Writing Assignment – 1

Rishabh Bhatt

Introduction:
User code often has branches. These branches may be conditional, unconditional, return etc. In order for a CPU to work efficiently it needs to handle these control dependencies and ensure that the pipeline is always full with the correct order of instructions. There are various ways in which CPU handles control dependencies: stalling the pipeline, branch prediction, belayed branching, predicated execution, multi-threading, etc.

1. Branch Prediction and its properties
In branch prediction the CPU predicts the outcome of a conditional branching instruction to get the next fetch address thus minimizing the number of times a CPU must wait for a conditional branch to complete before it can execute the next instruction. There are different types of branch predictions: **Static Branch Prediction** predicts that the branch will never be taken. **Dynamic Branch Predictions** uses previous executions of the same instruction to make the prediction for the outcome of the branch. To evaluate the performance of prediction algorithms we use the following: **Accuracy,** percentage of correctly predicted branches to the total number of branches. **Execution time,** time taken to make the prediction. **Hardware complexity**, amount of hardware resources required to implement the algorithm. **Coverage**, the different types of branching patterns the algorithm can predict, etc.

2. Alternative Implementations of Two-Level Adaptive Branch Prediction by Yeh and Patt
In this paper the authors argue that the traditional one-level branch prediction approaches are not very accurate. They go on to propose a new approach which uses a combination of global history and local history to predict the outcome of the branch. To implement **two-level adaptive branch prediction**, they use two tables: **Pattern History Table**, to track outcomes of previous branches and **Branch History Table** to track outcome of recent branches. They carry out different experiments on benchmark programs by changing the size, structure and update rules/algorithms of the tables. They use the results to compare their performance against the traditional one level branch prediction.

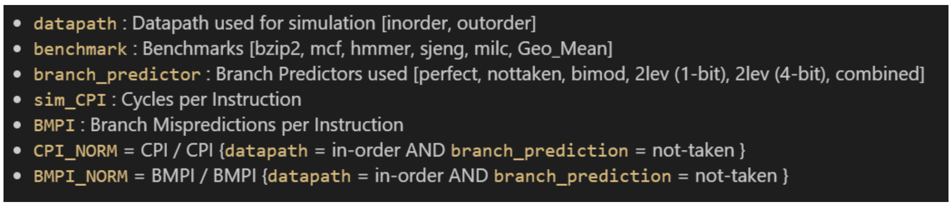
After examining the results of their experiment, the authors come to the following conclusions:
1. The proposed approach outperforms the traditional one-level approaches
2. The implementation (based on size, structure and algorithm used to update) can significantly impact performance depending on the program/benchmark being run

3. Simple Scalar and its supported branch predictors
Simple Scalar is a toolset that provides a framework for simulating computer architectures and analyzing their performance. It supports the following branch predictors used in our experiment: **Perfect Predictor** always makes the correct prediction for every branch, serves as a benchmark for other predictors. **Not-taken Predictor** always predicts that a branch will not be taken. Serves as a reasonable baseline for other predictors. **Bimodal (1-bit)** uses a table of 1-bit counters to keep track of the history of each branch and decide if the prediction is taken or not. **Two-level (1/4-bit) predictor** uses a history buffer to keep track of the outcomes of recent branches and a table of counters to make predictions based on that history. It allows a variation to table size (1/4 bit) to make more nuanced predictions. **Combined predictor** uses a combination of bimodal predictor (base predictor) and a two-level predictor (secondary predictor) to make more accurate predictions.

4. Experimental setup and data collection

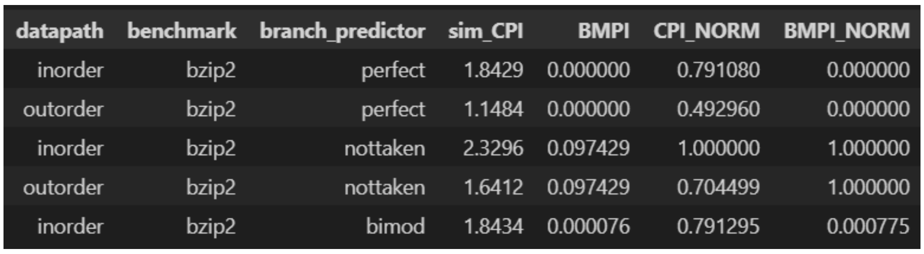
Process for collection of data:

- Ran the python script parse the simulation output and generate a data-frame with the following columns



- `datapath` : Datapath used for simulation [inorder, outorder]
- `benchmark` : Benchmarks [bzip2, mcf, hmmer, sjeng, milc, Geo_Mean]
- `branch_predictor` : Branch Predictors used [perfect, nottaken, bimod, 2lev (1-bit), 2lev (4-bit), combined]
- `sim_CPI` : Cycles per Instruction
- `BMPI` : Branch Mispredictions per Instruction
- `CPI_NORM` = CPI / CPI {`datapath` = in-order AND `branch_prediction` = not-taken }
- `BMPI_NORM` = BMPI / BMPI {`datapath` = in-order AND `branch_prediction` = not-taken }

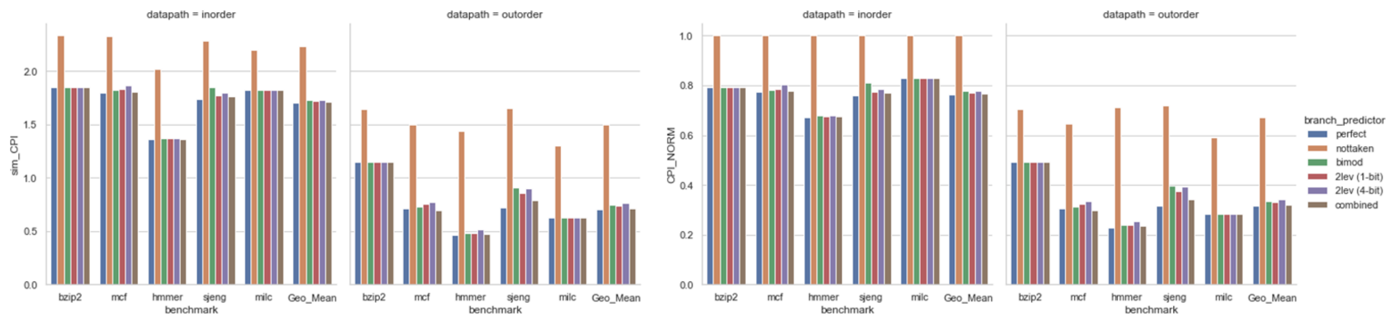
- Sample data-frame output



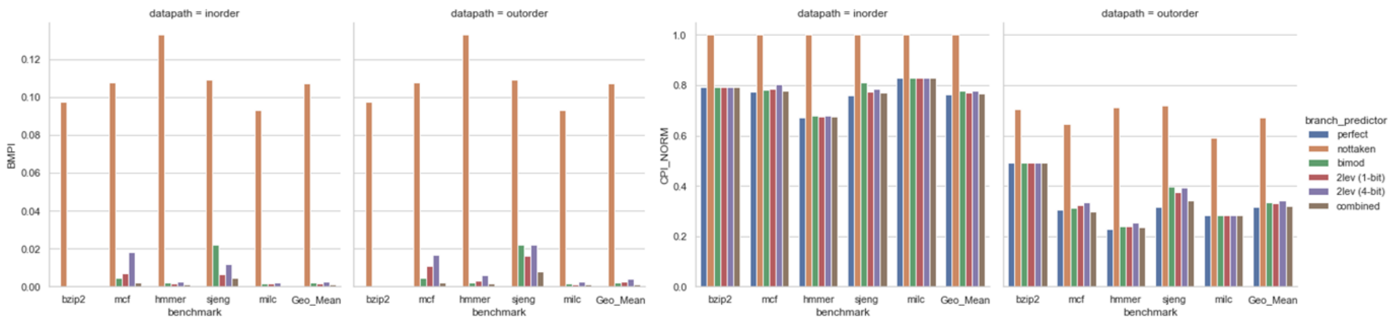
datapath	benchmark	branch_predictor	sim_CPI	BMPI	CPI_NORM	BMPI_NORM
inorder	bzip2	perfect	1.8429	0.000000	0.791080	0.000000
outorder	bzip2	perfect	1.1484	0.000000	0.492960	0.000000
inorder	bzip2	nottaken	2.3296	0.097429	1.000000	1.000000
outorder	bzip2	nottaken	1.6412	0.097429	0.704499	1.000000
inorder	bzip2	bimod	1.8434	0.000076	0.791295	0.000775

- Calculated **Geometric Mean** for each branch prediction type across all benchmarks
- Generated 4 clustered bar graphs for CPI, CPI_NORM, BMPI, BMPI_NORM, across 5 benchmarks and their Geometric Mean.
- **Note:** Each plot for CPI and BMPI contains Absolute Value Plot (left) and Normalized Value Plot (right)

Cycles Per Instruction



Branch Misprediction Per Instruction (BMPI)



5. Analysis and discussion:

1. **Which benchmark is the most sensitive to branch predictor accuracy? Which benchmark is the least sensitive?**
   From the data we collected for the 5 benchmarks and their geometric mean (for baseline) we can conclude that "hmmer" is the most sensitive to branch predictor accuracy and "bzip2" is the least sensitive.

   Sensitivity = (CPI_worst - CPI_best) / CPI_best * 100%

benchmark	Avg_sim_CPI	CPI_worst	CPI_best	Senstivity
hmmer	1.06	0.46	2.02	**77%**
milc	1.31	0.62	2.20	**72%**
mcf	1.38	0.69	2.32	**70%**
Geo_Mean	1.33	0.70	2.23	**69%**
sjeng	1.42	0.72	2.29	**68%**
bzip2	1.58	1.14	2.33	**51%**

2. For these 5 benchmarks, how much of the gap between static-not-taken and perfect prediction is achieved by implementable branch predictors, on average (geometric)?

benchmark	perfect	Not taken	gap	GM of gap covered
hmmer	0.9091	1.7257	0.8167	0.0203
milc	1.2207	1.7490	0.5283	0.0057
mcf	1.2528	1.9113	0.6585	0.0321
sjeng	1.2304	1.9659	0.7355	0.1197
bzip2	1.4957	1.9854	0.4898	0.0050

3. the relationship between the branch predictors simulated in SimpleScalar and those in a) and b), and

	Based on (A)	Based on (B)
Perfect	Static	one-level
Nottaken	Static	one-level
Bimod	Static	one-level
2lev (1 bit)	Dynamic	two-level adaptive
2lev (4 bit)	Dynamic	two-level adaptive
Combined	Dynamic	combination

4. the relative sensitivity of in-order and out-of-order pipelines to branch predictor accuracy
   From the given data we observe the relative sensitivity of out-order data path is more sensitive.

Data-path	Avg_sim_CPI	CPI_worst	CPI_best	Senstivity
inorder	1.81	1.36	2.33	**42%**
outorder	0.88	0.46	1.65	**72%**