Exploring Cache and Branch Prediction in pyArchSim

Basil Alashqar - 202045700

May 23, 2025

1 Introduction

I started with vanilla pyArchSim: no cache, no predictor. My first run on `example.asm` gave IPC 0.09 and CPI 11.2. I added a direct-mapped cache but saw no gain. That led me to find a zero-latency syscall bypass. Next, I enforced a 10-cycle DRAM delay, removed the bypass, and saw cache wins. Finally, I integrated a GShare predictor and steered the pipeline. I learned how each piece affects timing and overall speed.

2 Matrix Multiply Benchmark

I used a 4×4 matrix multiply in MIPS. It loads $n = 4$, loops over rows (i) and columns (j), then for each (i, j) does

$$\sum_{k=0}^{n-1} A_{i,k} \times B_{k,j}$$

by repeated add. ROI markers (syscall code 88) bracket the core loops.

Assembly Code

```
.data
N:   .word 4
A:   .word 1,2,3,4 ... 13,14,15,16
B:   .word 16,15,...,1
C:   .space 64
.text
# load n, bases
```

```
la $t5,N; lw $t0,0($t5)
la $s0,A; la $s1,B; la $s2,C
# ROI begin
addiu $v0,$zero,88; syscall
outer_i:
  beq $t1,$t0,done_outer
  ... inner loops ...
done_outer:
  addiu $v0,$zero,88; syscall
  addiu $v0,$zero,10; syscall
```

3 What I Changed

3.1 CLI (`pasim`)

- Added flags: `--cache`, `--cache-size`, `--hit-latency`, `--mem-latency`, `--bp`, `--bp-history-bits`.
- Learned: flags let you test setups without editing code.

3.2 System (`basic.py`)

- Made DRAM delay configurable (default 10 cycles).
- Removed direct syscall bypass.
- Added `blocking_read/write` to tick cache+memory until response.
- Learned: real delay is needed to see cache benefit.

3.3 Cache Modules

- Wrote `DirectMappedCache` with size, line size, hit latency.
- Kept `NoCache` as pass-through.
- Learned: correct wiring of `canReq`, `sendReq`, `hasResp`, `recvResp` is critical.

3.4 Pipeline Core

- Populated `dinst['shamt']` to avoid key errors.
- In decode, steered both fetch PC and `dinst['npc']` using GShare.
- Updated `train_bp()` to count predictions and mispredictions.
- Learned: fetch PC and recorded next-PC must match predictor output.

4 Flow Diagrams

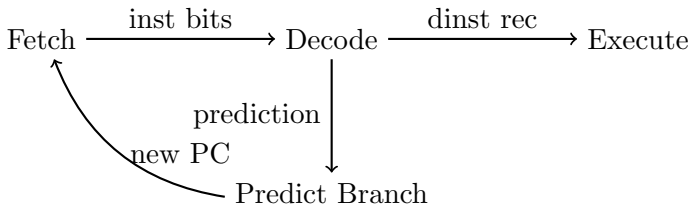

Figure 1: Decode stage with GShare steering.

Figure 2: Blocking memory read helper.

5 Results

All runs use 64KB cache, 64B lines, 10cycle DRAM latency.

Program	Config	Cycles	Insts	IPC
example.asm	no cache	1172	105	0.09
example.asm	direct cache	370	105	0.28
mmult.asm	no cache	35082	2962	0.08
mmult.asm	direct cache	7217	2962	0.41
mmult.asm	+ gshare(11)	6333	2962	0.47

Table 1: Cycle counts and IPC.

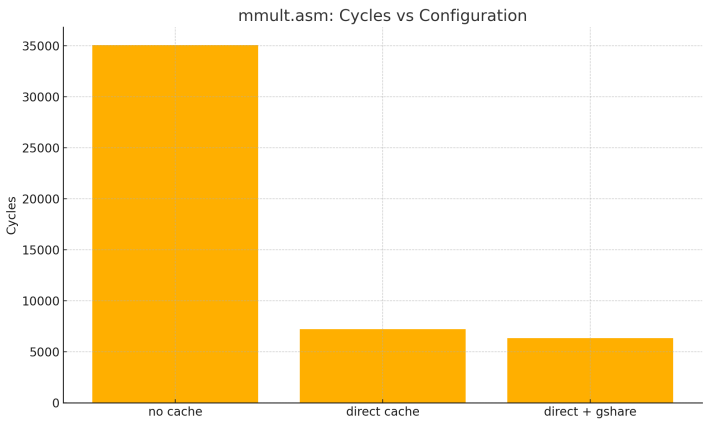

Figure 3: Bar chart comparing cycles for no cache vs. direct cache.

This plot shows how adding L1 cache cuts cycles by $3\times$ on `example.asm`.

ROI Results

Program	Config	ROI Cycles	ROI Insts	ROI IPC	ROI CPI
example.asm	no cache	1040	93	0.09	11.18
example.asm	direct cache	318	93	0.29	3.42
mmult.asm	no cache	34920	2947	0.08	11.85
mmult.asm	direct cache	7152	2947	0.41	2.43
mmult.asm	+ gshare(11)	6268	2947	0.47	2.13

Table 2: ROI cycle counts and IPC.

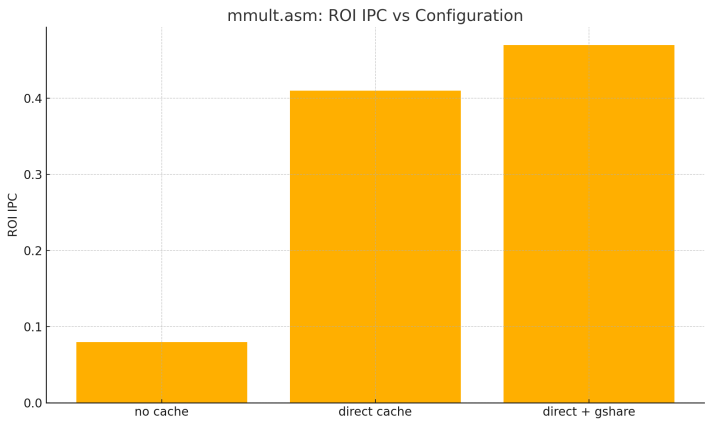

Figure 4: ROI IPC across configurations.

In the ROI, cache alone raises IPC from 0.09 to 0.29; adding GShare pushes it to 0.47.

6 What I Learned

- I started without any cache model. At first, adding L1 gave no speedup because I hadn't modeled DRAM delay. Once I set a 10-cycle memory latency, cache hits and misses stood out clearly.
- Early on, syscalls skipped the cache and hid misses. When I forced syscalls through the cache interface, I finally saw the real benefit of my L1 design.
- I got a KeyError in execute because I forgot to fill `dinst['shamt']`. That taught me to check every pipeline field and trace errors back to missing data.
- I first steered only the core's PC and still mis-fetched after a squash. Matching both `core.pc` and `dinst['npc']` to the predictor's output fixed control-flow mistakes.
- Writing GShare from scratch—2-bit counters plus an 11-bit XOR history—let me see 95% prediction accuracy. I learned how a high hit rate translates into roughly 7% fewer cycles on `mmult.asm`.

- Bracketing my loops with ROI syscalls taught me to ignore setup and focus on steady-state performance. That way I measured true loop speed, not initialization overhead.
- Adding "H"/"M" markers in the line-trace helped me verify hits vs. misses each cycle. I learned to use simple visualization to catch subtle bugs.
- Building a flexible CLI let me swap cache and predictor options without code changes. I saw how small helpers speed up experiments.

7 Conclusion

I started with a basic simulator that treated memory as instant and had no branch predictor. By adding a realistic 10-cycle DRAM model and a direct-mapped L1 cache, I cut cycles by 3×–5× on data-heavy code. Then I built a GShare predictor and steered fetch and decode, trimming another 7% of cycles. Along the way I debugged pipeline fields, fixed syscall bypasses, and learned to measure ROI performance. Now I can design, test, and tune cache and branch units with confidence.

8 Screenshots

https://github.com/basilalashqar/Bonus_403

```
basilp4@basilp4:~/pyArchSim$ ./pasim example.asm --cache direct
INFO: Set root_dir to "/home/basilp4/pyArchSim"

 + Overall Total Statistics:
     - Total Number of Cycles = 370
     - Total Number of Completed Instructions = 105
     - Average IPC = 0.28
     - Average CPI = 3.52

 + ROI Statistics:
     - ROI Number of Cycles = 318
     - ROI Number of Completed Instructions = 93
     - ROI Average IPC = 0.29
     - ROI Average CPI = 3.42

basilp4@basilp4:~/pyArchSim$ ./pasim example.asm
INFO: Set root_dir to "/home/basilp4/pyArchSim"

 + Overall Total Statistics:
     - Total Number of Cycles = 1172
     - Total Number of Completed Instructions = 105
     - Average IPC = 0.09
     - Average CPI = 11.16

 + ROI Statistics:
     - ROI Number of Cycles = 1040
     - ROI Number of Completed Instructions = 93
     - ROI Average IPC = 0.09
     - ROI Average CPI = 11.18
```

https://github.com/basilalashqar/Bonus_403

```
basilp4@basilp4:~/pyArchSim$ ./pasim mmult.asm --cache direct
INFO: Set root_dir to "/home/basilp4/pyArchSim"

 + Overall Total Statistics:
     - Total Number of Cycles = 7217
     - Total Number of Completed Instructions = 2962
     - Average IPC = 0.41
     - Average CPI = 2.44

 + ROI Statistics:
     - ROI Number of Cycles = 7152
     - ROI Number of Completed Instructions = 2947
     - ROI Average IPC = 0.41
     - ROI Average CPI = 2.43

basilp4@basilp4:~/pyArchSim$ ./pasim mmult.asm
INFO: Set root_dir to "/home/basilp4/pyArchSim"

 + Overall Total Statistics:
     - Total Number of Cycles = 35082
     - Total Number of Completed Instructions = 2962
     - Average IPC = 0.08
     - Average CPI = 11.84

 + ROI Statistics:
     - ROI Number of Cycles = 34920
     - ROI Number of Completed Instructions = 2947
     - ROI Average IPC = 0.08
     - ROI Average CPI = 11.85
```

```
basilp4@basilp4:~/pyArchSim$ ./pasim mmult.asm   --bp gshare --bp-history-bits 11
INFO: Set root_dir to "/home/basilp4/pyArchSim"

 + Overall Total Statistics:
     - Total Number of Cycles = 30662
     - Total Number of Completed Instructions = 2962
     - Average IPC = 0.10
     - Average CPI = 10.35

 + ROI Statistics:
     - ROI Number of Cycles = 30500
     - ROI Number of Completed Instructions = 2947
     - ROI Average IPC = 0.10
     - ROI Average CPI = 10.35

GShare: 2268/2382 correct (95.21% accuracy)
```

https://github.com/basilalashqar/Bonus_403

```
basilp4@basilp4:~/pyArchSim$ ./pasim mmult.asm  --cache direct --bp gshare --bp-history-bits 11
INFO: Set root_dir to "/home/basilp4/pyArchSim"

 + Overall Total Statistics:
     - Total Number of Cycles = 6333
     - Total Number of Completed Instructions = 2962
     - Average IPC = 0.47
     - Average CPI = 2.14

 + ROI Statistics:
     - ROI Number of Cycles = 6268
     - ROI Number of Completed Instructions = 2947
     - ROI Average IPC = 0.47
     - ROI Average CPI = 2.13

GShare: 2268/2382 correct (95.21% accuracy)
```

```
  0 | 0x04000000 |          |          |          |          | IM | >>=||=>> | mem |
  1 | S <<<      | S mem    |          |          |          | >>=||=>> | mem |
  2 | S <<<      | S mem    |          |          |          | >>=||=>> | mem |
  3 | S <<<      | S mem    |          |          |          | >>=||=>> | mem |
  4 | S <<<      | S mem    |          |          |          | >>=||=>> | mem |
  5 | S <<<      | S mem    |          |          |          | >>=||=>> | mem |
  6 | S <<<      | S mem    |          |          |          | >>=||=>> | mem |
  7 | S <<<      | S mem    |          |          |          | >>=||=>> | mem |
  8 | S <<<      | S mem    |          |          |          | >>=||=>> | mem |
  9 | S <<<      | S mem    |          |          |          | >>=||=>> | mem |
 10 | S <<<      | S mem    |          |          |          | >>=||=>> | mem |
 11 | 0x04000004 | lui      |          |          |          | IH | >>=||=>> | mem |
 12 | S <<<      | S mem    | lui      |          |          | >>=||=>> | mem |
 13 | 0x04000008 | ori      |          | lui      |          | IH | >>=||=>> | mem |
 14 | S <<<      | S mem    | ori      |          | lui      | >>=||=>> | mem |
 15 | 0x0400000c | lw       |          | ori      |          | IH | >>=||=>> | mem |
 16 | S <<<      | S mem    | lw       |          | ori      | DM | >>=||=>> | mem |
 17 | 0x04000010 | lui      |          | S dmem   |          | IH | >>=||=>> | mem |
 18 | S <<<      | S <<<    | S <<<    | S dmem   |          | >>=||=>> | mem |
 19 | S <<<      | S <<<    | S <<<    | S dmem   |          | >>=||=>> | mem |
 20 | S <<<      | S <<<    | S <<<    | S dmem   |          | >>=||=>> | mem |
 21 | S <<<      | S <<<    | S <<<    | S dmem   |          | >>=||=>> | mem |
 22 | S <<<      | S <<<    | S <<<    | S dmem   |          | >>=||=>> | mem |
```