COL718 Assignment - 1

B.V.N.S.Aditya (2019CS50471)

5th Sep, 2022

Contents

1 Part 1: Implement a Branch Predictor

1.a Budget: 2400

I have implemented GShare predictor, which takes 10 LSB from the PC. The size of global History Register is 10 bits. Hence, the size of predictor table is 2^{10}. Each entry in the predictor table is of 2 bits, representing a saturating counter.

Train:

1. First we XOR the 10 LSB with the GHR and then index into the Predictor table
2. If the branch is taken, the counter gets incremented by 1. Else it gets decremented by 1
3. Then, we update the GHR with the outcome

Predict:

1. First we XOR the 10 LSB with the GHR and then index into the Predictor table
2. If the value at that index is 0/1, we predict that branch is not-taken. Else we predict that branch is taken

Size of Predictor: 2058
Average Accuracy: 96.38

1.b Budget: 6400

I have implemented a Tournament Predictor. The two predictors I considered are GShare, PAp (Modified).

1.b.1 PAp Predictor

This takes 9 LSB from PC. The size of GHR table is 2^6. Each entry in GHR table is 3 bits. The size of Predictor table is 2^9, each entry in this table is of size 2 bits.
Train:

1. We index into the GHR table with 6 LSB of the PC
2. Then, we XOR the value at that index with 10 LSB of PC and store the output in 'val'. (This step is different from PAp where we concatenate both the bit strings). Then, the value at this index of GHR table gets updated based on the outcome
3. We index into the predictor table using the 'val' computed in previous step and then if the branch is taken, the counter gets incremented by 1. Else it gets decremented by 1

Predict:

1. We index into the GHR table with 6 LSB of the PC
2. Then, we XOR the value at that index with 10 LSB of PC and index into the predictor table using this output.
3. If the value at that index is 0/1, we predict that branch is not-taken. Else we predict that branch is taken

1.b.2 GShare Predictor

This takes 11 LSB from PC. The size of GHR is 11 bits. The size of Predictor table is 2^{11}, each entry in this table is of size 2 bits. The predict and training steps of GShare are as mentioned in section Budget: 2400

1.b.3 Choice Table

The size of choice table is 2^9, each entry is of size 2 bits.
Train:

1. We index into the choice table using 9 LSB of the PC
2. If both the predictors predict the same, then we won't train the table
3. If GShare predicted the correct outcome, then we decrement the counter in the choice table. Else we increment the counter in the choice table

Predict:

1. We index into the choice table using 9 LSB of the PC
2. If the value at that index is 0/1, we use GShare to predict. Else, we use PAp to predict the outcome

Size of Predictor: 6347
Average Accuracy: 97.66

1.c Budget: 9999

I have implemented a Tournament Predictor. The two predictors I considered are GShare, PAp (Modified). I have implemented Perceptron based predictor but the average accuracy is less than that of the tournament predictor. So, I submitted the tournament predictor.

1.c.1 PAp Predictor

This takes 9 LSB from PC. The size of GHR table is 2^8. Each entry in GHR table is 3 bits. The size of Predictor table is 2^9, each entry in this table is of size 2 bits. The predict and training steps of PAp are as mentioned in section Budget: 6400

1.c.2 GShare Predictor

This takes 11 LSB from PC. The size of GHR is 11 bits. The size of Predictor table is 2^{11}, each entry in this table is of size 2 bits. The predict and training steps of GShare are as mentioned in section Budget: 2400

1.c.3 Choice Table

The size of choice table is 2^9, each entry is of size 2 bits. The predict and training steps of Choice table are as mentioned in section Budget: 6400

Size of Predictor: 9995
Average Accuracy: 97.7

1.d Budget: 32000

I have implemented a Tournament Predictor. The two predictors I considered are GShare, PAp (Modified).

1.d.1 PAp Predictor

This takes 11 LSB from PC. The size of GHR table is 2^8. Each entry in GHR table is 3 bits. The size of Predictor table is 2^{11}, each entry in this table is of size 2 bits. The predict and training steps of PAp are as mentioned in section Budget: 6400

1.d.2 GShare Predictor

This takes 13 LSB from PC. The size of GHR is 13 bits. The size of Predictor table is 2^{13}, each entry in this table is of size 2 bits. The predict and training steps of GShare are as mentioned in section Budget: 2400

1.d.3 Choice Table

The size of choice table is 2^{12}, each entry is of size 2 bits. The predict and training steps of Choice table are as mentioned in section Budget: 6400

Size of Predictor: 29453
Average Accuracy: 98.11

2 Part 2: Comparison with ML Algorithm

Classifier Used: Random Tree
Test Options: Use Training Set

Trace-1:
Accuracy: 92%
Confusion Matrix:

True/Classified	a	b
a = Not Taken	1665336	135423
b = Taken	41493	371393

Trace-2:
Accuracy: 91.84%
Confusion Matrix:

True/Classified	a	b
a = Taken	343990	22877
b = Not Taken	123432	1302507

Trace-3:
Accuracy: 98.7%
Confusion Matrix:

True/Classified	a	b
a = Taken	1356472	2711
b = Not Taken	17407	170178

Trace-4:
Accuracy: 95.44%
Confusion Matrix:

True/Classified	a	b
a = Taken	802033	1245
b = Not Taken	39567	52968

Trace-5:
Accuracy: 79.84%
Confusion Matrix:

True/Classified	a	b
a = Not Taken	577802	447923
b = Taken	40275	1356020

The accuracy of the ML Algorithm is less than the algorithms implemented in Part 1. For trace 3, predictions are skewed towards "Taken" class. For trace 5, the predictions are skewed towards "Not Taken" class. In other cases, the predictions are not very skewed.

3 Part 3: The Tejas simulator's working

I have installed Tejas and have gone through PATMOS paper. Tejas is an open-source, Java-based multi-core architectural simulator, which is platform-independent. Binaries in different ISAs can be simulated using Tejas. On a high-level, Tejas consists of an emulator, which emulates the execution of the program and generates traces. These traces contains information about the instructions executed, load/store addresses, branch outcomes. These traces are sent to the timing simulator via transfer engine. Instructions are translated to VISA in the simulator. These stream of instructions are taken as input, pipelines are simulated. At the end of the simulation, statistics such as IPC, miss rates, power consumption rates are reported.

Emulator:
Tejas uses Intel Pin (downloaded during installation) to emulate x86 binaries and other emulators such as QEMU, GPU Ocelot, Jikes RVM. These emulators send the events of interest such as instruction type, operands, load/store addresses, branch outcomes to the simulator encoded in packets.

Transfer Engine:
The communication between Emulator and Simulator can be done by Shared Memory, Files, Pipes, Network Sockets in Tejas.

Simulator:
First, instructions in different ISAs are translated into VISA (Virtual Instruction Set Architecture) which is a RISC ISA. Then Tejas uses a semi event driven model to simulate hardware. For activities like decoding which have predictable latency, it uses an iterative approach where each structure is advanced through one cycle in each iteration. For activities like load/store which have unpredictable latency, it uses a priority queue where for each cycle we deque the instructions scheduled for that cycle and process them. Tejas uses a multi issue inorder pipeline and a complex out of order pipeline. The stages of pipeline are simulated in out of order to ensure correctness. The emulator merely informs simulator if the branch was taken or not. If simulator predicted the branch wrong, the pipeline will be stalled. So, many branch predictors such as Always taken, Always not taken, Bi-modal, GAg, GAp, PAg, PAp, GShare, TAGE and tournament predictors are implemented in Tejas.