COMPUTER ARCHITECTURE

16:332:563

Professor: NAGMEH KARIMI

EVALUATING BRANCH PREDICTION STRATEGIES

Designers

ANEESH ABHYANKAR, ana85

VISHALSINGH HAJERI, vsh15

CONTENTS

Evaluation of Branch Prediction Strategies

1. ABSTRACT

In order to increase the throughput of a computer we need to increase the number of instructions retiring from it. When the program is sequential fetching next instructions is trivial. But, when there are jumps and branches in the code determining the address of next instructions proves to be a bottleneck in the performance. Exploiting instruction level parallelism becomes crucial in the presence of branch instructions in the form of conditional and unconditional jumps. Branches interrupt the sequential flow of instructions and redirect it to some other location. In this case, fetching sequential instructions results in the eventual flushing of pipeline and re-fetching of instructions. This causes loss of performance in terms of throughput. In superscalar computers, that can fetch multiple instructions in a clock cycle, this loss is more pronounced as all the instructions fetched need to be flushed in the aftermath of a branch instruction. Thus, selecting the best branch prediction strategy is a crucial part of the design of a computer.

2. INTRODUCTION AND PROBLEM STATEMENT

2.1 INTRODUCTION

The ever-increasing need to improve performance has led to extensive research in the field of instruction level parallelism (ILP). The very early solution to the problem of wrongly fetching instructions after branches was to stall the processor until the address of the next instruction was calculated. But this proved to be a serious deterrent to the speed of operation of the processor. Branch prediction strategies provide a way to predict the direction of a branch and fetch the instruction at that location even before the processor calculates the branch target. Static prediction strategies predict a branch direction without considering the dynamic history of the behavior of that branch. Dynamic strategies will consider the recent history of that branch and optionally that of some previous branches before predicting. Sophisticated strategies have increased the processor performance tremendously, by accurately predicting the branch target almost 98% ["Combining Branch Predictors" – June 1993 – Scott McFarling] of the times.

2.2 PROBLEM STATEMENT

In this project we evaluate some famous strategies already published in literature, by implementing them in simplescalar. Our problem statement was to evaluate Strategy 6 and Strategy 7 from the paper "Study of Branch Prediction Strategies" by James E. Smith (1981) and a new strategy using simplescalar and to juxtapose all of them. The new strategy (not included in simplescalar) that we implement is the 'Tournament Predictor', which combines local and global predictors. We go ahead and evaluate basic strategies like 'always taken', 'always not taken', '1-bit branch predictor', '2-bit bimodal', and '2 level' also. We run SPEC95 little endian benchmarks on these strategies and analyze them to find that the new strategy, tournament predictor gives the maximum number of correct predictions, i.e. the minimum rate of misprediction.

3. SOFTWARE INFRASTRUCTURE

We use Simplescalar toolset, which is a modeling infrastructure that allows users to model and simulate computer microarchitecture for carrying out performance analysis and hardware-software co-verification. The users can, using this tool, simulate various processor environments, complete with static/dynamic scheduling, multiple issues of instructions, non-blocking caches and static/dynamic branch prediction strategies. Simplescalar can emulate Alpha and Portable Instruction Set Architectures (PISA). We used the PISA, which is a MIPS like architecture, since it is used for instructional and academic purposes. Simplescalar consists of a cross compiler which basically creates the illusion that the underlying computer is PISA even though it is something else. Todd Austin in the University of Wisconsin at Madison created Simplescalar, and several other researchers made further contributions to it.

4. BRANCH PREDICTION STRATEGIES

4.1 STATIC BRANCH PREDICTION

They are very primitive kinds of branch prediction strategies, which do not consider the dynamic behavior of the branches during run time of the program. They make a static decision for every branch at compile time, which remains the same during the entire lifetime of the program. They do not have the ability to adapt with the behavior of the program during run time. Examples are taken, not taken branch predictors, etc.

4.1.1 Predict Taken

This strategy always predicts all the branches to be taken and goes ahead with this prediction to fetch the next instruction at the branch target. This technique is found to work better with loop instructions, since they are almost always taken so as to loop back to the next iteration. It is only once after completion of the loop that the branch will be not taken. The problem with this strategy though is that the processor needs to calculate the branch target before checking the branch condition. This requires the branch target calculation hardware, i.e. ‘adder’ to be placed in the fetch cycle of the execution pipeline.

4.1.2 Predict Not Taken

This strategy will predict all branches to be not taken and carry the execution to the sequentially next instruction. This strategy fails miserably with loops for the reason mentioned above. Normally a typical program is made up of loops and this makes this strategy a bad choice in normal circumstances.

4.2 DYNAMIC BRANCH PREDICTION

These strategies use the information collected about a branch at run time to make a prediction about its direction. Unlike static predictors, which are compiler based, dynamic predictors are hardware implementations and perform a lot better than static predictors. The reason is mainly the fact that they take into consideration the dynamic behavior of branches while predicting their direction, which helps them correct a wrong decision taken in the past. The predictor may also become more inclined to make a particular prediction for a particular branch if that branch produces consistent outcomes. Examples are 1-bit branch predictor, 2-bit bimodal predictors, etc.

4.2.1 1-bit Branch Predictor

This is a dynamic prediction strategy, which maintains a 1-bit saturating counter. It has a history table that is indexed into by lower few bits of the Program Counter (PC). Entries of this table are 1-bit counters that help make predictions.

4.2.2 2-bit Bimodal Predictor

The problem with 1-bit Branch Predictor is that it changes its prediction very quickly. When a particular branch direction is mispredicted, the strategy changes its prediction for the next occurrence of that branch. Such a predictor fails when a branch is alternately taken. For example, for a program that prints all even numbers, the branch is taken alternately and this results in a very low accuracy. A simple solution to this problem is to provide hysteresis to the 1-bit predictor and to not change the prediction very quickly. 2-bit bimodal predictor uses a 2-bit saturating counter instead of a 1-bit counter in the earlier strategy. When the counter is 0 or 1, the branch is predicted not taken and when the counter is 2 or 3, the branch is predicted taken.

5. STRATEGIES WE IMPLEMENTED

5.1 STRATEGY 6 – 1-BIT BRANCH PREDICTOR

Strategy 6 described in the paper is like 1-bit branch predictor explained above. This strategy is implemented in simplescalar by defining a 1-bit saturating counter and updating it as per the figure 1. When the value of the counter is 1, the branch is predicted taken and if this proves to be a misprediction, the counter is decremented. When the counter value is 0, the branch is predicted not taken and the counter is incremented in the event of misprediction. As shown in Figure 2, the lower 'M' bits of PC are used to index into the prediction table that contains such 1-bit saturating counters in each entry.

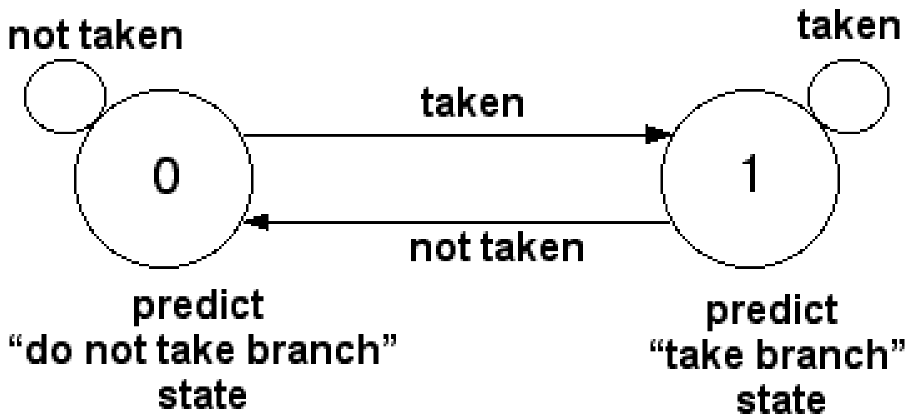


Figure 1 – State Machine of 1-bit Branch Predictor (courtesy: www.umd.edu)

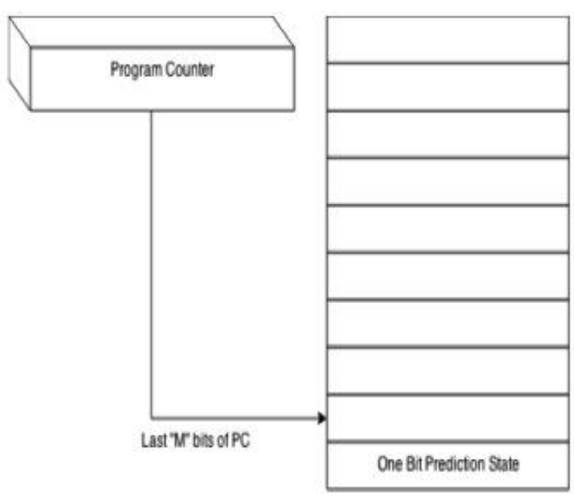


Figure 2 – 1-bit Branch Predictor

We increase the prediction table size in the predictor and obtain the number of prediction hits to study the effect of table size on the prediction accuracy. We see as expected that the prediction accuracy goes on increasing as we increase table size. Increasing table size implicitly means increasing the number of lower order bits of PC used to hash into the table. Results of the analysis are illustrated in figures 3 to 7 below.

1 bit Branch Predictor - Compress95		
Number of Branches	10992983	
Table Size	Misses	Hits
8	3440414	7552569
16	2763397	8229586
32	2351026	8641957
64	2350273	8642710
128	2347043	8645940
256	2345924	8647059
512	2344839	8648144
1024	2344464	8648519
2048	2344366	8648617
4096	2343704	8649279
Maximum Hits		8649279
Accuracy		78.6799998

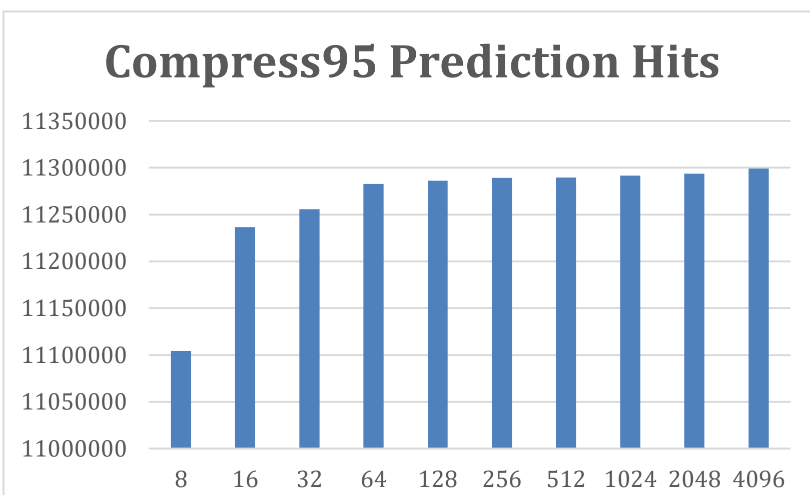


Figure 3 – Analysis of Strategy 6 against Compress95 Benchmark

1 bit Branch Predictor - GO		
Number of Branches	71660927	
Table Size	Misses	Hits
8	25981954	45678973
16	26327755	45333172
32	25986817	45674110
64	24231012	47429915
128	23111114	48549813
256	22179887	49481040
512	21391393	50269534
1024	20729531	50931396
2048	20400215	51260712
4096	20100890	51560037
Maximum Hits		51560037
Accuracy		71.95

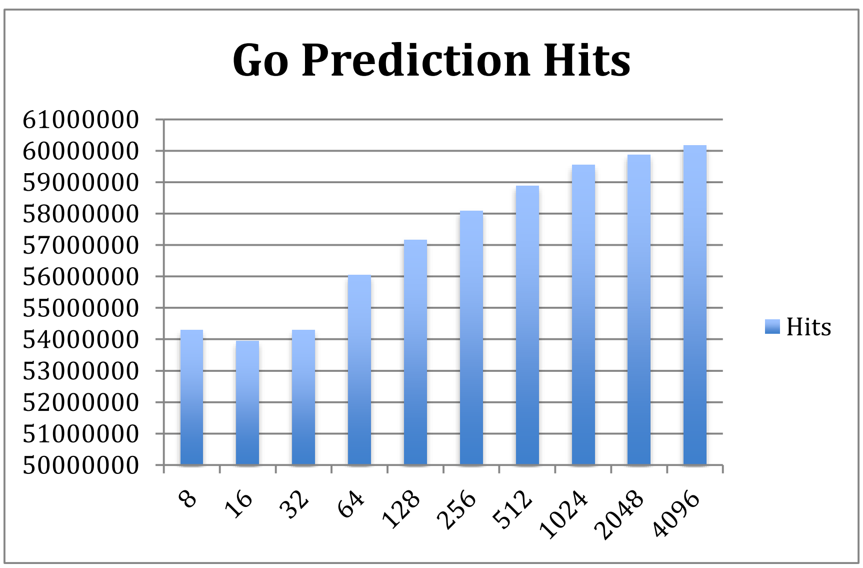


Figure 4 – Analysis of Strategy 6 against Go Benchmark

1 bit Branch Predictor - PERL		
Number of Branches	3061732	
Table Size	Misses	Hits
8	1374018	1687714
16	1244166	1817566
32	1052040	2009692
64	972603	2089129
128	841651	2220081
256	732944	2328788
512	614723	2447009
1024	556451	2505281
2048	529555	2532177
4096	512840	2548892
Maximum Hits		2548892
Accuracy		83.2500036

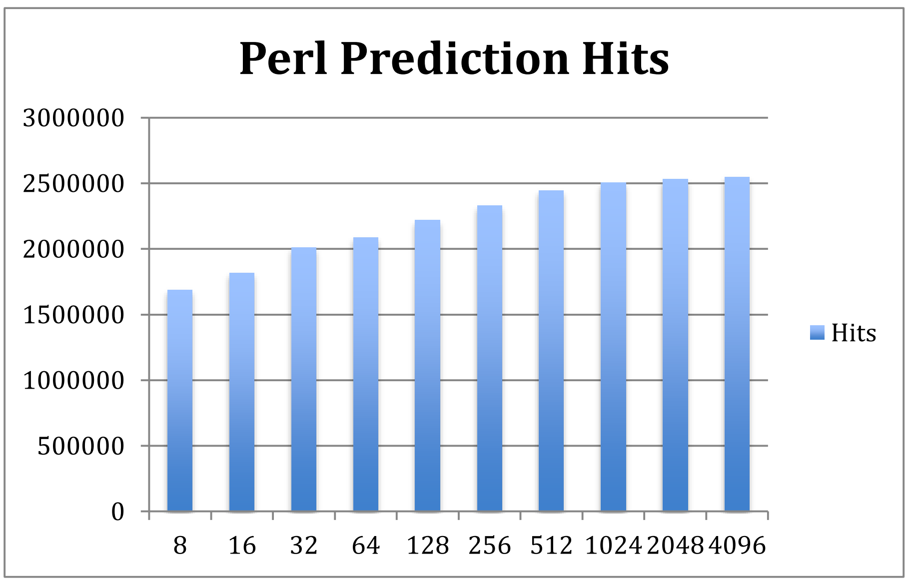


Figure 5 – Analysis of Strategy 6 against Perl Benchmark

1 bit Branch Predictor - CC1		
Number of Branches	41277100	
Local History Size	Misses	Hits
8	17492356	23784744
16	16420548	24856552
32	15189544	26087556
64	13785887	27491213
128	12137649	29139451
256	10752545	30524555
512	9614366	31662734
1024	8758463	32518637
2048	8140064	33137036
4096	7727072	33550028
Maximum Hits		33550028
Accuracy		81.2800027

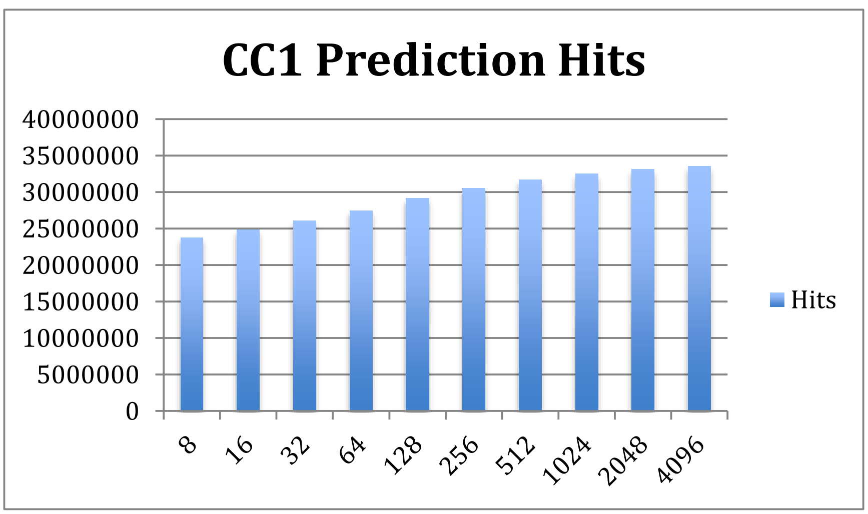


Figure 6 – Analysis of Strategy 6 against CC1 Benchmark

1 bit Branch Predictor – ANAGRAM		
Number of Branches	757	
Table Size	Misses	Hits
8	197	560
16	187	570
32	182	575
64	180	577
128	177	580
256	172	585
512	176	581
1024	172	585
2048	172	585
4096	174	583
Maximum Hits		585
Accuracy		77.2787318

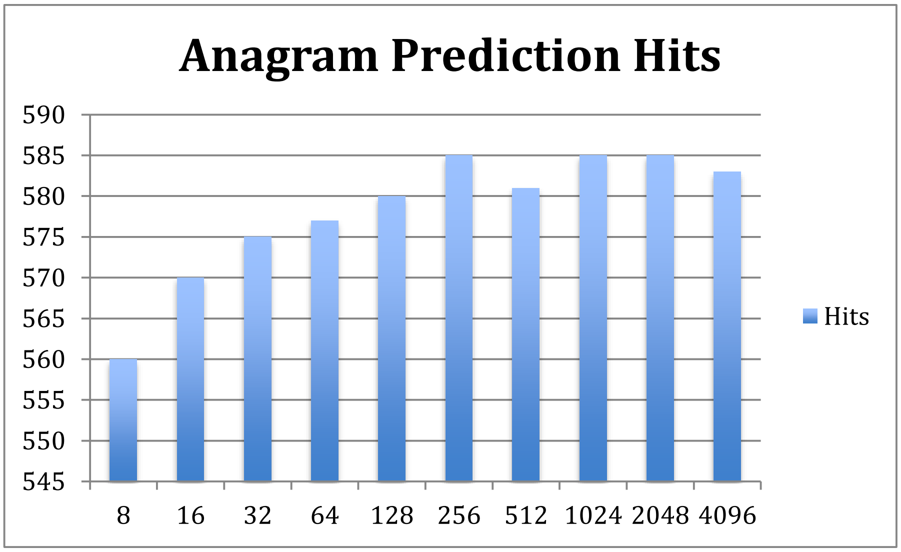


Figure 7 – Analysis of Strategy 6 against Anagram Benchmark

5.2 STRATEGY 7 – 2-BIT BRANCH PREDICTOR (BIMODAL PREDICTOR)

As discussed in section 4.2.2, 1-bit branch predictor faces problems when the actual branch outcomes change from taken to not taken frequently. This strategy provides an improvement over the 1-bit strategy by implementing 2-bit counters to take prediction decisions. The state machine implemented by this strategy to update the counter values is shown in figure 3 and the branch predictor algorithm is shown in figure 4.

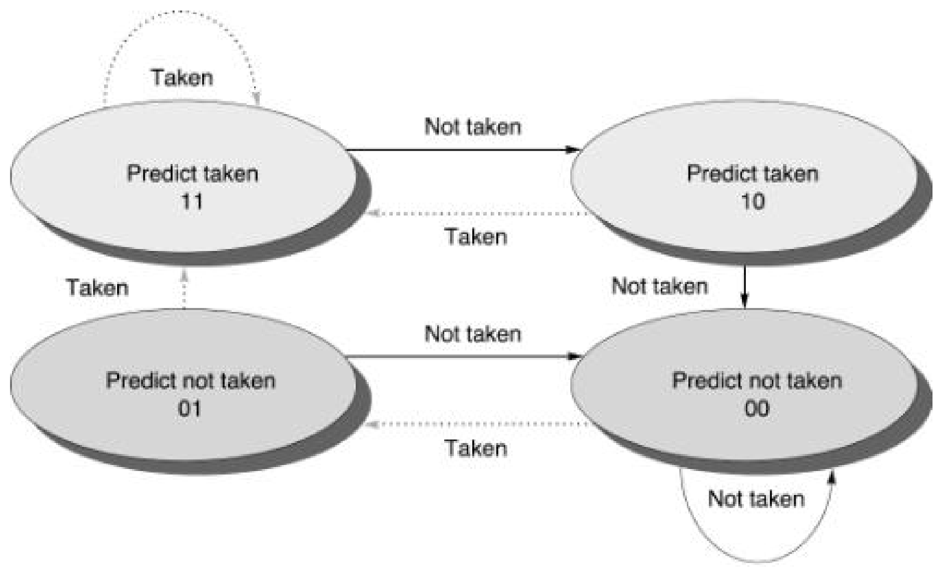


Figure 8 – State Machine of 2-bit Branch Predictor (courtesy: www.ncsu.edu)

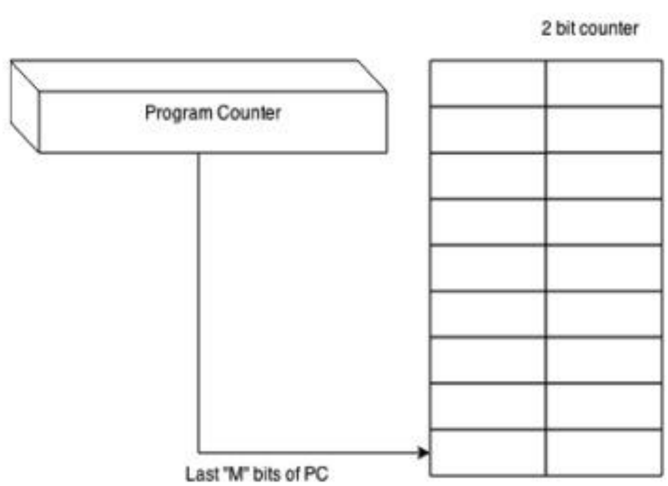


Figure 9 – 2-bit Branch Predictor (Bimodal Predictor)

We carry out the same analysis for 2-bit branch predictor as we did for - bit branch predictor to find the accuracy. We see that the performance of 2-bit predictor is better than that of 1-bit predictor as it adds hysteresis to it. The analysis has been illustrated in figures 10 to 14 below.

2 bit Branch Predictor - Compress95		
Number of Branches	14361040	
Table Size	Misses	Hits
8	1887445	11989635
16	1607468	12473595
32	1454012	12753572
64	1450936	12907028
128	1449448	12910104
256	1448817	12911592
512	1448274	12912223
1024	1448067	12912766
2048	1447658	12912973
4096	2343704	12913382
Maximum Hits		12913382
Accuracy		89.9195462

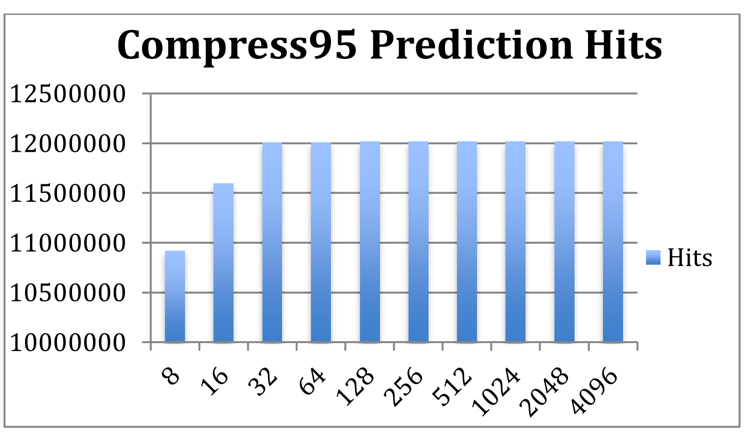


Figure 10 – Analysis of Strategy 7 against Compress95 Benchmark

2 bit Branch Predictor - GO		
Number of Branches	80274754	
Table Size	Misses	Hits
8	24869141	55405613
16	24816805	55457949
32	23832433	56442321
64	22355499	57919255
128	20640618	59634136
256	18919622	61355132
512	17630318	62644436
1024	16741821	63532933
2048	16039005	64235749
4096	15515991	64758763
Maximum Hits		64758763
Accuracy		80.6713939

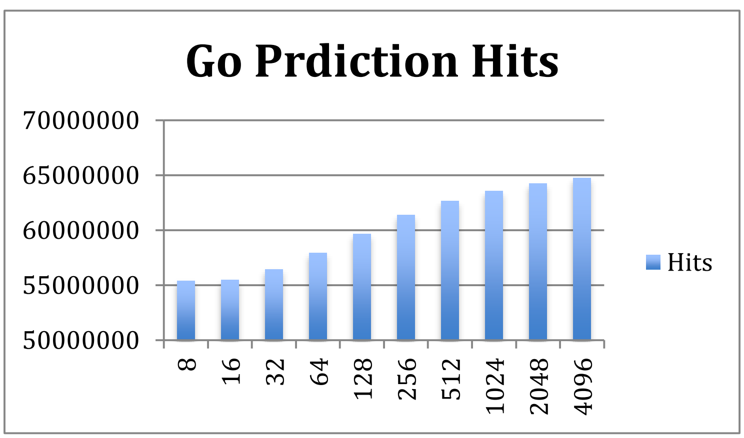


Figure 11 – Analysis of Strategy 7 against GO Benchmark

2 bit Branch Predictor - PERL		
Number of Branches	4366187	
Table Size	Misses	Hits
8	1264387	3101800
16	1139814	3226373
32	972756	3393431
64	818882	3547305
128	701926	3664261
256	615171	3751016
512	474220	3891967
1024	421029	3945158
2048	388377	3977810
4096	370076	3996111
Maximum Hits		3996111
Accuracy		91.524046

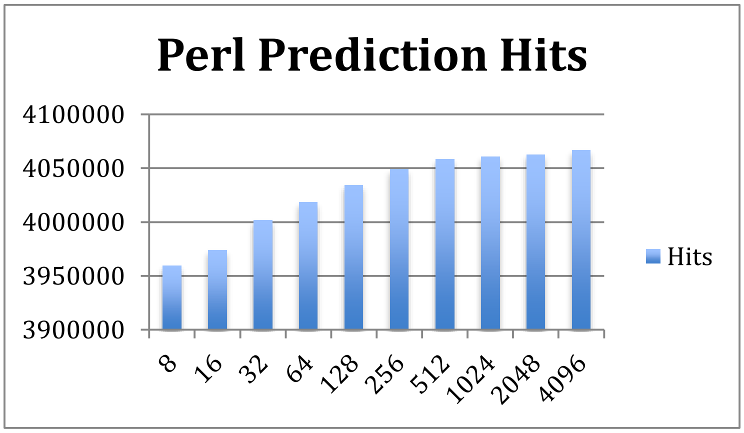


Figure 12 – Analysis of Strategy 7 against PERL Benchmark

2 bit Branch Predictor - Compress95		
Number of Branches	56338415	
Table Size	Misses	Hits
8	16036651	40301764
16	14794429	41543986
32	13507026	42831389
64	11985020	44353395
128	10378785	45959630
256	8868439	47469976
512	7748289	48590126
1024	6917451	49420964
2048	6317122	50021293
4096	5934658	50403757
Maximum Hits		50403757
Accuracy		89.4660544

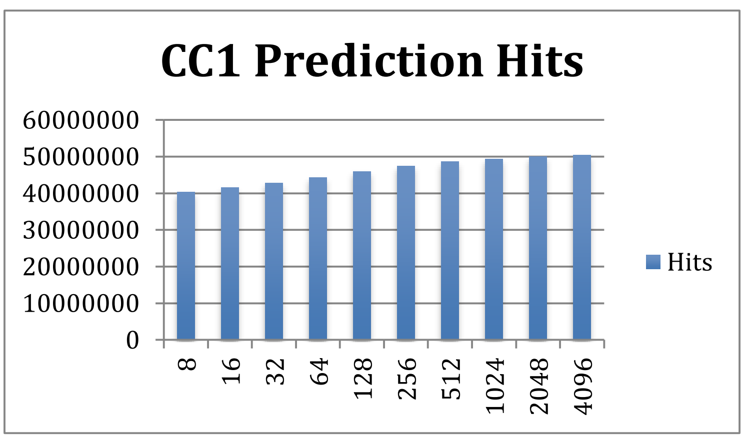


Figure 13 – Analysis of Strategy 7 against CC1 Benchmark

2 bit Branch Predictor - Compress95		
Number of Branches	971	
Table Size	Misses	Hits
8	186	785
16	178	793
32	164	807
64	165	806
128	146	825
256	152	819
512	147	824
1024	144	827
2048	145	826
4096	144	827
Maximum Hits		827
Accuracy		89.9195462

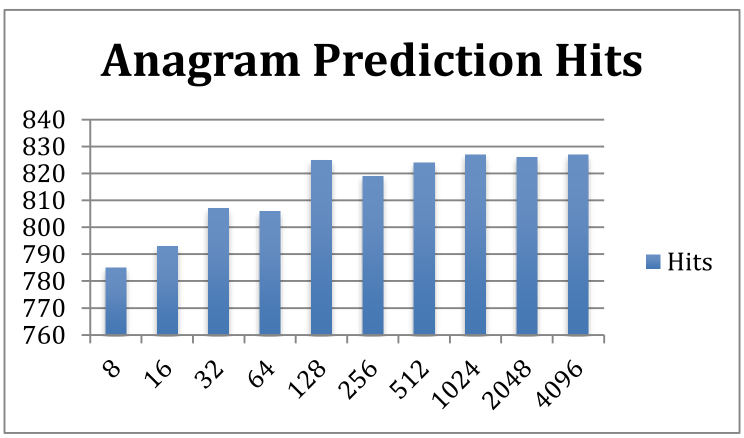


Figure 14 – Analysis of Strategy 7 against Anagram Benchmark

5.3 TOURNAMENT BRANCH PREDICTOR

The predictors analyzed above are local predictors, which means that they only consider the local branch history. They only consider the history of the branch of which direction is to be predicted. But, many times the outcome of a branch is not only dependent on the history of that branch, but also on the history of previous branches. This observation gave rise to the researchers developing global branch predictors, which provide consideration of the same. These predictors intuitively perform better than local predictors. However, neither global nor local predictors are good enough when we require accuracies in the range of 98%.

So, they developed a new type of a predictor called tournament predictor. The idea that is to introduce a predictor to predict which predictor will predict better. It combines the best performance of both local and global predictor. It consists of a local predictor table, a global predictor table and a selector to select between the two. So, the selector is a 2-bit saturating counter, which has the following states: 00 – strongly prefer local predictor, 01 – weakly prefer local predictor, 10 – weakly prefer global predictor and 11 – strongly prefer global predictor. The tournament predictor implemented with configuration: <L1size = 1024, L2 size = 1024, GHR size = 8, XOR = Yes> is shown in figure 15 below.

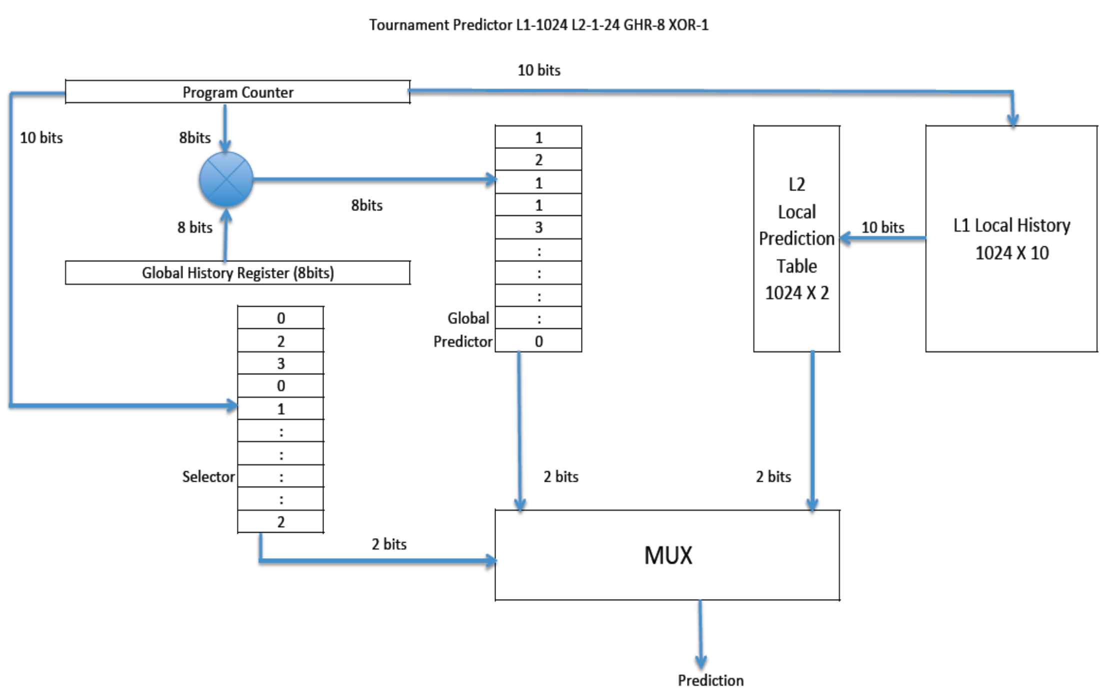
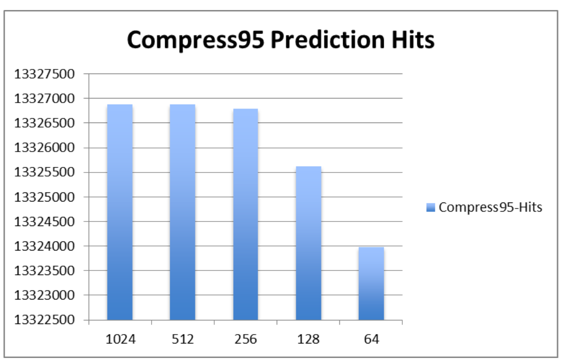


Figure 15 - Tournament Branch Predictor

Tournament Predictor - Compress95		
Number of Branches	14361040	
Config: (variable) 1024 8 1	Misses	Compress95-Hits
1024	1033548	13326879
512	1034161	13326879
256	1034243	13326797
128	1035416	13325624
64	1037055	13323985
Config: 1024 1024 (variable) 1	Misses	Compress95-Hits
4	1039536	13321504
5	896729	13464311
6	872487	13488553
7	936863	13424177
8	1033548	13327492
9	1090909	13270131
10	1099582	13261458



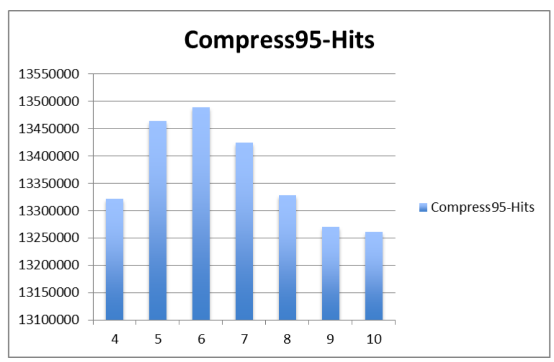
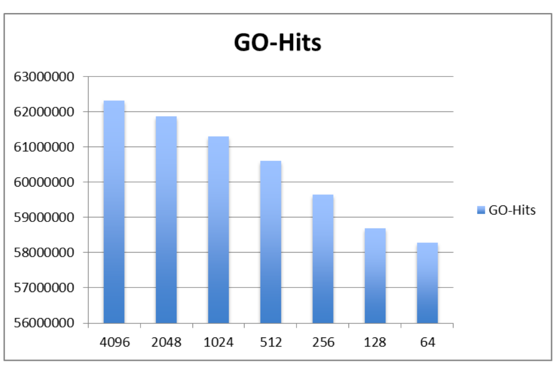


Figure 16 – Analysis of Tournament predictor against Compress95 Benchmark

Tournament Predictor - GO		
Number of Branches	80274754	
Config (variable) 1024 8 1	Misses	GO-Hits
4096	17948546	62326208
2048	18402108	61872646
1024	18977828	61296926
512	19663010	60611744
256	20622699	59652055
128	21584673	58690081
64	21998642	58276112
Config 4096 1024 (variable) 1	Misses	GO-Hits
4	17538823	62735931
5	17354122	62920632
6	15141708	65133046
7	16542892	63731862
8	17948546	62326208
9	18267429	62007325
10	18444566	61830188



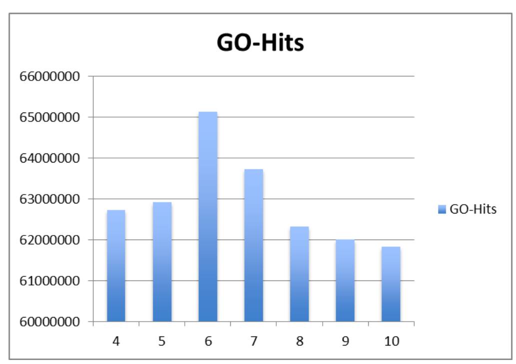
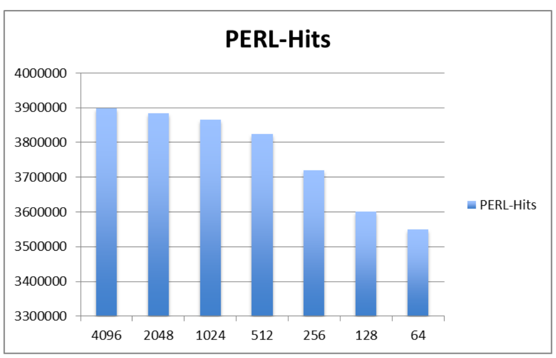
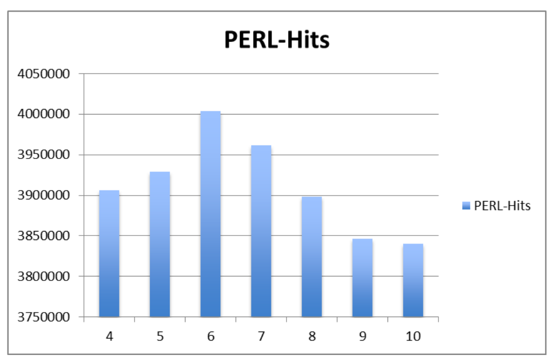


Figure 17 – Analysis of Tournament predictor against GO Benchmark

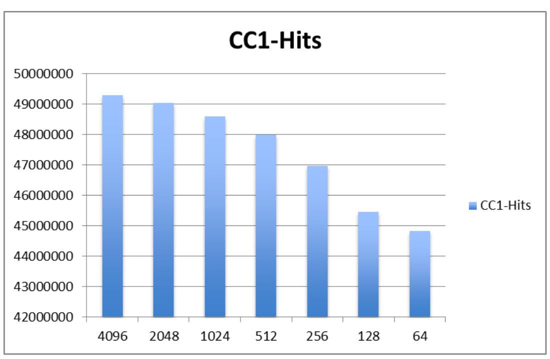
Tournament Predictor - PERL		
Number of Branches	4366187	
Config (variable) 1024 8 1	Misses	PERL-Hits
4096	468195	3897992
2048	483214	3882973
1024	500842	3865345
512	542404	3823783
256	645678	3720509
128	765438	3600749
64	816504	3549683
Config 4096 1024 (variable) 1	Misses	PERL-Hits
4	459853	3906334
5	437131	3929056
6	362770	4003417
7	404893	3961294
8	468195	3897992
9	519853	3846334
10	525583	3840604





18 – Analysis of Tournament predictor against Perl Benchmark

Tournament Predictor - CC1		
Number of Branches	56338415	
Config (variable) 1024 8 1	Misses	CC1-Hits
4096	7030925	49307490
2048	7289391	49049024
1024	7730431	48607984
512	8353360	47985055
256	9356793	46981622
128	10875931	45462484
64	11500972	44837443
Config 4096 1024 (variable) 1	Misses	CC1-Hits
4	6948023	49390392
5	6501195	49837220
6	5769734	50568681
7	6208653	50129762
8	7030925	49307490
9	7156983	49181432
10	7293613	49044802



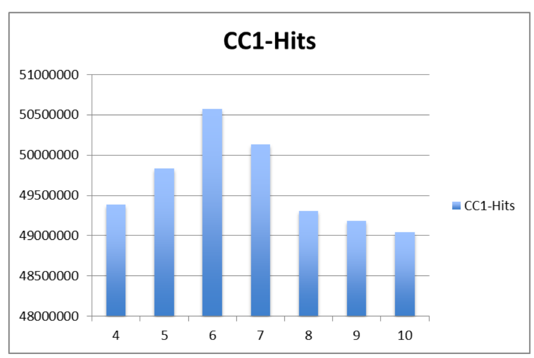
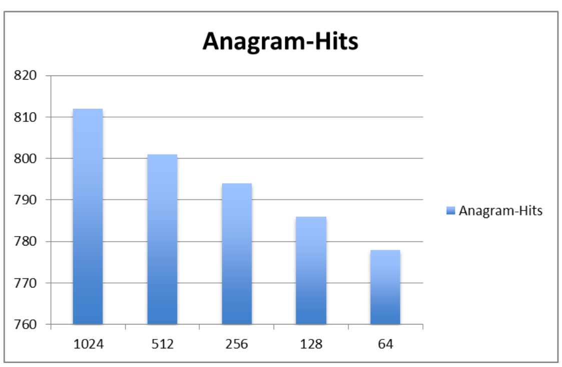


Figure 19 – Analysis of Tournament predictor against CC1 Benchmark

Tournament Predictor - Anagram		
	971	
Config = 1024 1024 8 1		
Config (variable) 1024 8 1	Misses	Anagram-Hits
1024	159	812
512	170	801
256	177	794
128	185	786
64	193	778
Config 1024 1024 (variable) 1	Misses	Anagram-Hits
4	171	800
5	166	805
6	159	812
7	160	811
8	155	816
9	169	802
10	176	795



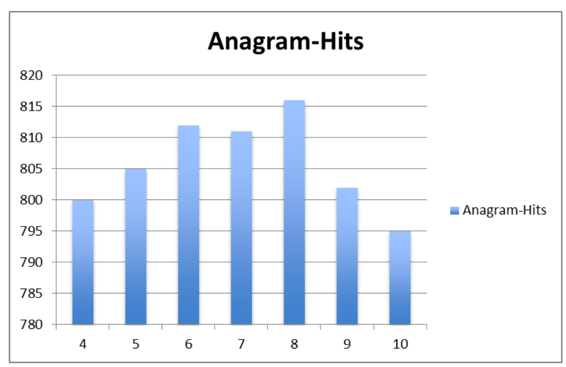


Figure 20 – Analysis of Tournament predictor against Anagram Benchmark

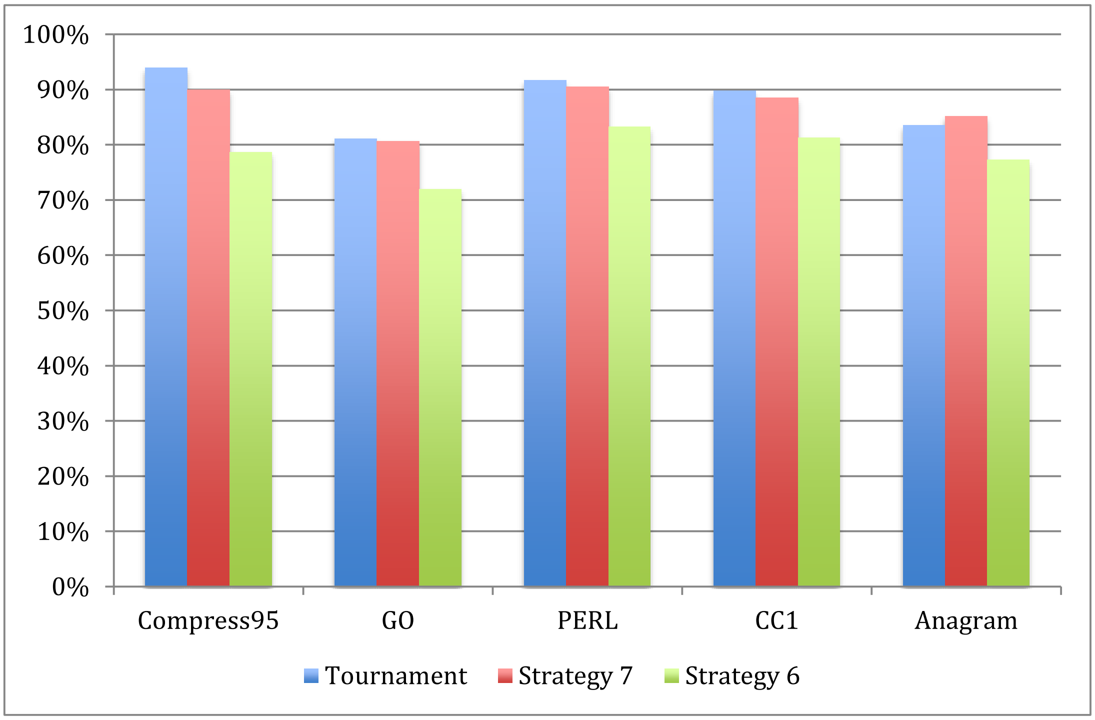


Figure 21 – Comparison of the performance of all Branch predictors on all Benchmarks

5.3.1 Warm-up Effect

As seen from the results of three different strategies, tournament predictor performs slightly less than strategy 7. But, as per our understanding since tournament predictor is a combination of local and global predictor, while 2-bit bimodal is just one level local predictor, tournament predictor should always perform better then strategy 7.

We dug deeper to understand this anomalous behavior of branch prediction strategies in case of Anagram benchmark. Anagram has only 971 branch instructions. The shift registers used in local predictor and branch history register, which store outcomes of previous predictions are initialized to some random bits (all 0's or all 1's) at the start of the program. So, Tournament predictor mispredicts initial few branch instructions as its predictions are not based upon any run-time data.

This warm-up effect is more evident in case of tournament predictors than Bimodal predictors. As seen in the graph, accuracy of tournament predictor is lower for fewer number of instructions, but as the total number of branch instructions goes on increasing so does the accuracy of tournament predictor. In the end, tournament predictor settles down to a much higher accuracy as compared to Bimodal predictor.

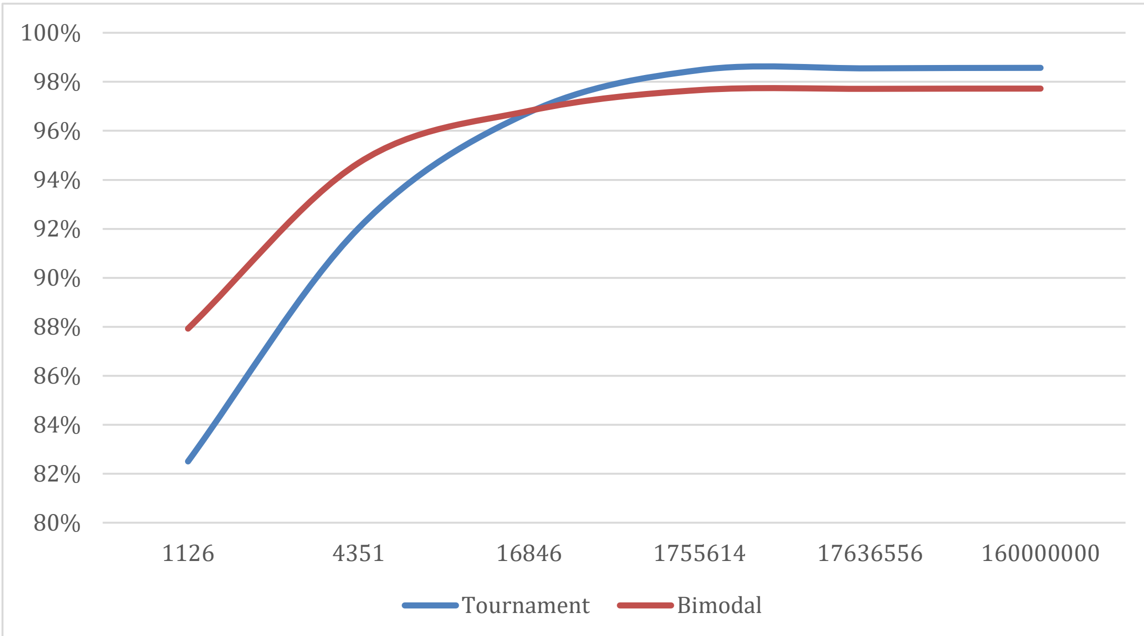


Figure 22 – Visualizing the effect of Warm up

6. CONCLUSION AND FUTURE SCOPE

6.1 Conclusion

As it is clearly seen from Figure 21 that Strategy 7 which uses 2-bit saturating counter performs better than strategy 6 which uses 1-bit saturating counter. The increased accuracy in strategy 7 can be contributed to one of its important feature of not being as impulsive as strategy 6 in making decisions.

Conditional branches encountered in the program could be either correlating branches i.e whose direction depend upon the outcomes of some ‘n’ previous branches or it could be non-correlating. Our understanding of branch predictor literature tells us that local branch predictors work well in case of non-correlating branches, while global predictors work well with correlating branches. Thus, our implemented strategy produces better results in comparison to previous strategies by combining both local and global branch predictor. It chooses one among the two predictors depending on the type of branch.

The one shortcoming of tournament predictor is that its accuracy is low for a benchmark having less number of branch instructions (<10000) because of warm-up effect. The w-bit shift registers used in local predictor and branch history register is initialized to some random bits (all 0’s or all 1’s). So, Tournament predictor mispredicts initial few branch instructions as it doesn’t have run-time data to make predictions upon.

In tournament predictor, the choice of global predictor is crucial. We chose gshare global predicted over gselect global predictor since gshare XOR’s branch PC and global branch history registers while gselect concatenates the same. XORing minimizes the effects of destructive aliasing, which are evident in gselct.

The Tournament predictor performs better than any single level branch predictor provided that there are higher number of branch instructions in the code.

6.2 Future Scope

An alternative to branch prediction could be executing both the directions of a branch speculatively. However, required hardware resources will increase and will be proportional to number of concurrent speculative executions.

7. REFERENCES

- A Study of Branch Prediction Strategies – James Smith

- Combining Branch Predictors – June 1993 – Scott McFarling

- Simple scalar user guide: http://www.simplescalar.com/

- http://ocw.mit.edu/courses/electrical-engineering-and-computer-science/6-823-computer-system-architecture-fall-2005/lecture-notes/l13_brnchpred.pdf

- http://users.ece.gatech.edu/~sudha/academic/class/ece4100-

- Smith JE. A study of branch prediction strategies. Proceedings of the 8th Annual International Symposium on Computer Architecture, Minneapolis, MN, 12–14 May 1981. ACM Press, 1981; 135–148.

- Hennessy J, Patterson D. Computer Architecture: A Quantitative Approach. Morgan Kaufmann: San Mateo, CA, 2003

- Nair R. Dynamic path-based branch correlation. Proceedings of the 28th Annual International Symposium on Microarchitecture, Ann Arbor, MI, 29 November–1 December 1995. ACM Press, 1995; 15–23.