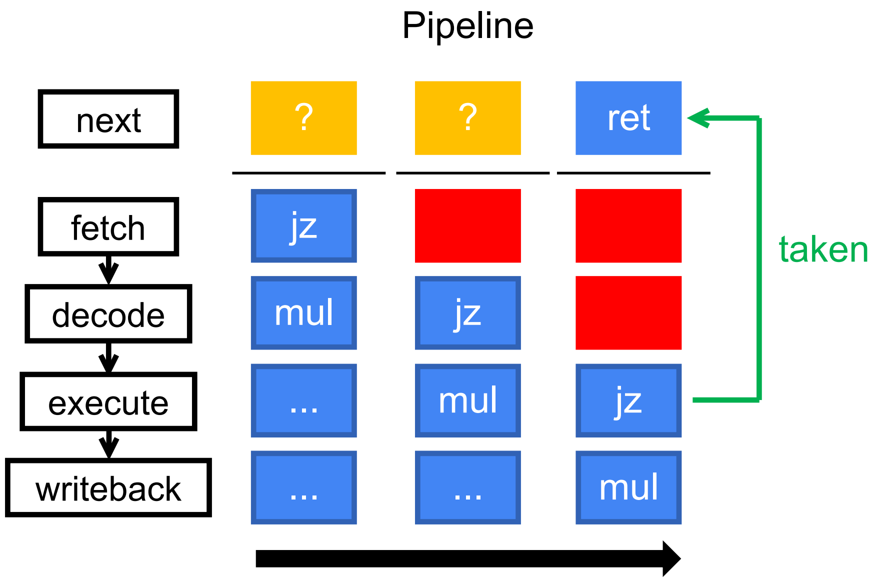
Mitigating Branch Predictor Latency with Hierarchical Design — an Analysis

Phillip Assmann
Advisor: Dr. David Schall
Chair of Computer Systems
https://dse.in.tum.de/

15.03.2025 – 15.07.2025

Branch prediction

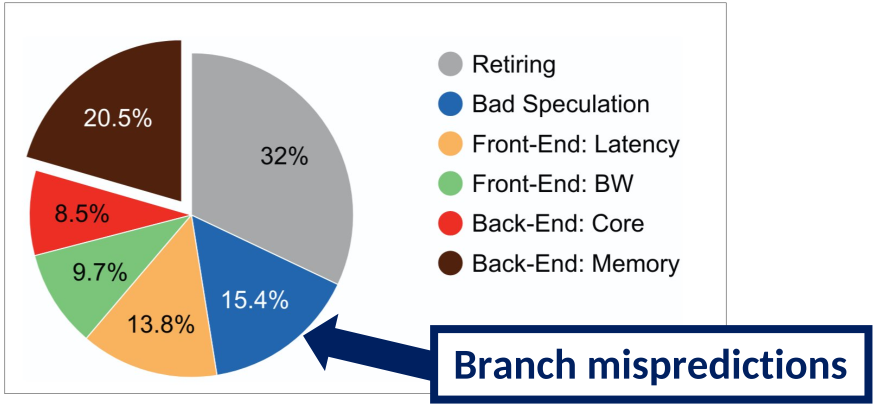


- Modern processors use **heavily pipelined** out-of-order execution [4]
- CPU cannot fetch the next instructions until branch direction is known

➡ **Branch prediction is required for effective pipelining**

Branch prediction

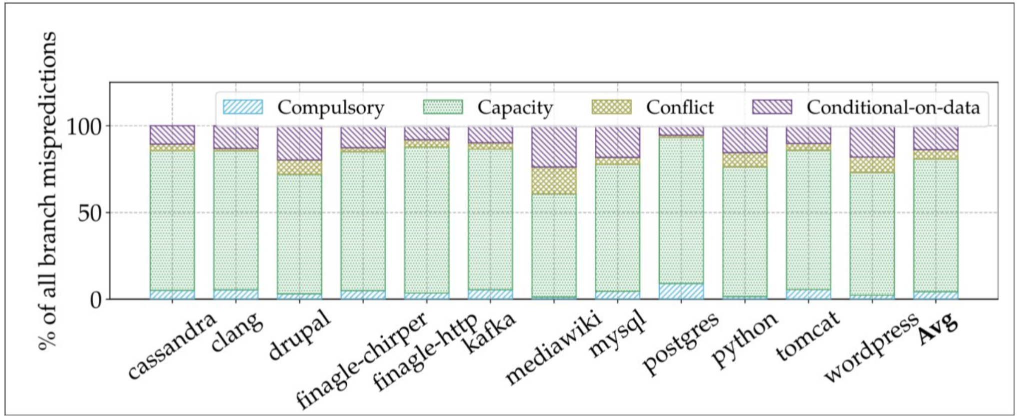
- Branch prediction is still a bottleneck in today’s processors:
  - Over 9% of cycles lost (avg) to mispredictions even on recent server processors [3]
  - 15.4% of pipeline slots wasted to branch mispredictions in Google’s datacenters [2]



Google; Top-down breakdown of search workload [2]

The capacity bottleneck

- The instruction footprint of datacenter workloads is growing [13]
- Branch predictors cannot keep up:
  - Over 75% of mispredictions in large server workloads are caused by a lack of storage capacity [14]

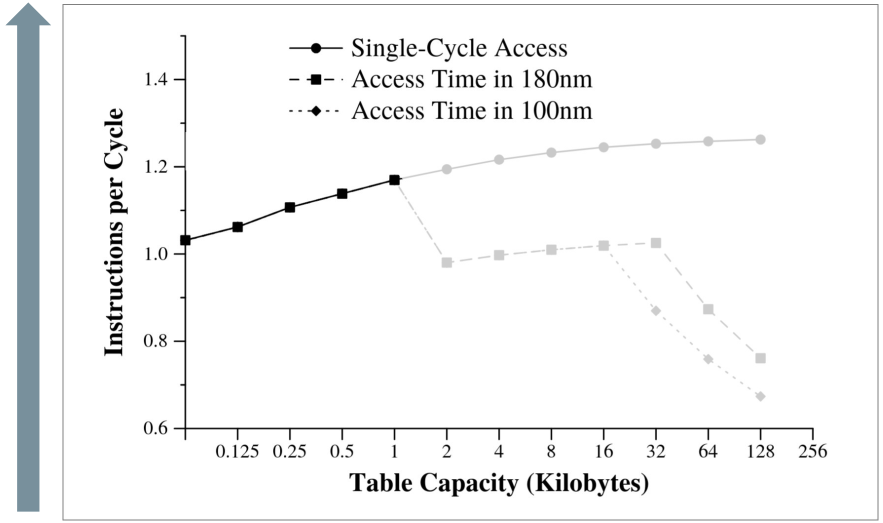


Whisper; Breakdown of branch mispredictions
Various server workloads [14]

➡ **Larger predictors needed to deal with growing branch working set**

Up-scaling alone is not the solution

Larger storage size ➡ Increased access latency



Theoretical improvement

With latency

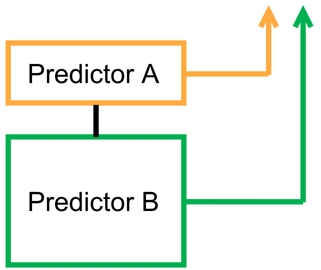
Jiménez et al; IPC vs table capacity, gshare predictor [8]

➡ Latency can be a performance limiter

Reduced latency with hierarchical designs

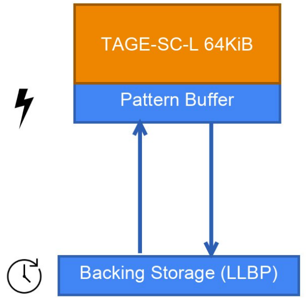
Overriding branch prediction

Commonly implemented in modern processors, **state-of-the-art**



The Last-Level Branch Predictor [3]

New design from MICRO '24

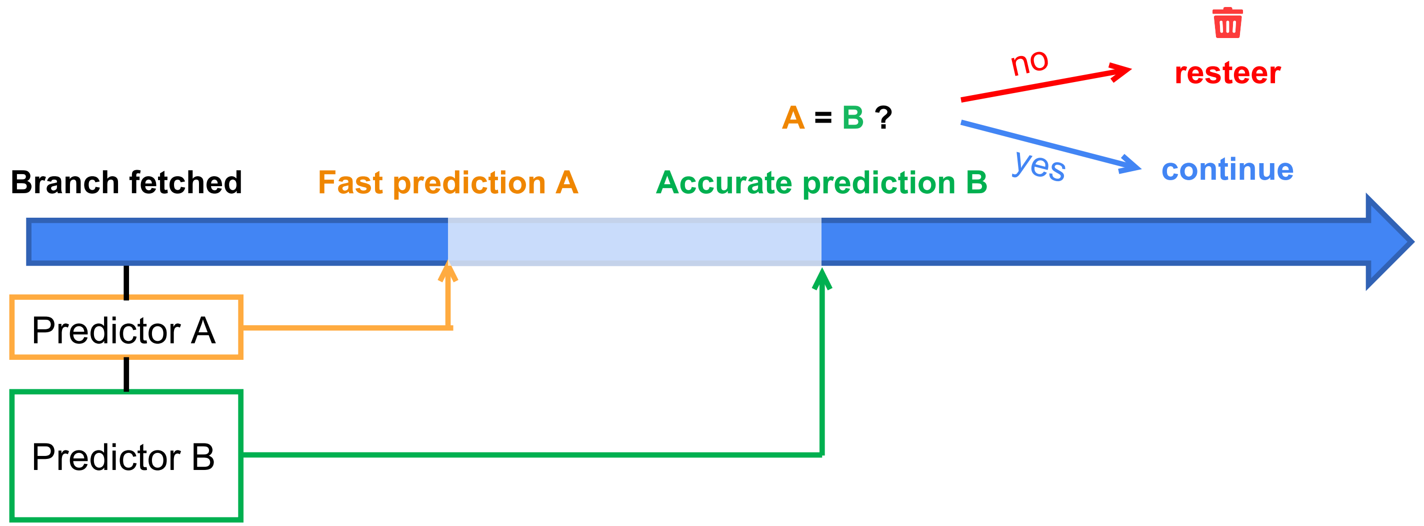


Overriding branch predictors

Two predictors:

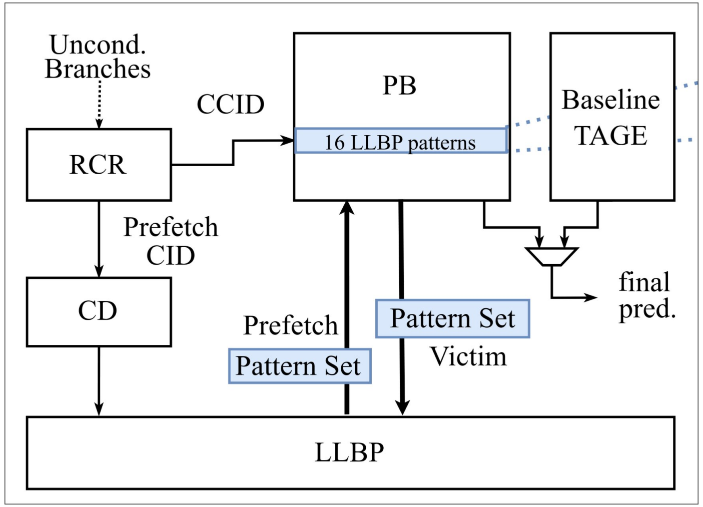
- Small, fast predictor A ➡ low latency
- Large, slow predictor B ➡ high accuracy

➡ **Hides long access latency of predictor B**

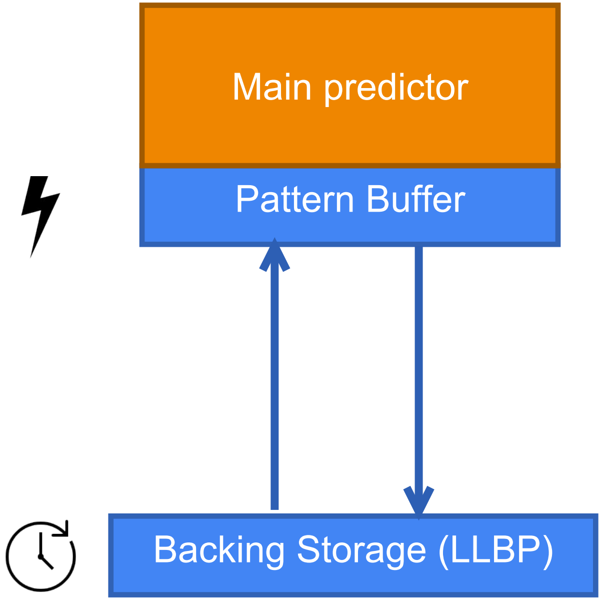


The Last-Level Branch Predictor (LLBP) [3]

- Extends a state-of-the-art TAGE-SC-L predictor
- Prefetching from backing storage ➡ **better accuracy — same latency**



Overview of LLBP from the paper [3]



Research gap & Problem statement

➡ **How effective are these approaches? Does predictor latency still have a large impact in modern processors?**

Problem Statement: Evaluating the impact of latency on state-of-the-art branch prediction and different latency mitigation schemes

Research Questions:

1. How large is the **impact of predictor latency** on performance?
2. Can **overriding prediction** reduce this penalty?
3. Is a recent approach, **The Last-Level Branch Predictor**, more effective?

The gem5 hardware simulator [9]

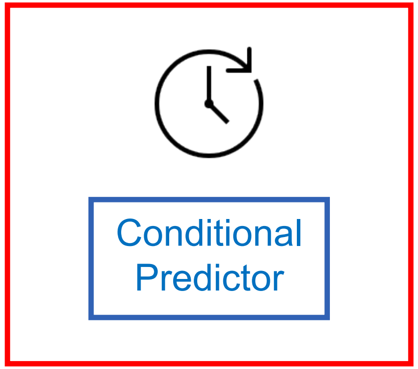
- Open-source execution-based hardware simulator (similar to hypervisor)
- Supports modeling a **detailed out-of-order CPU pipeline**

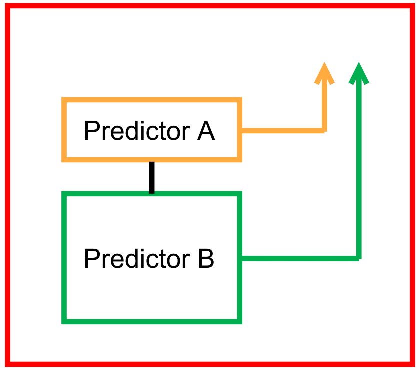
- False path execution is simulated, important for branch predictor evaluation
- **Models state-of-the-art decoupled frontend** [15]

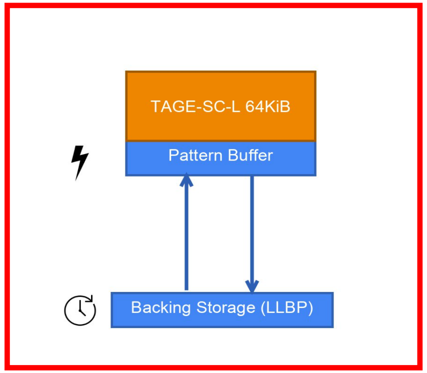
➡ **Allows for detailed evaluation of branch predictor behavior**

The missing link: Branch prediction in gem5

- gem5 is **missing features** for realistic branch prediction that are present in modern CPUs
- Most notably **latency modeling** and **overriding prediction** are not implemented, instead every prediction is available instantly







[1]

➡ **Need a framework for more realistic branch prediction simulation in gem5**

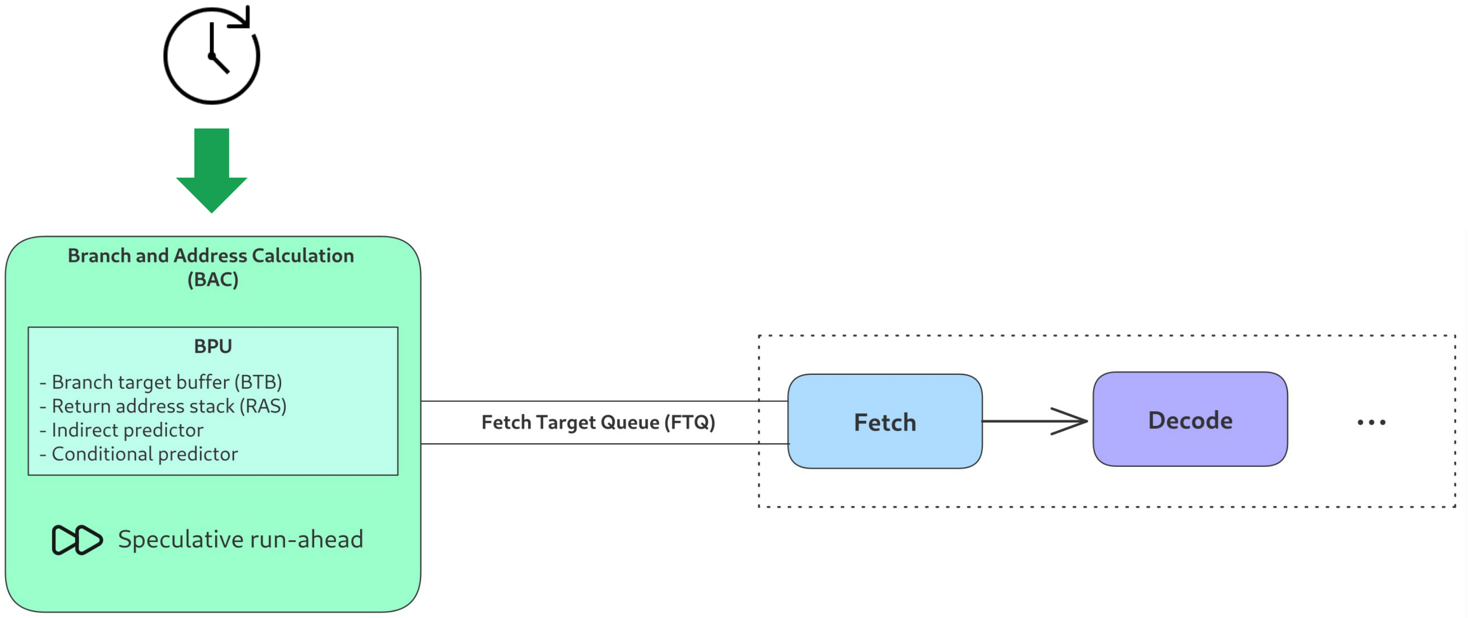
1 An initial prototype of LLBP for gem5 was implemented by the University of Wisconsin

Outline

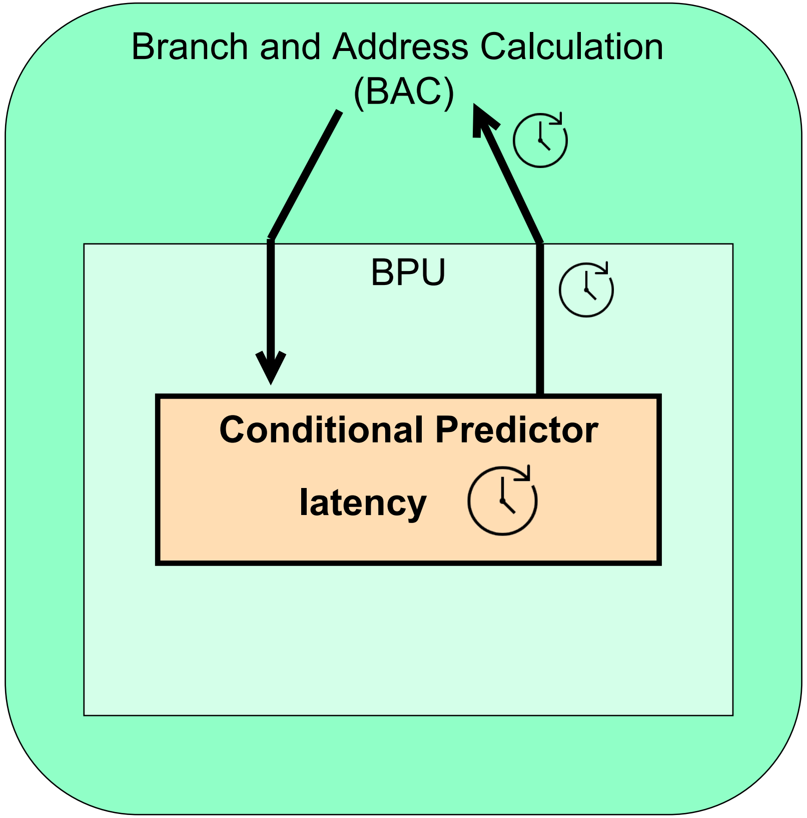
- ~~Motivation & Background~~
- Design
  - Modeling prediction latency in gem5
  - Overriding branch predictors
  - LLBP implementation
- Evaluation
- Conclusion

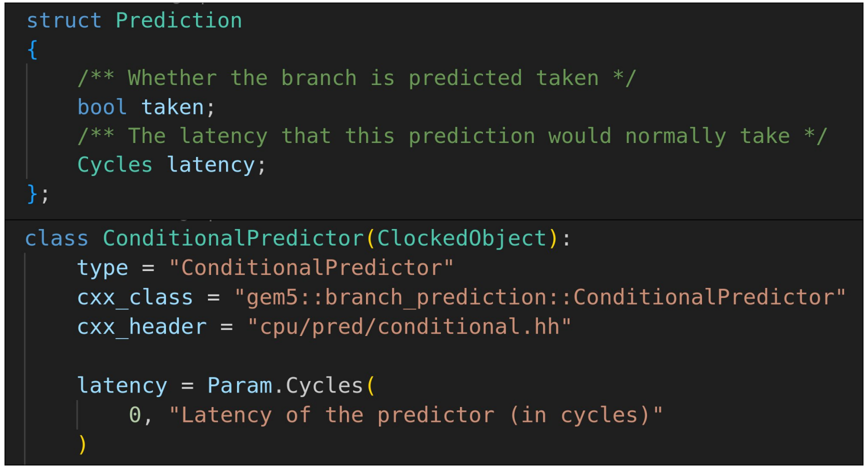
Modeling prediction latency in gem5

- Branch predictor is **decoupled from fetch**, runs ahead of instruction stream
- Predictor latency **only directly impacts the BAC stage**



Modeling prediction latency in gem5



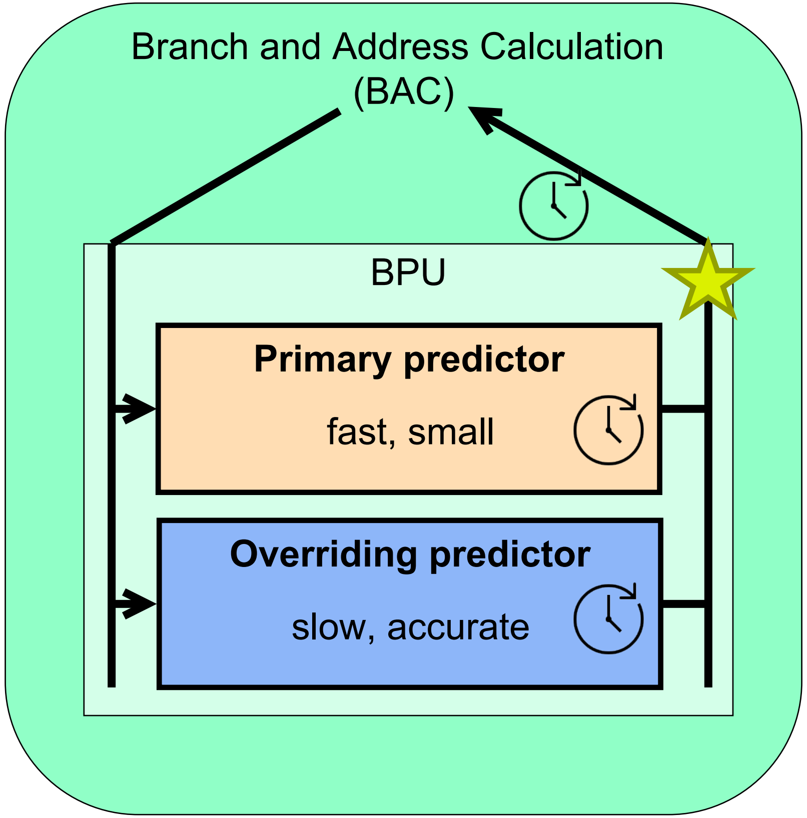


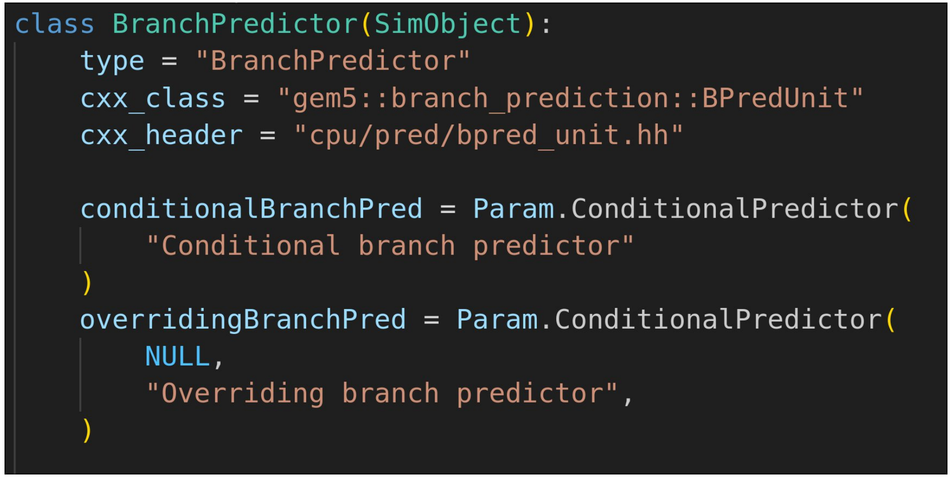
```
struct Prediction
{
    /** Whether the branch is predicted taken */
    bool taken;
    /** The latency that this prediction would normally take */
    Cycles latency;
};
```

```
class ConditionalPredictor(ClockedObject):
    type = "ConditionalPredictor"
    cxx_class = "gem5::branch_prediction::ConditionalPredictor"
    cxx_header = "cpu/pred/conditional.hh"

    latency = Param.Cycles(
        0, "Latency of the predictor (in cycles)"
    )
```

Overriding prediction in gem5

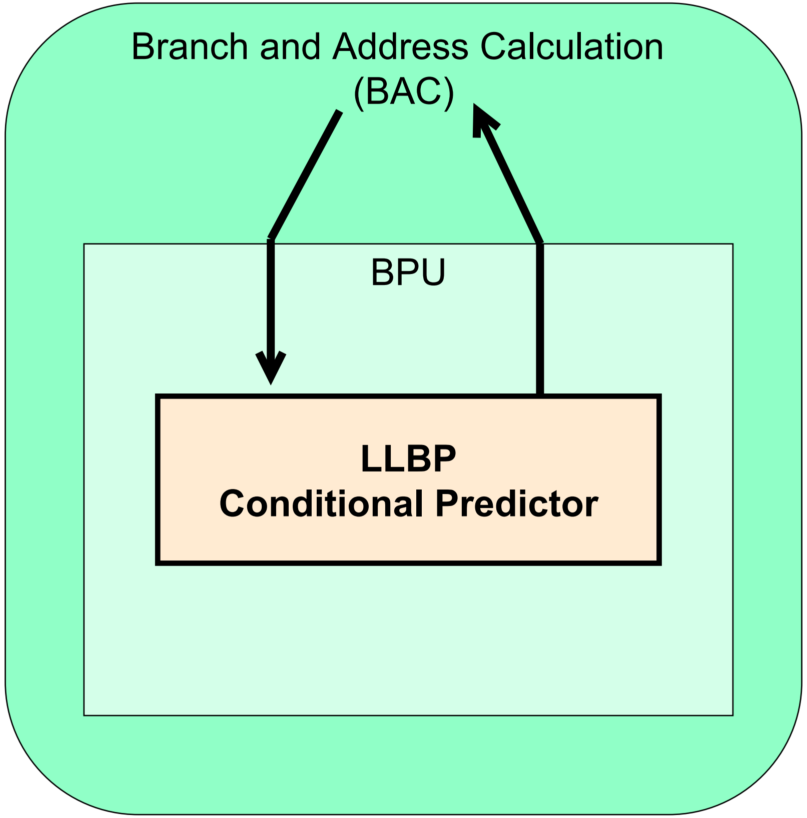


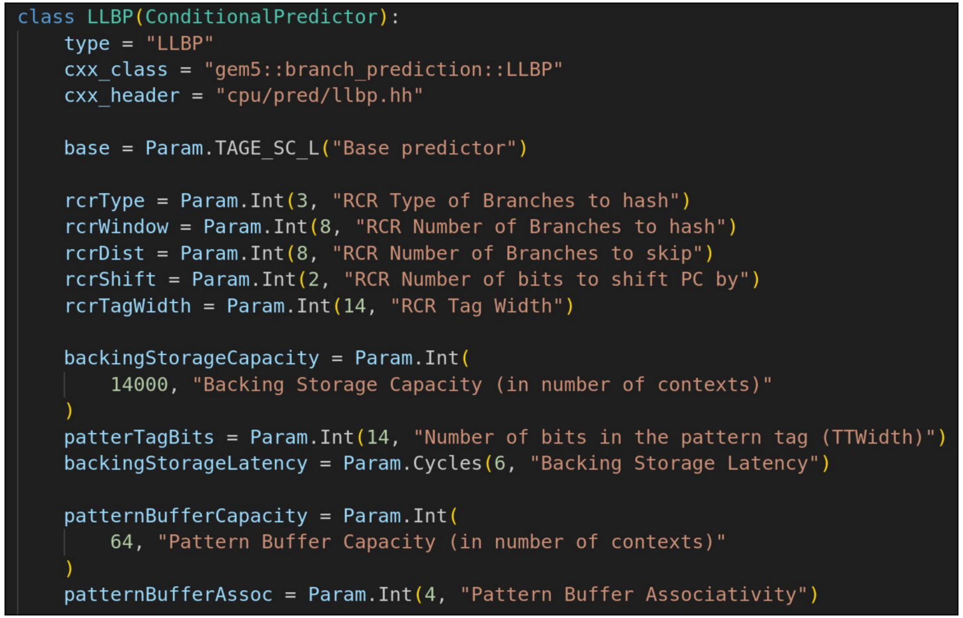


```
class BranchPredictor(SimObject):
    type = "BranchPredictor"
    cxx_class = "gem5::branch_prediction::BPredUnit"
    cxx_header = "cpu/pred/bpred_unit.hh"

    conditionalBranchPred = Param.ConditionalPredictor(
        "Conditional branch predictor"
    )
    overridingBranchPred = Param.ConditionalPredictor(
        NULL,
        "Overriding branch predictor",
    )
```

The Last Level Branch Predictor in gem5





```python
class LLBP(ConditionalPredictor):
    type = "LLBP"
    cxx_class = "gem5::branch_prediction::LLBP"
    cxx_header = "cpu/pred/llbp.hh"

    base = Param.TAGE_SC_L("Base predictor")

    rcrType = Param.Int(3, "RCR Type of Branches to hash")
    rcrWindow = Param.Int(8, "RCR Number of Branches to hash")
    rcrDist = Param.Int(8, "RCR Number of Branches to skip")
    rcrShift = Param.Int(2, "RCR Number of bits to shift PC by")
    rcrTagWidth = Param.Int(14, "RCR Tag Width")

    backingStorageCapacity = Param.Int(
        14000, "Backing Storage Capacity (in number of contexts)"
    )
    patterTagBits = Param.Int(14, "Number of bits in the pattern tag (TTWidth)")
    backingStorageLatency = Param.Cycles(6, "Backing Storage Latency")

    patternBufferCapacity = Param.Int(
        64, "Pattern Buffer Capacity (in number of contexts)"
    )
    patternBufferAssoc = Param.Int(4, "Pattern Buffer Associativity")
```

Outline

- ~~Motivation & Background~~
- ~~Design~~
- Evaluation
  - Parameters & Baseline
  - The impact of latency
  - Overriding branch prediction
  - The Last-Level Branch Predictor (LLBP)
- Conclusion

Configuration parameters for evaluation

- Base predictor: **TAGE-SC-L (state-of-the-art) with storage size from 64KiB - Infinite**

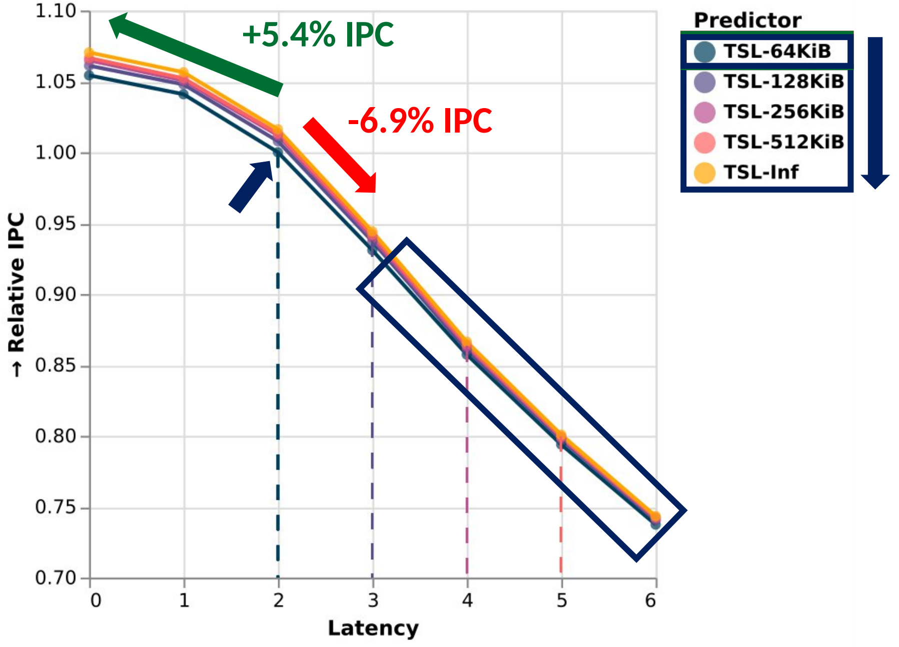
- Server workloads from the **Dacapo Benchmark Suite** [17] **and others** [16]
- **Warmup** for **100M** instructions, then **measuring** for **1B** instructions

- gem5 in **full system** mode on Arm ISA
- Simulated Processor: **O3 model** at 3GHz, 6-way OoO, 512 entry ROB
- Caches: 32KiB L1-I, 64KiB L1-D, 2MiB L2

RQ 1: The impact of latency

- **TAGE-SC-L (TSL)** with different sizes, latency sweep — **No latency mitigation**
  - **Baseline: TSL-64KiB with 2 cycles of latency**

IPC relative to TSL-64KiB with 2 cycles latency



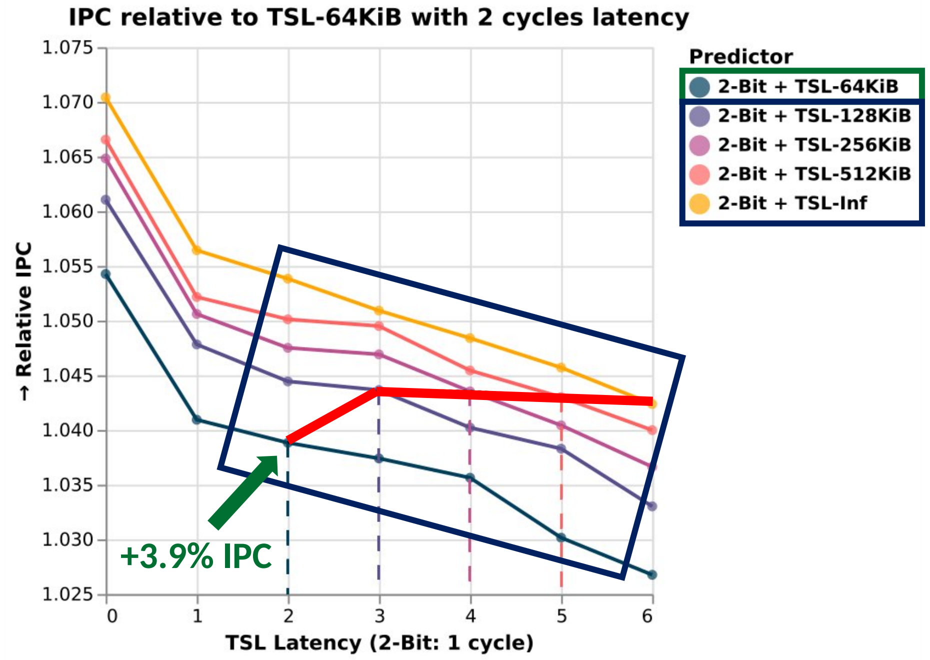
Latency has substantial performance impact

Eliminating latency can even benefit current predictor sizes of 64KiB

Increased storage capacity cannot offset latency — Mitigation is necessary

RQ 2: Overriding prediction

- **Small 2-bit predictor (fast) + TAGE-SC-L (accurate) — Overriding prediction**
    - **Fast predictor: 1 cycle of latency**



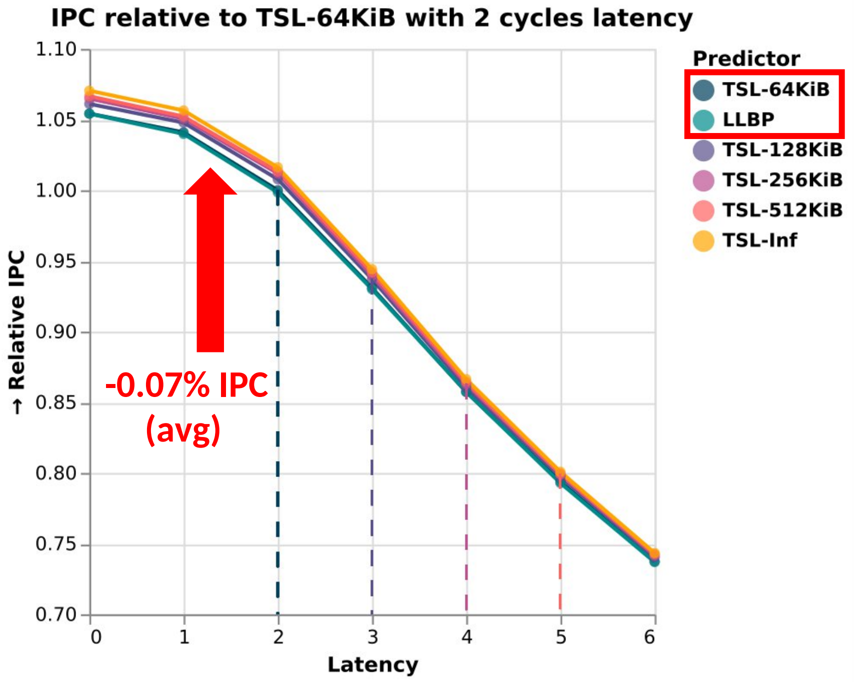
➡ **Overriding captures over 72% of the opportunity for latency elimination in the baseline predictor**

➡ **Increased storage capacity can now offset some latency**

➡ **Diminishing returns above 128KiB when assuming 2x storage = +1 cycle**

RQ 3: The Last-Level Branch Predictor (LLBP)

- **Pattern buffer + TSL-64KiB ("LLBP") – The Last-Level Branch Predictor**
  - **Same latency applied as unmodified TSL-64KiB**



➡ LLBP did not effectively mitigate latency

➡ Found an implementation bug near submission

Could not repeat the experiments due to time constraint

➡ Foundation for future work

Conclusion

1. How large is the impact of predictor latency on performance?

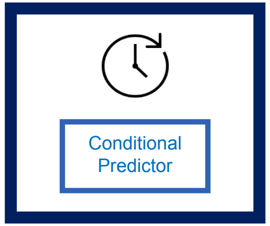
➡ Significant performance loss with state-of-the-art predictor up to 6.9% per cycle

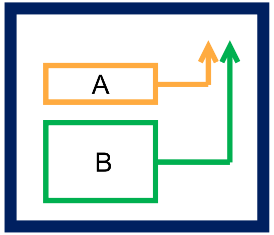
2. Can overriding prediction reduce this penalty?

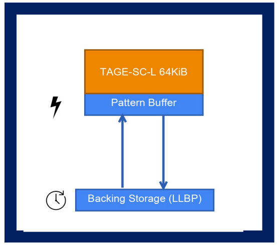
➡ Very effective, capturing over 72% of opportunity presented by reduced latency

3. Is a recent design, “The Last-Level Branch Predictor”, more effective?

➡ Performed worse than expected, implementation bug during evaluation







Contributions:

➡ Latency modeling framework for gem5 branch prediction

– Partially upstream, waiting for the decoupled frontend

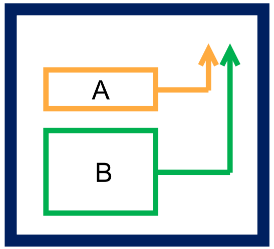
➡ Model of “The Last-Level Branch Predictor” for gem5

– Foundation for future research

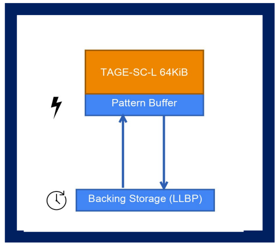
Future work

➡ **Repeat the experiments of "The Last-Level Branch Predictor"**

➡ **Use the latency framework to research a <u>combined latency mitigation strategy</u>**



+



References

- N. Adiga, J. Bonanno, A. Collura, M. Heizmann, B. R. Prasky and A. Saporito, "The IBM z15 High Frequency Mainframe Branch Predictor Industrial Product," 2020 ACM/IEEE 47th Annual International Symposium on Computer Architecture (ISCA), Valencia, Spain, 2020, pp. 27-39, doi: 10.1109/ISCA45697.2020.00014. [1]
- G. Ayers, J. H. Ahn, C. Kozyrakis and P. Ranganathan, "Memory Hierarchy for Web Search," 2018 IEEE International Symposium on High Performance Computer Architecture (HPCA), Vienna, Austria, 2018, pp. 643-656, doi: 10.1109/HPCA.2018.00061. [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]
- Anandtech.com, “Apple's Humongous CPU Microarchitecture”, Online: https://www.anandtech.com/show/16226/apple-silicon-m1-a14-deep-dive/2 [4]
- D. A. Jimenez, "Reconsidering complex branch predictors," The Ninth International Symposium on High-Performance Computer Architecture, 2003. HPCA-9 2003. Proceedings., Anaheim, CA, USA, 2003, pp. 43-52, doi: 10.1109/HPCA.2003.1183523. [5]
- André Seznec. TAGE-SC-L Branch Predictors Again. 5th JILP Workshop on Computer Architecture Competitions (JWAC-5): Championship Branch Prediction (CBP-5), Jun 2016, Seoul, South Korea. ⟨hal-01354253⟩ [6]
- André Seznec, Pierre Michaud. A case for (partially) tagged geometric history length branch prediction. The Journal of Instruction-Level Parallelism, 2006, 8, pp.23. ⟨hal-03408381⟩ [7]
- D. A. Jimenez, S. W. Keckler and C. Lin, "The impact of delay on the design of branch predictors," Proceedings 33rd Annual IEEE/ACM International Symposium on Microarchitecture. MICRO-33 2000, Monterey, CA, USA, 2000, pp. 67-76, doi: 10.1109/MICRO.2000.898059 [8]

References

- Jason Lowe-Power, Abdul Mutaal Ahmad, Adria Armejach, Adrian Herrera, Alec Roelke, et al.. The gem5 Simulator: Version 20.0+. 2021. ⟨hal-03100818⟩ [9]
- Irene Wang, Prasenjit Chakraborty, Zi Yu Xue, Yen Fu Lin, Evaluation of gem5 for performance modeling of ARM Cortex-R based embedded SoCs, Microprocessors and Microsystems, Volume 93, 2022, doi: 10.1016/j.micpro.2022.104599. [10]
- Gober, N., “The Championship Simulator: Architectural Simulation for Education and Competition”, arXiv e-prints, Art. no. arXiv:2210.14324, 2022. doi:10.48550/arXiv.2210.14324. [11]
- Lin, C.K. and Tarsa, S.J., “Branch prediction is not a solved problem: Measurements, opportunities, and future directions”, arXiv preprint, Art. no. arXiv:1906.08170, 2019. doi:10.48550/arXiv.1906.08170. [12]
- Svilen Kanev, Juan Pablo Darago, Kim Hazelwood, Parthasarathy Ranganathan, Tipp Moseley, Gu-Yeon Wei, and David Brooks. 2015. Profiling a warehouse-scale computer. In Proceedings of the 42nd Annual International Symposium on Computer Architecture (ISCA '15). Association for Computing Machinery, New York, NY, USA, 158–169. doi:10.1145/2749469.2750392. [13]
- T. A. Khan, M. Ugur, K. Nathella, D. Sunwoo, H. Litz, D. A. Jiménez, and B.Kasikci. “Whisper: Profile-Guided Branch Misprediction Elimination for Data Center Applications.” In: 2022 55th IEEE/ACM International Symposium on Microarchitecture (MICRO). 2022, pp. 19–34. doi: 10.1109/MICRO56248.2022.00017. [14]
- https://github.com/dhschall/gem5-fdp [15]
- https://github.com/dhschall/gem5-svr-bench [16]

References

- S. M. Blackburn, Z. Cai, R. Chen, X. Yang, J. Zhang, and J. Zigman. “Rethinking Java Performance Analysis.” In: Proceedings of the 30th ACM International Conference on Architectural Support for Programming Languages and Operating Systems, Volume 1, ASPLOS 2025, Rotterdam, Netherlands, 30 March 2025 - 3 April 2025. ACM, 2025. doi: 10.1145/3669940.3707217. [17]