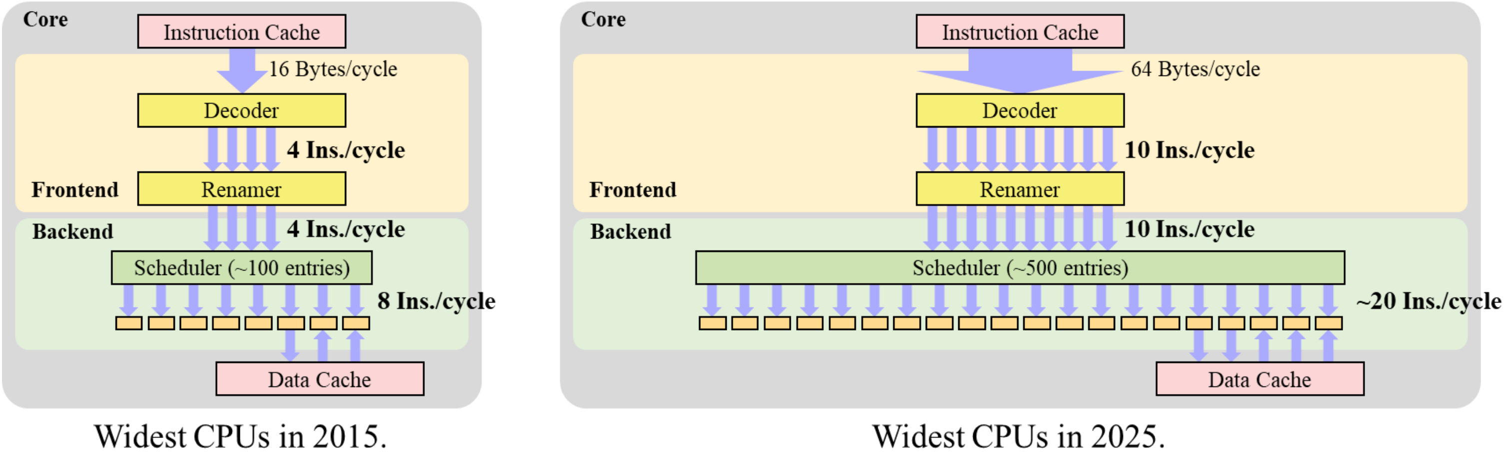
Evaluation of Multiple Branch Prediction's Potential in the Context of Wide-Pipeline Architectures

Steve Bambou Biangang
Advisor: Dr. David Schall
Systems Research Group
https://dse.in.tum.de/

28.04.2025 – 28.08.2025

Motivation

Instruction level-parallelism (ILP) is an inherent characteristic of programs

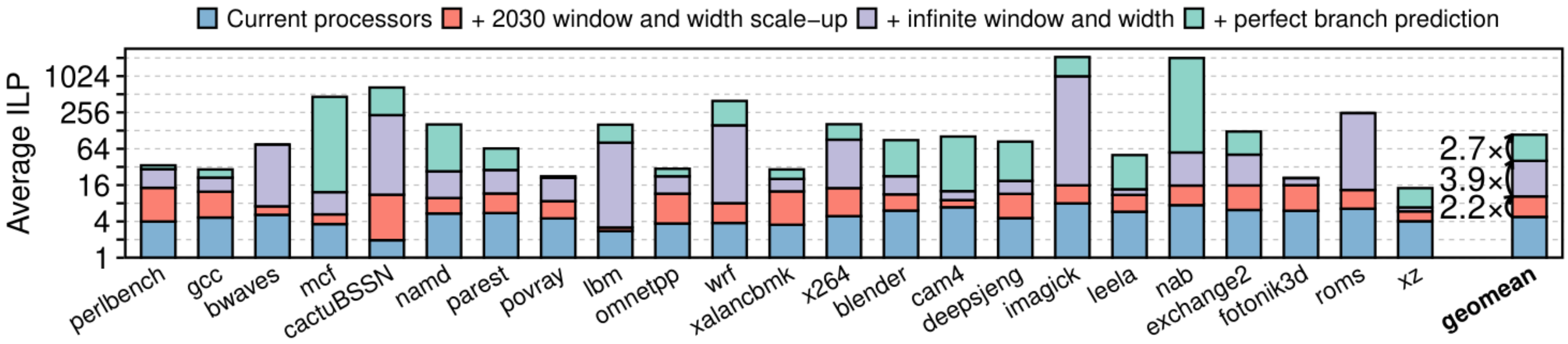


Comparison of widest CPUs in 2015 and 2025 [1]

Expectations are that instruction windows will reach 2^{12} by 2030s [6]

Motivation

ILP research work demonstrated that bigger instruction window can indeed improve performance



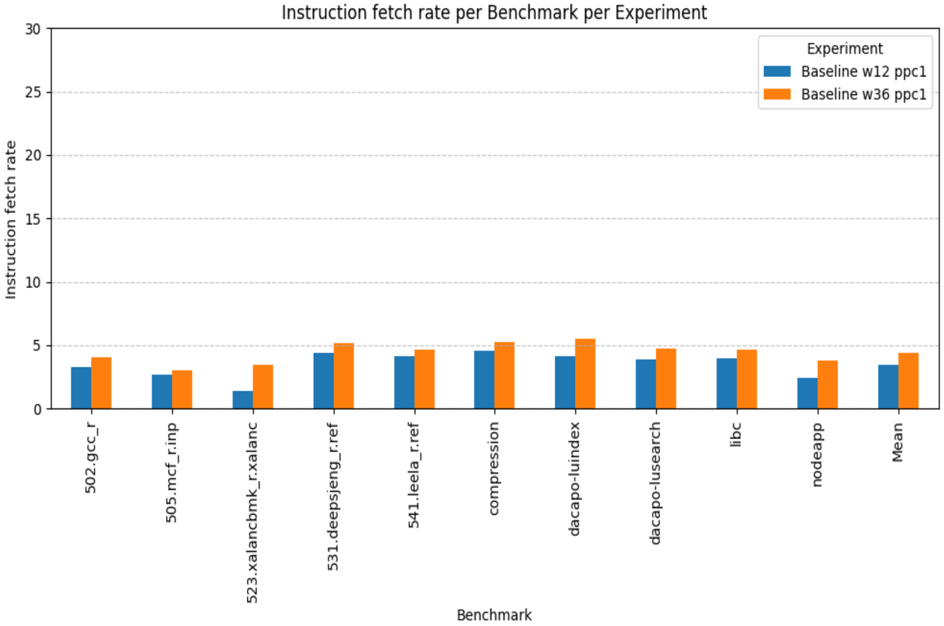
The average ILP limit for the SPEC CPU 2017 benchmarks [6]

Instruction supply is not modeled!

Motivation

Performance gain from scaling the frontend resources is limited

This shows the need for a scalable frontend design able to catch up with the evolution of backend capabilities



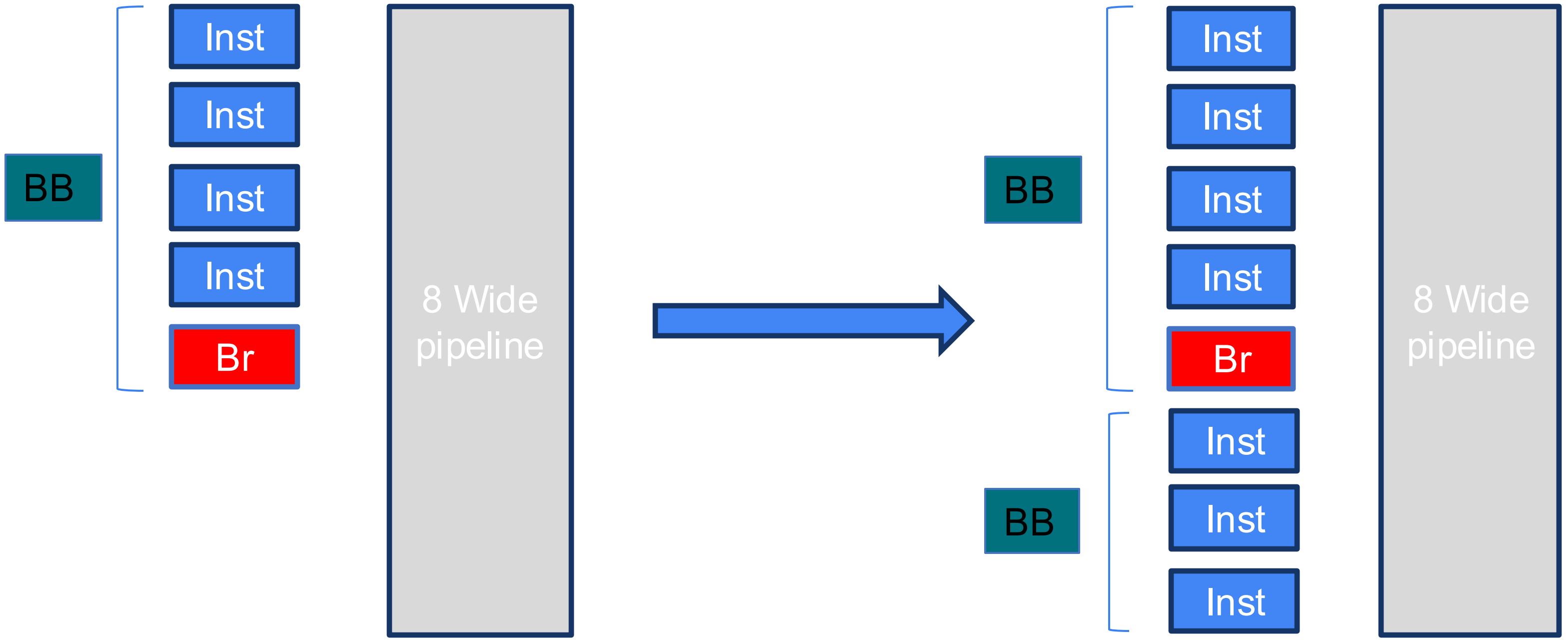
Instruction fetch rate: width 12 vs width 36

Scaling the pipeline resources is not enough!

State-of-the-art

Multiple branch prediction: effective to increase instruction fetch rate[9]

According to recent research work, around 20% of instructions are branches [3]



Research gap

- Scalable frontend designs are needed, effect of multiple branch prediction on scaled architectures not yet evaluated

- Previous ILP research work do not model frontend scenarios and other bottlenecks of real hardware (e.g. misprediction penatly, cache misses) -> need for a realistic assessment

Problem statement

Can multiple branch prediction improve CPU performance by filling the growing instruction window faster?

Research questions:

- What is the impact of multiple branch prediction on frontend performance as the width scales?
- Is scaling the instruction window sufficient to improve performance when comprehensive system modeling is employed?
- Which critical backend components are most susceptible to creating bottlenecks under increased speculative execution?

The gem5 hardware simulator

- Open-source **execution-based** hardware simulator
- Comprehensively models an **out-of-order(O3) CPU pipeline**
- Decoupled frontend is modeled[7]

Suited for scaling resources and to model multiple predictions per cycle

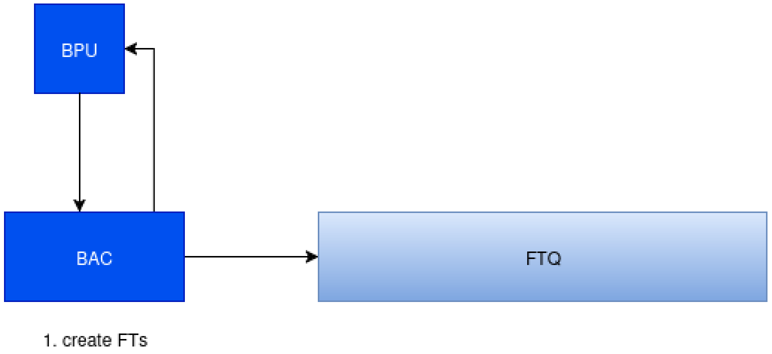
Study design and goals

- We implement multiple branch prediction in gem5
- We evaluate multiple branch prediction to improve instruction fetch rate for wide- pipeline architectures
- We highlight the bottlenecks of an increased speculative execution
- We estimate ILP using comprehensive modelling (gem5 O3 core)

Outline

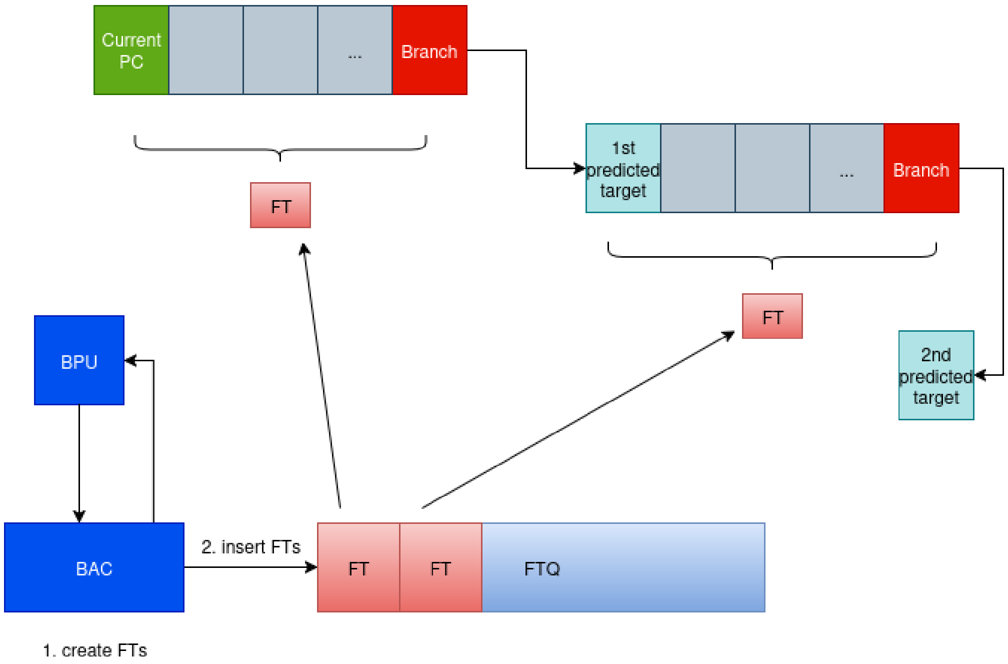
- ~~Motivation~~
- **Frontend**
  - Design
  - Evaluation
- Backend
- Summary

Design



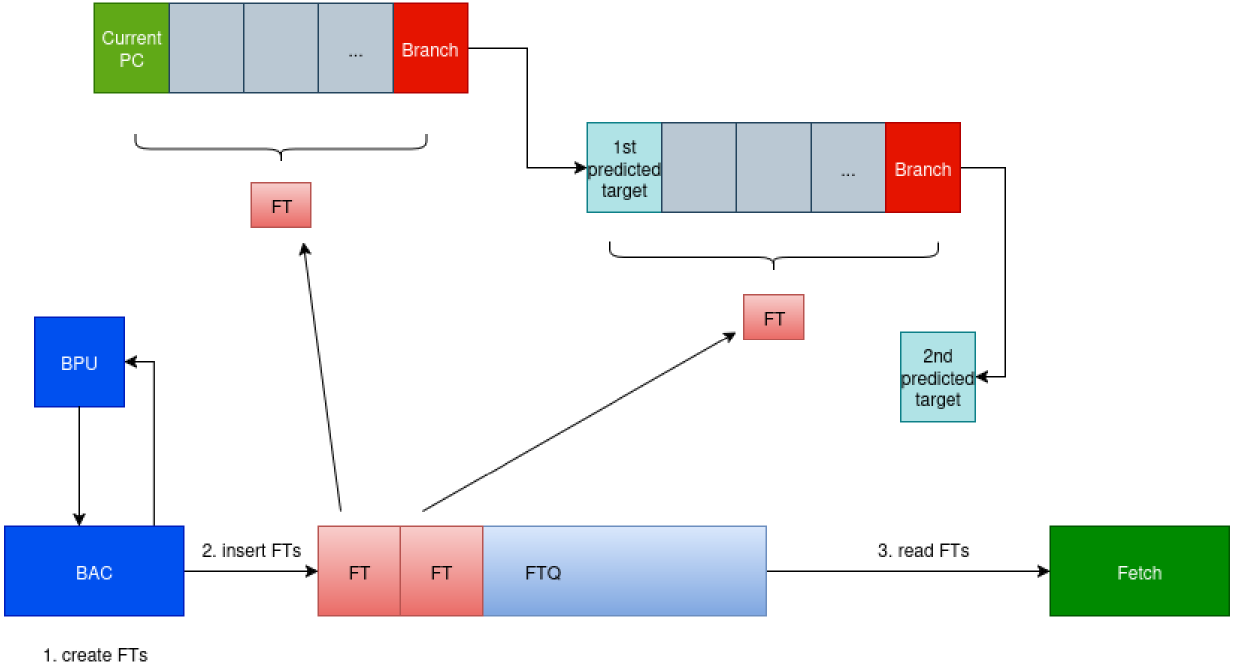
Multiple branch prediction in a decoupled frontend

Design



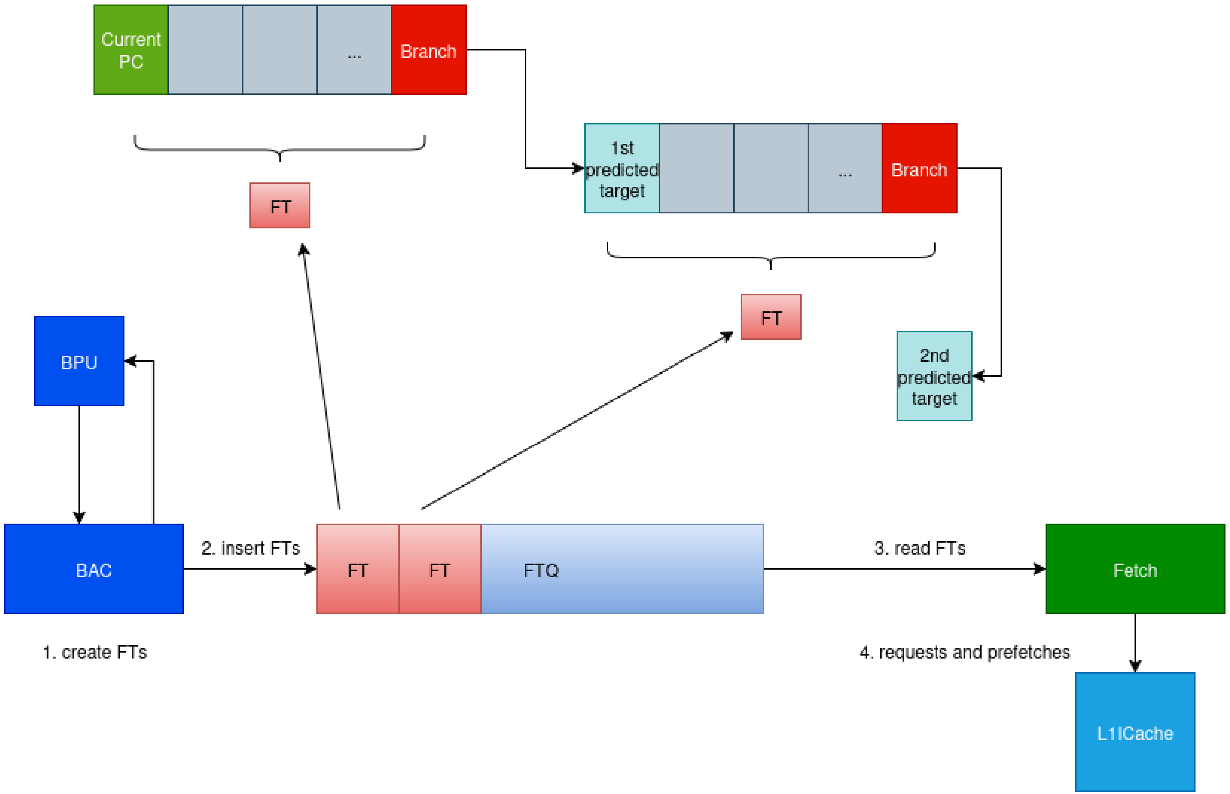
Multiple branch prediction in a decoupled frontend

Design



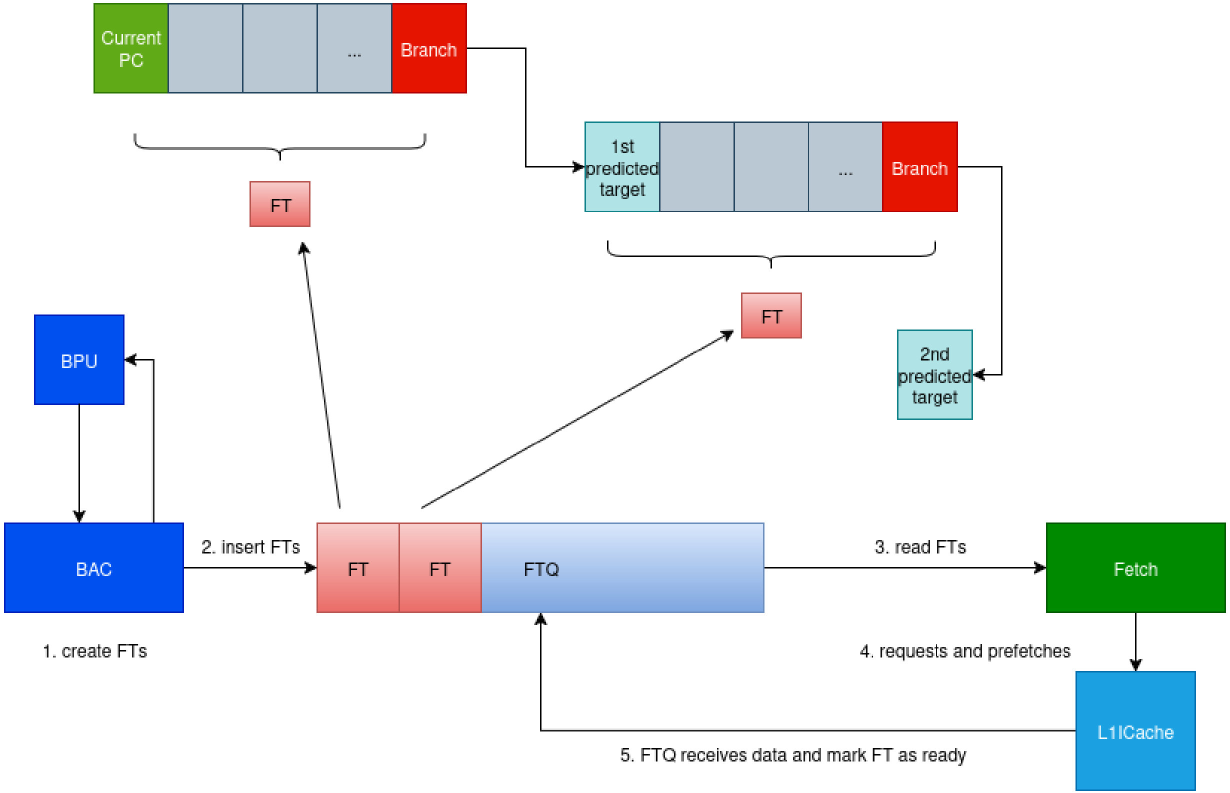
Multiple branch prediction in a decoupled frontend

Design



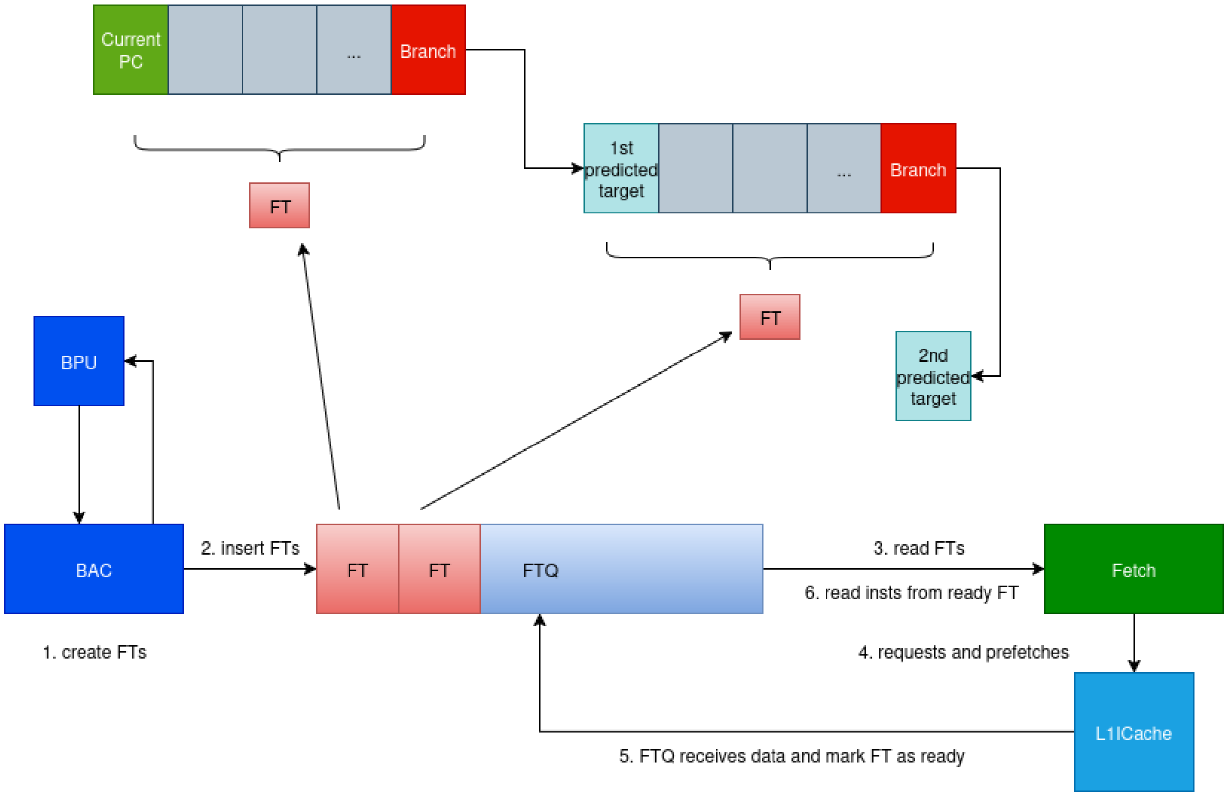
Multiple branch prediction in a decoupled frontend

Design



Multiple branch prediction in a decoupled frontend

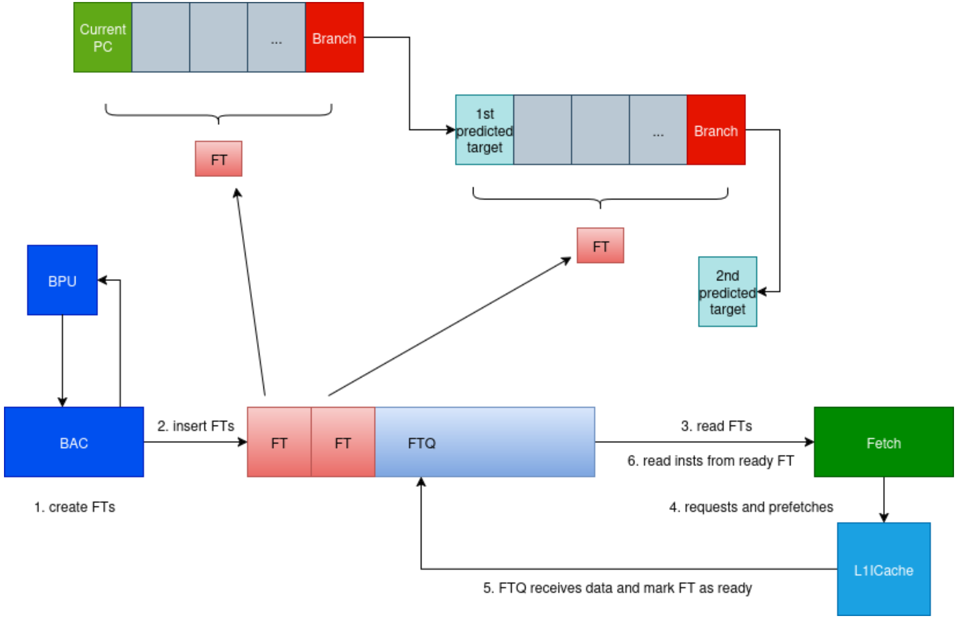
Design



Multiple branch prediction in a decoupled frontend

Design

- Multiple fetch targets fetched per cycle
- We hide miss latency with FDP
- We have an FTQ design hiding cache access latency



This ensures high throughput with low latency

Evaluation methodology

- Two pipeline configurations: w12, w36
- We use SPEC integer benchmarks and server workloads
- Warmup for at least 100M instructions, measuring for 200M
- Branch predictor: TAGE-SC-L 64KB [8]

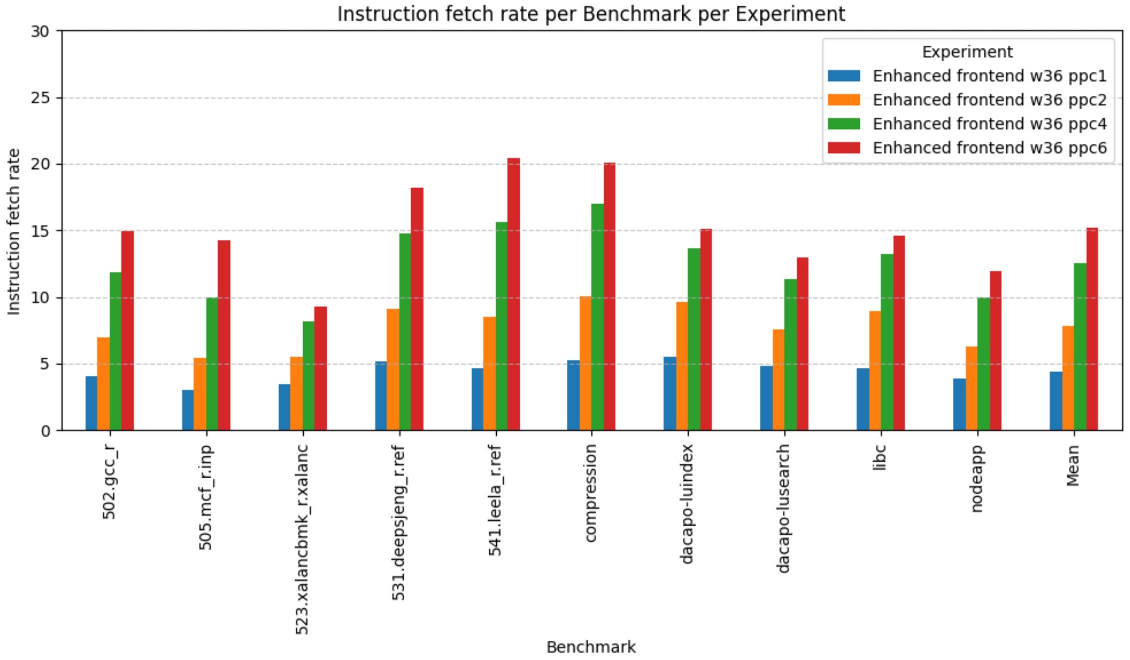
Configuration details:

Configuration	w12	w36
Pipeline width	12	36
L1Icache	64KiB	256KiB
BTB	32Ki entries	128 Ki entries
FTQ	50 fetch targets	150 fetch targets

Evaluation

Comparing six predictions per cycle to one:

We achieve an average increase of **3.4x**, with a maximum of **4.8x**



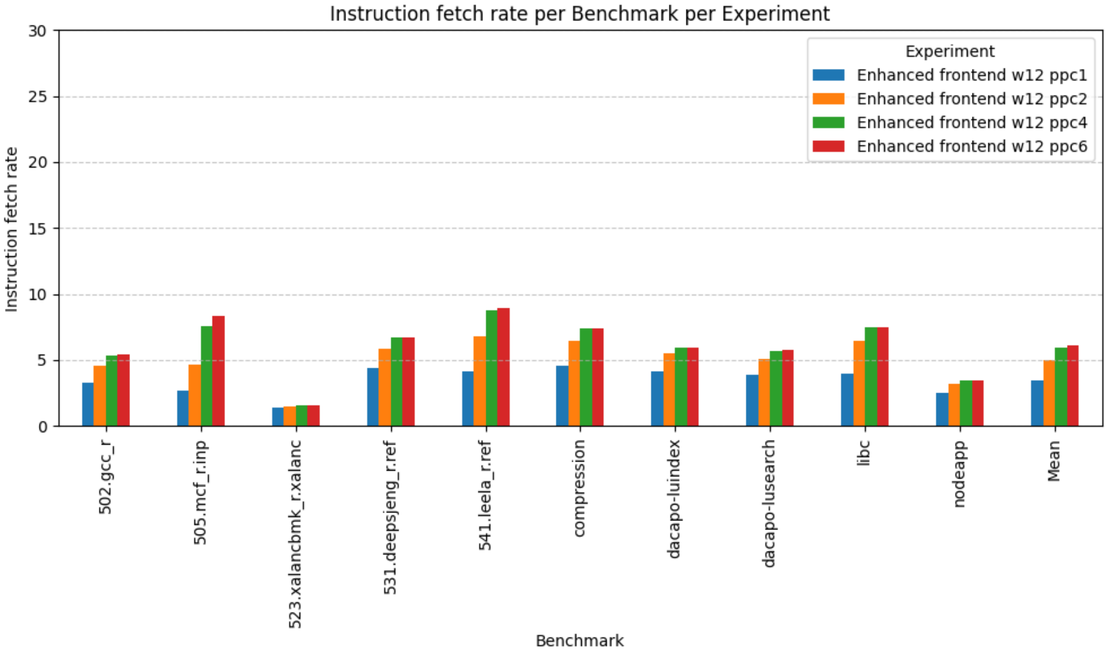
w36: Instruction fetch rate for multiple predictions per cycle

Multiple branch prediction is effective on a scaled pipeline

Evaluation

Comparing six predictions per cycle to one:

We achieve an average increase of **1.74x**, with a maximum of **3.4x**



w12: Instruction fetch rate for multiple predictions per cycle

There is also potential for current processors

Take-away

- Multiple branch prediction shows great potential at improving fetch rate for future processor design

- There are also (lower) opportunities for current architectures

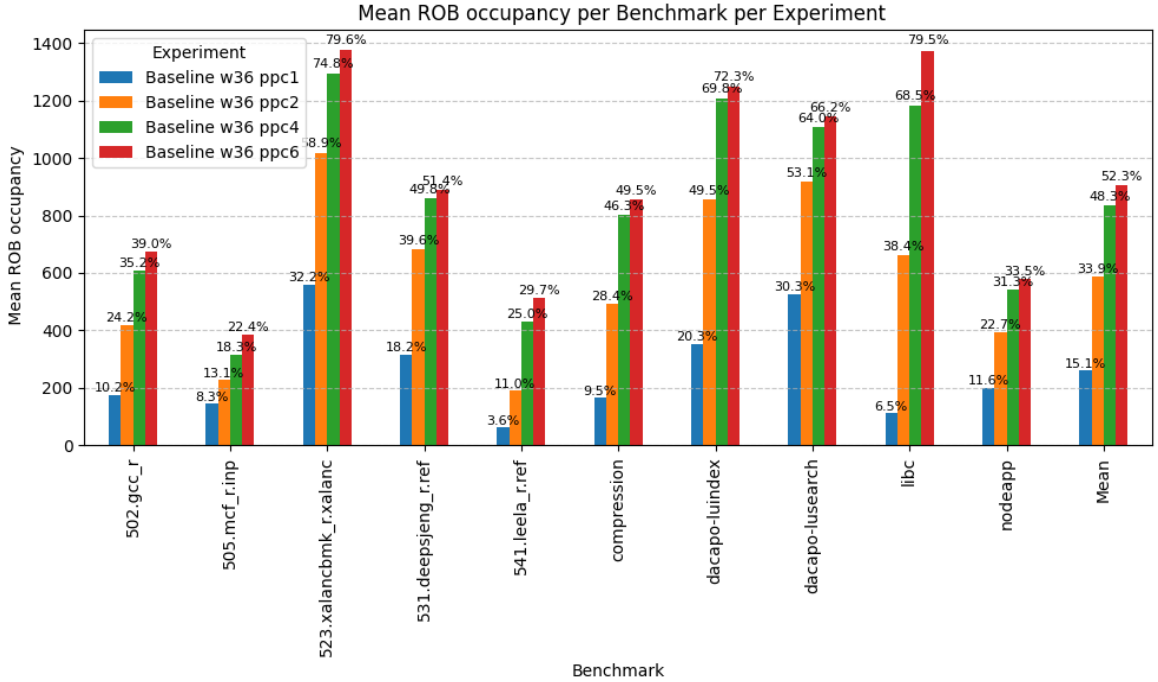
How well does the increase in instruction fetch rate translate to IPC improvement?

Outline

- ~~Motivation~~
- ~~Frontend~~
- **Backend**
  - Bottlenecks
  - Performance
- Summary

Impact on the instruction window

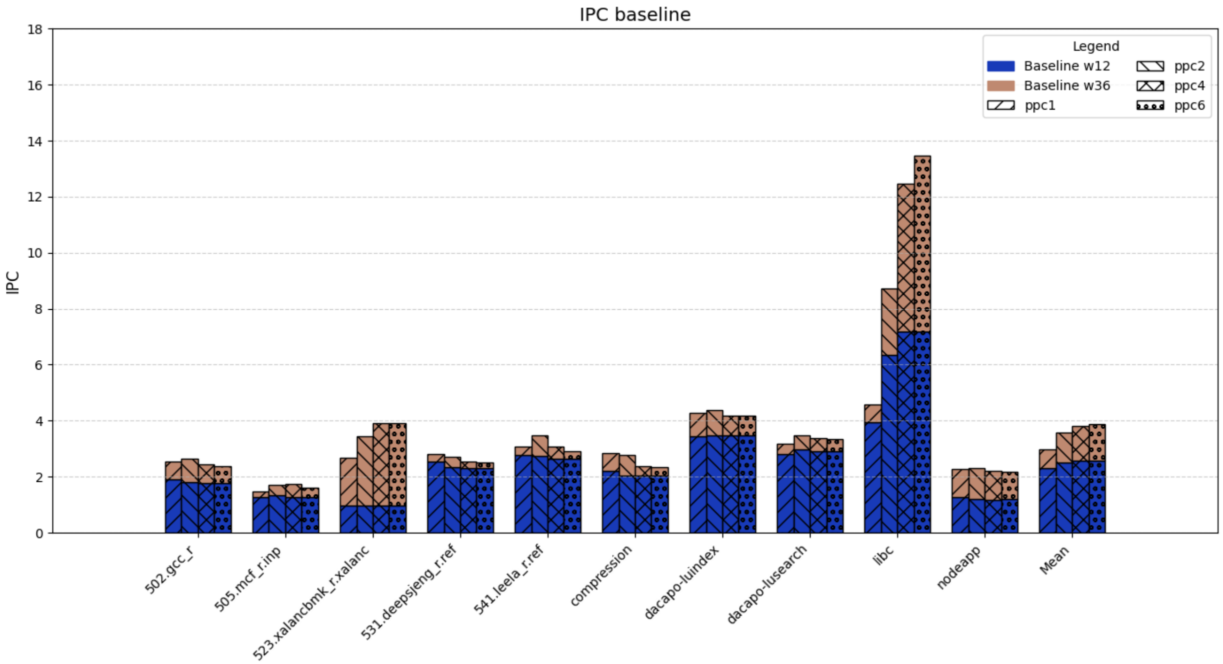
- The reorder buffer (ROB) fills faster
- Backend has more instructions available, which increase the opportunity to extract more ILP



Mean ROB occupancy for multiple predictions per cycle

Counterintuitive IPC results

- Overall IPC improvement from ppc6 over one limited
- A lot of benchmarks actually become worse

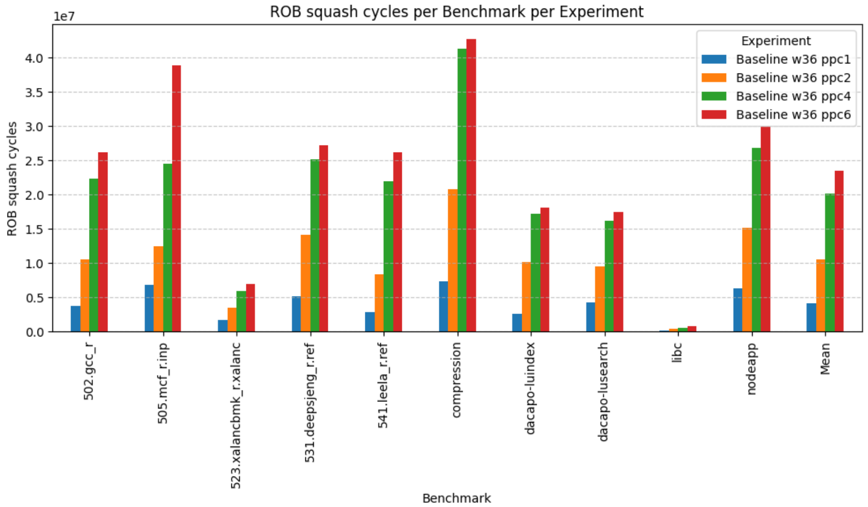


IPC baseline

Bottlenecks

- Increase of recovery penalty for the ROB:
  - Limited squash width
  - Branch misprediction
  - Memory dependency misprediction

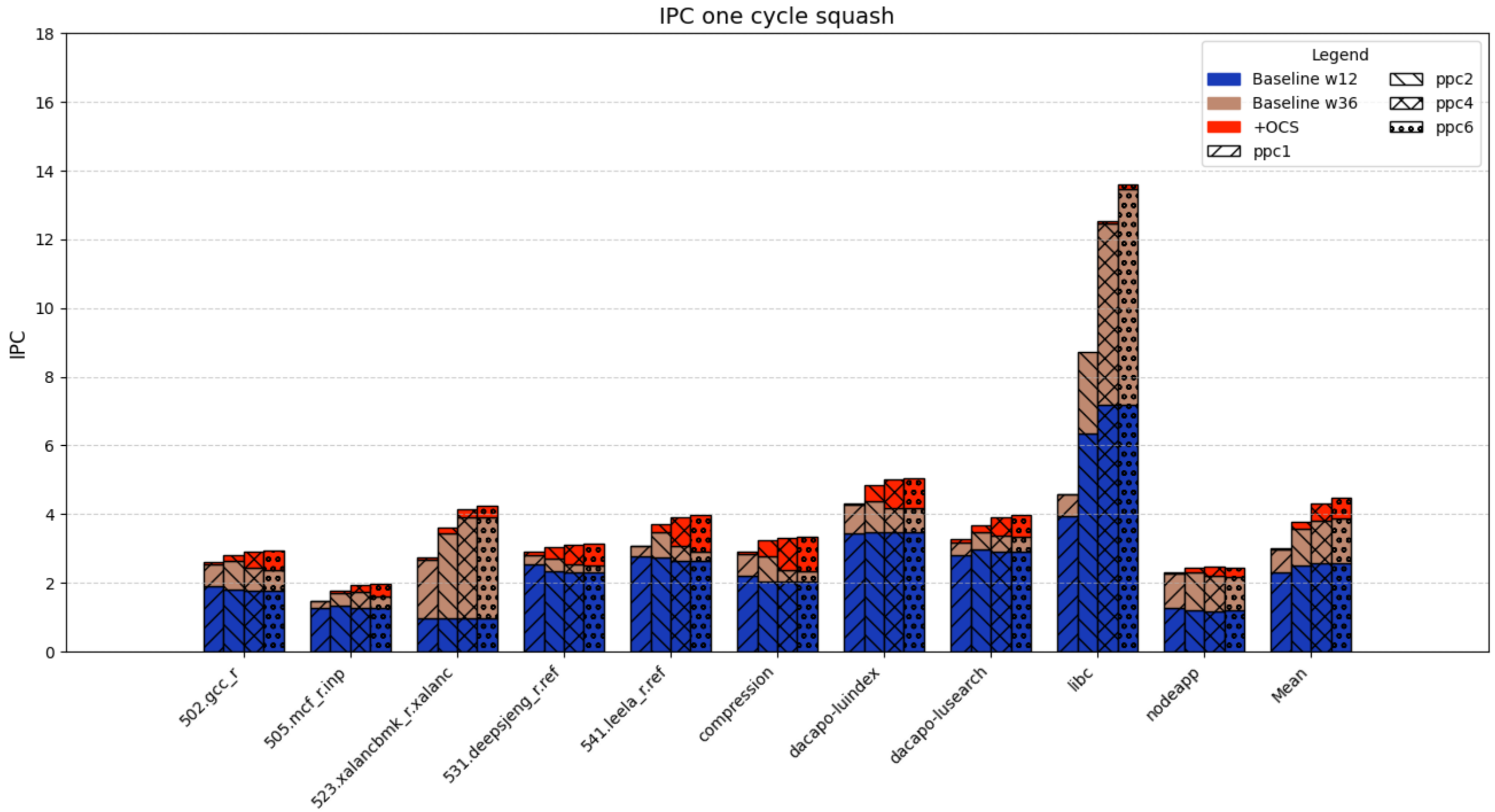
- Increase in simultaneous data cache misses (doesn’t degrade performance)



Backend optimizations

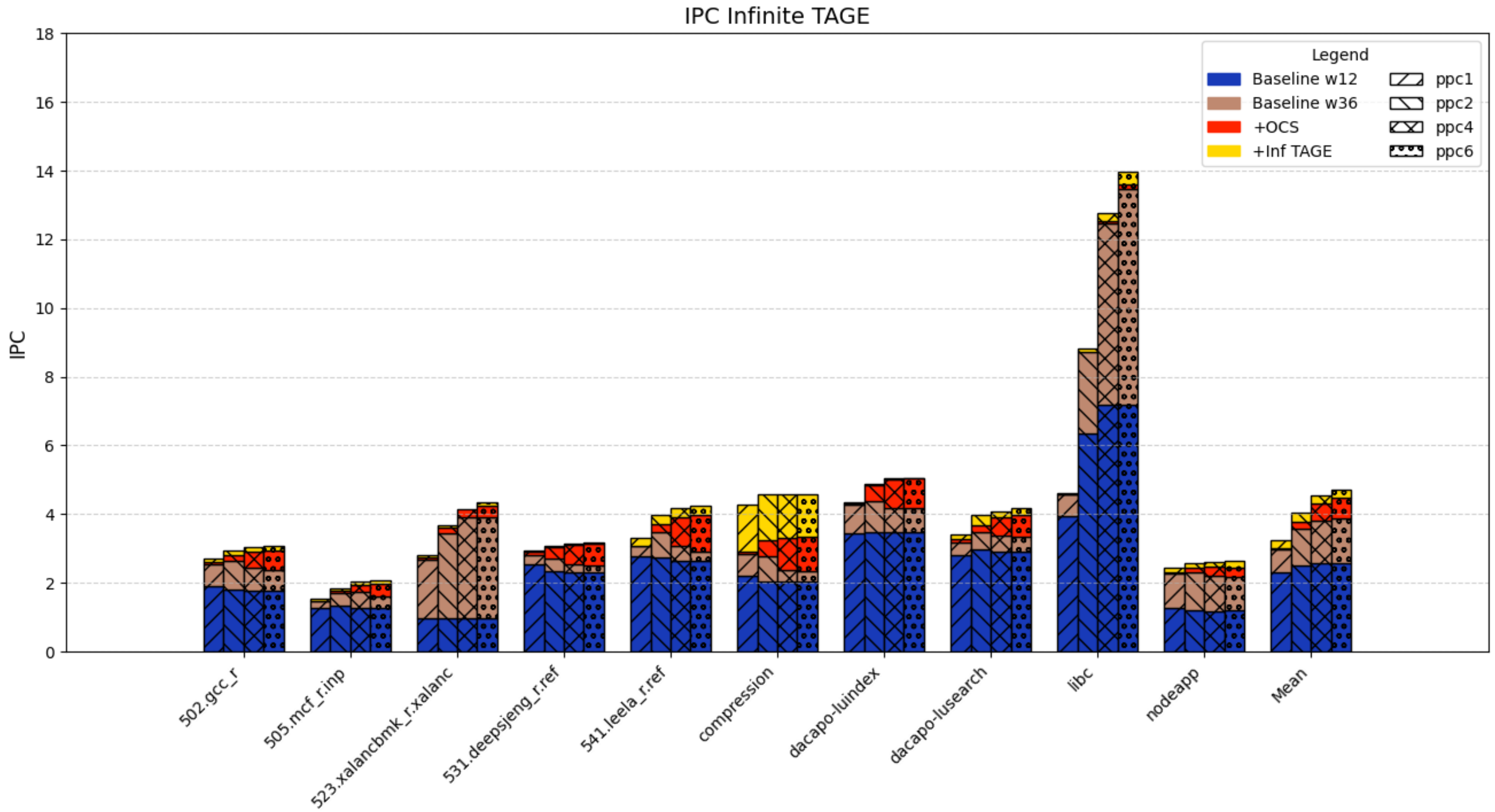
- One cycle squash (OCS) for the ROB
- TAGE-SC-L 64KB -> Infinite TAGE
- Store sets -> Infinite PHAST
- Normal cache-> Giant cache (32MiB)
- 1728 ROB entries -> 4608 ROB entries: bigger inst window

Evaluation



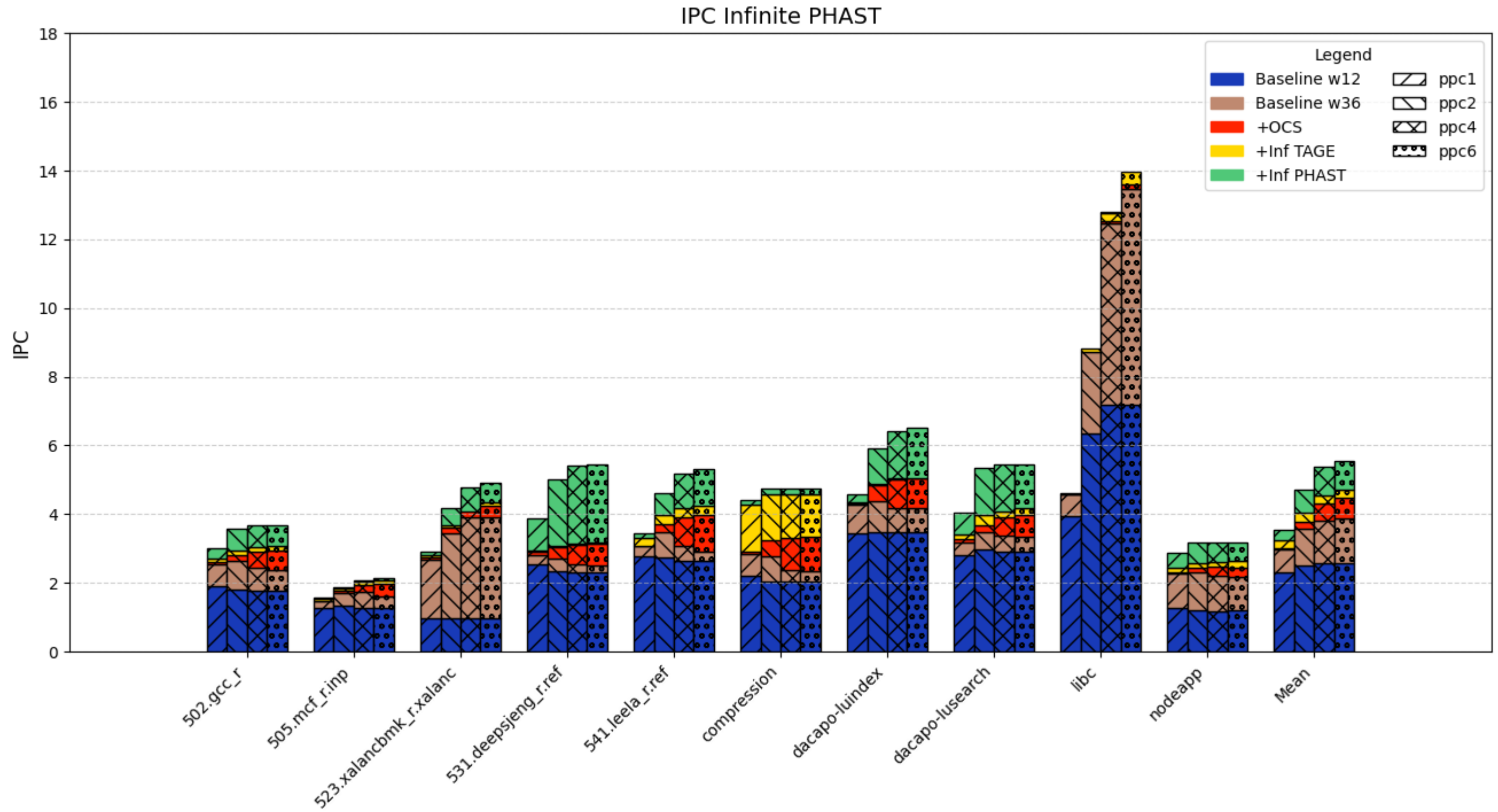
One cycle squash removes the negative effect on performance

Evaluation



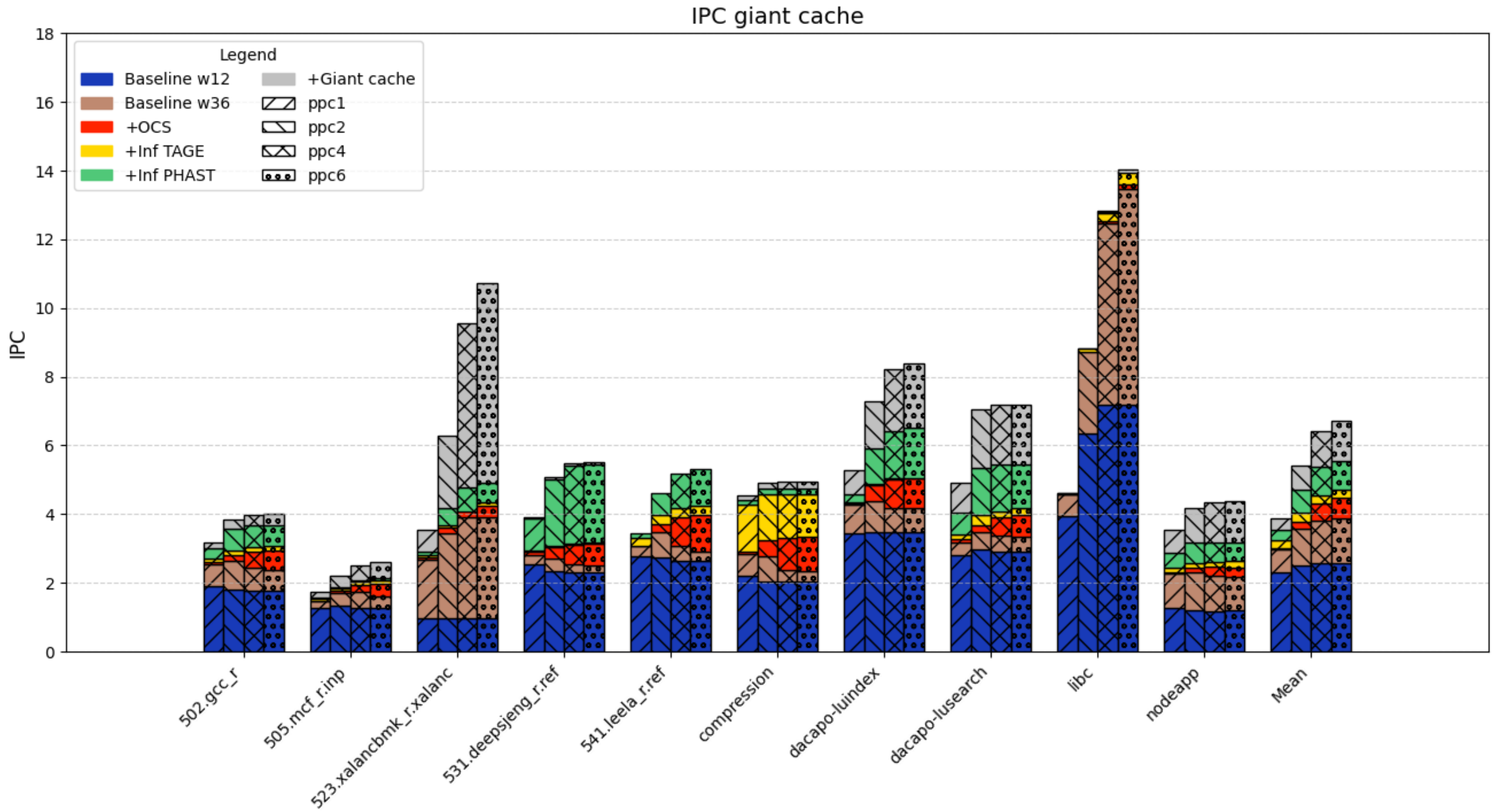
Performance improvement of Infinite TAGE is limited despite its infinite storage

Evaluation



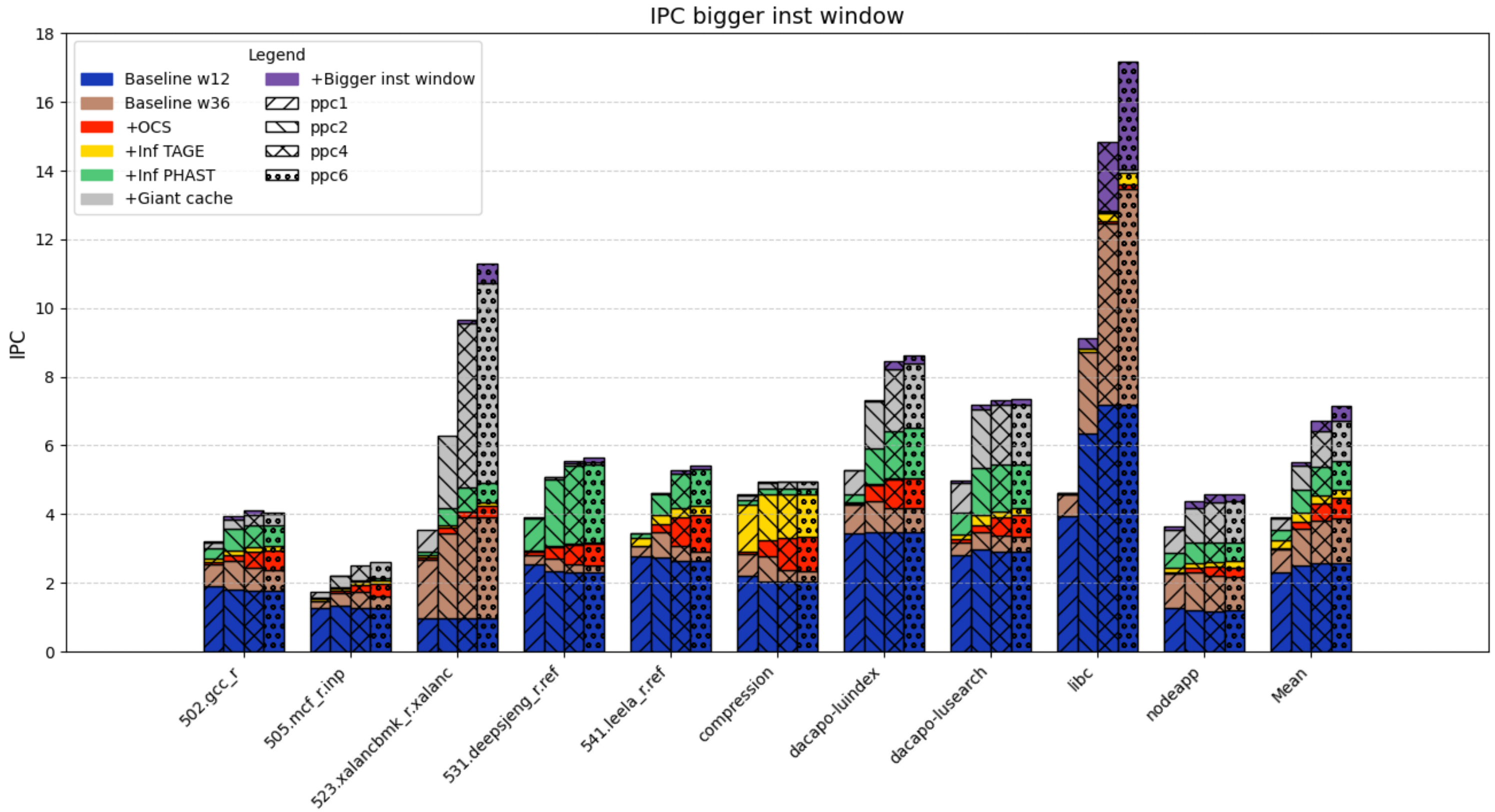
Infinite PHAST improves performance for most benchmarks

Evaluation



Reducing the data cache misses is indeed an important factor

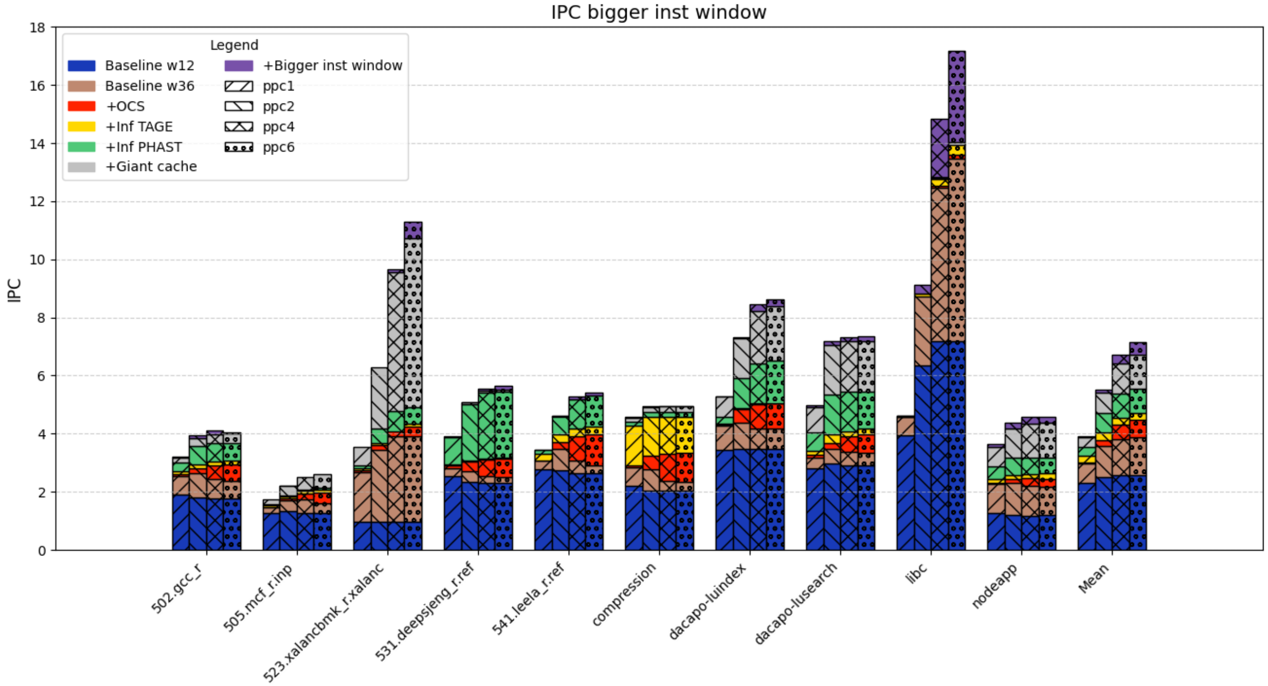
Evaluation



It's necessary to solve the other bottlenecks to profit from the larger ROB

Evaluation

- We reach **1.79×** speedup of ppc6 over ppc1 on this configuration



IPC improvement for bigger inst window

There are opportunities to improve performance with multiple branch prediction

Limitations

- Gem5 exhibit more serializing stalls than real hardware. They significantly limit out-of-order execution and therefore IPC
- We considered a delay between stages of one cycle. A longer delay could hide some of the recovery penalty
- We did not investigated optimizations of the instruction scheduler
- Gem5 does not model perfect predictors, which would have been useful to study the limit of ILP

Summary

- Multiple branch prediction shows great potential at improving fetch rate for future processor design
- The recovery penatly is the biggest bottleneck to an increased speculative execution in our model
- Backend's bottlenecks can mask the improved frontend capabilities

Holistical improvement of the pipeline is important for better performance

Future work

- Implementation of a state-of-the-art multiple branch prediction design for real hardware
- Find practical ways to reduce the recovery penalty in real hardware
- Remove the gem5-specific serializing stalls
- Implement perfect predictors in gem5

References

- Toru Koizumi, Ryota Shioya, Hidetsugu : Irie Distance-Based ISA for Efficient Register Management https://www.sigarch.org/distance-based-isa-for-efficient-register-management/ [1]
- C. -K. Lin and S. J. Tarsa, "Branch Prediction Is Not A Solved Problem: Measurements, Opportunities, and Future Directions," 2019 IEEE International Symposium on Workload Characterization (IISWC), Orlando, FL, USA, 2019, pp. 228-238, doi: 10.1109/IISWC47752.2019.9042108 [2]
- D. Schall, A. Sandberg and B. Grot, "The Last-Level Branch Predictor," *2024 57th IEEE/ACM International Symposium on Microarchitecture (MICRO)*, Austin, TX, USA, 2024, pp. 464-479, doi: 10.1109/MICRO61859.2024.00042 [3]
- André Seznec, Stéphan Jourdan, Pascal Sainrat, Pierre Michaud. Multiple-Block Ahead Branch Predictors. [Research Report] RR-2825, INRIA. 1996. ⟨inria-00073867⟩ [4]
- S. Wallace and N. Bagherzadeh, "Multiple branch and block prediction," *Proceedings Third International Symposium on High-Performance Computer Architecture*, San Antonio, TX, USA, 1997, pp. 94-103, doi: 10.1109/HPCA.1997.569645 [5]
- A. Chadwick, M. Erdos, U. Bora, A. Bhosale, B. Lytton, Y. Guo, R. Cooper, G.Gabrielli, and T. Jones. “The Future of Instruction-Level Parallelism (ILP).” In: May 2025, pp. 350–352. doi: http://dx.doi.org/10.1109/ISPASS64960.2025.00040 [6]
- D. Schall. Development repository for Fetch Directed Instruction Prefetching (FDP) in gem5. url: https://github.com/dhschall/gem5-fdp. [7]
- A. Seznec. “TAGE-SC-L Branch Predictors Again.” In: 5th JILP Workshop on Computer Architecture Competitions (JWAC-5): Championship Branch Prediction (CBP-5). Seoul, South Korea, June 2016. [8]
- T.-Y. Yeh, D. T. Marr, and Y. N. Patt. “Increasing the instruction fetch rate via multiple branch prediction and a branch address cache.” In: Proceedings of the 7th International Conference on Supercomputing. ICS ’93. Tokyo, Japan: Association for Computing Machinery, 1993, pp. 67–76. isbn: 089791600X. doi: 10.1145/165939.165956. [9]
- G. Reinman, B. Calder, and T. Austin. “Fetch directed instruction prefetching.” In: MICRO-32. Proceedings of the 32nd Annual ACM/IEEE International Symposium on Microarchitecture. 1999, pp. 16–27. doi: 10.1109/MICRO.1999.809439. [10]