Effects of Various Branch Predictors with Varying Table Sizes on SPEC2000 Test Benches

May 28, 2004

Andrew Wolan
andrew dot wolan at verizon dot net

Abstract

A common trend in microprocessor designs today is to increase the depth of its pipelines. Doing so can better exploit the available parallelism in code, which results in an increase in performance.

However, as the pipeline grows in length, so does the miss penalty. The solution to rising miss penalties has been installing more accurate branch predictors. However, the more accurate the branch predictor, the more chip area (silicon) is required.

The purpose of this paper is to see if there is a trend between various branch predictors, their table sizes (number of entries), and the applications that are commonly run on them to help more efficiently use available silicon area.

The paper explores the behavior of a g-share, bi-modal (BTB) and tournament predictor while running on ten of the SPEC200 test benches. The table sizes are varied to see if any interesting patterns emerge among the branch predictors.

TABLE OF CONTENTS

INTRODUCTION

Introduction

A common trend in microprocessor designs today is to increase the depth of its pipelines. Doing so can better exploit the available parallelism in code, which results in an increase in performance.

However, control statements, such as branches, can limit the available parallelism in code. The solution to this problem is a mechanism known as the branch predictor. This branch predictor tries to predict whether a given branch will follow the “true” path or the “false” path. This then allows the processor to schedule and issue future instructions into the pipeline without waiting for the branch to complex execution.

However, branch predictors are not always correct and poor branch predictors can reek havoc on performance. Whenever a branch predictors is wrong or miss predicts, a partial or a full flush of the pipeline is needed. This flushing is known as the miss penalty and is done to remove instructions that falsely executing within the pipeline. In the end, miss-predictions can lower pipeline utilization and lower overall performance.

The solution to date to combat miss penalties is to install more accurate branch predictors. However, as the pipelines in modern processor grow in length, so too does the miss penalty. Thus, there is a growing need for more accurate branch predictors. However, the more accurate the branch predictor, the more chip area (silicon) is required.

The purpose of this paper is to see if there is a trend between various branch predictors, their table sizes (number of entries), and the applications that are commonly run on them to more efficiently use available silicon area.

Objective

The objective of this paper can be summarized as follows:

Given a restricted table size and a SPEC2000 test bench, which branch predictor scheme works best?

In other words, what branch predictor and table size combinations yield the most accurate branch predictions for a given SPEC2000 test bench?

SIMPLESCALAR

The metrics for this paper were obtained by simulating executions of various SPEC2000 test benches using SimpleScalar. SimpleScalar is system simulator that allows architects and researchers to experiment with various hardware configurations via software. For more information on SimpleScalar, please refer to their website:

http://www.simplescalar.com/

In order to evaluate the behavior of various branch predictors, each SPEC2000 test bench had to be simulated multiple times. On each run, the branch predictor mechanism was varied.

For reference, the version of SimpleScalar used was version 3.0d. The in-depth, “out-of-order” version of the simulator was used so the desired metrics could be obtained.

Branch Predictors Used

For purpose of exploration, the following three “generic” branch predictors were used in this study: g-share, bi-modal and tournament. These predictors are summarized below. (An in-depth explanation of these predictors is beyond the scope of this paper.)

G-Share

This predictor is the simplest of the three in terms of implementation. It implements a global branch predictor, which means that the effects of various branch instructions can influence the prediction. The branch behavior of the last 10 or so branches is kept in a global buffer. This value is then XORed with the lower bits of the address of the branch instruction to for an index. This index is then used to select an entry in the table.

Bi-modal

This predictor is the second simplest of the three in terms of implementation. It implements a local branch predictor, which means that the effects of branches are localized. This means that branch predictions are handled individually, dependent on where the branch instruction is located in the program. It uses a branch target buffer (BTB) and 2-bit saturating counters.

Tournament

This branch predictor is a mix between a G-share and a Bi-modal predictor. Half of its entries are devoted towards one predictor and the rest toward the other. Selection of which predictor to use for a given branch is determined by the predictor depending on the complexity of predicting that branches instructions behavior. (Typically, the G-share is used if is simple and the B-modal if it is difficult.) In theory, this predictor should given the best “bang for the buck” in terms of silicon area.

Branch predictors in Simple Scalar

Each of the selected branch predictors are pre-built into Simple Scalar and are configuration via c command line argument.

- G-share - a “2lev” predictor in Simple Scalar
- Bi-modal – uses a 4-way BTB with 2-bit counters
- Tournament – a “comb” predictor in SimpleScalar.

Table Sizes

The table size of a predictor corresponds to the number of entries that can be used to store past branch history information. The following table sizes were used in this paper:

- 2^5 entries (32 entries)
- 2^7 entries (128 entries)
- 2^9 entries (512 entries)
- 2^11 entries (2K entries)
- 2^13 entries (8K entries)
- 2^15 entries (32K entries)
- 2^17 entries (128K entries)

SPEC2000 Test Benches

What is SPEC?

According to the SPEC website,

The Standard Performance Evaluation Corporation (SPEC) is a non-profit corporation formed to establish, maintain and endorse a standardized set of relevant benchmarks that can be applied to the newest generation of high-performance computers. SPEC develops suites of benchmarks and also reviews and publishes submitted results from our member organizations and other benchmark licensees.
- *www.spec.org*

In other words, it's a set of industry standard benchmarks used to evaluate and compare the performance of various architectures. Its widespread usage within the industry makes it the perfect suite of test benches which to run and evaluate for this paper.

SPEC Target

A file containing the executable program for a given architecture is called a binary. SimpleScalar accepts binary executable for various architectures. The version of SimpleScalar used in this paper supports three architectures: MIPS, Alpha and PISA, which stands for "portable instruction-set architecture".

For this paper, the Alpha processor was chosen as the target processor for simulation. The reason it was selected is because it is the most popular of the three supported architectures. Because of its popularity, it was easier to find files and support online for those using an Alpha target over the other architectures.

As one would imagine, the version of SimpleScalar used in this project was compiled and configured to run alpha binaries.

SPEC2000 Binaries

Because of the popularity of the Alpha architecture in the research community, precompiled binary executables of the SPEC2000 test benches are available freely online.
The executables used in this paper can be found at the Univerisity of Michigan website:

http://www.eecs.umich.edu/~chriswea/benchmarks/spec2000.html

From what I can tell, these executables were optimized for performance. This may very well translate into branch prediction accuracies that are at their maximum.

Reduced Input Set

As part of each SPEC Test Bench, each binary executable is to run a common input set of data so results from various architectures can be compared. However, do to the detailed simulation of SimpleScalar's "out-of-order" simulator, one run of a SPEC2000 test bench using the normal input data set could take days to execute! To reduce simulation time, a “reduced” input set was used instead.

The purpose of this “reduced” input set is to reduce simulation time while preserving much of the statistical behavior of the program. Simulation results from this reduced set will differ from those obtained from the full input set. However, since results can be obtained in much less time, it does provide a fairly accurate picture on how the actual normal input set would perform.

The reduced input set comes in three sizes: small, medium and large. According to some preliminary tests conducted by myself, the run time for these tests are as follows:

- Small - takes up to 15 minutes to simulate
- Medium – takes up to 1 hour to simulate
- Large – takes several hours to simulate
- Normal – actual input set. Takes days to simulate.

Furthermore, according to early simulation tests, its was determined that the large input set returned a higher, more accurate branch prediction percentage than using the smaller input set.

For more information regards this “reduced” input set, refer to the ART*i*C Labs website:

http://www.arctic.umn.edu/

SPEC2000 Test Benches Covered by the Reduced Input Set

The reduced input set encompassed most, but not all, of the SPEC2000 test benches. This reduced set consists of the following test benches:

164.gzip
175.vpr
176.gcc
177.mesa
179.art
181.mcf
183.equake
188.ammp
197.parser
253.perlbmk
255.vortex
256.bzip2 (3 parts)
300.twolf

Of these SPEC2000 test benches, the following test benches were dropped:

- 164.gzip – its input set consists of 5 separate input files, each of which must be simulated separately. Thus, a simulation of this test bench alone will take 5-times longer then the other test benches. It was dropped to reduce simulation time and because a similar test bench, bzip, will be used instead.
- 175.vpr – the program under simulation returns an error message half-way through execution. As a result, accurate test results cannot be obtained, so this test has been dropped.
- 255.vortex – only 400,000 of the more than 1 billion expected instructions were executed. There is an error internally with either the tool, the binary executable or the input set. Since the results it returns are useless, this test has been dropped.

Thus, the following set of SPEC2000 test benches were evaluated in this paper:

176.gcc	188.ammp
177.mesa	197.parser
179.art	253.perlbmk
181.mcf	256.bzip2 (3 parts)
183.equake	300.twolf

TEST ENVIRONMENT

SimpleScalar

As mentioned previously, SimpleScalar version 3.0d was used for this paper. The tool was configured for an Alpha processor target and was then compiled under the LINUX OS.

Scripts

In all, there are 273 separate simulation that need to be simulated. To simplify management of their executions, C-Shell scripts were created. These scaripts are available upon request.

Binaries

The binaries used for this paper were pre-compiled by the University of Michigan. These files can be found at:

http://www.eecs.umich.edu/~chriswea/benchmarks/spec2000.html

From what I can tell, they were optimized for performance.

Reduced Input Set

To reduce simulation time, the "reduced" input set for the SPEC2000 test benches were used. This set was developed by ART*i*C Labs and can be found at:

http://www.arctic.umn.edu/

This set does not encompass all of the SPEC2000 test benches. Of those that is covered, 3 were dropped for various reasons. Refer to the previous section for the complete list of SPEC2000 test benches that were evaluated.

Computers

Each simulation takes on average 4 hours to execute. Running the desired simulations 24/7 (non-stop) on one machine would take 70+ days to complete! To address this issue, multiple machines were used so that multiple simulations could be executed in parallel, reducing simulation time by the number of computers used.

The UMASS EDLAB and its array of 30+ Linux machines were used for this purpose. Each machine has a Pentium III processor running at either 733MHz or 1 GHz, has 256 MB of RAM and runs Red Hat Linux 3.2.2-5. Additional information on the configuration of these machines can be obtained at the EDLAB website: http://www-edlab.cs.umass.edu/

In theory, all of the necessary simulations could be completed in less than 3 days. However, since other students use the lab, the simulations were done during off-peak hours as much as possible. C-Shell scripts were written to simplify management of this process. These scripts are available upon request.

SIMULATION RESULTS

The results from the simulations are presented in the following graphs. Each graph contains the simulations results for a particular test bench. The plots compares branch prediction accuracies to the type of branch predictor used and the table sizes used.

A brief discussion is presented after each plot to summarize the results. The "sweet spot", i.e., the predictor and table size combination that netted the "best bang for the buck", is also presented.

176.gcc

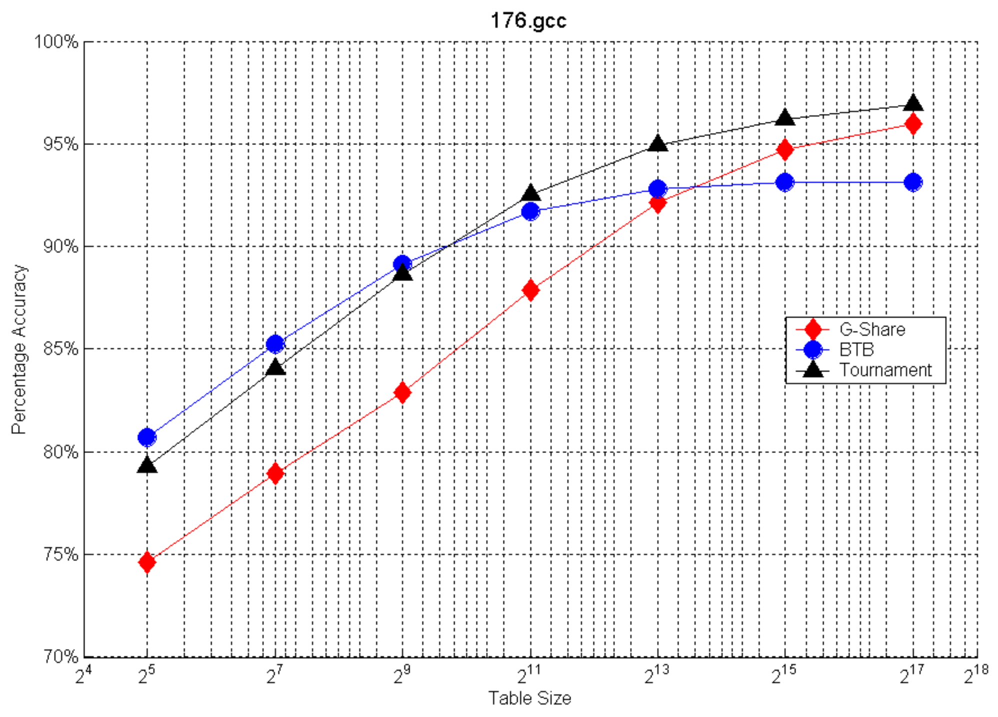

Figure 1 – 176.gcc

At lower table sizes, the BTB has an edge over the tournament predictor. At large table sizes, the tournament predictor is the clear winner. The sweet spot for this test bench is a tournament predictor with 32K entries.

177.mesa

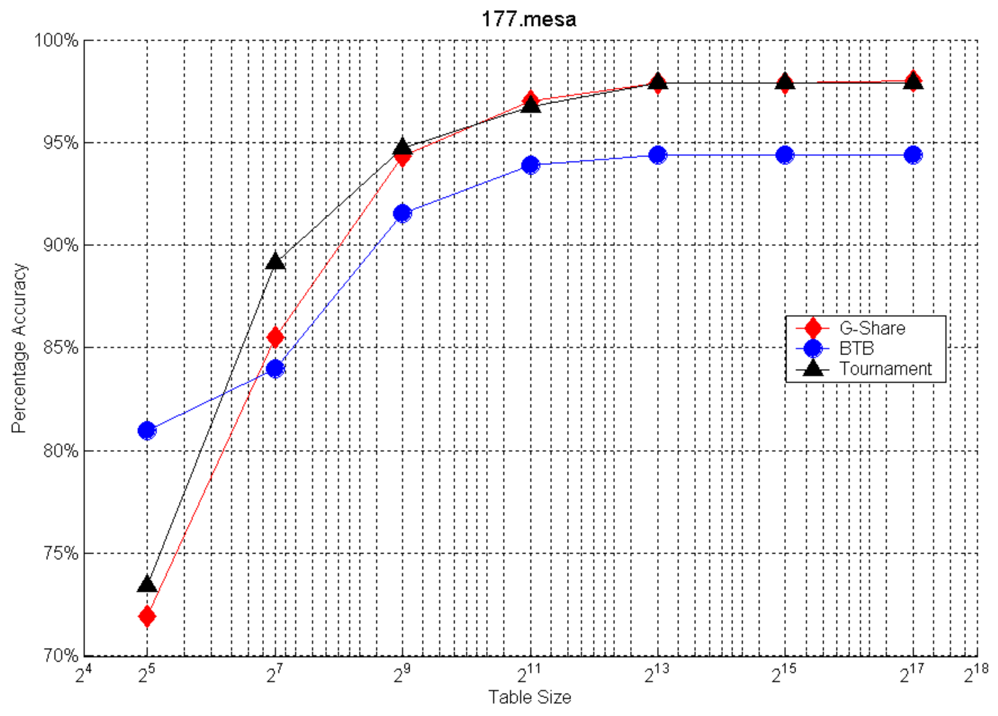

Figure 2 – 177.mesa

Though the BTB starts out good, the tournament predictor becomes the clear winner for small table sizes. At large table sizes, the accuracies of the predictors peak, and fail to improve in accuracy with an increase in table size. Both the G-share and the tournament predictors yield identical accuracies at higher table sizes. Since the G-share is simpler in terms of construction, the G-share is the winner for larger table sizes.

Since the plots peak at around 8K entries, the sweet spot is a tournament predictor with 8K entries.

179.art

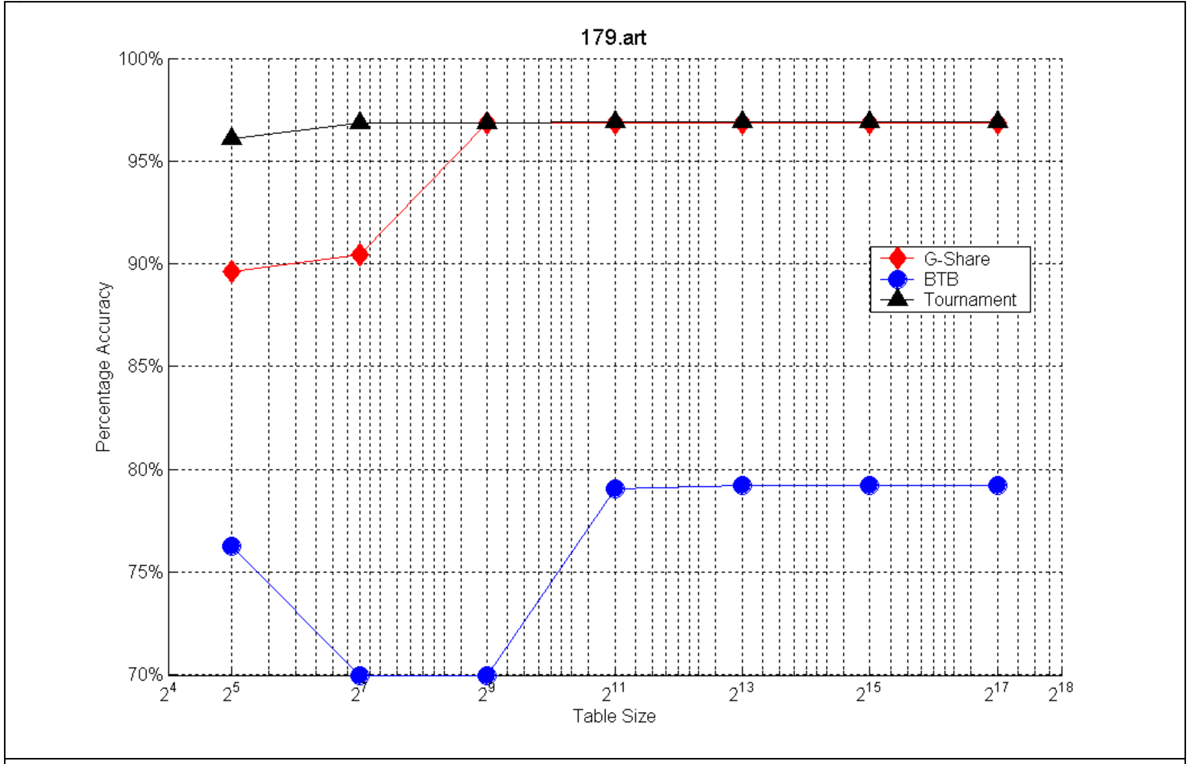

Figure 3 – 179.art

The plots in this graph are odd. The tournament predictor nets an accuracy of 96% with only 32 entries, making it the clear winning for small table sizes. Much like with the previous graph, the accuracies for all the predictors peak after a given table the size:
G-share, 512 entries; BTB, 2048 entries; tournament, 128 entries.

The Since the G-share and tournament predictors share nearly identical accuracies at larger table sizes, the G-share predictor is chosen due to its simplicity in implementation.

The sweet spot for this test bench is clearly a tournament predictor with only 32 entries.

As for the strange dip in accuracy for the BTB, it is not clear why its accuracy dips before peaking at 79%. This test bench uses a neural network for thermal image recognition. It is possible that this test bench exhibits more global branch behavior than it does local.

181.mcf

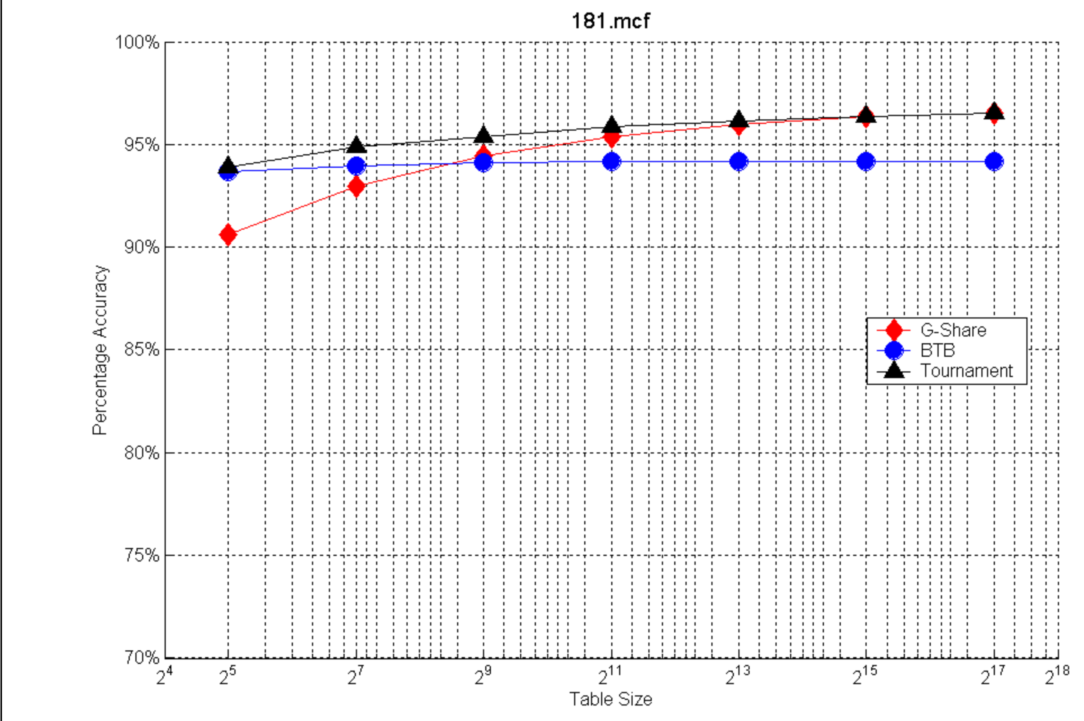

Figure 4 – 181.mcf

Like with the previous graphs, the tournament predictor is the winner for smaller table sizes and the g-share predictor the winner for larger table sizes. The accuracies of the predictors peak between 512 and 8K entries. The sweet spot is a tournament predictor with 512 entries.

183.equake

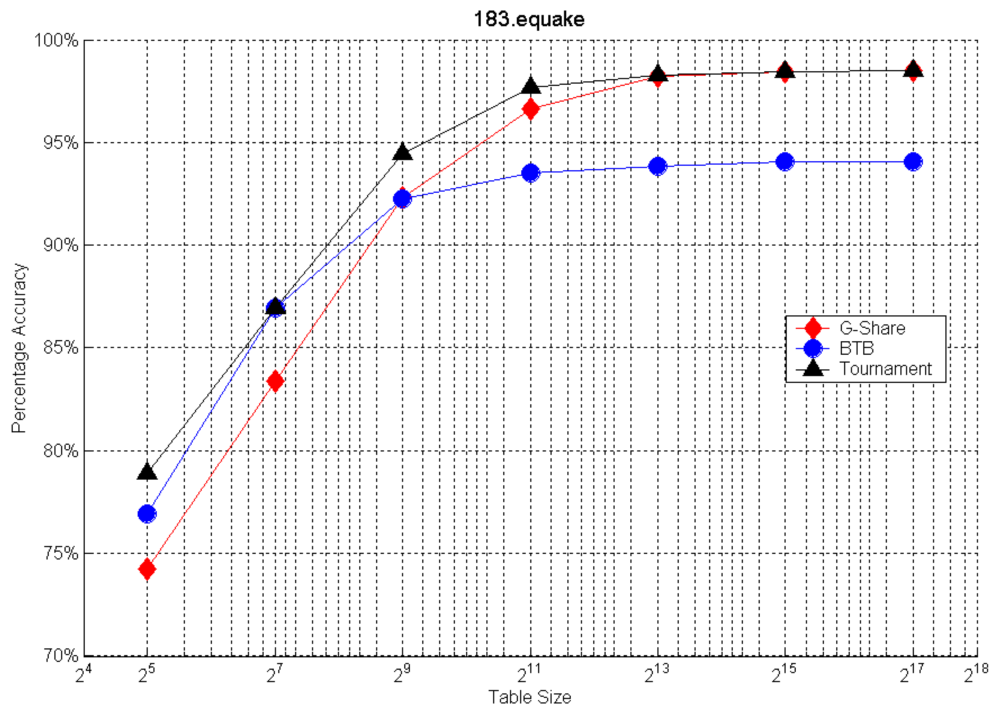

Figure 5 – 183.equake

Like with the previous graphs, the tournament predictor is the winner for smaller table sizes and the g-share predictor the winner for larger table sizes. The accuracies of the predictors peak at around 8K entries. The sweet spot is a tournament predictor with 8K entries.

188.ammp

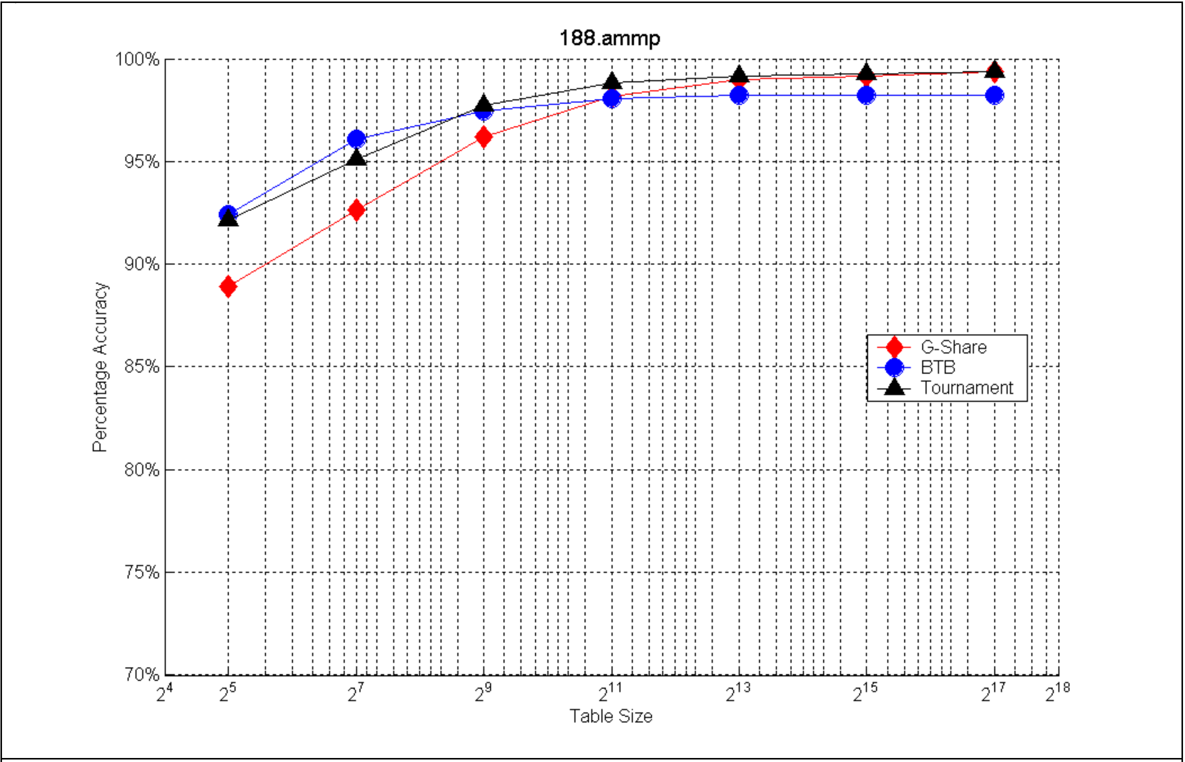

Figure 6 – 188.ammp

Unlike the previous graphs, the BTB maintains an edge over the tournament predictor for small table sizes, making it the winning. Much like with the previous graphs, the tournament and g-share predictors have identical accuracies at large table sizes. Again, the g-share predictor is chosen because it's simpler to implement.

The accuracies of the predictors peak at around 2K entries. Thus, the sweet spot is a tournament predictor with 2K entries.

Interestingly, this test bench netted accuracies as high as 99%, which means its behavior was very predictable.

197.parser

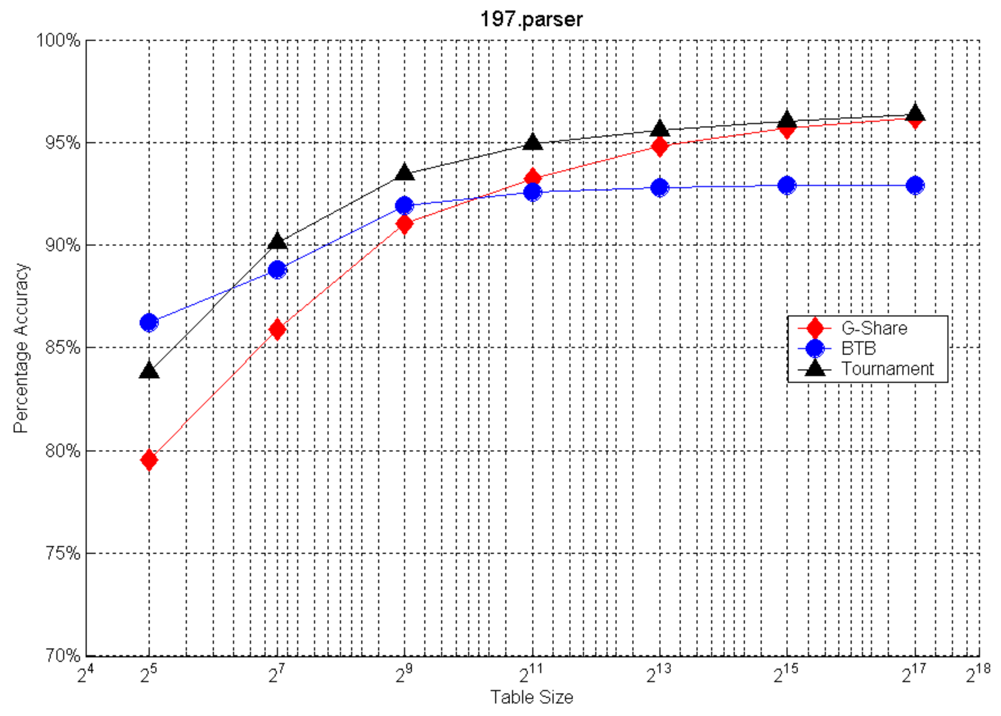

Figure 7 – 197.parser

For smaller table sizes, the BTB and tournament predictors share similar accuracies. Since the BTB is simpler to implement, it is chosen. For larger table sizes, the tournament predictor holds an edge over the g-share predictor, so it is the predictor of choose for large table sizes.

Interestingly, only the BTB predictor sees its accuracy peaks at around 8K entries. The other predictors continue to increase in accuracy with an increase in table sizes.

The sweet spot is a tournament predictor with 8K entires.

253.perlbmk

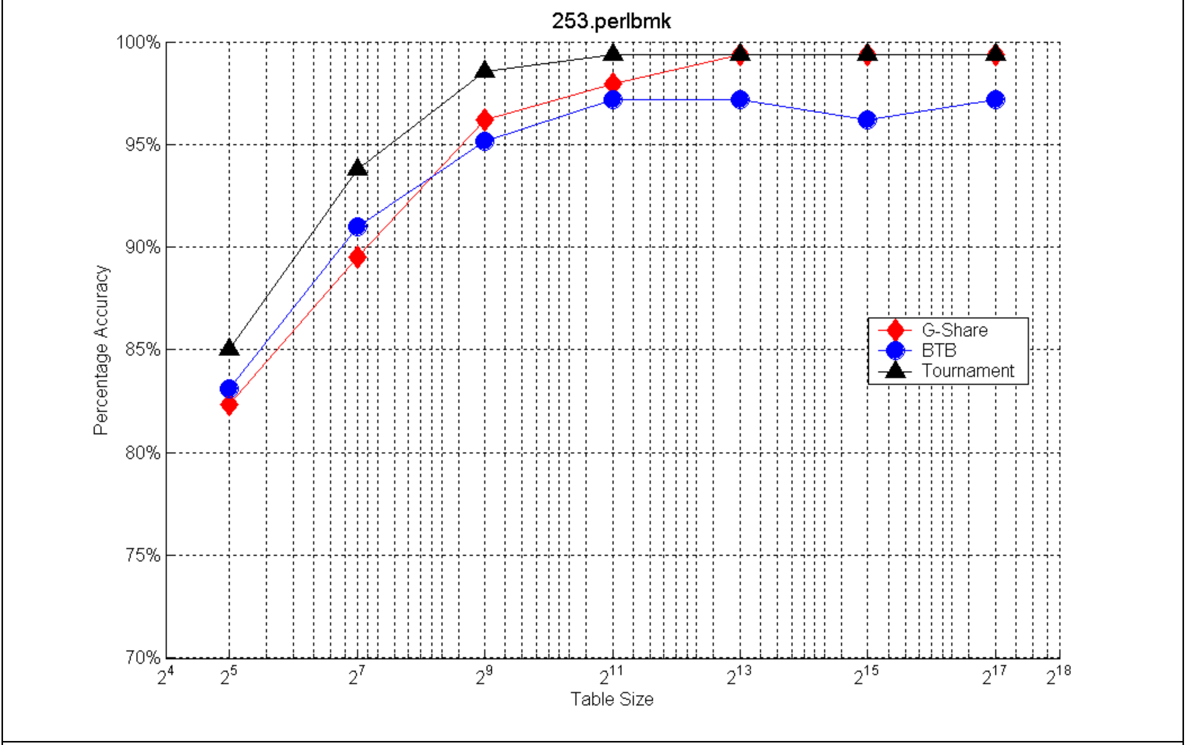

Figure 8 – 253.perlbmk

For small table sizes, it is obvious that the tournament predictor holds an edge over the other predictors. Much like with most of the previous graph, the g-share and tournament predictors share identical accuracies at larger table sizes, making the g-share the winner due to its simplicity in implementation.

The accuracies of the predictors peak at 2K entries, with the g-share peaking at 8K entries. The sweet spot is a 2K entry tournament predictor.

Much like with 179.art, the BTB dips strangely in accuracy at 32K entries before returning to its previous peak accuracy of 97%. This might be explained by a possible error in the simulation.

256.bzip-graphics

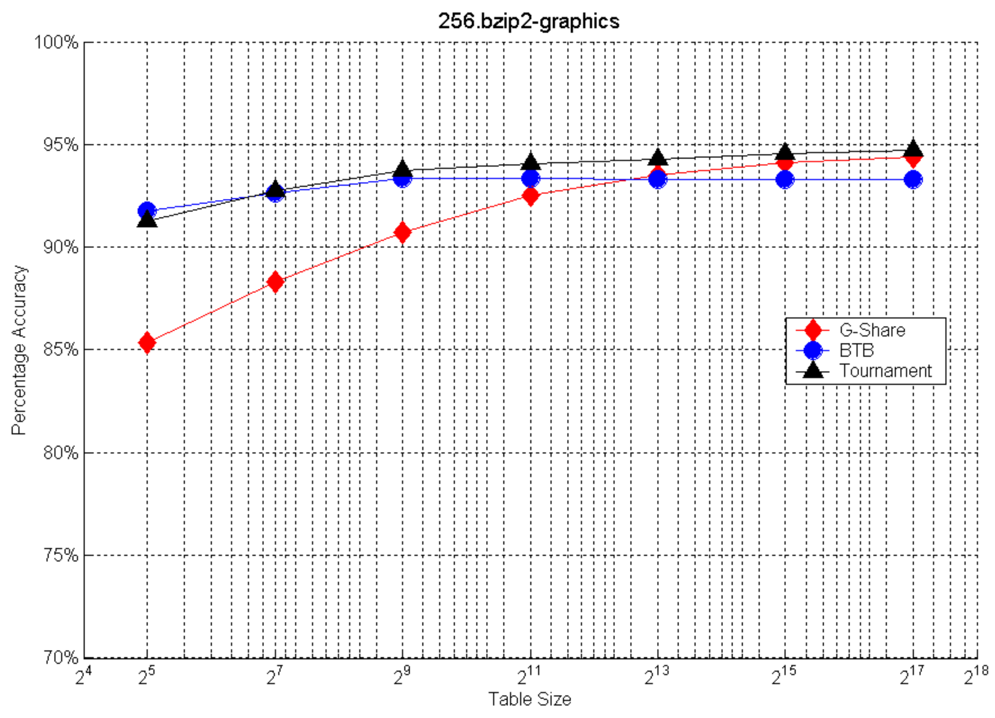

Figure 9 – 256.bzip-graphics

The bzip test bench is comprised of three separate input sets:

- graphics – an image file
- program – binary file for a program
- source - source code for a computer program

The results from each of these simulations were not combined to avoid bias in the results.

For smaller table sizes, the BTB has an edge over the tournament predictor, making it the winner. For larger table sizes, the tournament predictor holds an edge over the g-share predictor, making it the winner. Unlike most plots, only the BTB peaks in accuracy while the other do not. The sweet spot is an 8K entry tournament predictor.

256.bzip-program

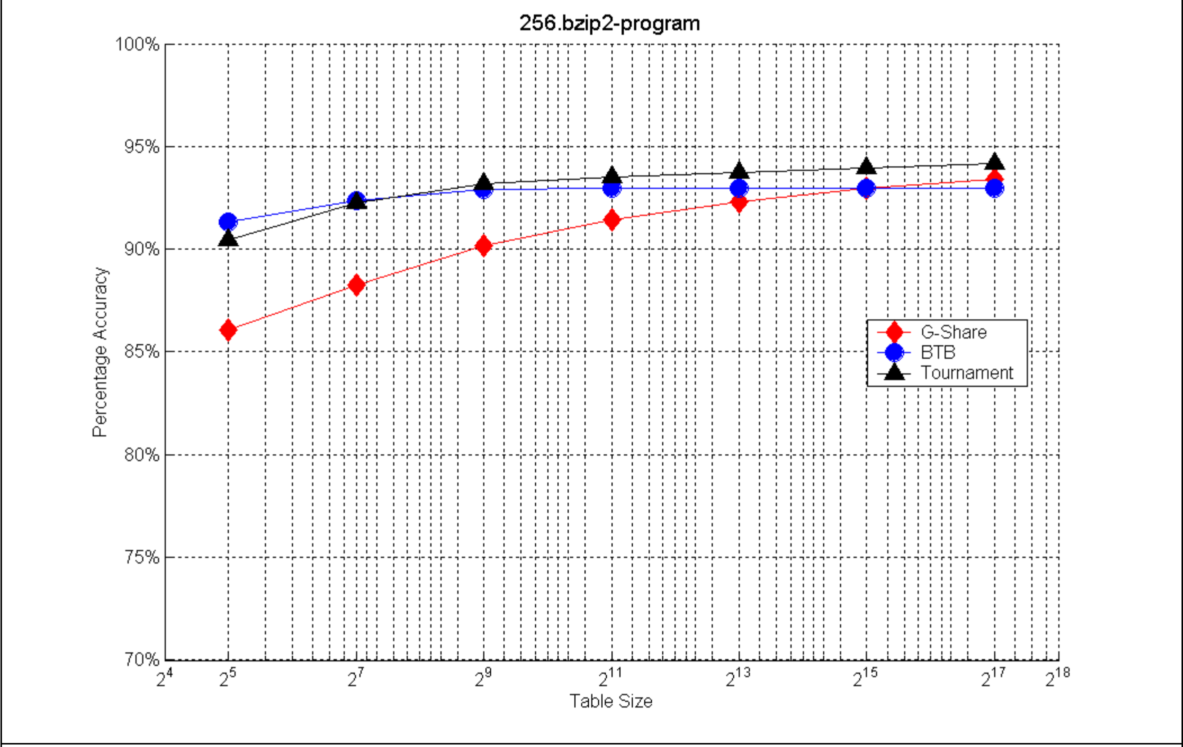

Figure 10 – 256.bzip-program

Like in the previous graph, the BTB has an edge over the tournament predictor for small table sizes while tournament predictor holds an edge over the others for large table sizes. Again, only the BTB peaks in accuracy while the other do not. The sweet spot is an 8K entry tournament predictor.

256.bzip-source

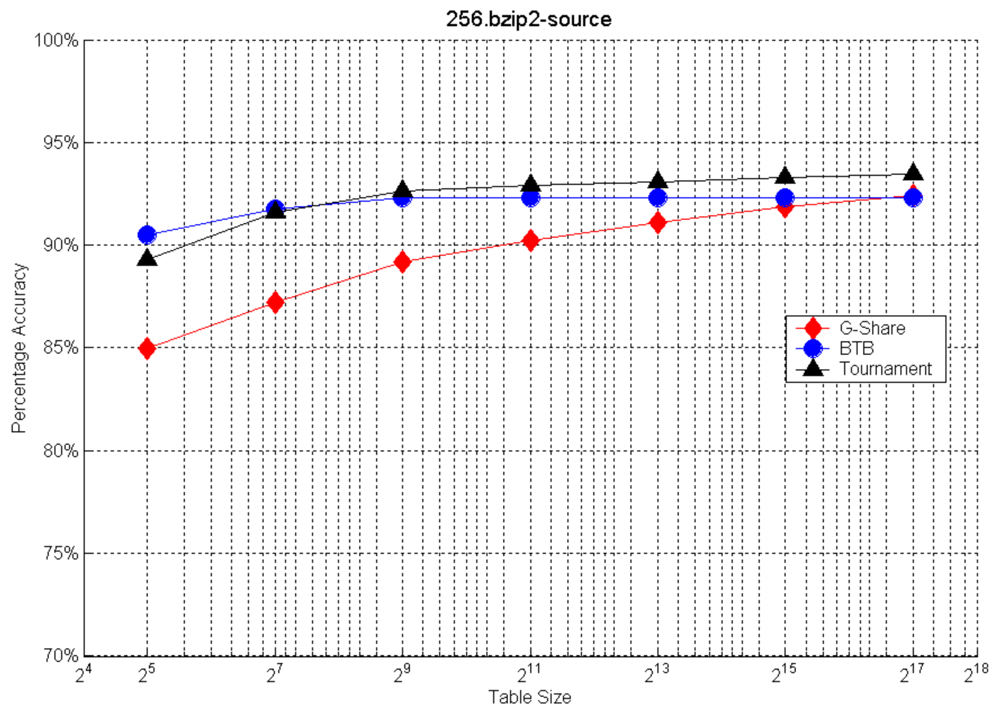

Figure 11 – 256.bzip-source

Like in the previous graph, the BTB has an edge over the tournament predictor for small table sizes while tournament predictor holds an edge over the others for large table sizes. Again, only the BTB peaks in accuracy while the other do not. The sweet spot is an 2K entry tournament predictor.

Unlike the other graphs, the g-share under-performs the BTB and the tournament predictor for all table sizes, with exception to a 128K-entry table where it matches that of the BTB.

300.twolf

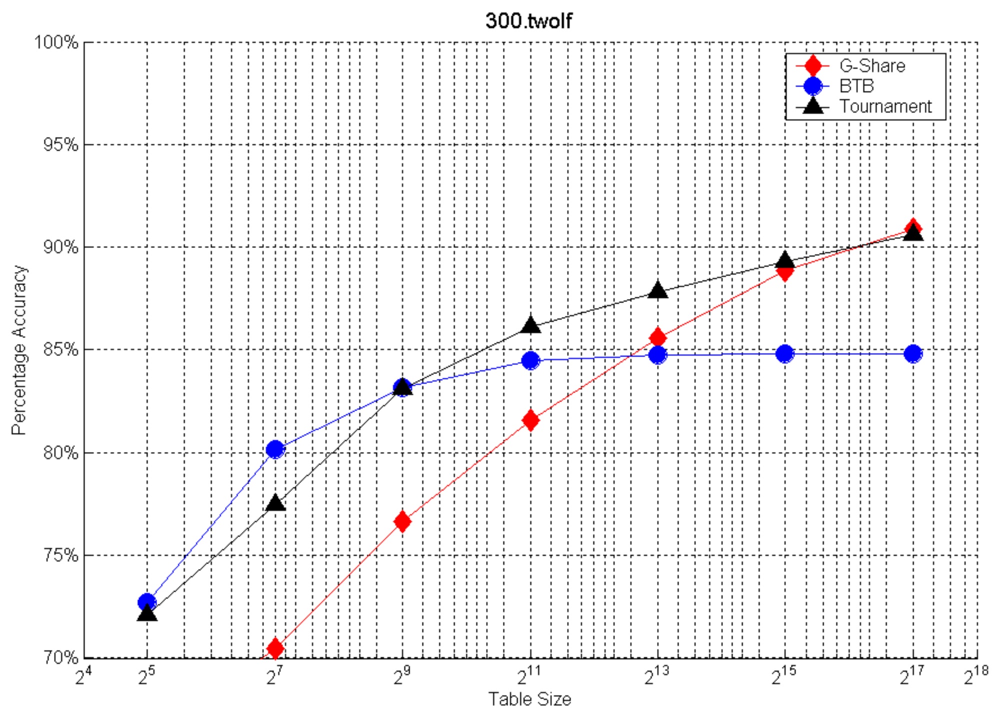

Figure 12 – 300.twolf

The BTB is the winner for small table sizes while the tournament predictor is the winner for larger table sizes. Like with the bzip test benches, the BTB peaks after 8K entries. Interestingly, the g-share predictor appears to perform the tournament predictor for table sizes greater than 128K entries. Since the accuracies of the g-share and tournament predictors are still improving with an increase in table size, it is difficult to say what the “sweet spot” is without any further experimentation. A tournament predictor with 32K entries is selected based on the data points obtained.

Summary

The results from the plots are summarized in the table below.

	Small			Large			
Test Bench	**G-Share**	**BTB**	**Tourn.**	**G-Share**	**BTB**	**Tourn.**	**Sweet Spot**
176.gcc		X				X	Tournament - 32K
177.mesa			X	X			Tournament - 8K
179.art			X	X			Tournament - 32
181.mcf			X	X			Tournament - 512
183.equake			X	X			Tournament - 8K
188.ammp		X		X			Tournament - 2K
197.parser		X				X	Tournament - 8K
253.perlbmk			X	X			Tournament - 2K
256.bzip2		X				X	Tournament - 8K
300.twolf		X				X	Tournament - 32K

Table 1 – Summary of Results

DISCUSSION

Results

This paper netted some interesting results.

First, it was discovered that the accuracy of most of the test benched peaked with a table size of around 8K entries. Thus increasing the table sizes for those test benches did not yield an improvement in branch prediction accuracies.

Second, it was discovered that both the BTB and the tournament predictor made better use smaller table sizes then the g-share. At larger table sizes, the accuracies of the predictors peaked, with both the g-share and tournament predictors having similar accuracies. Since the g-share is simpler to implement, it was consisted the predictor of choose for large table sizes. Interestingly with both the 300.wolf and 256.bzip2 test benches, the g-share predictor continued to improve in accuracy even after using 128K entries.

Finally, the sweet spot for most of the predictors was an 8K-entry tournament predictor. A processor with such a predictor should perform well under most of the SPEC2000 test benches. That is because most of the SPEC2000 test benches tend to peak in performance with 8K entries, showing negligible improvements afterwards. Only a few of the other test benches showed continued improvement with an increase in table size.

It shall be stated that a tournament predictor with only 8K entries will not yield the highest prediction accuracies for all of the SPEC200 test benches. Instead, it provides the best “bang for the buck” in regards to the evaluated SPEC2000 test benches.

Problems encountered

The documentation about SimpleScalar’s branch predictors was incomplete. I had to experiment with the tool to verify the behavior and interactions of various command line arguments. I ended up wasting a lot of time that could have been better used elsewhere.

My plan to only run my simulations in the EDLAB during off-peak hours did not work well. For whatever reason, either one simulation from a batch of simulation would complete overnight or none would complete at all. I had to resort to executing my simulation during the day so I could monitor their behavior. Doing so upset some students because I was utilizing several computers at once. I suspect that someone was resetting the machine over night, but I cannot confirm that for sure.

Future Work

One possible source of future work would be to rerun the simulations with the normal SPEC2000 input sets instead of the reduced input sets. Doing so would return the actual values from these SPEC benches. It is expected that the accuracies of the branch predictors will increase. It shall be noted even with 30 computers, it would take weeks to conduct all of the simulations.

The tournament predictor that was used was setup such that half of its table size was devoted towards the g-share predictor and the other half to the BTB. (A 50/50 split). It might be worth adjusting this ratio to see which test benches exhibit more “global” branch behavior over “local” and vise versa. I speculate that those test benches that favored the BTB for smaller table sizes will show more “local” branch behavior while the other will show more “global” branch behavior.

REFERENCES

Bibliography

Computer Architecture A Quantitative Approach, Third Edition,
Hennessy, J. L., Patterson, D. A., Morgan Kaufmann (2003)

The C-Shell Cookbook Version 1.1, Currie, Malcolm J,
http://www.starlink.rl.ac.uk/star/docs/sc4.htx/sc4.html, January 9, 1998

Special Thanks

Thomas F. Galliher Jr. for MatLab(R) assistance.

Matt Riechers for advice in creating shell scripts and for general UNIX tidbits.