Computer Architecture Assignment-1

Sohan Patidar, Megha Patidar

1. Introduction

Branch predictors are a critical component in modern processors, directly impacting instruction throughput and overall performance. It is a hardware mechanisms that predict the outcome of conditional branches in a program before they are actually resolved in a computer program. They are used to improve the performance of modern processors by reducing delays in program execution.

In this experiment we will aim to evaluate the performance of different branch predictors on various benchmarks. The different predictors that we are going to implement for various benchmarks are the bimodal predictor, G-share predictor, perceptron predictor, three variants of the TAGE (Tagged Geometric History Length) predictor and Hybrid predictor(G-share and TAGE predictor).

Bimodal Predictor is a simple branch predictor that uses a table of 2-bit saturating counters indexed by the branch address to predict whether a branch will be taken or not, based on past outcomes.

G-Share Predictor is a predictor that combines global branch history with the branch address using an XOR operation on PC and BHR to index into a table of counters, allowing for more context-sensitive predictions.

Perceptron Predictor is a machine learning-based predictor that uses a perceptron (a simple neural network) to predict branches by examining their weights, i.e., the correlation that they learn, it is easy to understand the decisions that they make.

TAGE Predictor (Tagged Geometric History Length Predictor) is a sophisticated predictor that uses multiple tables, each with different history lengths, to predict branches. It relies on a default tagless predictor backed with a plurality of (partially) tagged predictor components indexed using different history lengths for index computation. These history lengths form a geometric series.

ChampSim is an architectural simulator designed for research and educational purposes, focusing on the simulation of modern CPU components like branch predictors, cache hierarchies, and prefetchers. ChampSim simulates a heterogeneous multicore system with an arbitrary memory hierarchy, where each out-of-order core can be configured arbitrarily.

Benchmarks are standardized tests or sets of programs used to evaluate the performance of hardware or software systems. By running benchmarks, systems can be compared in a controlled manner to assess their efficiency, speed, and overall capability. In this experiment, we have used three **Cloudsuite benchmarks** to evaluate the performance of the branch predictors. Cloudsuite benchmarks focus on modern, real-world cloud-based applications and workloads, such as data analytics, web services, and big data processing. We have also used one of the given optional benchmarks, 631.deepsjeng_s-928B.champsimtrace.xz for the experiment.

This report will present the details about how this branch predictors have been implemented in 64KB size using ChampSim simulator.

2. Implementation

We have implemented the given experiment in three parts. It started from implementing the default predictors provided in ChampSim, which are bimodal predictor, G-share predictor, perceptron predictor. After evaluating these, we proceeded to build a custom TAGE predictor by integrating open-source code, creating three different versions of this predictor to explore variations in performance. Finally we have developed hybrid predictor by combining G-share and TAGE predictor, allocating memory equally between the two (50:50 ratio) used predictors. All these predictors have been built using a memory space of 64KB for their table sizes.

2.1 Installation

To implement the above mentioned predictors, first we need to install the ChampSim simulator. We have installed the simulator by cloning its git file from https://github.com/casperIITB/ChampSim in our system. We used a different version of ChampSim than the one mentioned in the file for this experiment. The implementation and evaluation followed the same experimental structure, but with the modifications according to this version of the simulator. We have used this version because given simulator was giving same output for all the predictors.

The benchmarks used in this experiment were obtained via a Dropbox link that was provided through an Outlook email. The email contained a pre-defined set of Cloudsuite benchmark traces- cassandra_phase5_core0.trace.xz, streaming_phase0_core1.trace.xz, cloud9_phase5_core3.trace.xz.

We have also used the optional benchmark- 631.deepsjeng_s-928B.champsimtrace.xz from : https://dpc3.compas.cs.stonybrook.edu/champsim-traces/speccpu/

2.2 Working

The workings of the bimodal, G-share, and perceptron branch predictors are provided within the simulator itself. These default implementations are included as part of the ChampSim codebase, allowing for direct evaluation of these branch prediction strategies. We have used the commands required to compile and run them with the required benchmark.
This ChampSim version takes seven parameters: Branch predictor, L1I prefetcher, L1D prefetcher, L2C prefetcher, LLC prefetcher, LLC replacement policy, and the number of cores.
//compilation
./build_champsim.sh [BRANCH_PREDCITOR] [L1I_PREFETCHER][L1D_PREFETCHER][L2C_PREFETCHER][LLC_PREFETCHER][LLC_REPLACEMENT][NUM_CORE]
For example, ./build_champsim.sh gshare no no no next_line lru 1, it builds a single-core processor with G-share branch predictor, no L1 instruction prefetcher, no L1/L2 data prefetchers, ip-stride LLC prefetcher and the baseline LRU replacement policy for the LLC.

We can run the branch predictors with the help of a script file or can run them directly with the help of command.
//run command
./run_champsim.sh [BINARY] [N_WARM] [N_SIM] [TRACE] [OPTION]
For example, ./run_champsim.sh bimodal-no-no-no-next_line-lru-1core 1 10 trace_name

We have used a script file for implementing all the branch predictors all together on the given benchmarks at once and stored their output in a txt file generated with the help of the script file(provide with the tar file).
//command for compiling in script file
./build_champsim.sh $predictor no no no next_line lru 1

//command for executing
(./bin/"$predictor"-no-no-no-next_line-lru-1core -warmup_instructions 200000000 -simulation_instructions 500000000 -c -traces ${TRACE_DIR}/${trace}) >> $output_file 2>&1

We have used 200 millions warmup instructions and 500 millions simulation instructions for all the predictions.

The default configurations (provided by the cloned ChampSim simulator) of bimodal, G-share, and perceptron branch predictors are not 64KB. So we were required to modify the configurations of the branch prediction table sizes for each predictor during the implementation. This step was crucial to ensure that all the predictors (bimodal, G-share, perceptron, and TAGE) adhered to the hardware storage budget of approximately 64KB (±2KB).

2.2.1. Bimodal predictor

The default bimodal predictor of ChampSim simulator is of size 6KB(it has 16384 entries in table, each of size 3 bits). So to transform it into a 64KB branch predictor we have used 262144 entry size table, with bimodal prime of 262139 and max counter bits is equals to 2.

2.2.2. G-share Predictor

The default G-share predictor of ChampSim simulator is of size 4KB(it has 16384 entries in table, each of size 2 bits). So to scale the G-share predictor to a 64KB branch predictor, we increased the table size to 262144 entries, while keeping the entry size at 2 bits. All other configurations remain unchanged.

2.2.3. Perceptron Predictor

The default Perceptron predictor of ChampSim simulator has a table size of approximately equals to 4KB. To scale the perceptron predictor to 64KB, we adjusted only the number of perceptrons, increasing it to 16280 while keeping all other configurations unchanged.

2.2.4. TAGE predictor

There is no default TAGE predictor in the ChampSim simulator. So we have used an open source TAGE predictor and integrated it with our version of simulator. We have used tage.bpred and tage.h file from https://github.com/KanPard005/RISCY_V_TAGE.
This predictor is using total 12 different components of TAGE, but we need to use only four as mentioned in the experiment. So we had made some changes so that it will use four components which result into approximately 64KB size table.
Changed TAGE_NUM_COMPONENTS from 12 to 4 for all the versions. Other changes are mentioned below:
For (a) part,
TAGE_MAX_INDEX_BITS 15
TAGE_INDEX_BITS[TAGE_NUM_COMPONENTS] = {14, 14, 13, 12}
TAGE_TAG_BITS[TAGE_NUM_COMPONENTS] = {7, 6, 5, 4}

Moving forward, for (b) part, in which we have to create some versions by exploring a different set of history lengths and different table sizes for the TAGE predictor, but keeping the storage budget for the predictor to 64KB .

Version 1:

TAGE_MAX_INDEX_BITS 15
TAGE_INDEX_BITS[TAGE_NUM_COMPONENTS] = {15, 13, 12, 10}
TAGE_TAG_BITS[TAGE_NUM_COMPONENTS] = {6, 5, 4, 4}

Version 2:

TAGE_MAX_INDEX_BITS 14
TAGE_INDEX_BITS[TAGE_NUM_COMPONENTS] = {14, 13, 13, 13}

TAGE_TAG_BITS[TAGE_NUM_COMPONENTS] = {10, 6, 5, 4}

Version 3:

TAGE_MAX_INDEX_BITS 15

TAGE_INDEX_BITS[TAGE_NUM_COMPONENTS] = {15, 11, 10, 10}

TAGE_TAG_BITS[TAGE_NUM_COMPONENTS] = {9, 7, 5, 4}

2.2.5. Hybrid Predictor

As for the third part(c) of the experiment, we need to create a hybrid predictor using G-share and TAGE predictor with a total storage of ~64KB (for the two predictors), such that the relative storage allocated to the individual predictors (g-share and TAGE) can be varied as 25:75, 50:50, or 75:25.

These variations have be achieved by following configuration:

2.2.5.1. 25:75 ratio

```
//G-share will get 16KB
GS_HISTORY_TABLE_SIZE = 65536;
//TAGE will get 48KB
TAGE_INDEX_BITS[TAGE_NUM_COMPONENTS] = {14, 13, 12, 12};
TAGE_TAG_BITS[TAGE_NUM_COMPONENTS] = {6, 6, 6, 5};
```

2.2.5.2. 50:50 ratio

```
//G-share will get 32KB
GS_HISTORY_TABLE_SIZE = 131072;
//TAGE will get 32KB
TAGE_INDEX_BITS[TAGE_NUM_COMPONENTS] = {13, 13, 11, 11};
TAGE_TAG_BITS[TAGE_NUM_COMPONENTS] = {6, 7, 5, 5};
```

2.2.5.2. 75:25 ratio

```
//G-share will get 48KB
GS_HISTORY_TABLE_SIZE = 196608;
//TAGE will get 16KB
TAGE_INDEX_BITS[TAGE_NUM_COMPONENTS] = {12, 12, 11, 10};
TAGE_TAG_BITS[TAGE_NUM_COMPONENTS] = {5, 5, 4, 4};
```

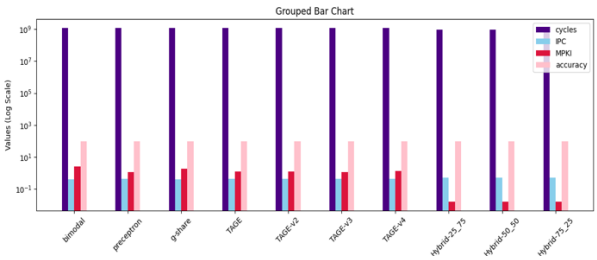
This predictor only was run on the ChampSim simulator which was provided in the question file. We have tried our best to make it run on the ChampSim version that is mentioned above and have been used for all other versions but it didn't work out. We will try to make it run on that version even after the submission to learn and improve ourselves.

3. Observation

We will analyse the performance of various branch predictors by comparing them across multiple benchmarks. This comparison will allow us to observe how each predictor behaves under different workload conditions, highlighting strengths and weaknesses relative to specific benchmark types.

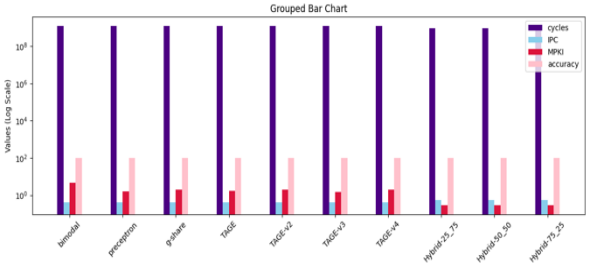
3.1. Cassandra_phase5_core0.trace.xz

trace	type	cycles	IPC	MPKI	accuracy
cassandra	bimodal	1198367812	0.417234	2.60625	97.8948
cassandra	preceptron	1171516422	0.426797	1.15828	99.0644
cassandra	g-share	1193667995	0.418877	1.94908	98.4257
cassandra	TAGE	1174415377	0.425744	1.21786	99.0163
cassandra	TAGE-v1	1175076871	0.425504	1.30092	98.9492
cassandra	TAGE-v2	1173138243	0.426207	1.18317	99.0443
cassandra	TAGE-v3	1175769809	0.425253	1.32583	98.9291
cassandra	Hybrid-25_75	972293142	0.5142	0.0157	99.41
cassandra	Hybrid-50_50	972292817	0.5142	0.0157	99.41
cassandra	Hybrid-75_25	972292817	0.5142	0.0157	99.41



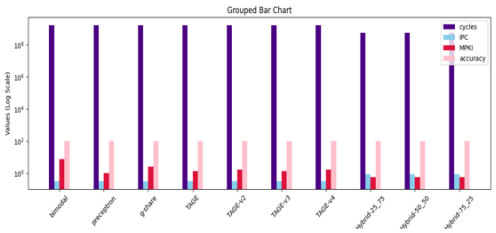
3.2. Cloud9_phase5_core3.trace.xz

trace	type	cycles	IPC	MPKI	accuracy
streaming	bimodal	1243617213	0.402053	4.6122	97.3394
streaming	preceptron	1219767574	0.409914	1.5727	99.0928
streaming	g-share	1226057265	0.407811	2.04094	98.8227
streaming	TAGE	1221458448	0.409347	1.65903	99.043
streaming	TAGE-v1	1225074721	0.408138	1.95435	98.8726
streaming	TAGE-v2	1220009683	0.409833	1.50622	99.1311
streaming	TAGE-v3	1225692821	0.407933	2.01093	98.84
streaming	Hybrid-25_75	910590862	0.5491	0.2788	97.93
streaming	Hybrid-50_50	910590862	0.5491	0.2788	97.93
streaming	Hybrid-75_25	910590862	0.5491	0.2788	97.93



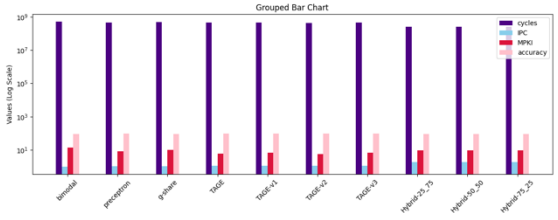
3.3. Cloud9_phase5_core3.trace.xz

trace	type	cycles	IPC	MPKI	accuracy
cloud	bimodal	1666236051	0.300078	7.53317	96.539
cloud	preceptron	1598795922	0.312735	0.982664	99.5485
cloud	g-share	1615572619	0.309488	2.43358	98.8819
cloud	TAGE	1597883721	0.312914	1.349	99.3802
cloud	TAGE-v1	1600966419	0.312311	1.68078	99.2278
cloud	TAGE-v2	1597701229	0.31295	1.32303	99.3921
cloud	TAGE-v3	1599351245	0.312627	1.65336	99.2404
cloud	Hybrid-25_75	581943321	0.8592	0.544	97.11
cloud	Hybrid-50_50	581943321	0.8592	0.544	97.11
cloud	Hybrid-75_25	581943321	0.8592	0.544	97.11



3.4. 631.deepsjeng_s-928B.champsimtrace.xz

trace	type	cycles	IPC	MPKI	accuracy
deepsjeng	bimodal	518215118	0.96485	13.8829	90.6168
deepsjeng	preceptron	465223799	1.07475	8.55407	94.2184
deepsjeng	g-share	476015672	1.05039	9.99821	93.2424
deepsjeng	TAGE	440163337	1.13594	5.89075	96.0185
deepsjeng	TAGE-v1	448450584	1.11495	6.76041	95.4307
deepsjeng	TAGE-v2	437995330	1.14156	5.67564	96.1639
deepsjeng	TAGE-v3	450132936	1.11078	6.93107	95.3154
deepsjeng	Hybrid-25_75	257612763	1.941	9.761	93.4
deepsjeng	Hybrid-50_50	257222732	1.944	9.71	93.44
deepsjeng	Hybrid-75_25	257222732	1.944	9.71	93.44



4. Observation and Result

We have tried to compare different branch predcitors on various benchmarks (631.deepsjeng_s-928B.champsimtrace.xz, cassandra_phase5_core0.trace.xz, cloud9_phase5_core3.trace.xz, streaming_phase0_core1.trace.xz), several trends emerge, which we have mentioned in the above charts and graph.

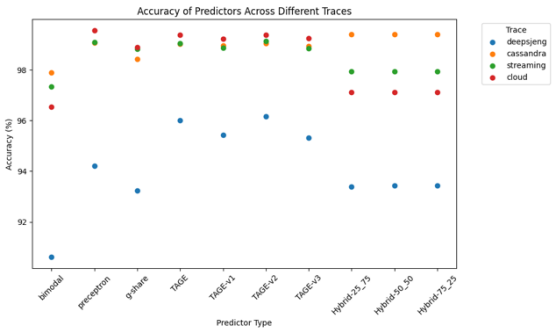
The TAGE family of predictors demonstrates strong performance, achieving high accuracy and efficiency, making it a reliable choice across all traces.

For the deepsjeng trace, hybrid predictors outshine others in terms of overall performance(but it was run on the simulator mentioned in the file, so not sure about correctness) , but TAGE predictors also show impressive results. In cassandra, simpler predictors like bimodal and g-share lag behind TAGE and preceptron. The streaming trace reveals a similar pattern, TAGE predictors performing better across all metrics. Lastly, in the cloud TAGE outperforms other predictors

In result we can conclude that the more complex TAGE predictors provide consistently better results compared to the simpler bimodal. g-share predictors and also the perceptron, making it the preferred options for achieving higher performance across the different traces.

But as for the Hybrid predictor, we cannot conclude anything for sure. It gave a slightly different output on the https://github.com/ChampSim/ChampSim for the deepsjeng trace.

Following is the graph on the accuracy of different predictors under various benchmark traces:-



The results show varying levels of accuracy across different predictors for each benchmark trace. In general, the perceptron-based predictors demonstrate high accuracy across most traces, outperforming simpler predictors like bimodal and g-share versions. The TAGE family of predictors, especially its various versions, also performs strongly, providing reliable and consistent results with only slight fluctuations between versions. In contrast, benchmarks like Deepsjeng and Cloud see a bit more variability across different predictor types but overall the high-performance of predictors continue to dominate. The simpler, earlier predictors tend to have lower accuracy compared to these advanced, adaptive techniques. And as for Hybrid predictor we are not sure what to conclude.

5. References

[1] N. Gober, et. al., “The Championship Simulator: Architectural Simulation for Education and Competition”, https://arxiv.org/pdf/2210.14324.pdf

[2] https://github.com/casperIITB/ChampSim

[3] https://github.com/ChampSim/ChampSim

[4] https://github.com/KanPard005/RISCY_V_TAGE

[5]https://github.com/chinmayachaudhry/Branch-Predictor

[6] https://chatgpt.com/

[7] https://google.com/