Computer Architecture (SIL 618)

Assignment 2 - Report

Prof. Smruti Sarangi Sir

Title:

Part 1: Branch prediction using ML Algorithms

Part 2: Tejas Simulator

15 October 2021

Indian Institute of Technology Delhi

Suraj Kiran Mate (2021JCS2387)

INTRODUCTION

In processor design it becomes very important to predict the branches in the program. Here branches are nothing but the jumps of program counter(PC) from one address to another address to execute the instructions in program order. For every branch there can be two possibilities, whether it is taken or not. If the branch is taken we need to update the PC address according to the location to which the branch is jumping to. Predicting the branch is necessary to improve the performance of our architecture. Also predicting the branches is possible due to their repeated nature in the program.

So we have created some predictors for knowing the branch outcome taking space constraints in mind.

After creating the predictors for branch prediction, we have already checked them and their accuracy in the terminal. Just one more stage of checking accuracy for each trace is using Machine Learning Algorithms.

For checking the accuracy using Machine Learning Algorithm I am going to use WEKA (Waikato Environment for Knowledge Analysis). It provides simplified GUI for many popular ML algorithms. So applying any algorithm takes just one click.

DISCUSSION

As we have already applied prediction on the branches using our designed predictors. The prediction results from them corresponding to every trace are as follows:

	Pred 2400	Pred 6400	Pred 9999	Pred 32000
Trace 1	89,37	91,84	93,68	98,22
Trace 2	93,39	96,30	96,91	95,40
Trace 3	98,46	98,26	98,65	97,89
Trace 4	98,86	98,17	98,23	96,61
Trace 5	95,51	96,00	97,00	96,18
Average	95,11	96,11	96,89	96,18

Table 1.1 - Prediction by predictors

All these prediction results were obtained on checking them on terminal using ant software. All these values represent the percentage of accurate results we are getting out of our different design schemes that we have used in these predictors.

All these predictors are working just on the basis of past history of the branch and its behaviour when it was invoked last time. With different design schemes we tried to maintain more and more history corresponding to the branches and use them for next prediction keeping the notion that more deep the history is the more accurate will be the prediction.

While applying ML tools for prediction we had certain options to choose from. Since we are feeding the data in the form of numerals ie. our data has only two attributes - PC (represents the address of branching instruction) and its outcome (0 or 1). Wafter feeding the data we have many options to choose from.

In preprocess we are going to feed the data in in the ".arff" extension. In the classify section we are choosing j48 classifier and in Test options we are choosing cross-validation with 10 folds. Cross-validation will iteratively take each fold of the data for training purpose and another fold for testing purpose. J48 is used to examine the data categorically.

With this tool using j48 and cross-validation the predictions accuracy we got is as follows:

Traces	Prediction Accuracy
Trace 1	91,99 %
Trace 2	91,83 %
Trace 3	98,66 %
Trace 4	95,40 %
Trace 5	79,83 %

Table 1.2 - Prediction by ML Algorithm

We have checked all the traces individually into this tool. Also the accuracy computed by this algorithm is quite close to what is predicted by our predictor but in some places it is less than our prediction.

The confusion matrix for all these predictions are as follows:

Confusion Matrix Trace 1

	0	1
0	75,19 %	6,15 %
1	1,85 %	16,79 %

Confusion Matrix Trace 2

	0	1
0	72,65 %	6,88 %
1	1,28 %	19,18 %

Confusion Matrix Trace 3

	0	1
0	10,94 %	1,18 %
1	0,145 %	87,72 %

Confusion Matrix Trace 4

	0	1
0	6,11 %	4,21 %
1	0,382 %	89,28 %

Confusion Matrix Trace 5

	0	1
0	23,75 %	18,59 %
1	1,57 %	56,07 %

Hence from all above confusion matrices we get the idea about the skewness behaviour of the algorithm base on the nature of the traces.

CONCLUSION

From trace 1 confusion matrix we can see that the skewness is not more dominant on final prediction even thought the nature of the branches towards not taken (0) is very high. In only 1.85% of the predictions it causes error due to skewness.

From trace 2 confusion matrix it was also skewed towards not taken (0) nature of the branches but no dominant effect on the outcome error in only 1.28% cases.

From trace 3 confusion matrix it was skewed towards taken (1) nature of the branches causing prediction error in 1.18% of the predictions.

Trace 4 confusion matrix was skewed towards taken (1) nature causing error in 4.21% of the cases. ie. in 4.21% cases algorithm predicted taken but actually it was not taken. Hence skewed towards taken nature of thee branches.

Trace 5 confusion matrix was also skewed towards taken (1) nature of the branches which leads to error in 18.59% of the cases.

Part 2: Tejas Simulator

INTRODUCTION

Simulators are used widely to understand the real time effects of different parameters on the actual implementation of any model. They are designed in such a way that the effects of different attributes changes can be studied in more detail. With the use of simulators there is no need to manufacture the model first and then check the effects. With simulators it becomes easy to understand the results with different input in testing stage.

Tejas is also one of the simulator for pipelined processors. This simulator helps to study the effects of different parameter changes on the actual working of the processor. Here different parameters can be the operating frequency, number of pipeline stages, types of pipelines like in order and out of order pipelines, also the major part of actual performance depends on the type of program which is getting used for testing.

Tejas is developed on java platform and is well equipped to understand the effects of these parameter changes on the outcomes. This simulator takes pipeline model as an input in terms of different parameters of pipeline and gives results based on them.

Use of Tejas makes it fast to understand the desirable and undesirable characteristics of various parameters and then finalise the model based on its required outcome.

Terms Overview

1) **Number of Cycles:** It defines the number of clock cycles required for the execution of the program. In almost all processors the clock cycles are the reference of comparison to the real time performance of the architecture. Hence in general lesser the number of cycles taken for execution faster is the processor considering that the processors under examination has same operating frequency.

2) **Instructions Per Cycle(IPC):** IPC is the good measure of defining the performance of the processor. It simply defines number of instructions executed per clock cycle. Hence it is more obvious that more is the IPC faster is the execution of the program in that pipeline and faster will be the overall processor. The term inverse of IPC is called CPI (cycles per instruction).
3) **Pipeline Types:** There can be two different types of pipelines (in order and out of order). In in order instructions are executed in program order. While in out of order pipeline instructions can be execute out of program order.

4) Pipeline Stages: Defines the number of stages in pipeline of the processor. These stages are nothing but the specific execution units made to execute certain tasks.

5) Core Frequency: It defines the frequency of the processor. This frequency is the reference for different pipeline stages for program execution in synchronisation. From frequency we can improve the performance of processor upto certain extent.

6) Time Taken: It simply defines the Time taken by the processor for the execution of any program.

Findings

When we tried to analyse the result of the program which is printing “Hello World.c” we could see following outputs. Since this program was mainly printing the “Hello World” in output. So we can understand the total computations required for this program and the computational resources in terms of clock cycles and energy consumed by it in the final generated file.

```
[Main Memory Configuration]
RAM frequency:800.0 MHz
Num Channels: 2
Num Ranks: 2
Num Banks: 8
Row Buffer Policy: OpenPage
Scheduling Policy: RankThenBankRoundRobin
Queuing Structure: PerRank
```

From the main memory configuration we can see that the frequency of RAM which we have used is 800MHz. And it is having two access channels.

```
[Translator Statistics]

Java thread      =       0
Data Read        =       230562 bytes
Number of micro-ops              =       128533
Number of handled CISC instructions      =       119787
Number of PIN CISC instructions  =       138744
Static coverage          =       95.6166 %
Dynamic Coverage =       82.7753 %
```

From the screenshot pasted above, this text represents the Translator Statistics for our processor parameters. This snip is the part of the actual file generated by the Tejas simulator for running the given program. Translator statistics represents the parameters of the translator. As we know for every instruction in high level language like c, cpp or java we have number of CISC instruction corresponding to this. This conversion is based on the compiler. This conversion is then made further using translator converted from CISC to micro-operations just to make the instruction structure more uniform.

CISC are converted into micro-operations because the CISC instructions are very less predictable in terms of their length and size in bits. Hence they are very difficult to fetch. Thus they are converted to

```
[Timing Statistics]

Total Cycles taken                =       332982

Total IPC                =                0.3244          in terms of micro-ops
Total IPC                =                0.2741          in terms of CISC instructions

core             =       0
Pipeline: outOfOrder
instructions executed     =       127931
cycles taken     =       329962 cycles
IPC              =                0.3244          in terms of micro-ops
IPC              =                0.2741          in terms of CISC instructions
core frequency   =       6400 MHz
time taken       =               62.6743 microseconds

number of branches        =       19744
number of mispredicted branches   =       2477
branch predictor accuracy=               87.4544 %

predictor type = TAGE
PC bits = 8
BHR size = 8
Saturating bits = 2
```

micro-ops for more uniformity and predictable behaviour during fetch operation. From above pasted snip we can see the instructions in c are converted into 128533 micro-operations.

In timing statistics we can see different timing parameters regarding the program. The timing statistics mostly depends on the auxiliaries that we are using for performance improvement of the processor. The core frequency is also an important parameter for calculation of performance of the processor. As mentioned we have the core frequency as 6400 MHz, if we again increase this frequency we will get more instruction throughput. As we are giving out of order pipeline for testing hence we can get better performance out of this. When we use TAGE type of predictor we get 87.4544% which is acting like bottleneck for the performance of out processor.

CONCLUSION

Hence after running the program in Tejas we found that we are getting different parameters for the simple program in pipeline. These parameters are based on the specifications that we have provided to this simulator. These parameters can change the overall performance of the complete architecture with large extent. Such simulations make it easy to understand the dependency of the designed architecture on the parameters we are giving as input.

Thus simulations is the most effective way to understand the functionality and dependence of the processor which is under study. It makes the work easy to test the performance under different testing conditions.