Modern Front-end Support in gem5

Bhargav Reddy Godala, Nayana Prasad Nagendra, Ishita Chaturvedi, Simone Campanoni, David I. August

PRINCETON UNIVERSITY

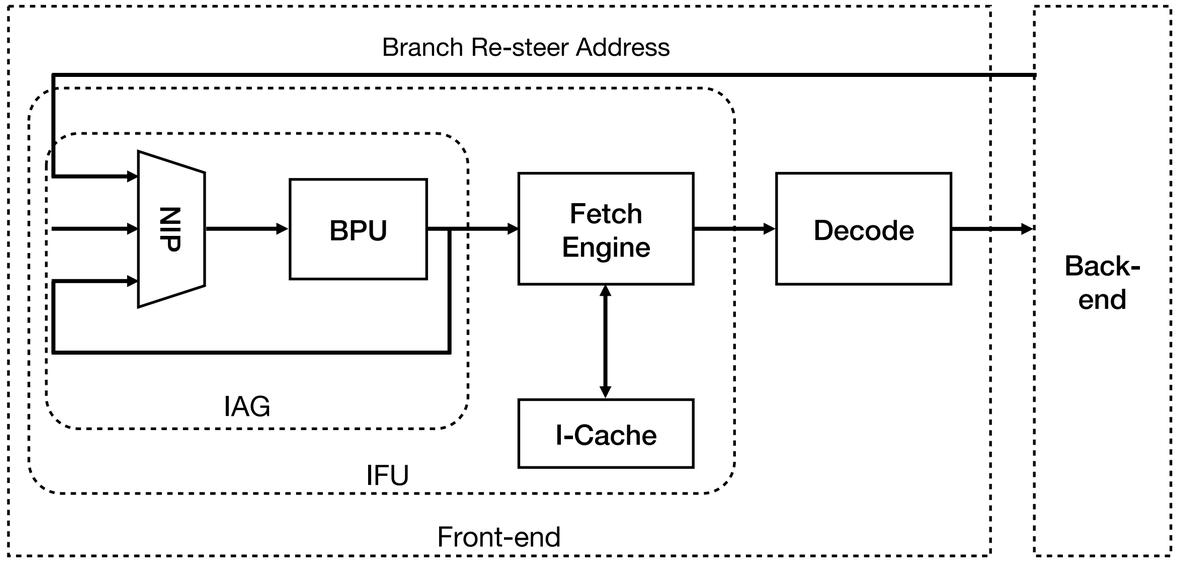
Liberty Research Group

Arcana Research Group

Introduction

- We have seen that aggressive Out-of-Order CPUs tolerate data miss latency.
- Modern CPUs employ decoupled front-end to tolerate instruction miss latency.
- What is a decoupled front-end?

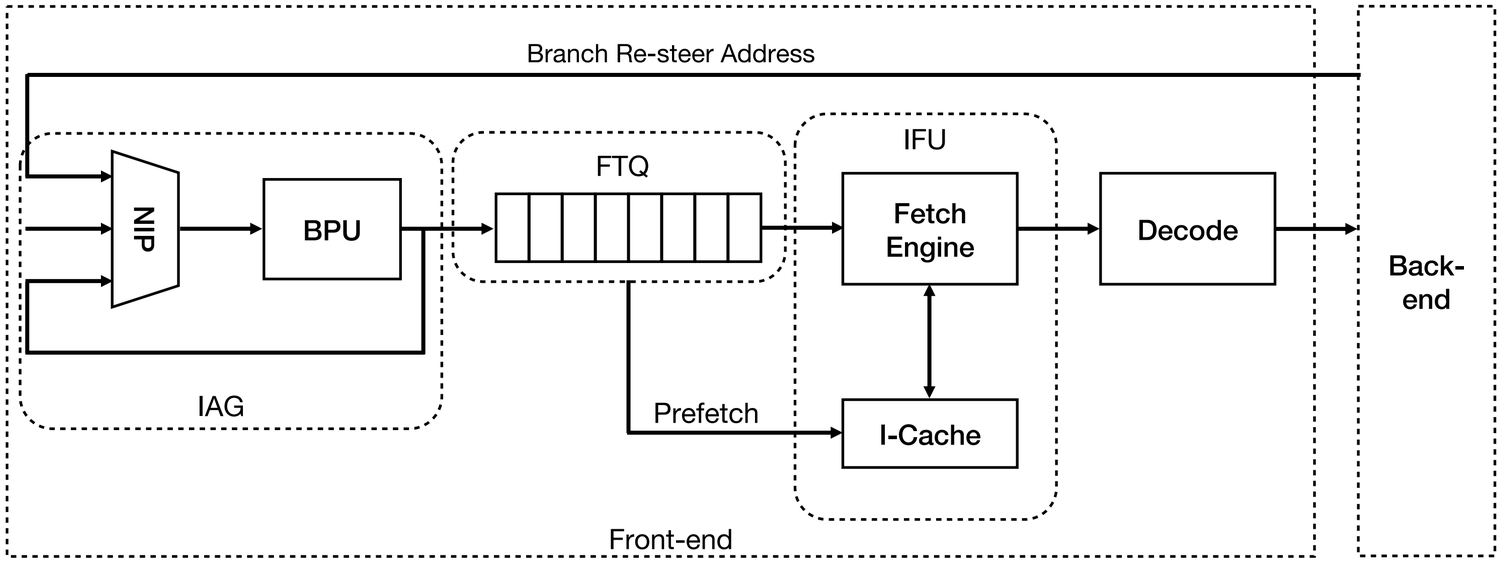
State-of-Art Front-end



Traditional Front-end

FTQ: Fetch Target Queue
IFU: Instruction Fetch Unit
BPU: Branch Prediction Unit
IAG: Instruction Address Generation
NIP: Next Instruction Pointer

State-of-Art Front-end



Fetch Directed Instruction Prefetching Pipeline (FDIP) [Glenn Reinman et al., MICRO'99]

FTQ: Fetch Target Queue
IFU: Instruction Fetch Unit
BPU: Branch Prediction Unit
IAG: Instruction Address Generation
NIP: Next Instruction Pointer

Key Idea: Prefetch in the predicted path

Design

Challenges in Implementing FDIP in gem5

- Fetch stage is already complex.
- Dynamic Instruction objects are constructed before BPU is invoked.
  - Branch Instruction is needed to invoke BPU.
- Sequence numbers are used to squash mis-speculated instructions.

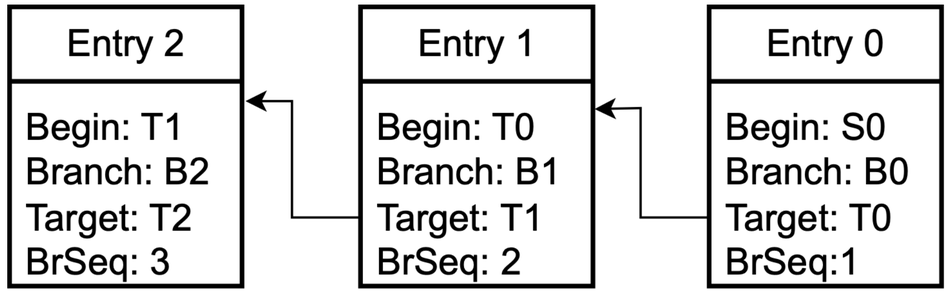
Branch Sequence Numbers

- Unique sequence number to identify branch.
- Every dynamic instruction contains:
  - A sequence number
  - Branch Sequence of prior branch

BrSeq Seq	Seq	Instruction
10	100	Br
10	101	I0
10	102	I1
10	103	I2
11	104	Br
11	105	I3
11	106	I4
11	107	I5

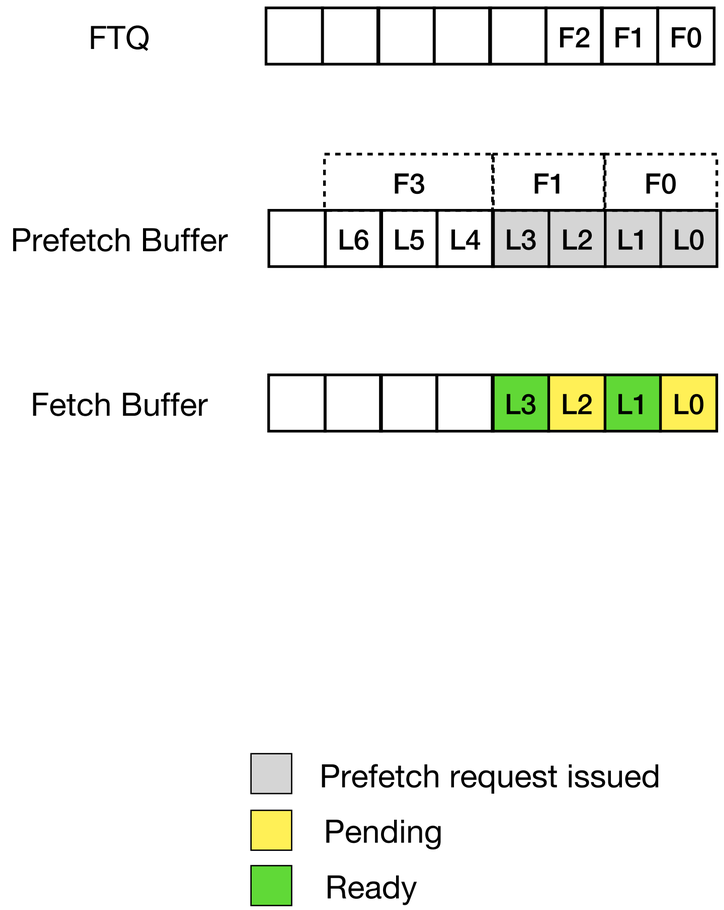
Fetch Target Queue (FTQ)

- Each entry consists of:
  - A begin address (target of prior branch)
  - End address (branch PC)
  - Target address
  - Branch Sequence number

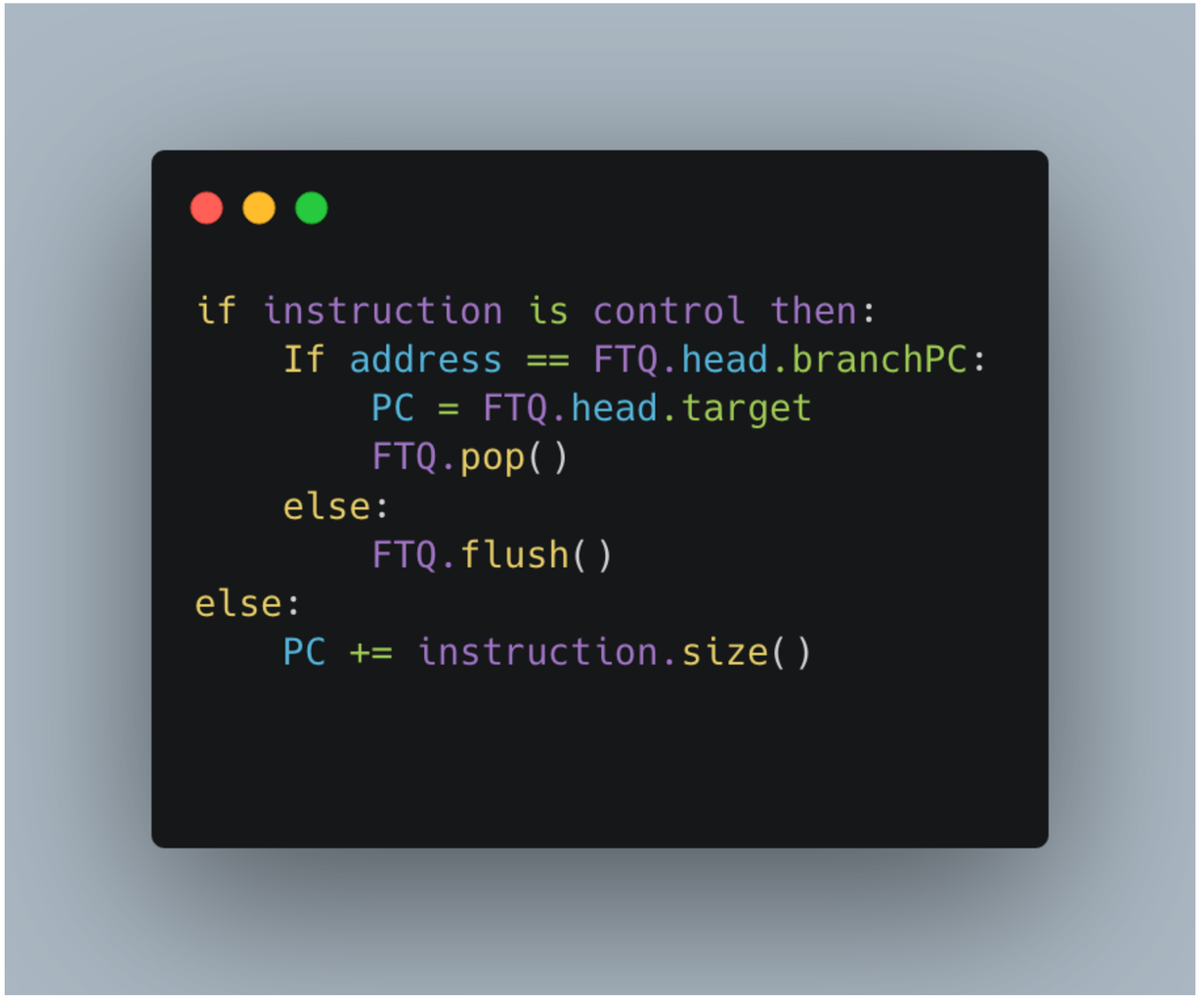


Prefetch Engine

- Prefetch Buffer:
  - Address to prefetch
  - Issue one prefetch and insert into Fetch Buffer



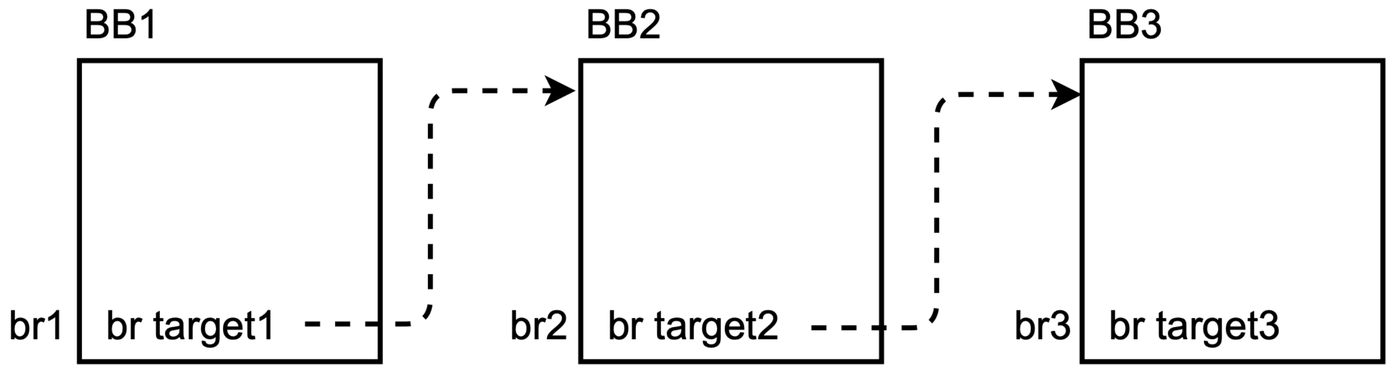
Modified Fetch Stage



```
if instruction is control then:
    If address == FTQ.head.branchPC:
        PC = FTQ.head.target
        FTQ.pop()
    else:
        FTQ.flush()
else:
    PC += instruction.size()
```

Optimizations

Basic Block Based BTB



Index	Traget
br1	target1
br2	target2
br3	Target3

PC based BTB

Index	Target	Branch
target1	target2	br2
target2	target3	br3

BBL based BTB

Pre-decode And Early Correction

- BBL BTB are indexed using beginning of a basic block.
- Beginning of a basic block is identified:
  - Using the next instruction following a branch instruction.
- Early Correction:
  - When an unconditional branch is predicted not taken.
  - Flush FTQ and restart by using the pre-decoded target.

Branch Predictor Changes

- BBL Based Branch Predictor lookup.
- Branch Sequence numbers.
- ITTAGE indirect predictor support.

X86 vs ARM

- X86:
  - Variable width instructions
  - Pre-decoding is very expensive
  - Micro Sequenced Ops
  - Exception handling using ROM
- ARM:
  - Fixed width instructions
  - Pre-decoding is not expensive

Micro Branches in X86

- In X86 there are instructions which are dynamically decoded to loops.
  - Example: String copy
- These branches are not inserted into BTB.
- This is handled as a special case:
  - These are not seen by the FDIP pipeline.
  - At the time of fetch; a back edge is predicted taken.
  - FTQ will not be flushed till a squash from later stages is received.

Performance Bug Fixes

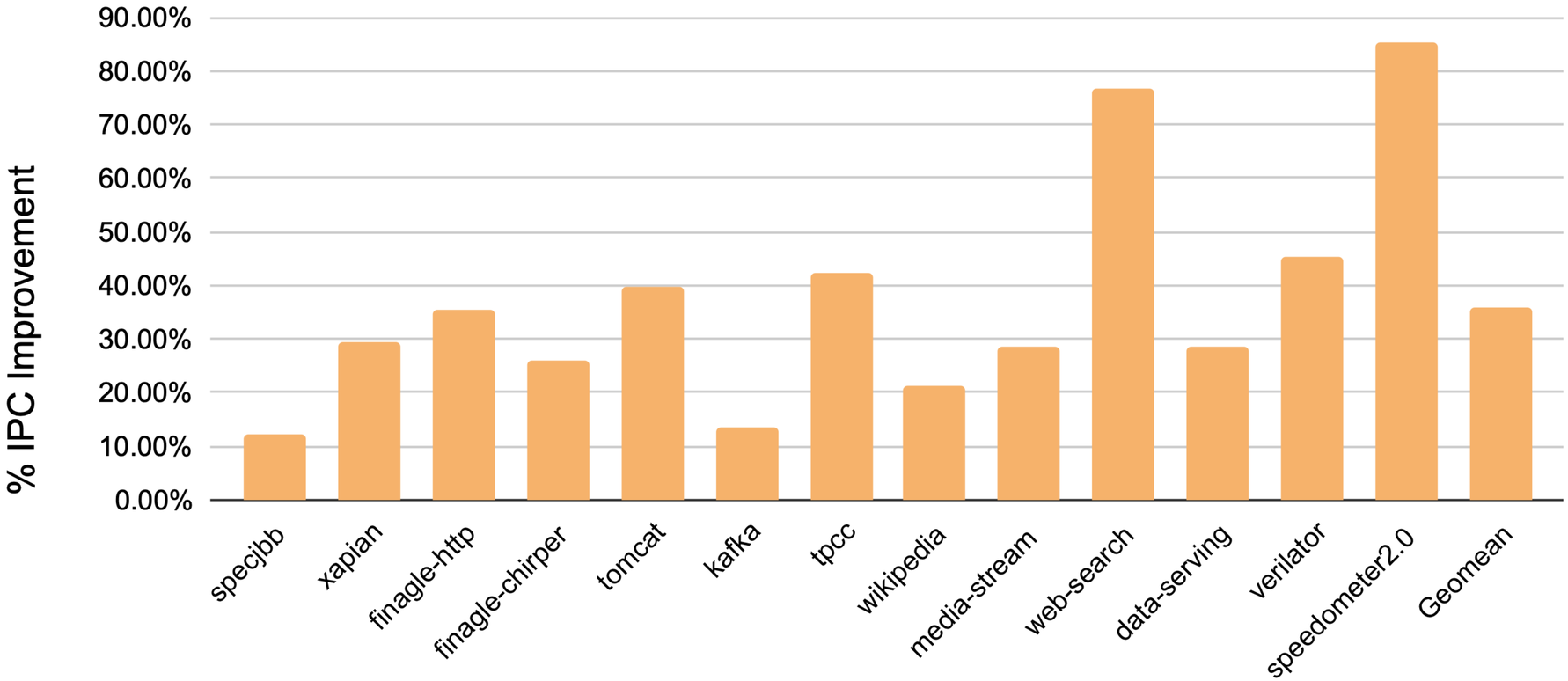
- Perfect recovery of branch history.
- TAGE Bimodal table roll back.

Evaluation

Performance of ARM workloads with FDIP

Field	Alderlake like
ISA	ARM 64-bit
L1I	32KB
L1D	64KB
L2	1MB (16-way)
L3	2MB
FTQ	24 entry 192 inst
Width	8-wide
ROB Size	512 entries
IQ/LQ/SQ	240/128/72
BPU	TAGE, ITTAGE
BTB	16K entries

gem5 O3 CPU simulation parameters

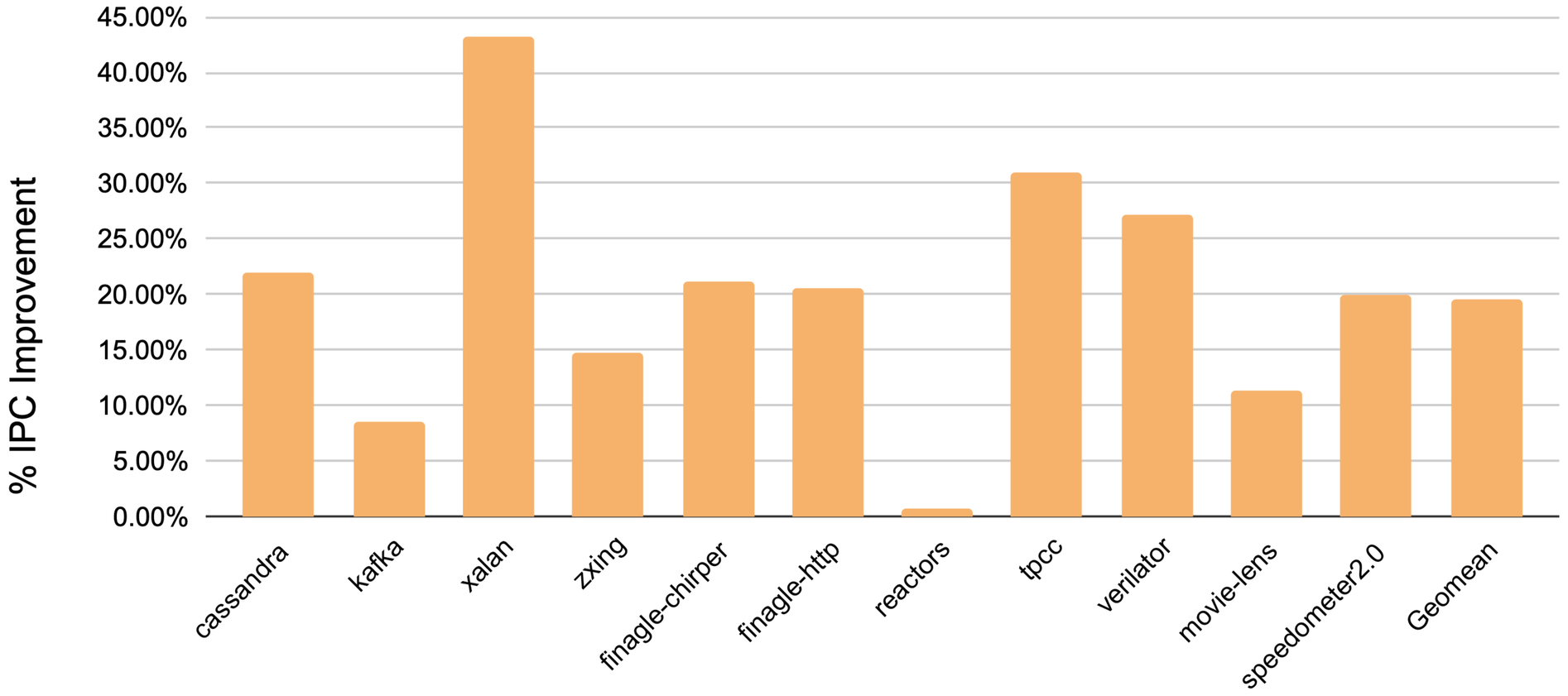


IPC Performance improvement of ARM workloads in % over No FDIP baseline

Performance of X86 workloads with FDIP

Field	Alderlake like
ISA	X86 64-bit
L1I	32KB
L1D	64KB
L2	1MB (16-way)
L3	2MB
FTQ	24 entry 192 inst
Width	8-wide
ROB Size	512 entries
IQ/LQ/SQ	240/128/72
BPU	TAGE, ITTAGE
BTB	16K entries

gem5 O3 CPU simulation parameters

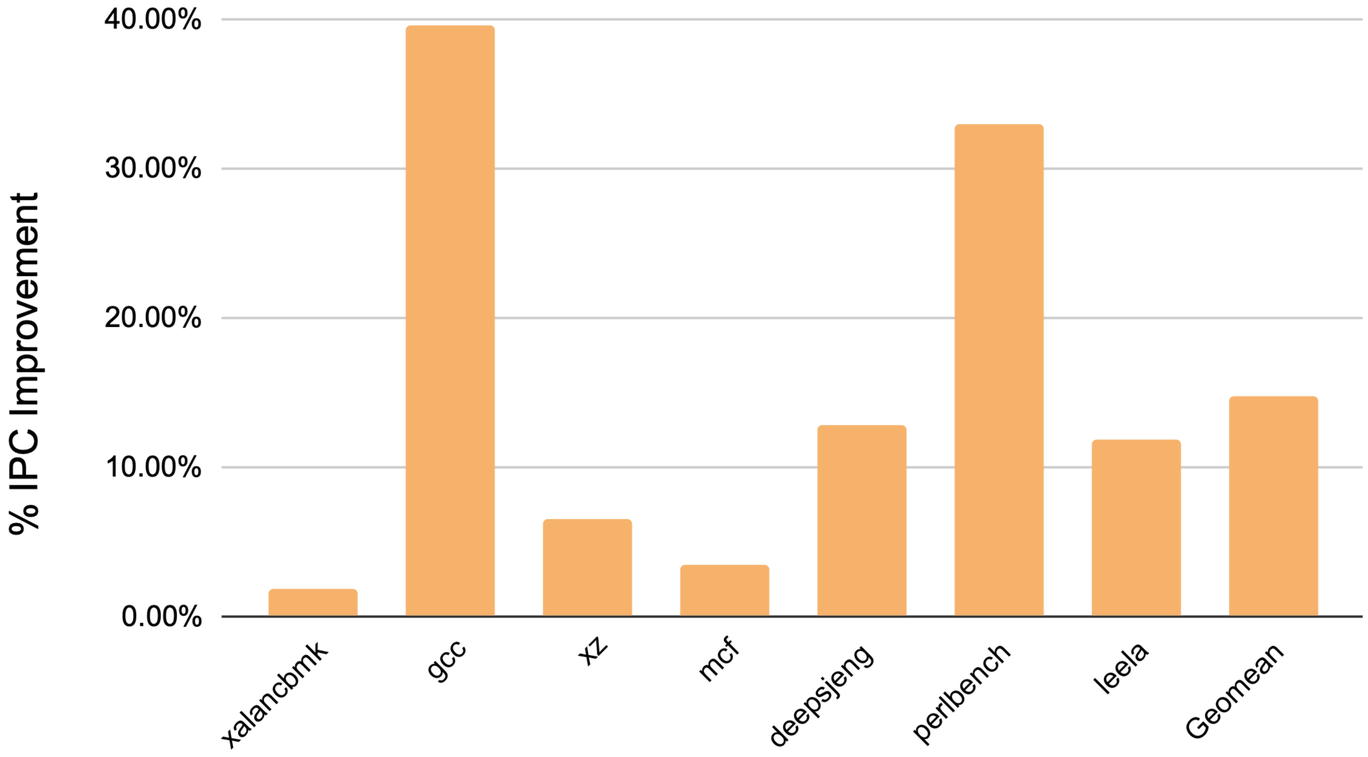


IPC Performance improvement of X86 workloads in % over No FDIP baseline

Performance of X86 SPEC17 workloads with FDIP

Field	Alderlake like
ISA	X86 64-bit
L1I	32KB
L1D	64KB
L2	1MB (16-way)
L3	2MB
FTQ	24 entry 192 inst
Width	8-wide
ROB Size	512 entries
IQ/LQ/SQ	240/128/72
BPU	TAGE, ITTAGE
BTB	16K entries

gem5 O3 CPU simulation parameters



IPC Performance improvement of X86 SPEC17 workloads in % over No FDIP baseline

Published Works

- EMISSARY: Enhanced Miss Awareness Replacement Policy for L2 Instruction Caching at ISCA’23
- Session 2B

Conclusion

- We implemented FDIP in gem5.
- A significant speedup over baseline.
- This work was used in EMISSARY [ISCA'23].
- Available at https://github.com/PrincetonUniversity/gem5_FDIP
- Workloads: https://tinyurl.com/yjsc2aw4

gem5 + FDIP

Workloads

Thank you
Questions?