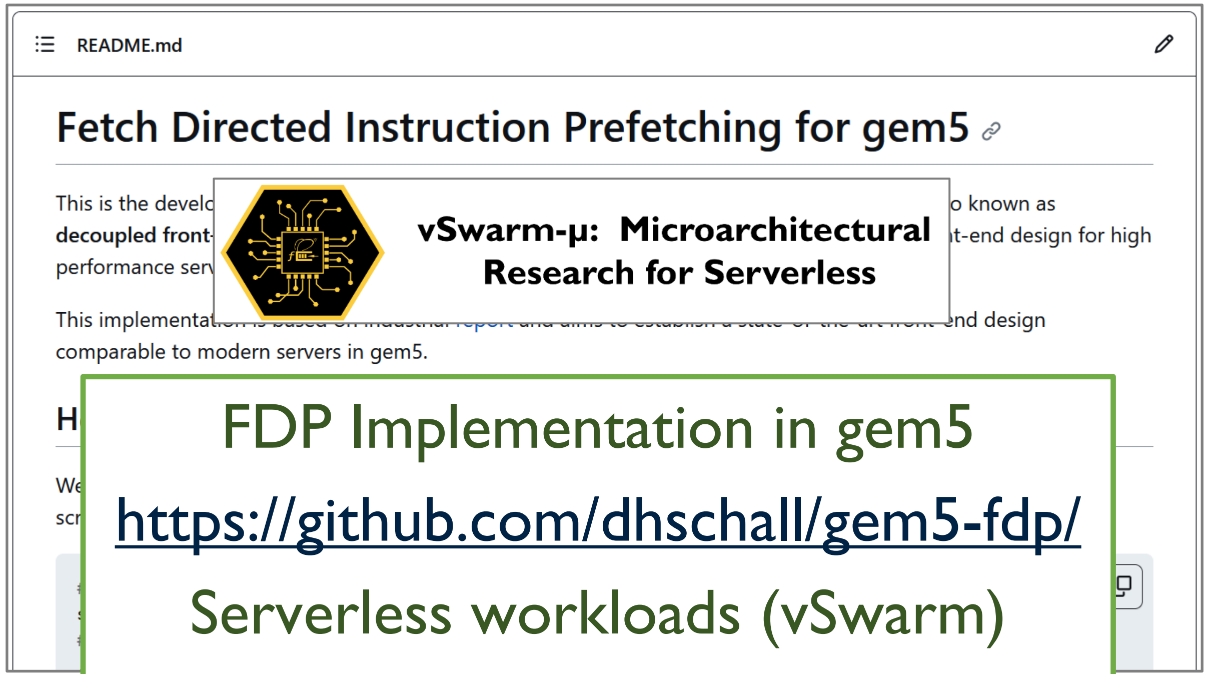
ACM Reference Format:
David Schall, Andreas Sandberg, and Boris Grot. 2023. Warming Up a Cold Front-End with Ignite. In *56th Annual IEEE/ACM International Symposium on Microarchitecture (MICRO '23), October 28–November 1, 2023, Toronto, ON, Canada.* ACM, New York, NY, USA, 14 pages. https://doi.org/10.1145/3613424.3614258

opular cloud programming
se a task graph of stateless
se to an invocation (e.g., a
serverless, developers pay
t only during a function's
s attractive since it offers
se basis. Meanwhile, cloud
infrastructure by shutting
le by their stateless nature.

Extensive characterization

Design details

Sensitivity studies



README.md

Fetch Directed Instruction Prefetching for gem5

This is the develo… o known as **decoupled front-**… t-end design for high performance serv…

vSwarm-μ: Microarchitectural Research for Serverless

This implementati… …nd design comparable to modern servers in gem5.

H

We
scr

FDP Implementation in gem5

https://github.com/dhschall/gem5-fdp/

Serverless workloads (vSwarm)

gem5 infrastructure (vSwarm-u):

https://github.com/vhive-serverless

arm