Hardware-Software Co-Designed Cache Bypass Mechanism on X86 Machine

EECS 583 Fall 2020

Group 11

Yichen Yang, Wentao Zhang, Wenqi Zhu, Zhiyi Pan

Motivation

Development of Hardware

↓

High Performance Computer with multiple processors

↓

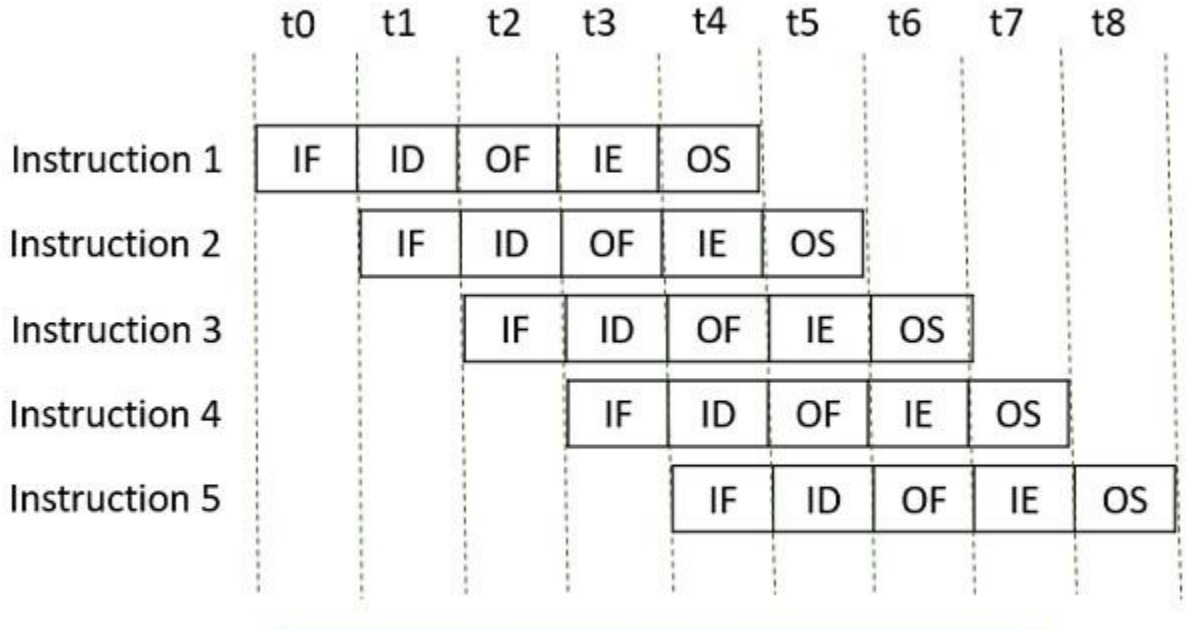
Fast instruction Execution

↓

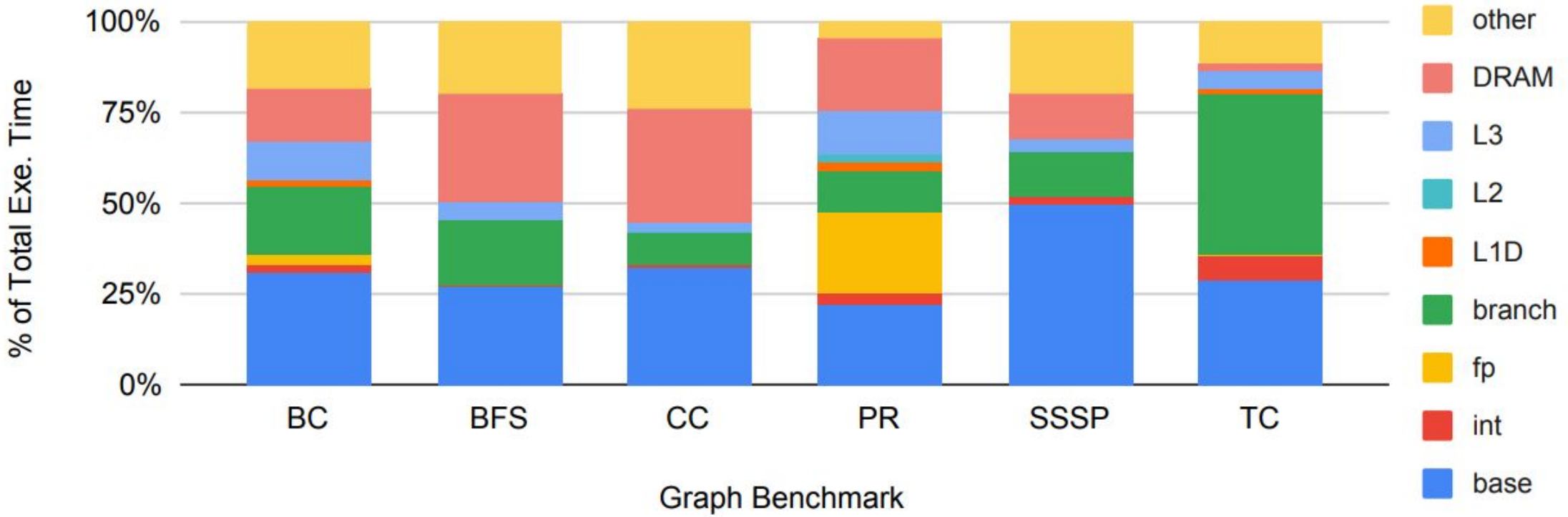
Low Memory Access Speed

↓

Cache



Motivation

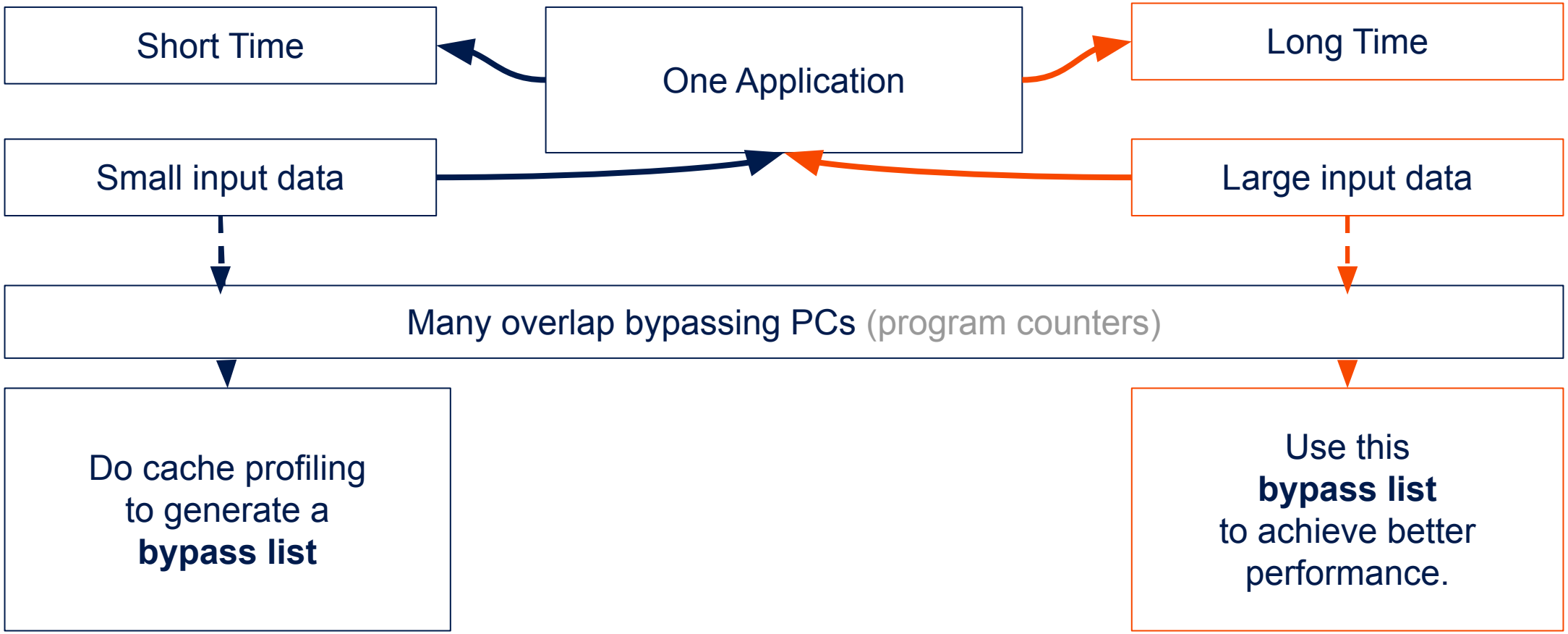


Percentage of total execution time spent on different part of the workload.

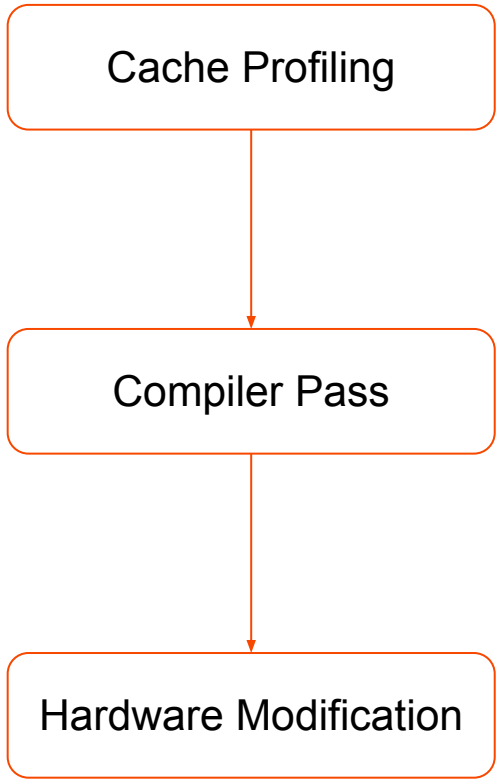
Motivation & Introduction

Prj:	Hardware-Software Co-Designed	Cache Bypass Mechanism	on X86 Machine
	• Cache profiling & compiler mark Inst. with bypass labels • ISA level modification	• Performance are bottle-necked by memory • Not cache infrequently used data	• **Previous work:** EPIC (in order) • **Modern Processors:** out-of-order (e.g. Intel X86)

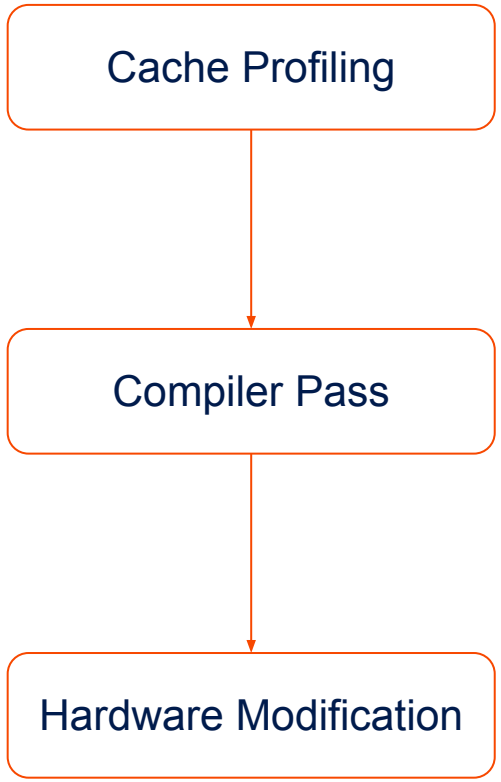
Background


Short Time
One Application
Long Time
Small input data
Large input data
Many overlap bypassing PCs (program counters)
Do cache profiling to generate a **bypass list**
Use this **bypass list** to achieve better performance.

Design

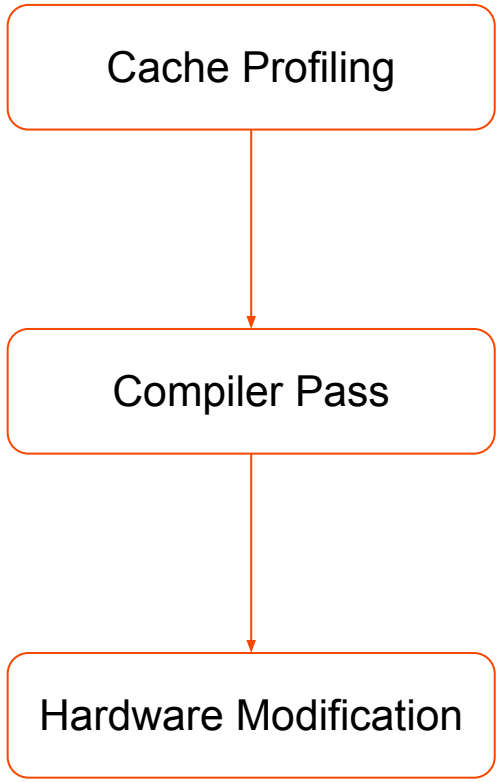


Design



- Profile Cache Performance
- Format:
  hit/miss rate per PC
  hit/miss rate per data Addr

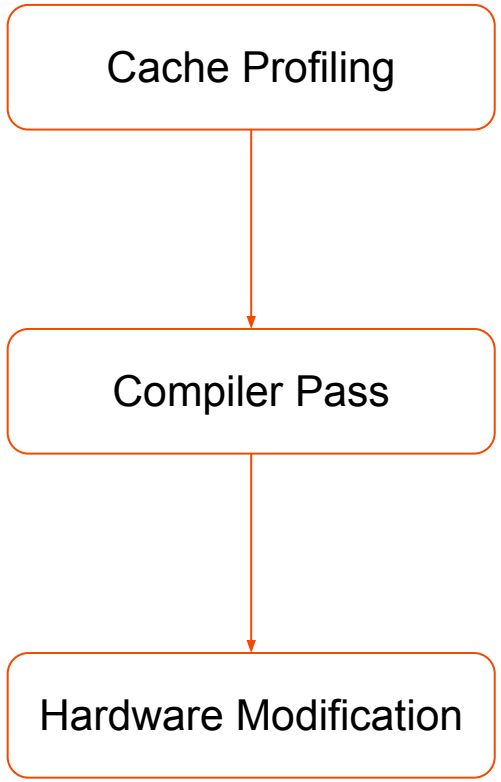
Design



- Profile Cache Performance
- Format:
  hit/miss rate per PC
  hit/miss rate per data Addr

- Analyze cache stats and select load insts to do bypass $mnrp = \frac{number\_misses\_no\_reuse}{number\_loads\_executed}$
- Algorithms:
  - Naive Miss Rate: insts that have miss rate higher than the threshold
  - Top % Miss Rate: insts that have top X% of miss rate
  - Reuse Probability: *mnrp* larger than the threshold

Design

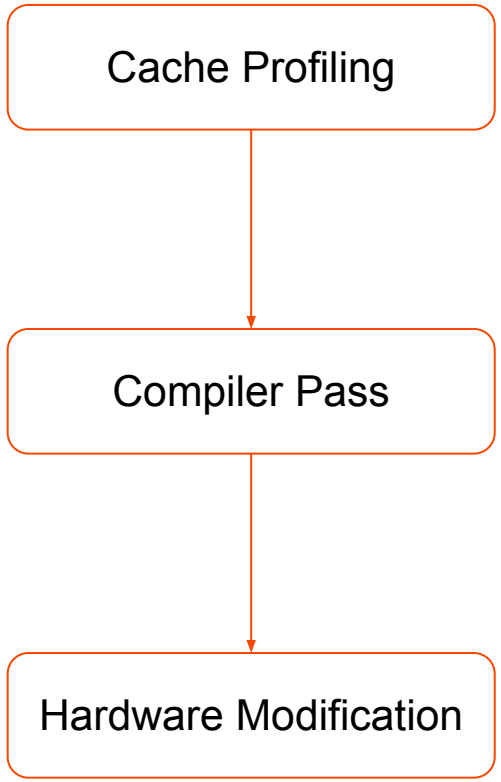


- Profile Cache Performance
- Format:
  hit/miss rate per PC
  hit/miss rate per data Addr

- Analyze cache stats and select load insts to do bypass $mnrp = \frac{number\_misses\_no\_reuse}{number\_loads\_executed}$
- Algorithms:
  - Naive Miss Rate: insts that have miss rate higher than the threshold
  - Top % Miss Rate: insts that have top X% of miss rate
  - Reuse Probability: *mnrp* larger than the threshold

- ISA level modification
  - Add 1 bit indicating bypass or not to load inst
- CPU core should issues the memory request to corresponding port

Implementation

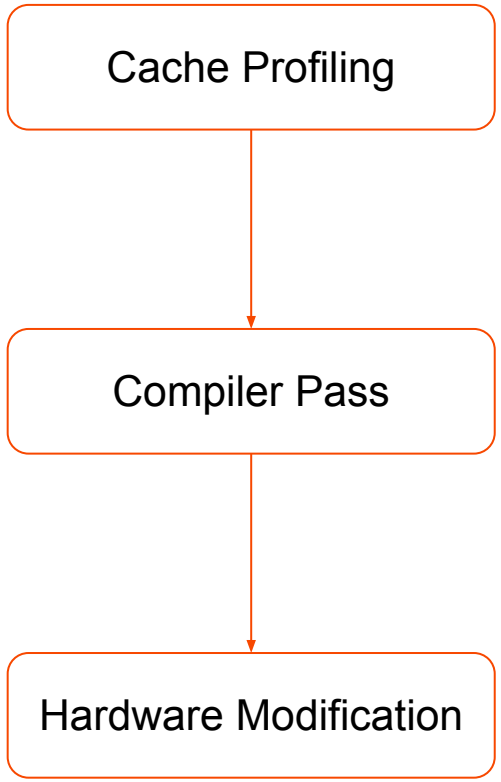


- Profile Cache Performance
- Format:
  hit/miss rate per PC
  hit/miss rate per data Addr

- Analyze cache stats and select load insts to do bypass
- Algorithms:
  - Naive Miss Rate: insts that have miss rate higher than the threshold
  - Top % Miss Rate: insts that have top X% of miss rate
  - Reuse Probability: *mnrp* larger than the threshold

- ISA level modification
  - Add 1 bit indicating bypass or not to load inst
- CPU core should issues the memory request to corresponding port

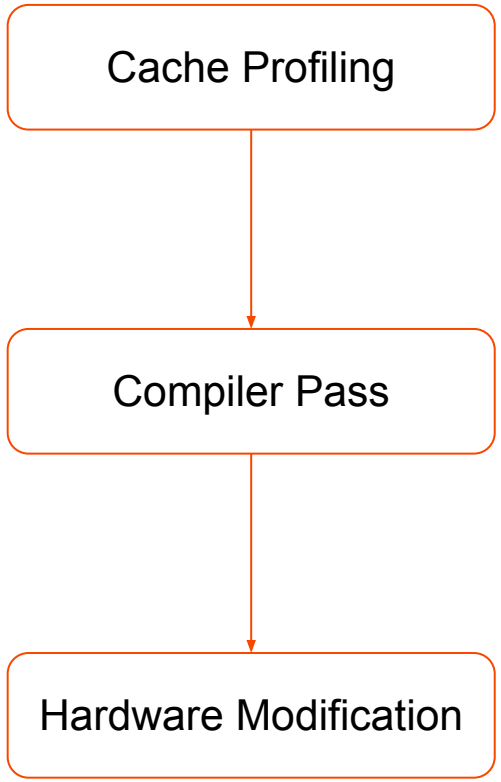
Implementation



- Dynamorio cache simulator
- Dump offline memory access trace
  - Format: PC, hit/miss, target data Addr

- Analyze cache stats and select load insts to do bypass
- Algorithms:
  - Naive Miss Rate: insts that have miss rate higher than the threshold
  - Top % Miss Rate: insts that have top X% of miss rate
  - Reuse Probability: *mnrp* larger than the threshold
- ISA level modification
  - Add 1 bit indicating bypass or not to load inst
- CPU core should issues the memory request to corresponding port

Implementation

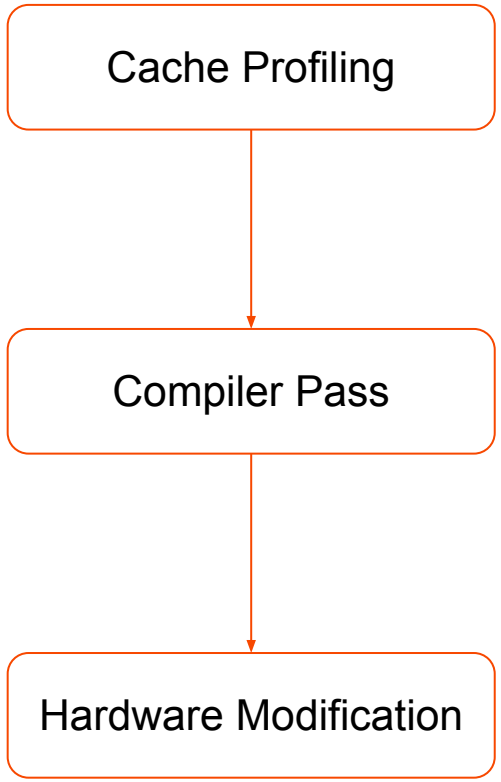


- Dynamorio cache simulator
- Dump offline memory access trace
  - Format: PC, hit/miss, target data Addr

- Implemented in Python
- Multiple bypass instruction selection algorithm
- Generate Bypass-List to a file
  - Common Bypass-List is used in the evaluation

- ISA level modification
  - Add 1 bit indicating bypass or not to load inst
- CPU core should issues the memory request to corresponding port

Implementation



- Dynamorio cache simulator
- Dump offline memory access trace
  - Format: PC, hit/miss, target data Addr

- Implemented in Python
- Multiple bypass instruction selection algorithm
- Generate Bypass-List to a file
  - Common Bypass-List is used in the evaluation

- Modify Dynamorio cache simulator
- Read Bypass-List and perform cache bypass
- Avoid ASLR (Address Space Layout Randomization)

Methodology

- Benchmark
  - gapbs: a graph benchmark suite
  - Breadth-First Search (BFS)

- Dataset from SNAP

Dataset	#Node	#Edges	Category
email-Eu-core (email)	1K	25.5K	Train/Test
p2p-Gnutella08 (p2p)	6.3K	20.7K	Train
wiki-Vote (vote)	7.1K	104K	Train
ca-GR-QC (qc)	5.2K	12.5K	Test
Slashdot-09 (slash)	82K	948K	Test
Pokec (pk)	1.6M	30M	Test

- Simulated cache specification

L1 Size	32KB
L1 Assoc	8
L2 Size	8MB
L2 Assoc	16
Cache Line Size	64

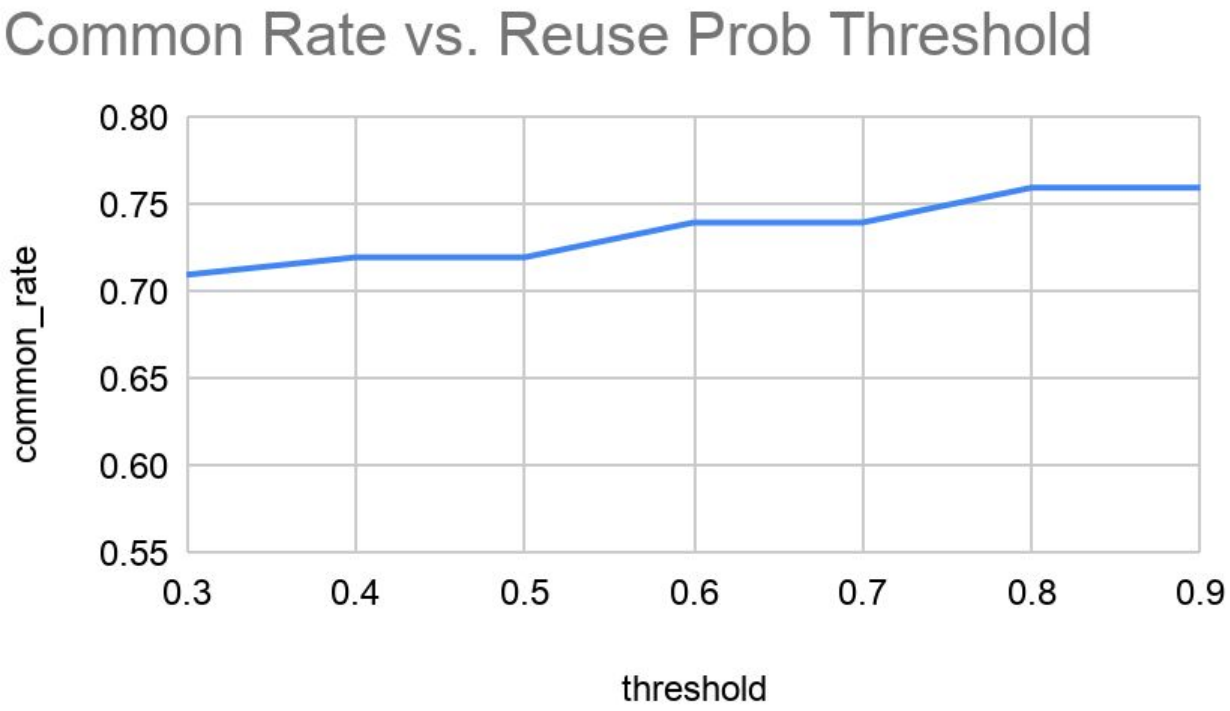
DEMO

Evaluation

- Bypass Lists Similarity
- Algorithm Evaluation
- Reuse Algorithm Evaluation
  - Cache Setup
  - Data Set Size
- Interpreted Performance Gain

Evaluation - Bypass Lists Similarity

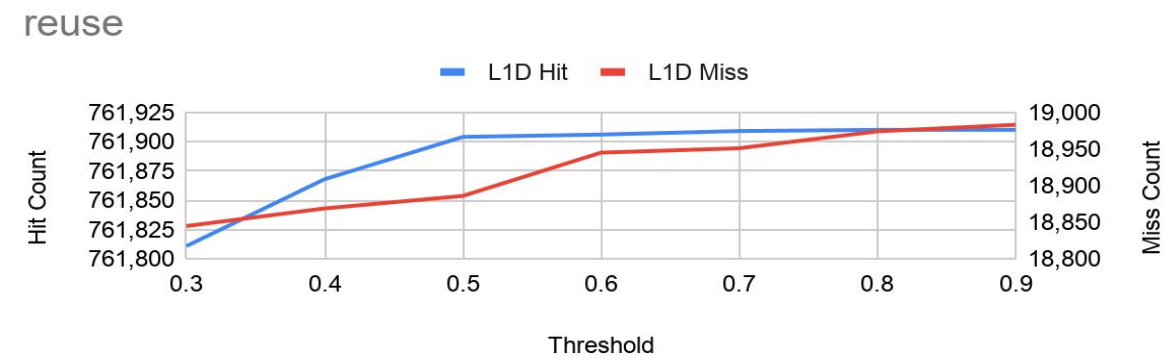
- Common rate = intersection(PCs) / Union (PCs)
- Average Common Rate: 73%

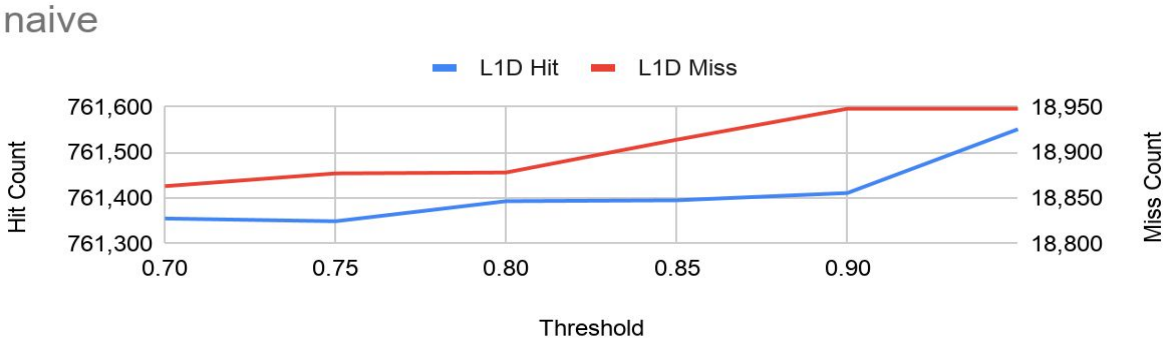


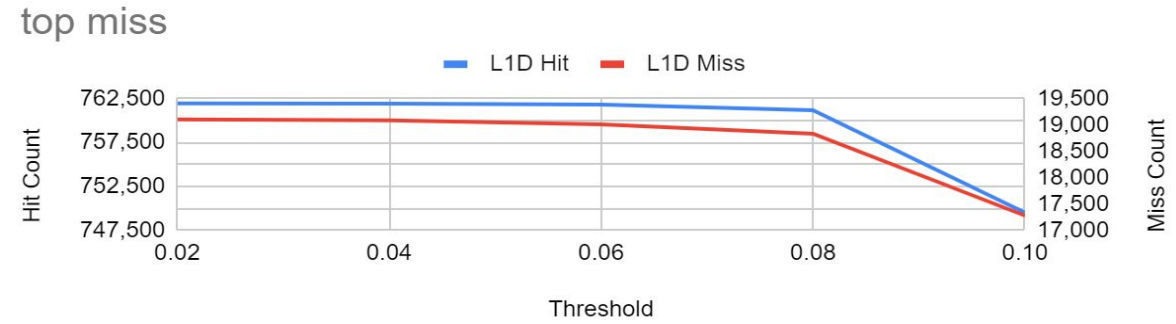
Evaluation - Algorithm

- Reuse Probability Algorithm has the best performance with threshold to be 0.5
- Both Naive and Top Miss Algorithm failed to provide the ideal miss reduction
- Possible Reason:

  Bypass one PC if its miss rate passes threshold. However, this may introduce more misses, since that cache line can be reused by other PCs.

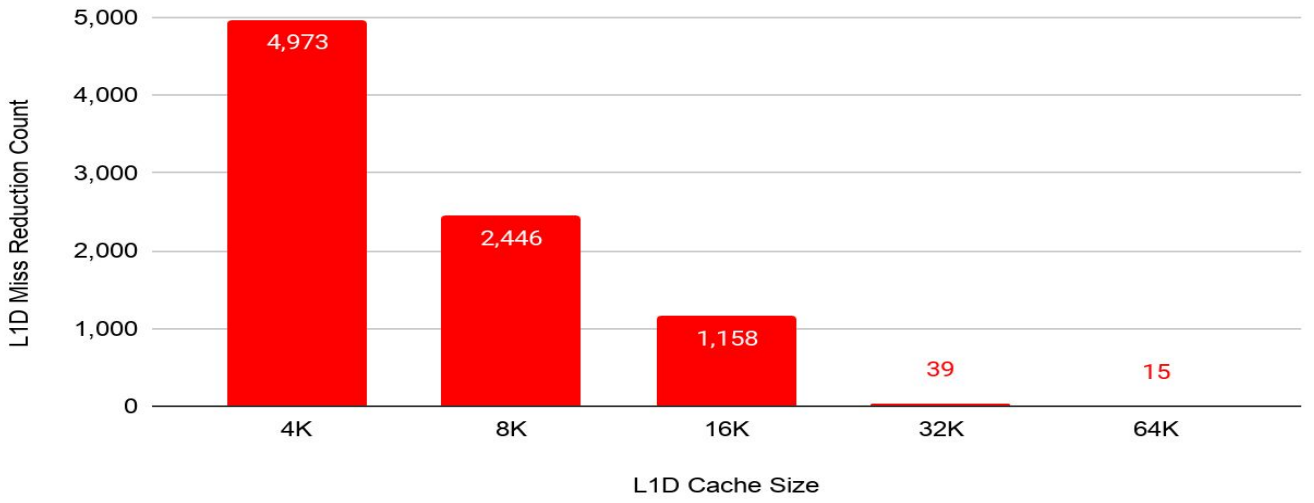


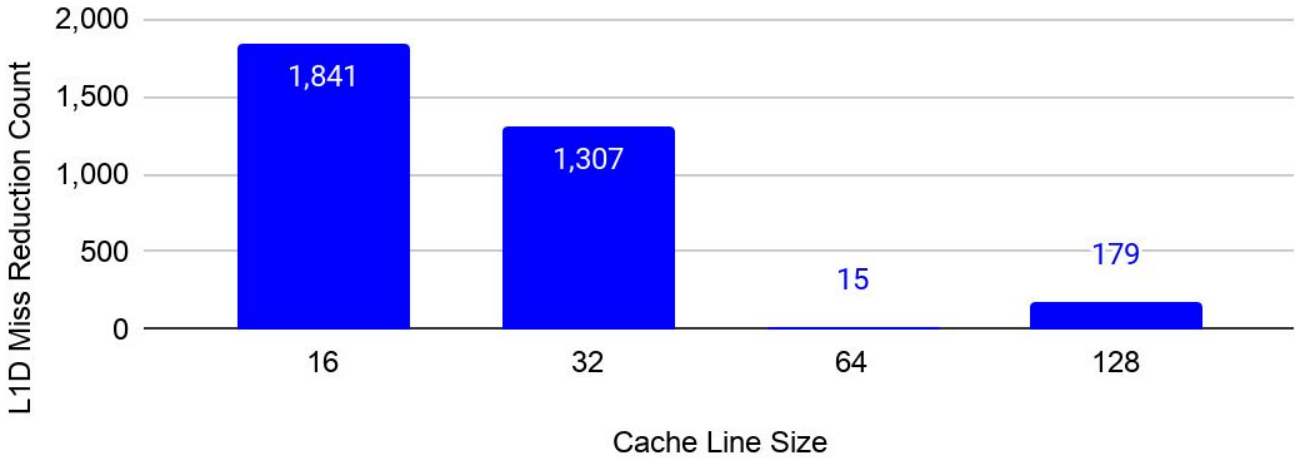




Evaluation - Cache Performance (Reuse)

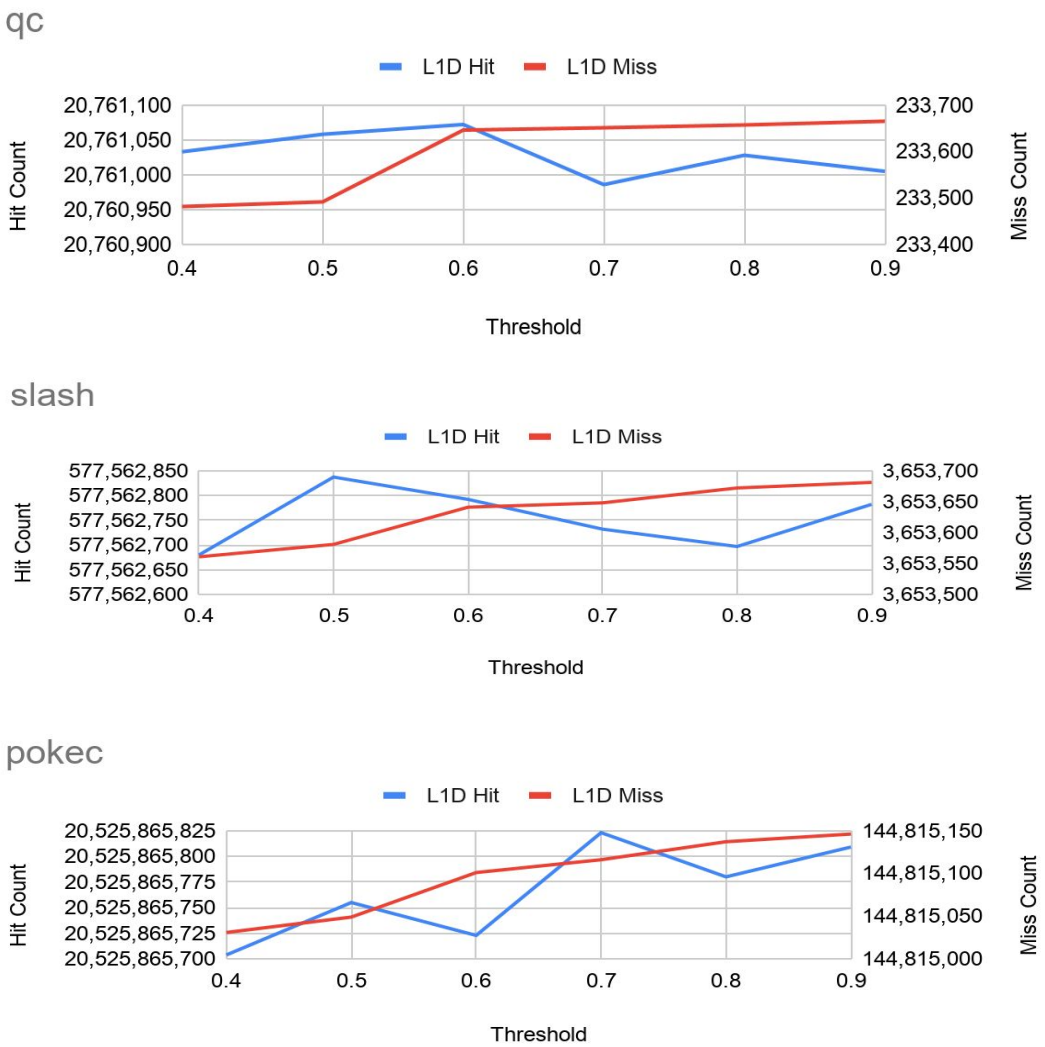
- Analyze the influence of cache size and cache line size
- The L1D miss without bypassing is 513K when cache size is 4k





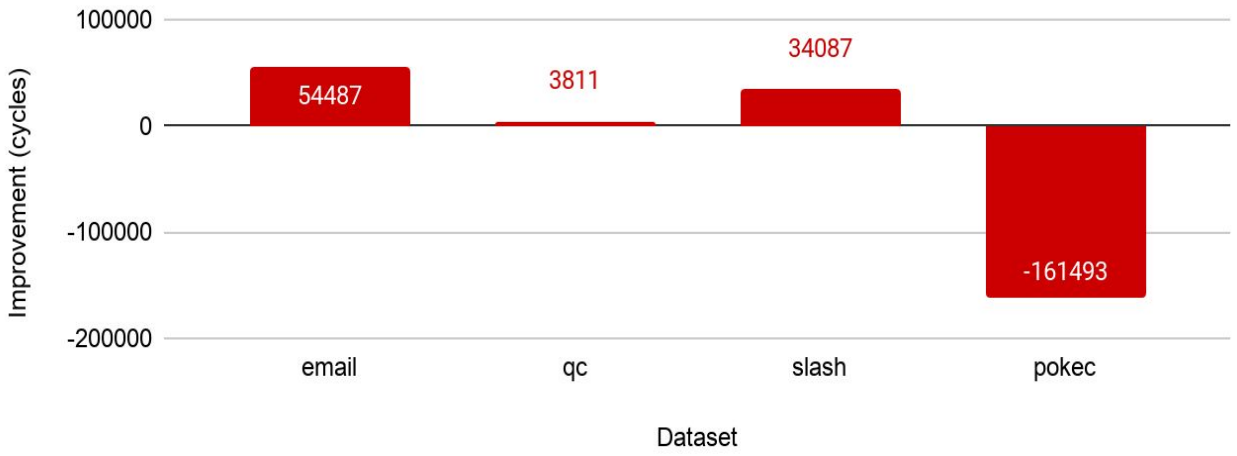
Evaluation - Data Set Size (Reuse)

- data set size (#Nodes): 5.2K, 82K, 1.6M
- bypass algorithm fails to provide ideal performance on large data set
- Possible Reason:
  - Less bypass opportunity
  - Increased number of LL miss



Evaluation - Interpreted Perf. Gain (Reuse)

- Set up
  - Intel i7-4770
  - L1 Data cache = 32 KB, 64 B/line, 8-WAY,
  - LL cache = 8 MB, 64 B/line
- Introduced performance lost on large data set (pokec)



Limitations and Future Works

- Not perform well on large dataset.
  - Possible reason: Data-access pattern on large dataset is different from that on small dataset.
  - Future work: Investigate data-access pattern in large dataset.

- A single iteration may not suffice for global optimal.
  - One iteration generates new miss!
  - Future work: Multiple iterations might finally converge to an optimal.

- Current method bypass the whole cache.
  - What if the load is never reused in L1 but frequently reused in L2?
  - Future work
    - Consider both L1D and LLC hit/miss rate.
    - Bypass selected layer of cache.

Conclusion

- What do we build?
  - An end-to-end tool chain to investigate the HW-SW co-designed cache bypass opportunity on X86 machine

- What is the result?
  - Reuse prob algorithm outperforms the others.
  - Works better on small cache size and small cache line size.
  - Limited benefit on large input dataset.


UNIVERSITY OF
MICHIGAN