Evaluating Branch Prediction Schemes with an Out-of-Order Core in gem5

Architecture for High Performance Computers / COL718

Arpit Prasad (2022EE11837)
ee1221837@iitd.ac.in

Devansh Panday (2022EE31583)
ee3221583@iitd.ac.in

September 10, 2025

1 Group Contributions

- **Arpit Prasad**
  - Extracted Results
  - Produced Report and Slides
- **Devansh Panday**
  - Configured all the Predictor.

2 Machine Configuration

- **CPU:** Intel(R) Core(TM) i5-1340P @ 3.70GHz
- **Cores:** 6 (12 threads)
- **L1d/L1i/L2/L3 Cache:** 448KiB / 640KiB / 9MiB / 12MiB
- **Memory:** 32 GB DDR4 @ 3200MHz
- **OS:** Ubuntu 22.04.1 LTS
- **gem5 Version:** [Specify the version or commit hash you used]

3 Background and Hypothesis

3.1 Background

Branch Prediction is a technique widely used to increase the IPC for modern day OOO Processors. The instructions to execute are fetched in blocks, however if one of the instruction turns out to be a branch, we face loss of IPC, since we would have to drop all the instructions ahead of that branch in the block of instrcution and fetch new intstruciton following after the Traget Branch.

3.2 Hypothesis

We hypothesize that more complex predictors like Tournament and Perceptron will significantly outperform simpler schemes like Bimodal, especially on the branch-heavy workload, leading to higher IPC. For the compute-heavy workload, we expect the difference in performance to be less pronounced, as branch mispredictions are a smaller bottleneck.

4 Experiment Methodology

4.1 Microarchitectural Parameters

The following parameters were kept constant across various iterations of the experiment:

1. Simulator: gem5 (version)
2. CPU Model: DerivO3CPU
3. ISA: X86
4. OOO core parameters

(a) ROB Size: 192 Instructions
(b) IQ entries: 64 Instructions
(c) Fetch Width: 4 Instructions
(d) Issue Width: 4 Instructions

4.2 Branch Predictors Evaluated

The following are the list of Branch Predictors that were used for Evaluation:

1. **BiModalBP** - This is a simple hardware level branch prediction mechnanism that uses two bit saturating counter to determine the surity of a branch prediction from previous prediction results
2. **GShareBP** - This is a dynamic Branch Predictor that takes into account for the Global History of Branch predictors along with the current program counter and makes prediction on the branch type
3. **LocalBP** - Similar to GShareBP, but uses a Local History Table instead of globally maintaining the last branch predictions
4. MultiperspectivePerceptron -
5. **TournamentBP**

4.3 Workloads

The following table tabulates the workloads that were experimented with and short description of why they are categorized either as Branch Heavy or Compute Heavy

Table 1: Workloads

Sl. No.	**Workload**	**Branch or Compute Heavy**	**Short Reason**
1	BasicMath	Compute Heavy	Lots of calulations
2	QuickSort	Branch Heavy	Many checks for Comparisions

4.4 Data Collection

The following are some key information in regards to the data collection process

1. Warmup strategy: The offset for Instructions was 2M for fast forwarding and 100M for checkpoints.
2. Region of Interest
3. Repetitions: The code was run three times to observer the variability of metrics across runs

5 Results

5.1 Summary Data

5.2 Graphical Analysis

Table 2: Performance Metrics for the *Branch-Heavy Workload*

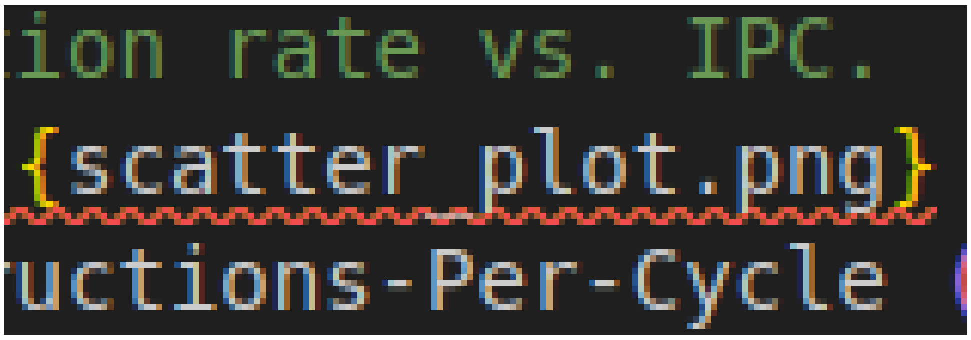
Predictor	IPC	Misprediction Rate (%)	Recovery Penalty (Cycles)
Baseline	0.42	0.00	20.0
BiModeBP	0.42	3.35	20.0
GShareBP	0.42	27.56	20.0
LocalBP	0.42	5.16	20.0
MultiperspectivePerceptron8KB	0.42	2.69	20.0
TournamentBP	0.42	3.02	20.0

Table 3: Performance Metrics for the *Compute-Heavy Workload*

Predictor	IPC	Misprediction Rate (%)	Recovery Penalty (Cycles)
Baseline	0.45	0.00	20.0
BiModeBP	0.45	2.04	20.0
GShareBP	0.45	31.60	20.0
LocalBP	0.45	3.99	20.0
MultiperspectivePerceptron8KB	0.45	0.87	20.0
TournamentBP	0.45	1.41	20.0

```
{ipc_plot.png}
ent Branch Predi
```

Figure 1: IPC Comparison Across Different Branch Predictors for Each Workload.



```
ion rate vs. IPC.
{scatter_plot.png}
uctions-Per-Cycle (
```

Figure 2: Misprediction Rate vs. Instructions-Per-Cycle (IPC).

6 Analysis and Discussion

7 Conclusion and Limitations

7.1 Conclusion

7.2 Limitations

References

[1] The gem5 Simulator System. http://www.gem5.org

[2] T-Y. Yeh and Y. N. Patt, "Two-Level Adaptive Branch Prediction," *in Proceedings of the 24th ACM/IEEE Annual International Symposium on Microarchitecture*, 1991.

A Run Script Example