Exploring the Impact of Normality and Significance Tests in Architecture Experiments

Pitchaya Sitthi-amorn, Dee A.B. Weikle, Kevin Skadron
Department of Computer Science
University of Virginia
{ps4wd, dweikle, skadron}@cs.virginia.edu

ABSTRACT

Computer architects often use statistical tools such as means in reporting results. While there has been discussion regarding which means to use for different metrics, the impact of the underlying assumptions involved in reporting results in the architecture community has been largely unexplored. This paper investigates the validity of assumptions such as the normality of the data gathered and the use of significance tests. These are demonstrated on actual branch predictor experiments using the SPECcpu2000 benchmark suite. We find our measures (IPC, branch misprediction, correct branch direction, and correct branch address rates) are mostly normally distributed in these experiments. Through the use of additional statistical tests, we also illustrate that simple visual inspection of results can be misleading, implying differences where no statistical difference exists or disguising a difference that is important.

1. INTRODUCTION

The field of computer architecture research relies heavily on results obtained from running large sets of experiments on benchmark suites, most often using a simulator or set of simulators. This methodology attempts to reduce a large set of data into a smaller, understandable, and yet useful set of information. Generally, this information is in the form of conclusions about the relative performance among certain hardware/software configurations according to particular evaluation metrics, or about general trends observed during the experiments. The goal is to determine whether a particular change in the architecture design will benefit the performance of the computer system and chosen workload as a whole.

This methodology requires several infrastructure components as shown in Figure 1. First, a benchmark suite is determined to represent the workload of interest, and then, because of resource constraints (simulation time, for example), the benchmark suite is sampled. Second, a simulator is designed to model both the new and old architectures. Operating system and compiler features are hopefully addressed in both the benchmarking component of the experiments and the simulator. Once the experiments have been run, the final step is analysis of the simulator results. Our focus is this last step of analysis.

The analysis process is where final conclusions are drawn and trends are discovered. It is the critical point where a researcher attempts to make sense of the data that has been gathered. This process makes use of many evaluation metrics, some of which are specific to the area of interest, such

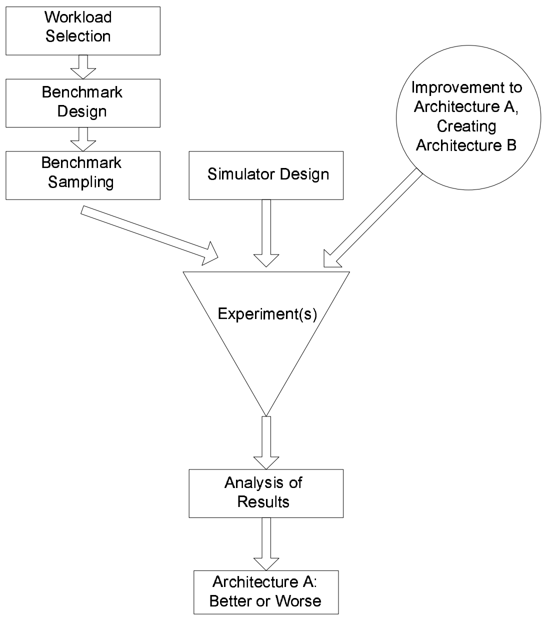

Figure 1: Graphical Representation of Architecture Experiment Process

as branch prediction rates, cache hit rates, or bandwidth usage. Nearly all architecture experiments, though, are interested in the effect of a new idea on overall execution time, so the metrics of IPC (Instructions Per Clock) and/or CPI (Clocks Per Instruction) are of particular interest.

Analysis requires statistics to summarize experimental results and infer their significance. The architecture community is becoming more aware of the importance of the formal statistical techniques in presenting and interpreting results. Discussions about which means to use for which evaluation metrics, as well as the importance of validating the underlying assumptions for using these statistics, are given in Mashey [7], John and Eeckhout [5], and Hennessy and Patterson [4].

Many statistical tests require validation of underlying assumptions about the distribution of the data. Since the majority of statistical techniques currently used in the architecture community require a normal distribution, we demonstrate the application of the Shapiro-Wilk test for normality. We also demonstrate two methods of determining the significance of differences in results, confidence intervals and the Student's Paired t-test. The main contribution is demonstrating these methods on data from actual experimental research results, and showing how conclusions may differ.

2. RELATED WORK

In [7], Mashey describes the history of the development of benchmark suites with emphasis on the series of SPEC suites. In addition, he discusses the importance of understanding statistics and their underlying assumptions, including distributions. Some textbook examples are given and a runtime example of an IBM eServer executing the SPECcpu2000 benchmark suite is explored. We apply some of the same techniques, but use the Shapiro-Wilk test for normality instead of the Coefficient of Determination, and analyze results from SimpleScalar branch predictor experiments. In addition, we address the issue of sampling of benchmark suites for the purpose of weighted analysis. Mashey also performs experiments to determine whether or not experimental data is distributed lognormally. For our data, an initial lognormal analysis was performed, but was inconclusive. Further investigation and reporting of these results is left as future work. John and Eeckhout [5] provide an overview of the current state of benchmarking, simulation techniques, and statistical experiment design. Chapter 4, in particular, provides an overview of the different means currently in use in the computer architecture community and discusses their proper use for different evaluation metrics. Almeldeen and Wood [1] use confidence intervals and hypothesis testing to describe the variability in results from experiments on multi-threaded workloads. We perform a similar analysis here for a set of branch predictor experiments, but also include some analysis using weighted means and the associated confidence intervals. Lilja [6] provides a detailed background on computer experimental methodology and related statistics, but does not perform tests on the underlying distribution.

3. BACKGROUND

While describing the history of benchmark suites, Mashey [7] also describes three separate methods of benchmark analysis. These are Workload Characterization Analysis (WCA), Sample Estimation of Relative Performance Of Programs (SERPOP), and Workload Analysis with Weights (WAW). WCA assumes information (detailed or general) has been gathered about existing workloads for a given machine or set of machines so some estimate is available of how prevalent any particular program may be in the original workload. SERPOP analysis creates a benchmark suite of several programs, including their inputs, and characterizes it as a sample of a larger population of programs. Ideally, this would include an appropriate model of the population's distribution. WAW assumes that extensive WCA has been performed and, if performed completely correctly, is able to predict the performance of actual *workloads* under different assumptions. He goes on to claim that, in practice, people do as much workload characterization analysis as they can afford, and use workload analysis with weights when they can, but frequently must employ sample estimation because they do not have enough data.

Using sample estimation of relative performance of programs (SERPOP) accurately, requires knowing the underlying distribution of the population being sampled. In most common practice for computer architects today, this implies knowing the underlying distribution of the SPECcpu2000 benchmark suite. Reporting means and other statistics on this suite, as is currently done, assumes that the SPECcpu2000 benchmark suite is a representative sample of a normally distributed population AND that the metrics that we report for that suite are also distributed normally. It is outside the scope of this paper to address whether the SPECcpu2000 benchmark suite is an appropriate sample of a normal distribution. However, if the community can understand that results are reported for the benchmarks given, which may or may not represent a particular workload, then we can focus on the method of compiling statistics appropriately for a particular set of programs and allow those programs that represent a given workload to be chosen by the user of our research. This paper focusses on applying appropriate statistics to a study of the relative performance of a set of branch predictors for the SPECcpu2000 benchmark suite. In particular we test the distribution of our metrics and then the significance of those results through the use of confidence intervals and the Student's T-test. Section 4 explains the metrics and analysis tools used to perform the analysis, building up from the simplest (means) to the more complex (hypothesis testing and significance tests). The following sections describe our application of these statistics to the data from our branch predictor study.

4. STATISTICAL ANALYSIS TOOLS

While the overall process of determining if a mean is an appropriate measure begins with determining the distribution (i.e. checking for normality), performing normality tests require first computing the mean and understanding hypothesis testing. So, we begin our discussion of tools by discussing means and weighting means. Then we introduce hypothesis testing. Hypothesis testing is used in both types of significance tests we describe, confidence intervals and the paired student's T-test. This section then ends with a discussion of testing a data set for normality.

4.1 Means and Weighted Means

There has been discussion in the architecture community about which mean should be used to report results. John and Eeckhout, Chapter 4 [5] details which mean to use for which metric. They choose weighting factors to develop a mean for each metric that would give the same result as running the entire suite as a single benchmark. Mashey [7] focusses on weighting individual benchmarks that contribute to a mean in such a way as to reflect a representative workload, and reports multiple means for each study. He also claims that the arithmetic mean or average is generally used when the data set has a normal distribution, or is the result of many additive effects. Hennessy and Patterson [4] use the arithmetic mean to predict performance. Similarly, we use the arithmetic mean for branch prediction results to provide consistency with average misprediction rates. We report a geometric mean in our tables for comparison, but do not make explicit conclusions regarding the geometric mean. A future analysis could address different means in the context of distributions and significance tests.

Though we use the arithmetic mean, we do weight the mean to address the differences in the individual benchmarks. For example, consider a suite of benchmarks each with a fixed number of instructions, and how it should be used to report an average branch prediction rate. Each benchmark will have a different number of branches for the same number of instructions. In the extreme case (albeit unrealistic), a benchmark could have only one branch that is always correctly predicted. In this case it would contribute a 100% correct prediction rate to the average, potentially skewing the results in such a way that would not be reflected in the actual performance of a final system. To eliminate this effect, we use a weighted mean to adjust the effect of a single point on the overall average.

The formula for the arithmetic mean is well-known as

$$\bar{x} = \frac{1}{N} \sum x_i.$$

where the x_i are the metric values being averaged, and N is the number of benchmarks in the suite.

For the weighted mean we use:

$$\bar{x}_w = \frac{1}{N} \sum w_i x_i$$

where

$$w_i = N \frac{v_i}{V}$$

are the weights.

v_i = number of the characteristic of interest in benchmark i, and V = the total number of the characteristic of interest in the whole suite. For example, in our branch predictor study, v_i = number of branches for benchmark i, and V = sum of all the v_i, or total number of branches in the suite. This weighting factor depends completely on the metric being weighted, and the goals involved in the weighting as described above. Some metrics already have the weights "built-in", such as misses-per-thousand-instructions, allowing them to be compared between benchmarks without further weighting. Still, if the goal is to aggregate such a metric over the suite in such a way that the mean of the suite is to be the same as running the suite as a single benchmark, the misses-per-thousand-instruction metric would need to be weighted with the number of instructions as the characteristic of interest.

4.2 Hypothesis Testing

In statistics, hypothesis testing is used to make a statement (hypothesis) that is true with a certain confidence level. Traditionally, the hypothesis that we want to prove is called the null hypothesis or H_0. Similarly, the opposite hypothesis (that H_0 is not true, is traditionally called the alternative hypothesis or H_1. The conclusion from hypothesis testing is then to either reject H_0 in favor of H_1 or do not reject H_0.

Unlike the some common usages of the term "hypothesis," statistical hypotheses are stated in a strictly true-false form for proof or disproof. Examples of H_0 from this study are "the data set is normally distributed", or "metric X on Architecture A is equal to metric X on Architecture B."

4.2.1 Types of Error in Hypothesis Testing

The errors that can occur with hypothesis testing are important because they are how the accuracy of the final conclusions are determined. These errors are classified into two different types. Type I errors occur when H_0 is wrongly rejected. The probability with which the type I error occurs is called the significance level and labeled α. The type II error occurs when the hypothesis test does not reject H_0 while H_1 is true, and is called β.

4.2.2 The Significance Test

The significance test is used to determine if a statistical value, p, differs from the hypothesis significance level, α. p values are determined by making a calculation for a given distribution, then using that calculation to index into a statistics table for that distribution to find the p value. The null hypothesis is rejected if p is at or below α. For example, if the null hypothesis states that it is true with an $\alpha = 5\%$ significance level, but the p value is 4%, then the null hypothesis is rejected.

4.3 Confidence Interval for Mean

If the data summarized by the mean is normally distributed, we can find the range (a low and high value) that the actual mean falls between with some probability (usually 95%). This interval is called the (95%) confidence interval (C.I.). If the confidence intervals of two different alternatives are not overlapped, it can be stated that they are significantly different with the confidence interval probability. However, if they are overlapped, the alternatives may or may not be significantly different. The general formula for the confidence interval is

$$(low, hi) = M \pm (t_{1-\alpha/2} \frac{s}{\sqrt{n}})$$

where M is the arithmetic mean, t is the value from a statistics table for the t-distribution (normal distribution) for the significance level α and s is the standard error computed by taking the square root of the variance:

$$s^2 = \sum_{i=1}^{n} \frac{(x_i - \bar{x})^2}{n-1}.$$

Note that the formula gives the confidence interval for the arithmetic mean. Statistics tables can be found in most statistics books such as [8], but another good resource is a Math CRC such as [2].

If we are using a weighted mean, we must compute the confidence interval using a weighted standard error. In this case, the variance is

$$s_w^2 = \frac{\sum w_i (x_i - \bar{x}_w)^2}{N-1}$$

where w_i are as described in Section 4.1 on Means.

4.4 Paired Student's T-test for Mean

Another way to analyze the significance of results is to use the paired student's T-test. In this study, we compare the performance of two processors differing only in their branch predictor on a benchmark suite and want to determine which processor performs better. The null hypothesis can be stated as H_0: the means of the performance metric for the suite using Architecture A are equal to the same means for the suite using Architecture B with significance level $\alpha = 5\%$. So if the p value is less than 5%, the null hypothesis is rejected, and the means have a 5% significant difference. The paired student's t-test is a good fit for this example, because it is computed by pairing the results from the two processors for each benchmark, computing the difference and then calculating the means of the differences.

The paired student's t-test is similar to the confidence interval for the mean. Let the first data set be x_i and the second be y_i and $c_i = x_i - y_i$. The sample standard deviation is

$$s_c = \sqrt{\sum_{i=1}^{n} \frac{(c_i - \bar{c})^2}{n-1}} = \sqrt{\frac{s_x^2}{n_x} + \frac{s_y^2}{n_y}}.$$

Note in our example the number of benchmarks in each suite is the same so $n_x = n_y$. s_x^2 is the standard deviation of the first data set, s_y^2 is the standard deviation of the second set, and $\bar{c}$ is the mean of the sample differences described above.

The t-statistic, T, is

$$T = \frac{\bar{c}}{s_c}$$

The value of T will be used to look up in the t distribution function table to find the p-value, which is then compared to our desired significance level α as described earlier. Note that there are two kinds of t-tests, one-tailed and two-tailed. The two-tailed t-test checks the alternative hypothesis that the two means of the performance metric are different. In contrast, the one-tailed t-test alternative is either that one mean is greater than the other OR that it is less than the other, but not both. These tests potentially have different outcomes.

Another way to calculate the student's t-test is to find the confidence interval for the mean of c. If zero is not contained in the interval, the null hypothesis is rejected.

Note that the student's t-test requires that the sample data comes from the normal distribution.

4.5 Distribution Testing

Since the paired student's t-test, the confidence interval, and even the arithmetic mean require that the distribution of the data is normal, the data must be tested for normality to insure the validity of the results.

There are several methods to test normality such as the Anderson-Darling test, Kolmogorov-Smirnov, D'Agostino-Pearson, or Shapiro-Wilk. We use the Shapiro-Wilk as it is recommended in [8]. The test is based on the hypothesis test where the null hypothesis states that the data is normal.

The test statistic of Shapiro-Wilk test is defined as

$$W = \frac{(\sum a_i x_i')^2}{\sum (x_i - \bar{x})^2},$$

where x_i are the ordered sample values and the a_i are constants generated from the means, variances and covariances of the order statistics of a sample of size n from a normal distribution. The calculation of the a_i is complicated, and not very enlightening. The reader is referred to the program used to generate results found on the NIST website Chapter 7.2.1.3 [8]. The test statistic is then used to index into a table to find the p-value as described above. If the p-value is greater than our desired error of type I, α then the data is normally distributed.

The Anderson-Darling test is a modification of the Kolmogorov-Smirnov test and has the advantage of being able to test for additional distributions, including normal, lognormal, Weibull, and extreme value Type I. Normal and lognormal are the most common and are the distributions for which most statistics packages are applicable.

5. CASE STUDY: BRANCH PREDICTION

In this section we report results of simulating samples of the SPEC2000 benchmark suite on the same processor with different branch predictors on a modified version of SimpleScalar as our case study. The branch predictors are bimod, gshare.32k, hybrid.32k, alpha.ev6, hashed perceptron, ogehl, and piecewise. For details on the configurations of the branch predictors see Table 2. For other details of the actual experiments see [3]. One large trace sample was taken from each benchmark to minimize warm-up effects. The single sample was determined with a traditional SimPoint analysis. We present results on the normality of the data, weight the means according to branch frequencies within the samples, and report results on IPC and branch prediction rates, including significance tests.

5.1 Analysis Method

When performing the analysis of the results, we used the following steps:

Choose a normality test, report results. Several distribution tests are described briefly in Section **??**. We use the Shapiro-Wilk test because it is recommended in the NIST Handbook [8]. Mashey [7] uses the coefficient of determination, which is available in Microsoft Excel. Which of these tests is better and what values are acceptable for reporting results for the computer architecture community will become more clear as more real data is analyzed.

Determine the group of benchmarks. It is important to note that the group of benchmarks chosen will affect the normality of the resulting data set. Suites should be chosen that are representative in some way to the user of the final analysis. When adding just one extra benchmark, the researcher may want to view the final data with and without that data point to see if the distribution changes significantly. Similarly, if the original suite turns out not to be distributed normally, or lognormally, benchmarks (and additional experiments) can be added with the hope of improving the distribution.

Choose mean(s) to report. A description of a variety of means and how to weight them for particular metrics so that the end result is representative of running a suite as one benchmark is in Chapter 4 of John and Eeckhout [5]. Mashey [7] uses several different means, describing the arithmetic mean as one to use on normal distributions where the values are the results of additive effects, and the geometric mean as one to use for logarithmic distributions where the values are the results of multiplicative effects. Hennessy and Patterson [4] use the arithmetic mean to enable prediction of run-times. We use the arithmetic mean here, weighted to be consistent with the goals in John and Eeckhout, and because our data is at least close to normal.

Choose weights for means. If the benchmarks directly represent a workload of interest, including the relative execution times, the metric values can be weighted with respect to execution times, instruction mix, etc. to ensure that each individual benchmark contributes to the mean appropriately. Now that most benchmarks are only being sampled, the sampling methodology may influence what kind of weighting can be used. For example, if an experiment includes samples from each benchmark such that the same number of instructions are simulated from each benchmark, weights using the number of instructions would not be appropriate as they would all be the same in the experiments but not in the "real world".

Choose significance test. Presented here are confidence intervals and the Paired Student's T-test. Confidence intervals are easier to see on a graph. The Paired Student's T-test may be more sensitive (i.e. register a significant difference more often), based on our limited experiments. However, is needs to be understood that these significance tests rely on the underlying distribution being normal, and only apply to the significance of the metrics as they are reported.

Evaluate significance in context of experiment. As mentioned above, statistical significance tests only capture the importance of variability in the results during the reporting step, they do not address the introduction of error in the other components of the methodology outlined in Figure 1. Error introduced by the other components of the infrastructure such as simulator error, sampling error, or variation in the benchmark suite add variability and decrease confidence and significance levels in most cases. Random errors in the infrastructure might be able to be ignored if the number of experiments are large enough. Systematic errors that affect all results consistently in the same direction should not change the relationship between the results. Quantifying this error and how to propagate it to the analysis stage is beyond the scope of this paper and a subject of future work.

The sections below demonstrate this procedure on the above branch predictor study.

5.2 Normality Test

To determine if the data set is normal, we perform Shapiro-Wilk tests and show results in Table 1. The null hypothesis of Shapiro-Wilk states that the data comes from the normal distribution. Hence, we want the p value to be greater than 5%. However, the p-values of ghare.32k, hashed perceptron, and ogehl are less than 0.05. While they appear "very close," this reduces our confidence levels in the tests on the data for these particular predictors in the significance test for integer IPC. All of the integer prediction rates appear to be normal. This is different from SPEC floating point where the IPC results all are normal according to the Shapiro-Wilk test, but several of the prediction rates are not. This differs from the results in Mashey [7] which indicate the SpecInt values were normally distributed while the SpecFloat values were lognormally distributed.

5.3 Analysis of IPC

From Table 7, we plot the mean IPC values for each predictor configuration and calculate the confidence intervals. The results are shown in Figure 2.

With only the arithmetic means and the confidence intervals, we may not be able to say which branch predictor is better than another. According to Figure 2, they are all

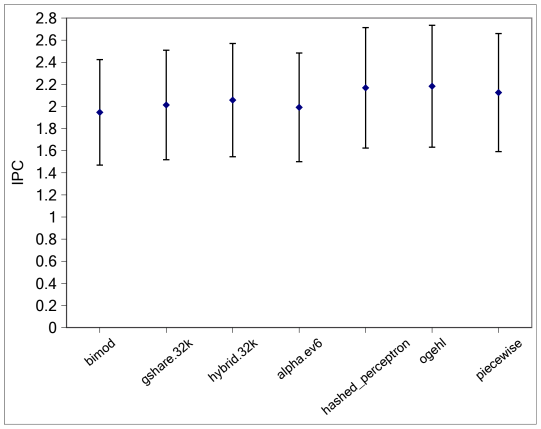

Figure 2: Confidence Interval for Mean of IPC of SPEC Integer benchmark suite

overlapped. Although not shown, the results for the floating point confidence intervals are also overlapped. Hence, the confidence interval suggests that there is no significant difference between these branch predictors. The average IPC of bimod and alpha.ev6 are very close. However, the p value of the paired Student's t-test between bimod and alpha.ev6 is 0.0248000. This implies that they are significantly different. On the other hand, the p value of the t-test between bimod and gshare.32k is 0.0565, so the null hypothesis is rejected and there is no 5% level significant difference between these two. Note that the arithmetic mean of gshare.32k is 2.014. The absolute difference between this and the mean of bimod is greater than that of bimod and alpha.ev6.

Another interesting pair is hashed perceptron and ogehl. The means of both of them are high, but very close together. The confidence intervals are almost impossible to differentiate, but the p value of the t-test in this pair is 0.0086000. (See Table 3.) Based on the t-test we might conclude that ogehl performance is better than hashed perceptron performance for SPEC integer IPC. Notice that from the traditional graph in Figure 5, we could not have seen this difference. From this we conclude that the paired student t-test gives the greatest opportunity for noticing a significant difference. This difference should be considered in light of the error within the experimental infrastructure, however. Simulation error, benchmark sampling error, and errors from insufficient warm-up would all likely make this difference insignificant.

In this paper, we use a one-tail t-test for the paired student's test, where the alternative hypothesis is that one mean is greater than another and the null hypothesis is that the means are equal. The p-values of the paired student t-test in these tables indicate whether the means for predictor a and b are equal. (i.e. if the p-value is less than 0.05 the predictors are considered significantly different). So, if the p-values are greater than 0.05, we do not reject the null hypothesis, so we cannot say that the difference between them is significant. However, because it is a one-tail test we cannot "accept" the alternative hypothesis either. When reading the student t-test tables it is important to realize they are not necessarily symmetric because of the lack of symmetry in the alternative hypothesis. A statistically significant

	SPEC Integer			SPEC Floating		
	IPC	Address	Direction	IPC	Address	Direction
Bimod	0.1158	0.1382	0.957	0.2515	**0.0242**	**0.0173**
Gshare	**0.0418**	0.5089	0.2439	0.2415	0.2195	0.2515
hybrid	0.0652	0.5145	0.4152	0.2457	0.3313	0.4218
alpha	0.0605	0.902	0.6484	0.2636	0.5574	0.7294
hashed perceptron	**0.0447**	0.4102	0.3979	0.1789	**0.0467**	**0.0431**
ogehl	**0.0448**	0.5984	0.3457	0.1785	**0.0011**	**0.001**
piecewise	0.057	0.4513	0.7966	0.1854	**0.0121**	**0.0115**

Table 1: Shapiro-Wilk test p-value of SPECinteger and SPECFloating across different branch predictors (H_0: Distribution is normal, p values greater than 0.05 are normal, highlighted values are not normal)

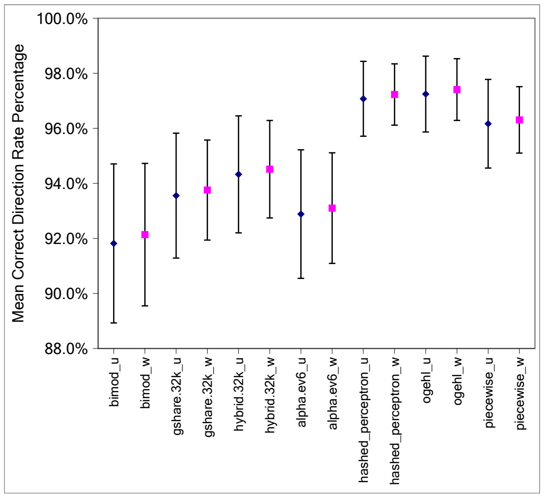

Figure 3: C.I. for Means of Unweighted vs Weighted Correct Direction Rate SPEC Integer benchmark suite

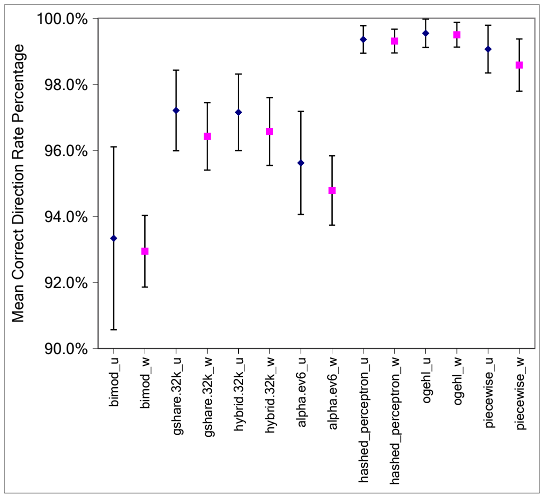

Figure 4: C.I. for Means of Unweighted vs Weighted Correct Direction Rate SPEC Floating Point benchmark suite

difference exists between two predictor means if *either* of the entries in the table for that pair is less than 0.05.

5.4 Branch Prediction Rates

The design specific metrics that we analyze here are the address prediction rate and the correct direction prediction rate. Because these experiments used sampled benchmarks, each one was simulated the same number of instructions. To compensate for this we weight the prediction rates by the number of branches simulated in the sample instead.

From Figure 3, we can see that gshare.32k and alpha.ev6 confidence intervals almost completely overlap those of bimod. (Note that weighted results are presented next to unweighted results, labelled with w and u repectively.) The result of the t-test agrees that there is no significant difference among them both in weighted and unweighted data for the Integer benchmarks.

Since the correct prediction rates are very close to one and the weights are very small, (see Table 9) the weighted rates are primarily influenced by the weight. Therefore, the normality testing of branch prediction rates is the result of testing the normality of the distributions of the branches of the benchmark suite with associated noise from the prediction rates. Note that the p value of only the branch weights for the integer benchmarks is 0.066, and p value of the Shapiro-Wilk test of SPEC integer branch prediction rates are close to 0.066 (i.e. they are equally normal).

The weights for the SPEC floating point benchmarks in Table 1 are also close to normally distributed. It is interesting to note, though, that for the floating point benchmarks the confidence intervals (particularly for bimod) are significantly tighter.

6. CONCLUSIONS

Means alone are very limited in showing the tradeoffs between alternative designs. With confidence intervals for the mean, we can visually see where different alternatives may overlap in performance. The students t-test provides a more sensitive method of determining significant differences, but are more difficult to view in table format. We describe these statistical methods in detail and apply them to a set of experiments with different branch predictors. We find that some results that were not significantly different visually, are statistically different according to the student's t-test. This significant difference is only a statistical significance, however. It does not consider the error from other aspects of the experimental infrastructure such as simulation error, or benchmark sampling error. We also test this data for normality using the Shapiro-Wilk test and find that, with a couple of exceptions, it is normally distributed.

7. FUTURE WORK

The results presented here are intriguing. Enough of the distributions turn out to be borderline normal that it would be interesting to test the distributions of other sets of data from real experiments, and to add a lognormal test to the experiments. The implications of different means such as the geometric mean in conducting significance tests could also be explored. In addition, we were unable to appropriately compute weighted p values for the paired student's t-test. This test appears to be more sensitive, so it would be useful to see the results on the weighted means. Finally, although these statistical results indicate significant differences here, the authors believe it is not really enough to account for possible errors in the other steps of the experimental methodology outlined in the introduction. An extension of this work would be to describe how to quantify an accumulated error or confidence interval through the whole experimental process so that the end result can be validated. The approach for this could be to see each step outlined in the process as a random variable with a certain amount of error.

Acknowledgments

This work is supported in part by the National Science Foundation under grant nos. NSF CAREER award CCR-0133634, and CNS-0340813, and a grant from Intel MRL. We would also like to thank Michele Co and John Mashey for their helpful input.

8. REFERENCES

[1] A. Alameldeen and D. Wood. Variability in architectural simulations of multi-threaded workloads. *Proceedings of the Ninth IEEE Symposium on High-Performance Computer Architecture*, 2003.

[2] William H. Beyer, editor. *CRC Standard Mathematical Tables, 27th Edition*. CRC Press Inc., Boca Raton, FL, USA, 1984.

[3] M. Co and K. Skadron. Evaluating trace cache energy-efficiency **(pending revisions)**. *ACM Transactions on Architecture and Code Optimization*, pages –, 2005.

[4] J. Hennessy and D. Patterson. *Computer Architecture - A Quantitative Approach, Third Edition*. Morgan Kaufmann Publishers, San Franciscon, CA, 2003.

[5] L. K. John and L. Eeckhout. *Performance Evaluation and Benchmarking*. CRC Press, Boca Raton, FL, USA, 2006.

[6] D. J. Lilja. *Measuring computer performance: a practitioner's guide*. Cambridge University Press, New York, NY, USA, 2000.

[7] J. R. Mashey. War of the benchmark means: Time for a truce. *SIGARCH Computer Architure News*, 32(4):1–14, 2004.

[8] NIST/SEMATECH. *e-Handbook of Statistical Methods*. 2006.

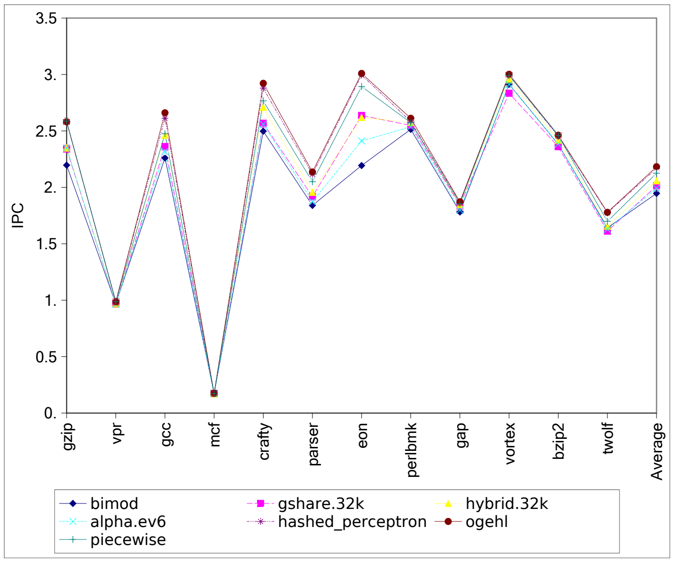

Figure 5: IPC of different SpecINT benchmarks across different branch predictors

Branch Predictor	Area (KB)	Configuration
gshare	8	L1: 15-bit, L2: 32K 2-bit counters, XOR: on
hybrid	8	8K-entry metachooser 8K-entry bimodal L1: 14-bit global history register L2: 16K-entry 2-bit counters
piecewise linear	32	34360 7-bit general weights, 2396 bias weights, 220 16-bit local history registers, 26-bit global history register
bimodal	1	4K-entry 2-bit counters
hashed perceptron	8	64 global history x 10 perceptrons per history, 10 local history
O-GEHL	8	8 2K-entry 4-bit counter tables, 1K-entry 1-bit tag table

Table 2: Branch predictor configurations evaluated.

	bimod	gshare.32k	hybrid.32k	alpha.ev6	hashed perceptron	ogehl	piecewise
bimod	-	0.9435	0.9949	0.9752	0.9966	0.9972	0.9947
gshare.32k	0.0565	-	0.9931	0.1647	0.9995	0.9996	0.9994
hybrid.32k	0.0051	0.0069	-	0.0032	0.9970	0.9981	0.9872
alpha.ev6	0.0248	0.8353	0.9968	-	0.9980	0.9984	0.9974
hashed perceptron	0.0034	0.0005	0.0030	0.0020	-	0.9914	0.0080
ogehl	0.0028	0.0004	0.0019	0.0016	0.0086	-	0.0062
piecewise	0.0053	0.0006	0.0128	0.0026	0.9920	0.9938	-

Table 3: p-value for Integer-IPC, H_0: The mean of the top predictor is less than the mean of the left predictor, values less than 0.05 are significant

	bimod	gshare.32k	hybrid.32k	alpha.ev6	hashed perceptron	ogehl	piecewise
bimod	-	0.9757	0.9972	0.9726	0.9994	0.9995	0.9987
gshare.32k	0.0243	-	0.9978	0.0388	0.9999	0.9999	0.9999
hybrid.32k	0.0028	0.0022	-	0.0005	0.9995	0.9997	0.9974
alpha.ev6	0.0274	0.9612	0.9995	-	0.9998	0.9999	0.9997
hashed perceptron	0.0006	0.0001	0.0005	0.0002	-	0.9769	0.0033
ogehl	0.0005	0.0001	0.0003	0.0001	0.0231	-	0.0028
piecewise	0.0013	0.0001	0.0026	0.0003	0.9967	0.9972	-

Table 4: p-value for Integer Correct Direction Rate, H_0: The mean of the top predictor is less than the mean of the left predictor, values less than 0.05 are significant

	bimod	gshare.32k	hybrid.32k	alpha.ev6	hashed perceptron	ogehl	piecewise
bimod	-	0.9927	0.9933	0.9808	0.9933	0.9938	0.9967
gshare.32k	0.0073	-	0.9752	0.0046	0.9738	0.9797	0.9940
hybrid.32k	0.0067	0.0248	-	0.0049	0.9706	0.9777	0.9937
alpha.ev6	0.0192	0.9954	0.9951	-	0.9862	0.9884	0.9961
hashed perceptron	0.0067	0.0262	0.0294	0.0138	-	0.9197	0.1782
ogehl	0.0062	0.0203	0.0223	0.0116	0.0803	-	0.1203
piecewise	0.0033	0.0060	0.0063	0.0039	0.8218	0.8797	-

Table 5: p-value for Floating IPC, H_0: The mean of the top predictor is less than the mean of the left predictor, values less than 0.05 are significant

	bimod	gshare.32k	hybrid.32k	alpha.ev6	hashed perceptron	ogehl	piecewise
bimod	-	0.9955	0.9953	0.9920	0.9997	0.9997	0.9994
gshare.32k	0.0045	-	0.3316	0.0024	0.9984	0.9991	0.9986
hybrid.32k	0.0047	0.6684	-	0.0032	0.9988	0.9994	0.9989
alpha.ev6	0.0080	0.9976	0.9968	-	0.9998	0.9998	0.9998
hashed perceptron	0.0003	0.0016	0.0012	0.0002	-	0.9699	0.1797
ogehl	0.0003	0.0009	0.0006	0.0002	0.0301	-	0.0651
piecewise	0.0006	0.0014	0.0011	0.0002	0.8203	0.9349	-

Table 6: p-value for Floating Correct Direction Rate, H_0: The mean of the top predictor is less than the mean of the left predictor, values less than 0.05 are significant

	bimod	gshare.32k	hybrid.32k	alpha.ev6	hashed perceptron	ogehl	piecewise
gzip	2.196	2.34	2.351	2.349	2.582	2.581	2.587
vpr	0.968	0.968	0.971	0.965	0.991	0.985	0.986
gcc	2.261	2.362	2.463	2.317	2.612	2.661	2.476
mcf	0.173	0.173	0.173	0.171	0.176	0.174	0.173
crafty	2.498	2.568	2.711	2.559	2.879	2.922	2.767
parser	1.84	1.918	1.956	1.873	2.116	2.136	2.05
eon	2.193	2.637	2.62	2.412	2.995	3.01	2.894
perlbmk	2.514	2.549	2.574	2.537	2.584	2.612	2.573
gap	1.782	1.841	1.845	1.798	1.869	1.872	1.855
vortex	2.914	2.834	2.957	2.912	2.993	3.002	2.991
bzip2	2.382	2.361	2.418	2.384	2.453	2.463	2.46
twolf	1.642	1.612	1.653	1.628	1.775	1.779	1.699
Average	1.947	2.014	2.058	1.992	2.169	2.183	2.126

Table 7: SPEC Integer IPC on different types of branch predictors

	bimod	gshare.8kb	hybrid.8kb	alpha.ev6	hashed perceptron	ogehl	piecewise
wupwise	1.813	1.92	1.925	1.879	1.962	1.993	1.989
swim	0.675	0.675	0.675	0.675	0.675	0.675	0.675
mgrid	1.308	1.309	1.309	1.308	1.309	1.309	1.309
applu	1.123	1.123	1.123	1.123	1.123	1.123	1.123
mesa	2.852	2.953	2.972	2.904	3.053	3.064	3.028
art	2.405	2.474	2.483	2.414	2.753	2.753	2.561
equake	0.407	0.408	0.408	0.408	0.408	0.408	0.408
facerec	2.551	2.644	2.649	2.598	2.701	2.703	2.704
ammp	2.305	2.323	2.328	2.301	2.436	2.438	2.433
lucas	0.744	0.744	0.744	0.744	0.744	0.744	0.744
fma3d	1.12	1.121	1.121	1.12	1.125	1.125	1.125
apsi	2.428	2.559	2.56	2.508	2.583	2.585	2.584
Average	1.644	1.688	1.691	1.665	1.739	1.743	1.724

Table 8: SPEC Floating IPC on different types of branch prediction

	bimod	gshare.32k	hybrid.32k	alpha.ev6	hashed perceptron	ogehl	piecewise	Weight
gzip	0.8945	0.9218	0.924	0.9239	0.9629	0.9616	0.9635	0.8065
vpr	0.9382	0.9384	0.9422	0.9352	0.9564	0.9528	0.954	0.8084
gcc	0.9276	0.9434	0.9595	0.9379	0.977	0.9825	0.9601	1.1237
mcf	0.9045	0.9106	0.9134	0.9049	0.9613	0.9629	0.9491	1.3673
crafty	0.8917	0.9124	0.9361	0.9061	0.9635	0.9696	0.945	0.852
parser	0.9101	0.9319	0.938	0.9181	0.9731	0.9762	0.9613	1.1639
eon	0.8302	0.926	0.9244	0.8877	0.9934	0.9945	0.9827	0.8721
perlbmk	0.9788	0.9847	0.9884	0.9831	0.9927	0.9942	0.9889	0.9676
gap	0.962	0.9855	0.9868	0.9681	0.9947	0.997	0.9898	0.9879
vortex	0.9927	0.989	0.9963	0.9928	0.9988	0.9991	0.9986	1.34
bzip2	0.9123	0.9121	0.9237	0.9146	0.9343	0.9371	0.936	0.7949
twolf	0.8755	0.8708	0.8865	0.8737	0.9407	0.9419	0.911	0.9159
AM	0.9182	0.9356	0.9433	0.9288	0.9707	0.9725	0.9617	
GM	0.9172	0.935	0.9428	0.9282	0.9705	0.9722	0.9614	
WAM	0.9214	0.9376	0.9451	0.931	0.9723	0.9741	0.9631	
WGM	0.9203	0.9369	0.9446	0.9303	0.9721	0.9739	0.9628	

Table 9: Integer Correct Direction Rate

	bimod	gshare.8kb	hybrid.8kb	alpha.ev6	hashed perceptron	ogehl	piecewise	Weight
wupwise	0.9348	0.9817	0.9838	0.959	0.9927	0.9998	0.9994	2.0212
swim	0.9938	0.9942	0.9942	0.9939	0.9969	0.997	0.9969	0.2842
mgrid	0.9343	0.9923	0.9795	0.9714	0.9989	0.9989	0.9988	0.0622
applu	0.9222	0.9721	0.9667	0.949	0.9902	0.9994	0.9901	0.0549
mesa	0.9384	0.9674	0.9701	0.9519	0.9874	0.9897	0.9821	1.7839
art	0.927	0.9446	0.9459	0.9287	0.9998	0.9998	0.9643	2.534
equake	0.9493	0.9736	0.9749	0.9731	0.9778	0.9773	0.9754	0.7756
facerec	0.9141	0.9728	0.9748	0.952	0.9912	0.9926	0.9926	1.2557
ammp	0.9405	0.9463	0.9479	0.9406	0.9914	0.9921	0.9899	1.5585
lucas	0.9956	0.9994	0.9995	0.9995	0.9994	0.9994	0.9995	0.3464
fma3d	0.9288	0.9431	0.9426	0.9344	0.9994	0.9996	0.9997	0.6301
apsi	0.8217	0.9775	0.9783	0.9208	0.9979	0.9998	0.9991	0.6933
AM	0.9334	0.9721	0.9715	0.9562	0.9936	0.9955	0.9907	
GM	0.9325	0.9719	0.9714	0.9559	0.9936	0.9954	0.9906	
WAM	0.9294	0.9642	0.9657	0.9479	0.9931	0.995	0.9858	
WGM	0.9289	0.9641	0.9656	0.9477	0.9931	0.995	0.9857	

Table 10: Floating Correct Direction Rate Raw Data