Evaluation of SimPoint for Specific Architectural Studies

Veynu Narasiman
narasima@ece.utexas.edu

Aater Suleman
suleman@ece.utexas.edu

***Abstract*—**

Computer architects often times run simulations to test the impact their particular architectural enhancements may have on performance. However, the increasing complexity of modern microprocessors coupled with the emergence of longer benchmark applications has dramatically increased the time it takes to run such simulations. Therefore, much research has been done attempting to find ways to reduce this simulation time. SimPoint is a tool that selects *representative* samples from a benchmark. Only simulating these samples will significantly reduce the total simulation time. However, it is imperative that there be little compromise in accuracy for the results obtained using SimPoint.

Our goal is to evaluate the effectiveness of SimPoint with respect to two specific architectural studies - prefetching and branch prediction. For prefetching, we want to see if SimPoint can accurately capture the actual relative performance improvement obtained from prefetching. For branch prediction, we want to see if SimPoint can be used to identify individual branches that exhibit a specific type of behavior.

Our results show that SimPoint was unable to accurately capture the relative improvement in hit ratio due to prefetching. Likewise, SimPoint was also unable to capture the individual phase behavior exhibited by certain branches. Therefore, we conclude that SimPoint is not an appropriate tool to be used for these particular architectural studies.

1 Introduction

In order to gauge the potential impact a particular architectural enhancement may have on performance, computer architects have to perform detailed simulations. The increasing complexity of modern microprocessors and the emergence of larger and larger benchmark applications has drastically increased the time it takes to run such simulations. Therefore, there is a definite need to reduce this simulation time. SimPoint [1] is a popular tool that attempts to alleviate this problem.

SimPoint selects the most representative samples from a benchmark application. Only these samples are simulated in detail, instead of the entire benchmark, resulting in a significantly shorter simulation time. However, it is imperative that the results obtained using SimPoint are still accurate. Otherwise, the results of an experiment performed using SimPoint could not be trusted.

In this project, we are interested in evaluating the accuracy of SimPoint for two particular architectural studies - prefetching and branch prediction. Specifically, we want to see if SimPoint can accurately capture the performance improvement obtained from cache prefetching. Secondly, we want to see if SimPoint can accurately identify individual branches within a program that exhibit a certain kind of phase behavior.

2 Background and Motivation

In this section, we will provide some background information on how SimPoint operates. In addition, we will present some of the key results from previous work that is related to our project.

2.1 SimPoint

Using Basic Block Vectors combined with clustering, SimPoint attempts to identify similar intervals within a large application and strategically chooses sample intervals that are highly representative of the entire program. Only simulating these sample intervals instead of the entire program greatly reduces the total simulation time.

SimPoint first divides an entire program into fixed length intervals also known as slices (for example, 100-million instructions per interval). It then profiles the entire program, gathering the Basic Block execution behavior during each interval. SimPoint then compares the Basic Block execution behavior of each interval and groups similar intervals into clusters using a K-means clustering algorithm. This is an iterative process whose end result is the grouping of each interval into one of the K clusters. One representative from each cluster is then chosen with an accompanied weight based on the total number of members in that cluster. These are the samples that SimPoint reports as the most representative of the entire program. This is only a very basic description of how SimPoint operates. More detailed information can be found in [10] or at SimPoint's webpage at [1].

2.2 Related Work

Calder *et al.* in [10] performed an experiment where the actual IPC of a benchmark (obtained from complete simulation) was compared to the IPC

calculated using information from only the intervals SimPoint selected. They showed that the estimated IPC (obtained using SimPoint) was within three percent of the actual IPC, and thus concluded that SimPoint models actual performance fairly accurately. However, in another paper from the same authors, [9], several other metrics (in addition to IPC) are compared to determine SimPoint's accuracy. The results showed that SimPoint was able to accurately estimate certain metrics, such as overall branch prediction accuracy, but poorly estimated others such as data cache miss rate. This suggests that SimPoint may not be appropriate for certain architectural studies.

A recent paper from Hawkins *et al.*, [11], evaluates the effectiveness of various sampling techniques including SimPoint. This paper uses five different criteria to compare these sampling techniques. The important result here is that sampling the benchmark proved to be the best technique for reducing the length of a benchmark for detailed execution while still maintaining accurate results.

The above studies evaluated SimPoint's ability to estimate overall performance, but do not target any specific architectural studies like we are proposing to do. Specifically, we have not seen any literature that evaluates SimPoint's ability to capture the actual performance improvement of prefetching, nor have we seen any studies relating to SimPoint's ability to identify individual branches that exhibit a certain kind of phase behavior.

3 Motivation

Hawkins *et al.* in [11] declare that several architects are hesitant to use SimPoint for their experiments because they are unsure about whether or not it will yield accurate results. From this project, we hope to either validate or invalidate the use of SimPoint for studies relating to prefetching and particular branch prediction studies.

Architects are constantly inventing new prefetching algorithms, and experiments are being performed to test their impact on performance. If we show that SimPoint can accurately capture the performance improvement of prefetching, then in the future, those performing prefetching studies would be able to confidently use SimPoint thereby expediting their research. On the other hand, if we show that SimPoint cannot accurately capture the improvement, researchers in this field will know that they cannot use SimPoint for their studies and will have to rely on complete simulation or some other alternative to reduce the simulation time.

Architects are also performing research on how to handle branches whose prediction accuracy exhibits a certain kind of phase behavior. For example, individual branches that have a very high prediction accuracy during some intervals (easy to predict), but also have a low prediction accuracy during other intervals (hard to predict) are particularly interesting. Special optimizations can be made on such branches to reduce the total number of branch mispredictions. The first step for this kind of research is to identify branches that actually exhibit this kind of special behavior. In this project we will test whether or not SimPoint can be used to identify these branches. If so, future studies in this field can be performed using SimPoint, once again expediting the research process.

4 Methodology

We decided to use the SPEC 2000 benchmark suite with reference inputs for our analysis. In order to evaluate SimPoint's accuracy, we must first measure the actual statistics obtained from complete simulation of the entire benchmark and compare these with the estimated version obtained using SimPoint. We used the SPEC 2000 benchmark with reference inputs because we believe that they are the most representative of real world applications. However, a drawback of using the reference input set is very long simulation times, especially if simulating a benchmark in its entirety. Therefore, instead of using a simulator such as Simple Scalar, we decided to use an instrumentation tool for x86 architectures known as PIN [8] to collect the data.

4.1 Prefetching

We created a cache simulator that interfaced with the PIN framework and recorded the total number of cache hits and misses. We simulated a 32 KB L1-cache, and a 1 MB L2-cache with a block size of 32 bytes and an associativity of 32. The replacement policy was round robin. We implemented a simple stream prefetcher similar to the one implemented on the PowerPC [6] for the L2-cache only. We created the prefetcher in such a way that it could be easily enabled or disabled so that we could measure the relevant cache statistics both with and without a prefetcher. We recorded the number of cache hits and misses for the L2-cache only.

In order to measure the statistics for each interval in the benchmark, we added a special instruction counter so that we could differentiate between intervals. We assumed a slice size of 100-million instructions. Whenever the counter reached 100-million, we reset it back to zero and noted that a new slice was about to begin. In this manner, we recorded the total number of hits and misses that occurred in each slice of the program. An excerpt of this raw data can be found in Appendix I. Summing the data from all of the slices gives us the actual hit ratio. We wrote a Perl script that parsed the data and computed this

hit ratio for us (a list of all the scripts that we wrote for this project can be found in Appendix III). We first calculated the actual hit ratio obtained with the prefetcher disabled, and then repeated the process with the prefetcher enabled. We compared these two numbers and calculated the actual relative improvement in cache hit ratio due to prefetching. We repeated this entire process for several benchmarks in the SPEC 2000 suite.

Since we had already recorded the hit and miss statistics for every slice within the entire program, we had all the data that we needed to calculate the SimPoint estimated version of the cache hit ratio. We simply calculated the cache hit ratio for each slice number that SimPoint reported, then took the weighted average of these hit ratios using the weights SimPoint reported. Once again, we wrote a Perl script that parsed the data and computed these values for us. We did this for both cases (with and without the prefetcher). Then, just as before, we compared these two hit ratios and calculated the relative improvement in cache hit ratio. This new number is the relative improvement estimated using SimPoint. Just as before, we repeated this procedure for several of the benchmarks in the SPEC 2000 suite.

Lastly, we compared the actual improvement in hit ratio to the SimPoint estimated improvement for each benchmark. The results we obtained are reported later in the results section.

4.2 Branch Prediction

For the branch prediction information we first had to create a branch predictor simulator. We implemented a G-SHARE branch predictor with an 8-KB Pattern History Table that interfaced with the PIN framework. In order to find individual phase behavior, we needed to record the branch prediction accuracy during every interval within the program for every static branch in the program. We accomplished this by implementing a hash table that used the Program Counter (PC) of the branch instruction as the key and inserted prediction statistics for each branch. Just as in the prefetching case, we added in an instruction counter and whenever it reached 100-million (the size of a slice), we noted that a new slice was about to begin. In this manner, we recorded the number of correct predictions and mispredictions for every static branch in the entire program for every interval within the program. An excerpt of the raw data obtained using this procedure can be found in Appendix II. We repeated this process for several of the SPEC 2000 benchmark programs. This process resulted in an incredibly large amount of data (certain raw data text files were greater than 500 MB in size). The next step was to search through this data and create meaningful statistics for our experiment.

In order to verify our methodologies, we first wanted to calculate the overall branch prediction accuracy obtained from complete simulation of a benchmark, and also compute the estimated overall branch prediction accuracy using SimPoint. As mentioned before in the related work section, previous studies very similar to this have been done before. We thought it would be a good idea to validate our results with their results before we began studying the individual phase behavior of the branches. Since we had recorded the total number of correct predictions and mispredictions for each static branch in the program during every interval, we simply had to sum this data over all branches, and then over all intervals to come with the overall branch prediction accuracy. We wrote a Perl script that automatically calculated this actual overall branch prediction accuracy for us. We repeated this process for several of the benchmarks in the SPEC 2000 suite.

For the SimPoint estimated version, we first calculated the overall branch prediction accuracy for only those intervals that SimPoint reported and then took the weighted average of those prediction accuracies according to the weights SimPoint reported. Once again, we wrote another Perl script that parsed the data, and calculated these values for us. Just as before, we repeated this process for several benchmarks in the SPEC 2000 suite.

Now that we had the actual overall branch prediction accuracies and the SimPoint estimated versions for each benchmark, we could make comparisons to see how well SimPoint did. Furthermore, we performed a "before and after" comparison test to determine whether or not the difference between the actual overall branch prediction accuracy and the SimPoint estimated accuracy was statistically significant. Our results are reported later in the results section. Before continuing, we verified that our results matched the results of Calder *et al.* in [9].

After validating our branch prediction methodology, we could now continue with our individual branch phase behavior study. The next step was to identify only those branches that exhibited the special kind of phase behavior we were interested in. Specifically, we wanted to identify branches that had a high branch prediction accuracy for some intervals (greater than 95%), but also had a low prediction accuracy (anything less than 90%) for other intervals.

Since we had recorded the branch prediction statistics for every static branch during each interval, we simply found the mean and standard deviation of the branch prediction accuracies across all intervals for each static branch. We only selected those branches that had a mean above 75% and a standard deviation above 3%. Branches that met these criteria displayed

the type of behavior we were looking for. The numbers we chose (75% and 3%) were selected by an iterative process starting with some initial guesses and then graphically viewing the branches that were selected, and changing the numbers accordingly. The final requirements (mean $> 75\%$ and stdev $> 3\%$) resulted in a selection of branches that definitely exhibited the phase behavior we were looking for, except for a few cases where there were one or two outliers that severely skewed the standard deviation and thus the branch met the criteria but did not really display the type of phase behavior we were looking for. We eliminated such branches by adding a third requirement. We measured the percent of data points that lied above the mean and made sure this number was between 20% and 80%. This eliminated those branches that met the previous two requirements only because their mean and standard deviation were skewed by a few outliers. We again wrote Perl scripts that parsed the raw data, did the necessary calculations, and reported these results.

One important note to make is that for any given branch, there were slices where the total number of times that branch was executed was quite low. This could result in a very skewed prediction accuracy for that interval. For example, if a branch was only accessed twice within a given interval and predicted correctly once, but mispredicted the other time, that would result in a 50% prediction accuracy for that interval. This 50% accuracy could dramatically change the overall mean and standard deviation over all intervals, thereby falsely selecting this branch. To prevent this from happening, prediction accuracies calculated for intervals where there were less than 1000 accesses to a particular branch were ignored and not used when calculating the mean and standard deviation for that branch.

The procedure described above identified several static branches within each of the SPEC benchmarks that displayed the type of phase behavior we were looking for. To see if SimPoint could also identify these branches, we simply took the branch prediction accuracies of each branch from only those intervals that SimPoint reported and once again calculated the mean and standard deviation. We used the same criteria as before (mean $> 75\%$, and standard deviation $> 3\%$) to identify the branches we were interested in. This was all done using new Perl scripts. Lastly, to test how SimPoint did, we compared the number of branches SimPoint identified to the number of branches actually identified. Our results are reported in the next section.

5 Results

In this section we will discuss the results of our experiments. In addition we will also attempt to provide explanations for the results.

5.1 Prefetching Results

We first ran all SPEC INT and SPEC FP benchmarks using our tool and gathered the relavant data. The results obtained can be seen in Table I on the following page. Table I illustrates a comparison between the L2-cache hit ratio actually observed from the complete simulation of the benchmark, and the hit ratio that was estimated using SimPoint. The first nine rows show the data obtained from the SPEC INT benchmarks. The last row is the data from the only SPEC FP benchmark we were able to run successfully to completion in the short amount of time we had for this project. It can be seen that the difference between the actual hit ratio and SimPoint hit ratio is less that 1% in the case of eon, vpr, gzip, crafty, and perlbmk. The difference is close to 5% or less in the case of vortex and gcc. However, the difference is substantially higher in the case of gap, bzip2, and equake. On careful observation, it can be seen that the main difference between these three benchmarks and the other benchmarks is the noticeably low cache hit ratio. The actual cache hit ratio is 61% in case of bzip2, 17.83% in case of gap, and 4.78% in case of equake, whereas most of the other benchmarks have hit ratios higher than 90%. This suggests that a strong correlation exists between the cache hit ratio for a program and the difference between the actual hit ratio and the SimPoint estimated hit ratio. Therefore, it can be concluded that SimPoint does not estimate the cache behavior very accurately for programs that result in a poor cache performance.

Table II on the next page is similar to Table I except that Table II shows the results obtained when the cache tool was used with the stream prefetcher enabled. We will first look at the *Actual-HR* column. If we compare these hit ratios from Table II to the actual hit ratios recorded in Table I, we will see that the hit ratio has increased in the case of most benchmarks due to prefetching. It can also be seen that this increase in hit ratio is most visible in the benchmarks that had lower hit ratios without the prefetcher. Some noticeable improvements are: gap (from 17.83% to 51.55%), bzip2 (from 60.93% to 67.20%), and equake (from 4.79% to 27.03%). The reason for this behavior is because prefetchers tend to perform better on programs that are more memory intensive compared to programs that put less pressure on the memory system. These improvements are similar to the results seen in the literature [5]. Now if we compare the actual hit ratio with the

TABLE I

COMPARISON OF ACTUAL AND SIMPOINT ESTIMATED HIT RATIO (HR) WITHOUT PREFETCHER

Benchmark	Slices	Sim-HR	Actual-HR	Percent Difference
gap_00	2268	10.006093	17.839695	-43.911075
vortex_00	1013	82.260692	85.649938	-3.957091
gcc_00	199	71.988477	76.043075	-5.331976
eon_00	3683	99.980576	99.933566	0.047041
vpr_00	1094	95.312556	95.363824	-0.053760
bzip2_00	208	46.622580	60.935978	-23.489239
gzip_00	541	99.260283	99.668525	-0.409599
crafty_00	2155	99.162037	99.138497	0.023745
perlbmk_00	329	94.377155	94.870957	-0.520499
equake_00	1510	5.570253	4.787959	16.338784

TABLE II

COMPARISON OF ACTUAL AND SIMPOINT ESTIMATED HIT RATIO WITH PREFETCHER

Benchmark	Slices	Sim-HR	Actual-HR	Percent Difference
gap_00	2268	16.285250	51.553131	-68.410745
vortex_00	1013	83.577690	86.565478	-3.451477
gcc_00	199	74.841063	77.955966	-3.995721
eon_00	3683	99.980577	99.937977	0.042627
vpr_00	1094	94.174617	94.203570	-0.030735
bzip2_00	208	59.349353	67.208511	-11.693695
gzip_00	541	99.686524	99.828241	-0.141961
crafty_00	2155	98.775854	98.751424	0.024738
perlbmk_00	329	95.036322	95.543706	-0.531048
equake_00	1510	21.949688	27.034940	-18.809925

TABLE III

COMPARISON OF ACTUAL AND SIMPOINT ESTIMATED IMPROVEMENT DUE TO PREFETCHING

Benchmark	Slices	Sim-Impr	Actual-Impr	Difference
gap_00	2268	62.753328	188.979882	-126.226554
vortex_00	1013	1.601005	1.068932	0.532074
gcc_00	199	3.962560	2.515536	1.447024
eon_00	3683	0.000001	0.004414	-0.004413
vpr_00	1094	-1.193903	-1.216660	0.022757
bzip2_00	208	27.297444	10.293645	17.003799
gzip_00	541	0.429417	0.160247	0.269170
crafty_00	2155	-0.389447	-0.390436	0.000990
perlbmk_00	329	0.698439	0.709119	-0.010680
equake_00	1510	294.051886	464.644356	-170.592470

estimated SimPoint hit ratio, it can be seen that the accuracy of the SimPoint estimated hit ratio varies accross benchmarks just as in Table I. The difference between the actual hit ratio and SimPoint estimated hit ratio is significantly higher for gap, bzip2, and equake (similar to Table I).

Data in Table I and Table II was used to compute Table III. The *Actual-Impr* column lists the relative improvement in cache hit ratio due to prefetching obtained from complete simulation of the entire benchmark. The *Sim-Impr* column lists the relative improvement estimated using SimPoint. The *Difference* column in Table III is simply the difference between the two relative improvements (Sim-Impr - Actual-Impr). In the case of most benchmarks, the difference in the improvement is less than or very close to 1%. These benchmarks (where the difference is low) are known to be the non-memory intensive benchmarks. This implies that SimPoint can accurately estimate the relative improvement due to a prefetcher for the non-memory intensive programs but the same cannot be said about the memory intensive benchmarks (gap, bzip2, and equake). The difference in improvement for each of these benchmarks suggests that SimPoint performs very poorly on estimating the relative performance improvement of memory intensive benchmarks and therefore SimPoint should not be used for research related to prefetching.

Based on our results, we decided to further study these benchmarks in order to explain this poor accuracy of SimPoint. Figure 1 on the next page shows the number of cache misses observed during each 100-million instruction slice of bzip2. On the x-axis we have the slice numbers and on the y-axis we see the

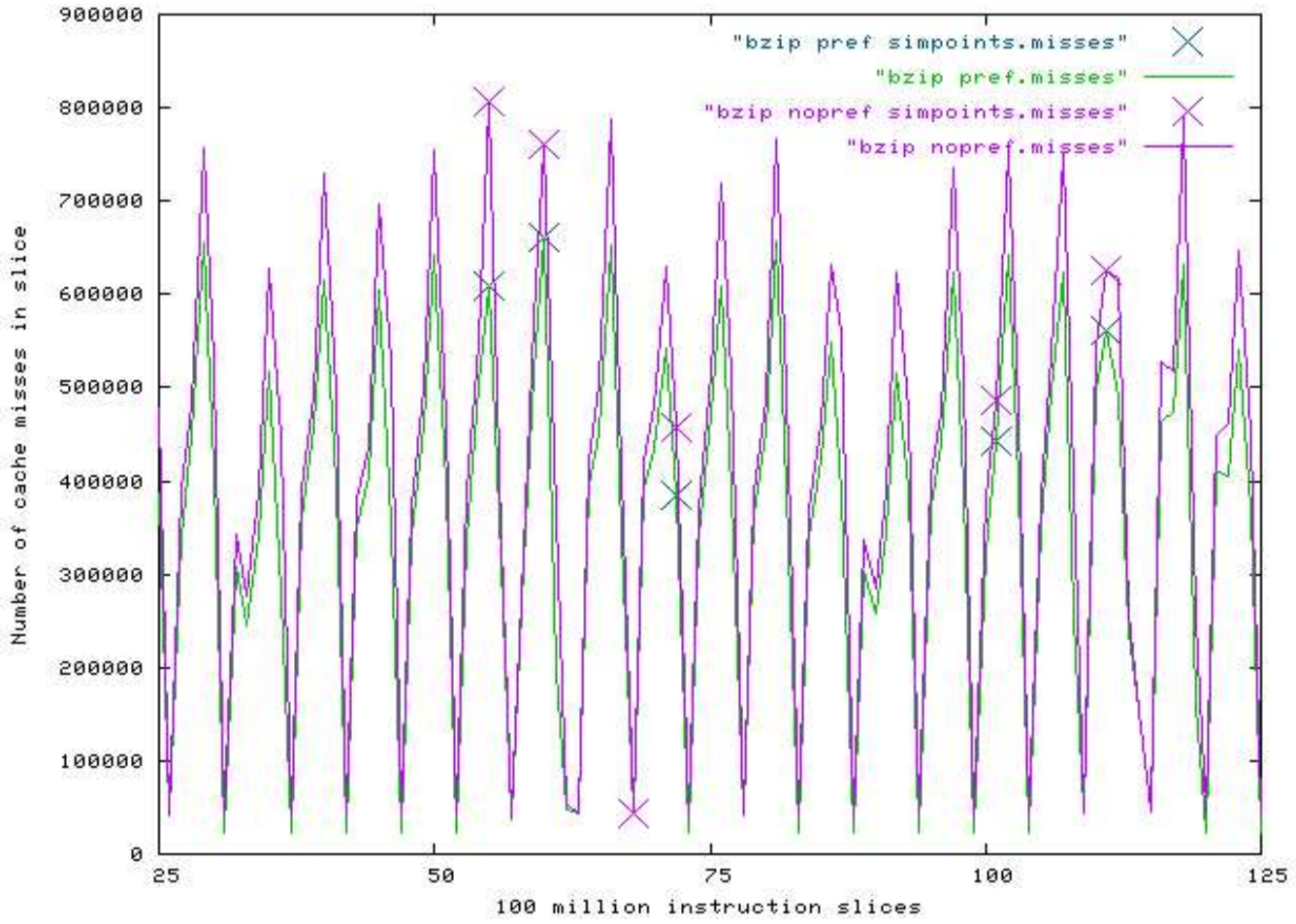


Fig. 1. Number of cache misses in every slice of bzip without prefetcher, with prefetcher, and the SimPoints

TABLE IV
COMPARISON OF ACTUAL AND SIMPOINT ESTIMATED OVERALL BRANCH PREDICTION ACCURACY

Benchmark	Slices	Sim-BPA	Real-BPA	Percent Difference
gap_00	2266	97.439887	95.559914	1.967324
mcf_00	514	93.314229	93.508401	-0.207651
gcc_00	199	96.919811	97.319026	-0.410213
vpr_00	1094	89.130415	89.088314	0.047257
bzip2_00	208	92.977164	91.578189	1.527629
gzip_00	541	93.004881	93.549842	-0.582536
crafty_00	2152	87.797056	87.953863	-0.178284
perlbmk_00	329	94.942455	94.680599	0.276568
galgel_00	3615	98.199102	97.800976	0.407078
fma3d_00	3673	94.050250	95.799894	-1.826353
equake_00	1510	82.046102	82.048140	-0.002484
apsi_00	4580	96.592628	92.585496	4.328034
mesa_00	2867	93.345454	93.445023	-0.106553
lucas_00	2479	95.441976	90.849094	5.055506
facerec_00	3438	90.884490	91.437499	-0.604794
swim_00	2237	99.889948	99.893738	-0.003794
art_00	1076	95.401496	95.472843	-0.074730
applu_00	3873	80.922845	78.618597	2.930919
mgrid_00	4819	99.222424	98.767044	0.461065
ammp_00	3921	97.091106	96.438017	0.677211

number of cache misses in each slice. The two lines show the number of cache misses with and without the prefetcher. It can be seen that the number of misses with the prefetcher are much lower compared to the number of misses without the prefetcher when the total number of misses is very high. On the other hand, the number of misses with and without the prefetcher are approximately the same in the slices where the total number of misses is very low. The X markers on the graph indicate the points that were chosen by SimPoint and were used to estimate the cache hit ratio. It can be noticed that most of the points appear to be in the upper half of the graph (the slices with very high number of misses) and therefore SimPoint severely overestimates the reduction in the number of misses due to the prefetcher. We believe that using more SimPoint slices or a different criteria (such as suggested in [4]) instead of the Basic Block Vectors [10] may be more effective in the case of these memory intensive benchmarks.

5.2 Branch Prediction Results

5.2.1 Overall Branch Prediction Accuracy

Table IV on the preceding page lists the actual overall branch prediction accuracy (obtained from simulation of the entire benchmark) as well as the SimPoint estimated branch prediction accuracy for several of the SPEC 2000 benchmark applications. The top half of the table summarizes the results for eight of the SPEC INT benchmarks, and the bottom half contains the results for twelve benchmarks from the SPEC FP suite. In the last column, the difference between the actual and SimPoint estimated branch prediction accuracies is calculated. As you can see, for most of the benchmarks (14 out of the 20), the difference between the SimPoint estimated prediction accuracy and the actual prediction accuracy is less than one percent. This suggests that SimPoint does a pretty good job of predicting the overall branch prediction accuracy. The two worst cases were both floating point benchmarks (lucas and aspi) where the difference rose to as high as five percent.

To make a stronger claim about SimPoints ability to estimate the overall branch prediction accuracy, we conducted a "before and after" significance test to determine whether or not the difference between the actual and SimPoint estimated branch prediction accuracies was statistically significant. Taking the mean and standard deviation of the differences column in Table IV, we computed the 95% confidence interval to be [-0.11, 1.48]. Since this interval contains zero, we can conclude that the difference is not statistically significant. This further attests that SimPoint does a good job of estimating the overall branch prediction accuracy.

As mentioned in the methodologies section, the reason we performed this experiment was so that we could compare our results with some previously done related work. Calder et al. in [9] performed a similar experiment to the one we did above and reached the same conclusion that we made. Specifically, their results also showed that most of the time, the estimated branch prediction accuracy obtained using SimPoint was able to come within one percent of the actual branch prediction accuracy obtained from complete simulation of the benchmark. Since our results closely matched other previous results, we were confident that our methodologies were correct and were ready to perform the branch phase behavior experiments.

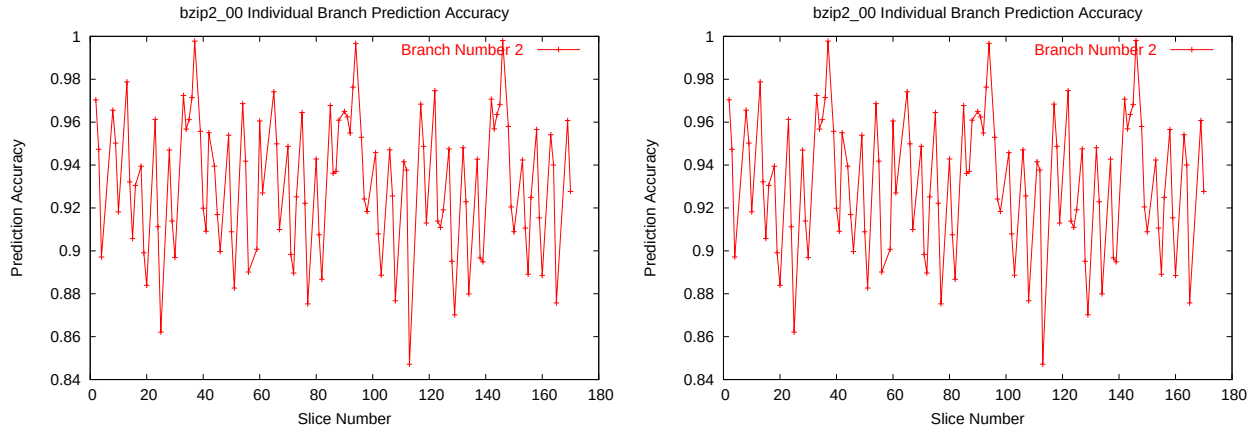
5.2.2 Individual Branch Phase Behavior Results

Figure 2 on the next page graphically illustrates some of the branches that were identified that exhibited the type of phase behavior we were interested in. The top two graphs are from bzip, the middle two from vpr, and the bottom two are from gap. The x-axis on the graphs corresponds to the slice or interval number, and the y-axis is the branch prediction accuracy during that particular interval. All of these branches had several intervals where the prediction accuracy was very high (greater than 95%) but also had several intervals where the prediction accuracy was low (less than 90%). Sometimes the variation was sporadic changing dramatically between consecutive intervals (bzip). On the other hand there were also branches such as those from vpr where the accuracy remained fairly constant for a while and then suddenly changed.

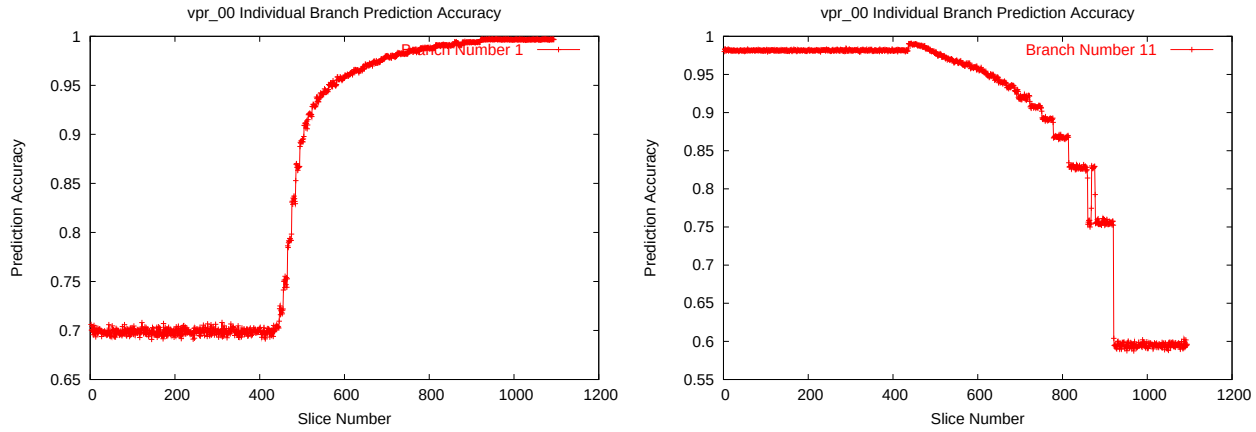
Table V on page 9 shows the results of the individual branch phase behavior experiments that we conducted. The top half of the table lists the results for eight benchmarks from SPEC INT 2000, and the latter half contains the results of twelve benchmarks from SPEC FP 2000. The middle column lists the actual number of static branches identified that exhibited the specific type of phase behavior we were looking for. The last column shows the number of branches identified when SimPoint was used.

The results from the SPEC FP benchmarks were not that meaningful. As you can see, for more than half of them the number of branches actually identified was too small for any evaluation of SimPoint to be done. For the remaining benchmarks, the data shows that SimPoint was only able to identify a small subset of the actually identified branches. The worst case was galgel, where SimPoint was not even able to identify a single one of the 20 actually identified branches.

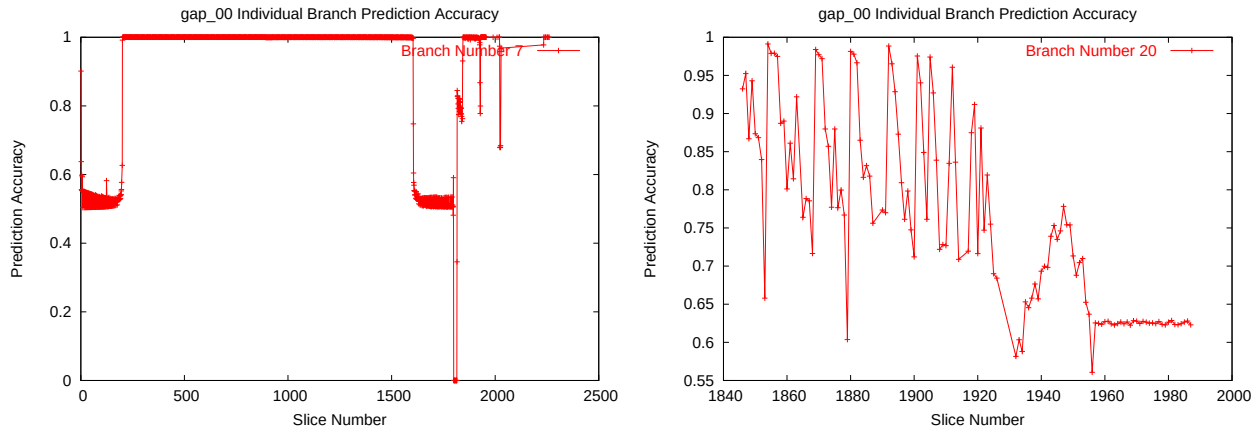
The SPEC INT benchmarks however provided better results. As you can see, for most of the benchmarks, SimPoint only identified a very small percent-



(a) Two branches from bzip that show phase behavior



(b) Two branches from vpr that show phase behavior



(c) Two branches from gap that show phase behavior

Fig. 2. Examples of branches identified by our criteria

TABLE V
COMPARISON OF THE NUMBER OF BRANCHES ACTUALLY IDENTIFIED AND THE NUMBER SIMPOINT IDENTIFIED

Benchmark	Actual	SimPoint
bzip2_00	24	9
gcc_00	468	49
gzip_00	23	9
mcf_00	32	6
perlbmk_00	75	1
vpr_00	15	15
crafty_00	493	302
gap_00	167	6
ammp_00	29	18
applu_00	0	0
apsi_00	0	0
art_00	1	1
equake_00	3	0
facerec_00	0	0
fma3d_00	47	17
galgel_00	20	0
lucas_00	0	0
mesa_00	5	3
mgrid_00	0	0
swim_00	0	0

age of the branches that were actually identified. The exceptions were crafty and vpr. For crafty, out of the 493 branches actually identified, SimPoint identified 302 of them (about 61%). For vpr, SimPoint was able to identify all 15 of the branches that were actually identified. However, for the remaining benchmarks, SimPoint performed rather poorly. The worst case was for perlbmk where SimPoint only identified 1 out of the 75 actually identified branches.

This data suggests that SimPoint cannot capture specific phase behavior of individual branches within a program. We believe that the fundamental reason for this is that in order to capture variation, many data points are needed, and although SimPoint tries to accurately capture the most representative samples from a program, the number of data points reported is not enough to capture phase behavior very well.

6 Future Work

Our interest in this research has increased substantially as we have made progress on this project. We plan to further investigate our conclusions derived from the study of prefetching as well as branch prediction. We will discuss our specific plans in the following sub-sections.

6.1 Prefetching

In the prefetching area, we first want to validate our setup and tools in a greater detail because we feel that a completely validated system is pivotal for further research. We plan to accomplish this by running SPEC CPU 2000 benchmarks on an x86 simulator that is confidently used by many researchers. This process will require a lot of time hence we have not been able to perform this task so far. Once we validate the setup, the next task will be to expand our model from a cache simulator to a complete timing model of a super scalar processor so we can compute IPC, and make a stronger claim regarding the validity of SimPoint. We also need to run more benchmarks. For this report, we were able to run 9 out of 12 SPEC INT benchmarks and only 1 out of 14 SPEC FP benchmarks. Our claim from the results so far is that SimPoint does not estimate the improvement due to a hardware prefetcher accurately for memory intensive benchmarks, but to establish this claim we may have to run similar analysis on other benchmarks which are known to be more memory intensive compared to SPEC CPU 2000 benchmarks. These results, once validated, can be used in the future as a guideline for computer architects doing research in the field of prefetching. The conclusion may also be useful in enhancing the performance of tools similar to SimPoint.

6.2 Branch Prediction

The study of individual branch behavior has provided us multiple directions in which further research can be conducted. We believe that studying the behavior of individual branches can lead to improved branch predictors or compiler mechanisms that may improve overall branch prediction accuracy. Some of the ideas we plan to explore are listed below:

1) Track the branches that show phase behavior back to the source code level and try to understand the reason for such behavior and its dependence on the input data, ISA, and the branch predictor.

2) Try different criteria for selecting branches with phase behavior and try to find the best criteria for identifying such branches using the complete reference input set and just using SimPoint.
3) Conduct a similar study for other code reduction techniques like minneSpec [7], truncation, etc. (a study similar to the one presented in [11] but with different criteria).

7 Conclusion

In this project, we evaluated the effectiveness of using SimPoint for two particular architectural studies, prefetching and branch prediction. For prefetching, we wanted to see if SimPoint could accurately captured the actual relative performance improvement obtained from prefetching. For branch prediction, we wanted to see if SimPoint could be used to identify individual branches that exhibited a specific type of behavior.

We concluded that SimPoint was able to capture the relative improvement due to prefetching effectively for the benchmarks that exhibited an already high cache hit ratio but it was not capable of estimating this improvement for benchmarks with low cache hit ratios. These particular benchmarks (with low cache hit ratios) are known to be more memory intensive and therefore they are good candidates for testing the effectiveness of a prefetcher. Since SimPoint was unable to accurately capture the relative improvement for the benchmarks that were most relevant for prefetching, we conclude that SimPoint cannot be confidently used for prefetching studies. We believe that using a different criteria rather than the Basic Block Vectors for grouping intervals could yield better results. Specifically, for cache studies it may be a better idea to use the spatial and temporal locality distributions introduced in [2], [3], and [4] to group similar intervals together.

Our branch prediction data shows that although SimPoint does estimate the overall branch prediction accuracy quite well, it cannot capture the phase behavior exhibited by certain individual branches. We believe that in order to capture variation in behavior of a branch a larger set of data points is needed than used by SimPoint. We plan to further investigate other criteria used to identify branches that show phase behavior in the hope of identifying more of these branches both with and without the use of SimPoint.

References

[1] Simpoint. http://www.cs.ucsd.edu/calder/simpoint/, March 2005.

[2] Thomas M. Conte and Wen mej W. Hwu. Benchmark characterization for experimental system evaluation. In *1990 Hawaii international Conference on System Sciences (HICSS)*, volume 1, pages 6–18, 1990.

[3] Thierry Lafage and Andre; Seznec. Choosing representative slices of program execution for microarchitecture simulations: a preliminary application to the data stream. pages 145–163, 2001.

[4] Jeremy Lau, Stefan Schoenmackers, and Brad Calder. Structures for phase classification. In *Proceedings of the IEEE International Symposium on Performance Analysis of Systems and Software*. IEEE Computer Society, 2004.

[5] Onur Mutlu, Hyesoon Kim, and Yale Patt. Efficient processing in runahead execution engines. In *ISCA '05: Proceedings of the 32nd annual international symposium on Computer architecture*, 2005.

[6] Frank P. O'Connell and Steven W. White. Power3: The next generation of powerpc processors. *IBM Journal of Research and Development*, 44(6):873–884, 2000.

[7] A. J. Klein Osowski and David J. Lilja. Minnespec: A new spec benchmark workload for simulation-based computer architecture research. *Computer Architecture Letters*, 1, 2002.

[8] Vijay Janapa Reddi, Alex Settle, Daniel A. Connors, and Robert S. Cohn. Pin: A binary instrumentation tool for computer architecture research and education. In *Proceedings of the Workshop on Computer Architecture Education*, June 2004.

[9] Timothy Sherwood, Erez Perelman, and Brad Calder. Basic block distribution analysis to find periodic behavior and simulation points in applications. In *PACT '01: Proceedings of the 2001 International Conference on Parallel Architectures and Compilation Techniques*, pages 3–14. IEEE Computer Society, 2001.

[10] Timothy Sherwood, Erez Perelman, Greg Hamerly, and Brad Calder. Automatically characterizing large scale program behavior. In *ASPLOS*, pages 45–57, 2002.

[11] Joshua J. Yi, Sreekumar V. Kodakara, Resit Sendag, David J. Lilja, and Douglas M. Hawkins. Characterizing and comparing prevailing simulation techniques. Laboratory for Advanced Research in Computing Technology and Compilers, February 2005.

Appendix I
Prefetch Data Sample

```
########################SLICE NUMBER1
1 DL2          L2 Data Cache:
1 DL2          Load-Hits:               1851    3.28%
1 DL2          Load-Misses:            54531   96.72%
1 DL2          Load-Accesses:          56382  100.00%
1 DL2
1 DL2          STORE-Hits:               430    0.39%
1 DL2          STORE-Misses:          110987   99.61%
1 DL2          STORE-Accesses:        111417  100.00%
1 DL2
1 DL2          PREFETCH-Hits:              0     nan%
1 DL2          PREFETCH-Misses:            0     nan%
1 DL2          PREFETCH-Accesses:          0     nan%
1 DL2
1 DL2          Total-Hits:              2281    1.36%
1 DL2          Total-Misses:          165518   98.64%
1 DL2          Total-Accesses:        167799  100.00%

########################SLICE NUMBER2
2 DL2          L2 Data Cache:
2 DL2          Load-Hits:             494086   64.75%
2 DL2          Load-Misses:           268954   35.25%
2 DL2          Load-Accesses:         763040  100.00%
2 DL2
2 DL2          STORE-Hits:            493072   54.49%
2 DL2          STORE-Misses:          411735   45.51%
2 DL2          STORE-Accesses:        904807  100.00%
2 DL2
2 DL2          PREFETCH-Hits:              0     nan%
2 DL2          PREFETCH-Misses:            0     nan%
2 DL2          PREFETCH-Accesses:          0     nan%
2 DL2
2 DL2          Total-Hits:            987158   59.19%
2 DL2          Total-Misses:          680689   40.81%
2 DL2          Total-Accesses:       1667847  100.00%

########################SLICE NUMBER3
3 DL2          L2 Data Cache:
3 DL2          Load-Hits:            1303796   66.37%
3 DL2          Load-Misses:           660645   33.63%
3 DL2          Load-Accesses:        1964441  100.00%
3 DL2
3 DL2          STORE-Hits:            660322   50.01%
3 DL2          STORE-Misses:          660067   49.99%
3 DL2          STORE-Accesses:       1320389  100.00%
3 DL2
3 DL2          PREFETCH-Hits:              0     nan%
3 DL2          PREFETCH-Misses:            0     nan%
3 DL2          PREFETCH-Accesses:          0     nan%
3 DL2
3 DL2          Total-Hits:           1964118   59.79%
3 DL2          Total-Misses:         1320712   40.21%
3 DL2          Total-Accesses:       3284830  100.00%
```

Appendix II
Branch Prediction Data Sample

```
########################SLICE NUMBER1

References =15604926
Mispredicts =142544
NumItems 1048
DATA:START
#  counters
# :
0x004a685d:            1            0
    .
    .
    .

0x0804869d:            0            1
0x080486f1:            0            1
0x0804c052:            1       608073
0x0804c06d:        57026       673185
0x0804c075:           23       122137
0x0804c07a:            2       608072
0x0804c08d:            1       608072
0x0804c092:        29756       578317
0x0804c096:            1        27946
0x0804c0be:        23290       126794
0x0804c13d:            1          255
0x0804c15f:            1            0
0x0804c16b:            1            0
0x0804c18b:            1       608072
0x0804c19b:        17792       590281
0x0804c1a4:        10715       579960
0x0804c1ad:            0       580126
0x0804c22e:            2       608071
0x0804c236:            0       608073
0x0804c260:         1822         8727
0x0804c350:            1            0
0x0804c374:            0            1
0x0804c3a8:            0            1
0x0804c3d1:            0            1
0x0804cf4c:            1            0
0x0804cf8f:            1            0
0x0804f4f3:            1            0
0x0804f4fd:            1            0
0x0804f507:            1            0
0x0804f50b:            0            1
0x0804ff52:            0            1
0x0804ff9c:            1            2
0x0804ffa2:            1            2
0x0804ffba:            3        89085
0x0804ffc3:            1            2
0x08050281:            1            0
0x080502a6:            0            1
0x080502c6:            2           72
0x080502ca:            1           72
0x080502da:            0           73
```

```
0x080502f5:           1           0
0x08050304:           1           1
0x0805030f:           1           1
0x0805033b:           1           1
0x080504a3:           1     1338284
0x080504a8:           1     1338284
0x080504bf:           0     1338285
0x08050502:           1     1338284
0x08050563:           1      608073
0x0805056c:           1      608073
0x0805057b:           1      608073
0x0805058f:           2      608072
0x08050598:           1      608073
0x08050687:           1           0
0x08050692:           0           1
0x0805069b:           1           0
0x080506a7:           1           0
0x080506b0:           1           0
0x080506e2:           1           0
0x080506fe:           0           1
0x0805070b:           1           0
0x08050721:           1           0
0x0805072b:           1           0
0x0805074c:           2       29608
0x08050759:           0           1
0x08050784:           0           1
0x08050b1c:           1           2
0x08050b21:           1           2
0x08050ba3:           0           1
0x08050bfe:           1           0
DATA:END
```

Appendix III
List of Scripts

III.1 run_pin

Runs PIN instrumentation tool with a benchmark, input set, and a PIN tool specified from the command line.

III.2 get_gshare_data.pl

Parses the raw branch prediction data file to gather all branch data and identify branches of interest based on a criteria specified from the command line. Also picks branches identified using simpoints based on the same criteria and returns output in a format specified by the command line.

III.3 get_branch_plots.tcsh

Calls the get_gshare_data.pl with +plot option to generate data files that can be plotted to generate graphs similar to 2. This also generates a gnuplot script that can be used to graphically view all the data files.

III.4 get_gshare_indiv_specfp.tcsh

Calls the get_gshare_data.pl and reports the number of branches of interest identified (using all slices and using only SimPoint slices) in raw branch prediction data output files from SPECFP.

III.5 get_gshare_indiv_specint.tcsh

Calls the get_gshare_data.pl and reports the number of branches of interest identified (using all slices and using only SimPoint slices) in a raw branch prediction data output files from SPECINT.

III.6 get_gshare_simpoints.pl

Parses the branch prediction raw data file and the output from SimPoint to generate the actual branch prediction accuracy and simpoint estimate branch prediction accuracy.

III.7 get_gshare_simpoints_specfp.tcsh

Calls get_gshare_gshare_simpoints.pl for all the branch prediction files from SPEC FP.

III.8 get_gshare_simpoints_specint.tcsh

Calls get_gshare_gshare_simpoints.pl for all the branch prediction files from SPEC INT.

III.9 get_prefetch_simpoints.pl

Parses the prefetching raw data file and the output from SimPoint to generate the actual hit ratio and simpoint estimate hit ratio.

III.10 get_prefetch_simpoints_specfp.tcsh

Calls get_gshare_gshare_simpoints.pl for all the branch prediction files from SPEC FP.

III.11 get_prefetch_simpoints_specint.tcsh

Calls get_gshare_gshare_simpoints.pl for all the branch prediction files from SPEC INT.

III.12 parse_gshare_simpoints.awk

Parses the output from get_gshare_simpoints_specint.tcsh and get_gshare_simpoints_specfp.tcsh to generate a table similar to Table IV.

III.13 parse_identified.awk

Parses the output from get_gshare_indiv_specfp.tcsh and get_gshare_indiv_specint.tcsh to generate a table similar to Table V.

III.14 parse_prefetch_simpoints.awk

Parses the output from get_prefetch_simpoints_specint.tcsh and get_prefetch_simpoints_specfp.tcsh to generate a table similar to Table I.