Analyzing the Processor Bottlenecks in SPEC CPU 2000

Joshua J. Yi[1], Ajay Joshi [2], Resit Sendag [3], Lieven Eeckhout[4], and David J. Lilja[5]

[1] - Networking and Computing Systems Group
Freescale Semiconductor, Inc.
Austin, TX
joshua.yi@freescale.com

[2] - Department of Electrical and Computer Engineering
University of Texas
Austin, TX
ajoshi@ece.utexas.edu

[3] - Department of Electrical and Computer Engineering
University of Rhode Island
Kingston, RI
sendag@ele.uri.edu

[4] – Department of Electronics and Information Systems
Ghent University
Ghent, Belgium
leeckhou@elis.ugent.be

[5] - Department of Electrical and Computer Engineering
University of Minnesota
Minneapolis, MN
lilja@ece.umn.edu

Abstract

The performance of a processor is limited by the specific bottlenecks that a benchmark exposes while running on that processor. Since the quantification of these bottlenecks can be extremely time-consuming, our prior work proposed using the Plackett and Burman design *as a statistically-rigorous, but time-efficient method of determining the processor's most significant performance bottlenecks. In this paper, we use the Plackett and Burman design to quantify the magnitude of the bottlenecks in the SPEC CPU 2000 benchmark suite from the viewpoints of both performance and energy consumption. We then use* Principal Components Analysis*, and* hierarchical *and* K-means clustering *algorithms to determine the similarity of the benchmarks based on their energy-delay production bottlenecks.*

1 Introduction

When running a benchmark on a processor, the bottlenecks that the benchmark exposes in the processor ultimately determines the execution time of the benchmark. For example, if the various queues in the load-store unit (LSU) are too small, then trying to decrease the execution time of the benchmark solely by increasing the processor's issue width or its number of simple integer ALUs is futile since the performance bottleneck is the LSU. However, while the LSU may be the performance-limiting bottleneck for one benchmark, the size of the L1 D-cache may be the limiting bottleneck for another benchmark. In an analogous way, processor components may be energy consumption bottlenecks in that those components ultimately determine the overall energy consumption of the processor.

Although it is very important to precisely identify the bottlenecks for a specific processor and benchmark, determining which processor components are the bottlenecks is a non-trivial task, and determining the relative significance and ordering of these bottlenecks is even more difficult. To minimize the difficulty of determining the significance of a processor's bottlenecks, Yi *et al*. [20] proposed using the *Plackett and Burman* (P&B) *design* [13] to determine which processor parameters, or bottlenecks, have the most effect on performance.

In this paper, we use the P&B design to determine the most significant performance and energy consumption bottlenecks for the benchmarks in the SPEC CPU 2000 benchmark suite. Then, we use *Principal Components Analysis* (PCA), and *hierarchical* and *K-means clustering* algorithms to determine which benchmarks have similar performance and energy consumption bottlenecks.

This paper makes the following contributions:

1. It quantifies and analyzes the performance and energy consumption bottlenecks in the SPEC CPU 2000 benchmarks.
2. It determines the similarity of benchmarks based on their bottlenecks.

The remainder of this paper is organized as

follows: Sections 2, 3, and 4 describe some related work, the P&B design, and the benchmarks and simulation methodology that we used in this paper, respectively. Sections 5 and 6 present the results, while Section 7 summarizes.

2 Related Work

Yi *et al*. [20] proposed using the P&B design to improve the statistical rigor of simulation methodology. More specifically, they proposed using P&B design as the foundation to choose processor parameters, select a subset of benchmarks, and analyze the effect of a processor enhancement. This paper builds on that work by using the P&B design to determine the most significant performance and energy consumption bottlenecks for the entire SPEC CPU 2000 benchmark suite, and similarity of benchmarks based on those bottlenecks.

Eeckhout *et al*. [3, 4] characterized benchmarks by gathering a set of metrics such as the instruction mix, branch prediction accuracy, cache miss rates, and basic block lengths for each benchmark and input set pair. After gathering these metrics, they used PCA to determine the principal components for each pair, and then clustered the pairs based on their principal components. Phansalkar *et al*. [12] also used PCA to characterize the benchmark and input set pairs, but they used K-means clustering and the Bayesian Information Criterion instead to cluster the pairs. Vandierendonck and De Bosschere [17] analyzed the SPEC CPU 2000 benchmark suite peak results on 340 different machines representing eight architectures, and used PCA to identify the redundancy in the benchmark suite. Finally, Eeckhout *et al*. [5] compared the efficacy of using Independent Components Analysis (ICA) instead of PCA for benchmark subsetting. One key difference between these papers and ours is that the focus of these papers is benchmark subsetting while this paper primarily focuses on analyzing the performance and energy consumption bottlenecks in the SPEC CPU 2000 benchmarks.

Finally, Tune *et al*. [15, 16] and Fields *et al*. [6, 7, 8] proposed techniques to predict the criticality of instructions to improve execution efficiency of the processor. The key difference between their papers and ours is that we use a statistically-rigorous technique to quantify the significance of each bottleneck across the entire run of the benchmark while they use heuristics to dynamically estimate the criticality of individual instructions.

3 The Plackett and Burman Design: Finding Processor Bottlenecks

To determine the bottlenecks in the processor, we used the P&B design, with foldover [11], as described in [20]. For computer architects, the P&B design is a statistical technique that can be used to determine the significance of the processor's bottlenecks, at an $O(N)$ simulation cost, where N is the number of bottlenecks. By comparison, using a design such as ANOVA [10] requires $O(2^N)$ simulations for only a little additional accuracy.

3.1 Mechanics of the Plackett and Burman Design

The first step to use a P&B design is to construct the design matrix. Since P&B designs exist only in sizes that are multiples of 4, the base P&B design requires X simulations, where X is the next multiple-of-four that is greater than N. The rows of the design matrix correspond to different processor configurations while the columns correspond to the parameters' values in each configuration. When there are more columns than parameters, then the extra columns serve as "placeholders" and have no effect on the simulation results.

Table 1. Plackett and Burman design, with foldover (X = 8)

A	B	C	D	E	F	G	Exec. Time
+1	+1	+1	-1	+1	-1	-1	9
-1	+1	+1	+1	-1	+1	-1	11
-1	-1	+1	+1	+1	-1	+1	20
+1	-1	-1	+1	+1	+1	-1	10
-1	+1	-1	-1	+1	+1	+1	9
+1	-1	+1	-1	-1	+1	+1	74
+1	+1	-1	+1	-1	-1	+1	7
-1	-1	-1	-1	-1	-1	-1	112
-1	-1	-1	+1	-1	+1	+1	17
+1	-1	-1	-1	+1	-1	+1	76
+1	+1	-1	-1	-1	+1	-1	6
-1	+1	+1	-1	-1	-1	+1	31
+1	-1	+1	+1	-1	-1	-1	19
-1	+1	-1	+1	+1	-1	-1	33
-1	-1	+1	-1	+1	+1	-1	6
+1	+1	+1	+1	+1	+1	+1	4
-34	-224	-96	-202	-110	-170	32	

For most values of X, the design matrix is simple to construct. For these values of X, the first row of the design matrix is given in [13]. The next X – 2 rows are formed by performing a circular right shift on the preceding row. The last line of the design matrix is a row of "-1"s. The gray-shaded portion of Table 1 illustrates the construction of the P&B design matrix for X = 8, a design appropriate for investigating 7 (or fewer) parameters. When using foldover, X additional rows are added to the matrix. Although this doubles the simulation cost, the advantage of using foldover is that it filters out the effects of interactions from the single parameter bottlenecks, while also allowing the user to determine the significance of two-parameter

interactions. The signs in each entry of the additional rows are the opposite of the corresponding entries in the original matrix. Table 1 shows the complete P&B design matrix with foldover; rows 10 to 17 show the rows that were added for foldover.

A "+1", or high value, for a parameter represents a value that is higher than the range of normal values for that parameter while a "-1", or low value, represents a value that is lower than the range of normal values. Ideally, the high and low values for each parameter should be just outside of the normal range of values. For example, if 16KB and 32KB are typical L1 D-cache sizes, then an appropriate low value might be 8KB while an appropriate high value might be 64KB.

The set of low and high values that we used in this study is similar to those found in [20]. We fixed the issue width to be 4-way to eliminate any potential issue-width dependency for the other parameters.

3.2 Calculating the Significance of Bottlenecks Using the Plackett and Burman Design

To compute the effect of each parameter, we multiply the output value (*e.g.*, execution time, energy consumption, *etc*.) by the parameter's P&B value (+1/-1) for that configuration and sum the resulting products across all configurations. For example, we compute the effect of parameter A is computed by weighting the Execution Time Column with Column A in Table 1:

$$\text{Effect}_A = (1 * 9) + (-1*11) + \ldots + (-1*6) + (1*4) = -34$$

Only the magnitude of an effect is important; its sign is meaningless. The effect that a parameter has represents how much of the total variation in the output value is attributable to that parameter. Therefore, a parameter that has a large effect on the execution time accounts for a significant amount of variability in the execution time, which makes it a significant performance bottleneck (since changing the parameter's value results in large changes in the execution time).

After simulation, we computed the percentage of the variability in the output value across all configurations that can be assigned to each bottleneck on a per-benchmark basis, in a manner similar to how one computes the percentages for ANOVA [10]. By examining the percentage effect that each bottleneck has on the average cycles-per-instruction (CPI) or average amount of energy-per-instruction (EPI) for that suite or benchmark, we can determine absolute and relative significance of the performance and energy consumption, respectively, bottlenecks in the processor.

4 Simulation Methodology

In this paper, to gather the profiling data for the P&B design simulations, we used SMARTS [19], which estimates both the performance and energy consumption (`cc3` power consumption measurement), after adding user-configurable instruction latencies and throughputs. All simulations were run until the sampling frequency was greater than the recommended frequency.

Since we wanted to use a set of benchmarks that had a wide range of behavior and represented a wide range of applications, *i.e.*, general-purpose computing, we decided to use the SPEC CPU 2000 benchmark suite. We downloaded pre-compiled Alpha binaries from [18], and evaluated all benchmark and `reference` input set pairs, with the exceptions of *vpr-place* and *perlbmk-perfect* – for a total of 46 benchmark and input set pairs – as they both crash SMARTS.

Note that, in the remainder of this paper, for brevity, we often use the term "benchmarks" to represent both the benchmark and its input set.

5 Quantifying the Bottlenecks in SPEC CPU 2000

5.1 Processor Performance Bottlenecks

Table 2 shows the results of a P&B design with foldover (X = 44), where the bottlenecks are sorted in descending order of their average percentages. This table shows that, based on the average percentage, there are seven significant bottlenecks. We draw this conclusion based on the relatively large difference between the average percentage of the seventh most significant bottleneck, the number of Integer ALUs, and the average percentage of the eighth most significant bottleneck, the number of LSU entries. Although the seven most significant bottlenecks for each benchmark are completely different, one bottleneck, the number of ROB entries, is significant for all benchmarks since it has a high percentage for each benchmark; it has the highest percentage in 23 of 46 benchmark and input set pairs. Therefore, for these benchmarks, the number of ROB entries is the biggest performance bottleneck in the processor.

Therefore, of all these bottlenecks, the architect needs to be especially careful when choosing a value for the number of ROB entries since a poor choice can significantly affect the processor's performance. The L2 cache size, L1 I-cache size and memory latency is the most significant performance bottleneck for 10, 6 and 4 benchmarks, respectively. Although it is significant for some benchmarks, the L1 I-cache size is not one of the 5 most significant bottlenecks because it does not have a high average percentage. These results clearly show that the average percentage may not reflect the significance of a bottleneck for a subset of benchmarks and is best used only to gain a big-picture view of the results.

Table 2. Plackett and Burman design results for all performance bottlenecks; sorted in descending order of the percentage of the total variation accounted for by each bottleneck.

Parameter	*gzip-graphic*	*gzip-log*	*gzip-program*	*gzip-random*	*gzip-source*	*wupwise*	*swim*	*mgrid*	*applu*	*vpr-route*	*gcc-166*	*gcc-200*	*gcc-expr*	*gcc-integrate*	*gcc-scilab*	*mesa*	*galgel*	*art-110*	*art-470*	*mcf*	*equake*	*crafty*	*facerec*	*ammp*	*lucas*	*fma3d*	*parser*	*sixtrack*	*eon-cook*	*eon-kajiya*	*eon-rushmeier*	*perlbmk-diffmail*	*perlbmk-makerand*	*perlbmk-splitmail 535*	*perlbmk- splitmail 704*	*perlbmk- splitmail 850*	*perlbmk-s splitmail 957*	*gap*	*vortex-1*	*vortex-2*	*vortex-3*	*bzip2-graphic*	*bzip2-program*	*bzip2-source*	*apsi*	*twolf*	*Average*
ROB Entries	26.7	28.3	20.0	28.6	23.6	25.9	12.7	18.9	15.4	11.5	4.1	11.9	8.6	5.4	9.8	13.0	36.7	15.1	15.2	5.1	19.1	5.7	32.4	24.3	3.8	19.4	17.4	56.8	16.6	14.4	16.4	13.1	7.5	19.5	18.2	21.8	18.6	10.4	12.4	10.4	12.5	22.7	24.7	25.1	9.3	33.0	17.9
L2 Cache Size	1.1	1.5	0.8	1.3	1.1	4.1	2.2	5.8	2.8	17.8	18.4	14.9	12.8	15.7	13.4	0.8	12.7	28.0	28.0	35.4	1.5	1.1	17.1	24.0	3.7	2.4	16.1	0.1	0.4	0.4	0.5	1.8	0.7	0.5	0.9	1.6	1.7	2.3	5.2	4.5	5.3	11.1	10.0	14.7	30.0	8.5	8.4
Memory Latency First	0.5	0.7	0.0	0.8	0.3	12.9	29.9	14.7	22.3	15.6	9.5	7.0	5.9	7.7	6.1	0.5	6.2	12.0	12.0	27.8	39.6	0.7	10.0	15.1	9.2	16.5	7.0	0.3	0.2	0.2	0.2	0.6	0.1	0.7	0.8	0.7	1.1	5.7	3.1	2.4	3.2	4.3	3.7	5.8	15.5	12.0	7.6
L2 Cache Latency	2.3	6.0	5.6	2.9	6.3	4.2	0.2	6.3	1.3	1.4	0.8	5.7	4.7	1.6	6.4	16.0	1.6	1.5	1.5	0.6	2.4	18.5	1.3	1.2	0.7	9.8	5.1	6.8	15.6	16.2	15.4	11.8	21.0	2.9	4.2	4.3	7.3	14.2	11.9	13.1	12.1	1.3	1.7	2.0	5.3	5.0	6.3
BPred Type	16.6	8.7	24.4	10.7	17.5	1.8	0.3	0.7	0.6	3.9	1.1	4.8	4.0	1.9	4.7	0.9	0.1	0.1	0.1	0.3	0.5	1.8	1.4	0.7	0.7	0.7	7.8	0.4	2.7	3.0	3.6	8.8	4.3	12.7	12.5	16.2	14.5	4.2	4.4	3.5	4.1	9.6	10.4	5.7	1.3	0.1	5.2
L1 I-Cache Size	0.3	0.2	0.1	0.4	0.2	1.8	0.1	0.1	0.1	0.0	0.0	2.9	2.9	0.2	4.2	21.6	0.0	0.2	0.2	0.1	0.0	22.6	0.0	0.1	0.2	7.6	0.8	5.0	14.9	15.5	13.8	12.0	21.0	2.3	4.1	4.0	7.0	17.2	13.9	16.9	13.6	0.1	0.1	0.2	2.0	2.2	5.1
Int ALUs	16.4	12.4	8.3	18.2	10.0	2.4	0.0	0.0	0.0	1.2	0.7	3.4	2.4	1.2	2.6	1.4	0.2	0.3	0.3	0.2	0.0	1.6	2.6	0.6	0.0	0.5	5.6	0.1	2.2	1.9	2.2	4.8	1.4	3.0	3.4	10.0	8.5	2.7	6.1	4.5	5.9	10.7	11.2	9.3	1.6	1.1	4.0
LSU Entries	3.6	4.1	2.0	3.3	2.9	1.7	0.0	0.3	0.2	1.7	0.1	1.5	0.9	0.2	1.3	2.5	3.2	2.7	2.7	0.3	2.0	1.2	3.8	1.4	0.0	1.6	2.2	3.0	4.5	4.5	4.7	4.1	2.4	4.0	4.1	6.2	5.5	1.6	3.1	3.0	3.0	5.3	4.8	4.1	0.4	3.1	2.6
L1 D-Cache Latency	4.3	5.0	3.9	4.1	3.9	1.9	0.3	1.3	0.5	0.9	1.7	3.0	3.1	2.3	3.1	1.4	1.0	0.1	0.1	0.1	0.4	1.2	1.2	0.7	0.3	1.2	2.8	1.2	2.1	2.2	2.2	2.9	1.9	3.1	3.1	4.5	3.9	3.6	3.3	2.9	3.3	3.3	3.6	2.9	0.9	0.7	2.2
L1 I-Cache Block Size	0.1	0.0	0.0	0.1	0.0	1.8	0.3	2.3	1.2	0.1	1.1	1.7	2.1	1.3	2.4	5.4	0.1	0.1	0.1	0.0	0.0	5.4	0.1	0.1	0.6	5.7	0.3	2.4	4.8	5.3	4.8	2.1	3.8	1.3	1.6	0.8	1.3	3.8	1.7	2.3	1.6	0.1	0.1	0.1	0.8	0.8	1.6
Memory Bandwidth	0.1	0.2	0.0	0.3	0.1	2.2	4.2	2.1	3.2	2.4	1.7	1.7	1.5	1.6	1.6	0.5	1.1	2.2	2.2	4.2	5.1	0.4	2.0	2.2	1.4	3.0	1.5	0.0	0.1	0.1	0.1	0.4	0.2	0.0	0.0	0.2	0.4	1.8	1.3	1.2	1.4	0.9	0.8	1.2	2.6	2.3	1.4
L1 D-Cache Size	0.7	6.9	8.2	0.8	8.6	0.0	0.9	1.1	0.4	0.7	1.0	0.0	0.1	0.7	0.0	1.0	0.6	0.0	0.0	0.0	0.3	2.5	0.0	0.1	0.0	0.5	1.0	2.2	2.6	2.5	2.5	2.1	1.3	0.2	0.2	0.3	0.7	0.1	0.9	0.9	1.0	0.1	0.2	0.3	0.1	1.6	1.2
L1 D-Cache Block Size	0.2	0.1	0.2	0.4	0.0	2.3	5.6	3.5	5.4	0.1	5.2	2.4	4.2	6.0	3.0	0.0	0.7	0.1	0.1	0.1	1.8	0.0	0.9	0.7	3.3	1.4	0.6	0.2	0.0	0.0	0.1	0.1	0.1	0.0	0.0	0.0	0.0	0.8	0.1	0.0	0.1	0.0	0.1	0.0	0.3	0.5	1.1
D-TLB Size	1.2	0.5	0.0	1.9	0.3	1.5	0.8	2.5	1.3	5.1	1.2	1.0	1.0	1.4	0.9	0.5	0.5	0.5	0.5	0.2	1.0	0.2	0.6	0.8	1.0	0.5	0.8	0.1	0.1	0.0	0.1	0.2	0.7	5.0	4.2	0.5	0.3	0.5	0.3	0.4	0.3	1.6	1.1	1.2	1.0	1.0	1.0
I-TLB Page Size	0.3	0.5	0.3	0.4	0.7	0.4	0.5	0.0	0.5	7.7	0.6	0.0	0.3	0.6	0.1	0.3	5.1	0.7	0.7	0.6	0.1	0.4	0.0	0.2	4.8	0.1	0.2	0.0	0.0	0.0	0.0	0.5	0.1	1.9	1.3	0.0	0.3	0.1	0.7	0.9	0.8	0.9	0.5	1.0	0.6	0.8	0.8
L1 D-Cache Associativity	1.2	0.8	0.6	1.7	0.5	3.1	2.0	4.0	2.3	0.3	1.1	0.9	1.4	1.7	1.1	0.0	0.5	0.2	0.2	0.1	1.0	0.0	0.2	0.1	1.7	1.0	1.1	0.0	0.0	0.0	0.0	0.4	0.9	0.0	0.0	0.2	0.2	1.1	0.2	0.2	0.2	0.9	0.8	0.5	0.0	0.9	0.8
L2 Cache Associativity	0.4	0.7	0.4	0.7	0.4	2.0	2.3	1.1	1.8	0.5	1.9	1.5	1.9	2.0	1.7	0.1	0.6	0.1	0.1	0.1	0.3	0.0	0.4	0.7	3.4	1.3	0.7	0.0	0.0	0.0	0.0	0.4	0.1	0.6	0.5	0.3	0.1	0.1	0.1	0.1	0.1	0.2	0.2	0.2	0.1	0.0	0.7
Int ALU Latencies	2.7	3.6	3.4	2.2	3.1	0.0	0.2	0.1	0.5	0.0	0.1	0.1	0.0	0.0	0.1	0.2	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	1.5	0.2	0.1	0.1	0.1	0.5	0.6	0.0	0.0	1.3	1.2	0.3	0.4	0.5	0.3	0.7	0.9	0.6	0.0	0.0	0.6
BPred Misprediction Penalty	1.4	0.4	1.9	0.8	1.1	0.0	0.2	0.7	0.6	0.0	0.2	0.0	0.0	0.1	0.0	0.0	0.1	0.1	0.1	0.0	0.0	0.0	0.2	0.1	1.1	0.1	0.8	0.0	0.2	0.2	0.2	0.9	0.5	1.0	1.1	1.4	1.1	0.4	0.5	0.3	0.4	0.6	0.7	0.3	0.0	0.1	0.4
D-TLB Associativity	0.0	0.1	0.1	0.0	0.1	0.0	0.1	0.0	0.2	0.1	0.1	0.0	0.0	0.1	0.0	0.3	0.0	0.4	0.4	0.0	0.2	0.2	0.1	0.0	6.5	0.0	0.0	0.1	0.1	0.1	0.1	0.2	0.4	4.5	3.6	0.4	0.2	0.0	0.1	0.1	0.1	0.1	0.1	0.1	0.1	0.0	0.4
FP Square Root Latency	0.1	0.1	0.1	0.2	0.1	1.5	1.9	2.2	2.2	0.1	1.6	0.5	0.9	1.7	0.6	0.1	0.4	0.2	0.2	0.0	0.4	0.0	0.2	0.4	0.8	0.4	0.3	0.0	0.1	0.1	0.0	0.0	0.0	0.1	0.1	0.0	0.0	0.1	0.0	0.1	0.0	0.1	0.1	0.0	0.0	0.0	0.4
L2 Cache Block Size	0.0	0.1	0.0	0.1	0.1	0.4	0.9	0.0	0.0	1.2	0.3	0.3	0.5	0.4	0.2	0.0	0.0	0.1	0.1	0.2	8.1	0.1	0.1	0.3	0.3	0.2	0.0	0.0	0.1	0.1	0.1	0.1	0.0	0.4	0.3	0.1	0.1	0.0	0.1	0.1	0.1	0.4	0.3	0.4	0.4	0.6	0.4
FP ALU Latencies	0.1	0.1	0.1	0.1	0.0	0.0	1.1	0.1	0.8	0.7	1.5	0.6	1.1	1.5	0.7	0.2	0.0	0.7	0.7	0.0	0.1	0.0	0.0	0.1	0.3	0.0	0.0	3.4	0.3	0.3	0.3	0.0	0.1	0.3	0.2	0.1	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.2	0.3
Placeholder #2	0.3	0.3	0.2	0.4	0.3	0.4	0.3	0.6	0.7	0.2	0.6	0.7	0.7	0.8	0.6	0.0	0.1	0.4	0.3	0.1	0.0	0.1	0.2	0.2	0.5	0.4	0.3	0.0	0.2	0.1	0.3	0.3	0.0	0.4	0.4	0.3	0.2	0.1	0.2	0.1	0.2	0.4	0.4	0.4	0.1	0.1	0.3
FP Multiply Latency	0.2	0.2	0.2	0.2	0.2	0.5	0.0	0.2	0.3	0.2	0.1	0.3	0.3	0.2	0.4	0.4	0.0	0.2	0.2	0.0	0.1	0.5	0.1	0.3	1.0	0.6	0.1	2.5	0.5	0.4	0.4	0.4	0.1	0.0	0.0	0.1	0.2	0.1	0.3	0.3	0.3	0.2	0.2	0.2	0.2	0.2	0.3
Placeholder #1	0.1	0.0	0.0	0.2	0.0	0.2	2.0	1.2	1.5	0.3	1.5	0.4	1.0	1.3	0.5	0.0	0.0	0.1	0.1	0.1	0.2	0.0	0.2	0.3	0.7	0.3	0.1	0.0	0.0	0.0	0.0	0.0	0.1	0.0	0.0	0.0	0.0	0.0	0.0	0.1	0.0	0.1	0.0	0.0	0.1	0.1	0.3
I-TLB Latency	0.1	0.0	0.0	0.2	0.1	0.0	0.0	0.2	0.0	1.5	0.0	0.0	0.0	0.0	0.0	0.4	2.1	0.1	0.1	0.0	0.1	0.2	0.0	0.1	2.4	0.2	0.0	0.1	0.2	0.2	0.3	0.2	0.3	0.9	0.8	0.1	0.1	0.1	0.0	0.2	0.0	0.1	0.1	0.2	0.1	0.2	0.3
Instruction Fetch Queue Entries	0.0	0.0	0.0	0.0	0.0	0.1	1.0	0.5	1.1	0.0	1.5	0.7	1.1	1.1	0.8	0.1	0.1	0.0	0.0	0.0	0.0	0.1	0.1	0.1	0.4	0.0	0.3	0.4	0.2	0.2	0.2	0.0	0.0	0.0	0.0	0.0	0.0	0.2	0.1	0.0	0.1	0.0	0.0	0.0	0.1	0.1	0.2
FP Mult/Div	0.0	0.1	0.1	0.0	0.1	0.3	0.5	0.2	0.6	0.1	0.9	0.5	0.6	1.0	0.5	0.1	0.4	0.1	0.1	0.2	0.1	0.1	0.5	0.4	0.0	0.1	0.2	0.2	0.0	0.0	0.0	0.0	0.2	0.4	0.4	0.1	0.0	0.0	0.0	0.0	0.0	0.1	0.1	0.1	0.1	0.8	0.2
Int Mult/Div	0.0	0.0	0.1	0.0	0.1	0.0	1.3	0.8	1.0	0.0	1.1	0.3	0.7	0.9	0.3	0.0	0.0	0.1	0.1	0.0	0.0	0.1	0.0	0.0	0.8	0.0	0.0	0.0	0.1	0.1	0.1	0.3	0.0	0.2	0.3	0.0	0.0	0.0	0.2	0.1	0.2	0.0	0.0	0.1	0.0	0.0	0.2
Return Address Stack Entries	0.0	0.1	0.0	0.0	0.0	0.2	0.4	0.7	0.8	0.0	0.5	0.4	0.5	0.5	0.5	0.1	0.0	0.1	0.1	0.0	0.0	0.3	0.0	0.1	0.8	0.4	0.0	0.0	0.2	0.2	0.2	0.2	0.0	0.3	0.4	0.1	0.1	0.3	0.1	0.1	0.2	0.0	0.0	0.0	0.1	0.1	0.2
L1 I-Cache Associativity	0.5	0.3	0.1	0.5	0.2	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.2	0.1	0.1	0.1	0.1	0.1	0.1	0.3	0.1	0.1	0.0	0.7	0.1	0.0	0.0	0.0	0.8	0.0	0.2	0.1	0.3	0.2	0.3	0.3	0.3	0.4	0.8	0.8	0.8	0.0	0.0	0.2
FP ALUs	0.0	0.0	0.0	0.0	0.0	0.0	0.7	0.2	0.5	0.0	0.9	0.1	0.4	0.6	0.2	0.2	0.0	0.0	0.0	0.0	0.1	0.5	0.0	0.1	0.8	0.1	0.0	0.4	0.6	0.6	0.6	0.3	0.0	0.1	0.2	0.0	0.1	0.0	0.2	0.2	0.1	0.0	0.0	0.0	0.0	0.1	0.2
BTB Entries	0.2	0.0	0.1	0.3	0.0	0.8	0.1	0.2	0.2	0.2	0.2	0.1	0.2	0.3	0.2	0.0	0.1	0.0	0.0	0.0	0.0	0.0	0.2	0.0	0.4	0.0	0.5	0.0	0.1	0.1	0.1	0.2	0.2	0.0	0.0	0.1	0.0	0.5	0.1	0.1	0.1	0.4	0.3	0.3	0.0	0.0	0.2
L1 I-Cache Latency	0.0	0.1	0.2	0.0	0.2	0.3	1.1	0.3	0.9	0.0	0.7	0.1	0.3	0.9	0.2	0.0	0.0	0.1	0.1	0.0	0.2	0.1	0.0	0.0	0.6	0.0	0.0	0.1	0.1	0.1	0.1	0.0	0.0	0.0	0.0	0.0	0.0	0.1	0.0	0.0	0.1	0.0	0.0	0.1	0.0	0.1	0.2
Int Divide Latency	0.1	0.0	0.0	0.1	0.0	0.0	0.0	0.0	0.0	0.1	0.1	0.0	0.0	0.0	0.0	0.2	0.0	0.1	0.1	0.0	0.1	0.4	0.0	0.0	0.0	0.2	0.1	0.1	0.6	0.7	0.5	0.1	0.0	0.9	0.9	0.0	0.0	0.1	0.1	0.1	0.1	0.1	0.1	0.0	0.0	0.2	0.1
FP Divide Latency	0.0	0.1	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.1	0.0	0.0	0.1	0.0	0.4	0.1	0.2	0.2	0.0	0.0	0.2	0.0	0.0	0.4	0.1	0.0	0.1	0.4	0.4	0.3	0.0	0.3	0.2	0.1	0.0	0.0	0.1	0.0	0.0	0.0	0.0	0.0	0.0	0.1	1.8	0.1
Memory Ports	0.1	0.0	0.0	0.0	0.0	0.2	0.0	0.0	0.0	0.3	0.0	0.0	0.0	0.0	0.0	0.3	0.0	0.1	0.1	0.2	0.0	0.0	0.0	0.1	0.1	0.1	0.0	0.1	0.1	0.1	0.1	0.1	0.4	0.3	0.2	0.2	0.1	0.1	0.1	0.1	0.1	0.0	0.0	0.0	0.1	0.0	0.1
Int Multiply Latency	0.0	0.1	0.2	0.0	0.1	0.0	0.0	0.0	0.1	0.0	0.0	0.1	0.1	0.0	0.1	0.2	0.1	0.0	0.0	0.0	0.0	0.2	0.3	0.0	0.0	0.0	0.0	0.0	0.3	0.2	0.2	0.1	0.0	0.4	0.4	0.1	0.1	0.1	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.1
I-TLB Associativity	0.0	0.0	0.0	0.1	0.0	0.5	0.0	0.2	0.1	0.0	0.2	0.1	0.2	0.3	0.2	0.1	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.1	0.1	0.1	0.0	0.0	0.0	0.0	0.0	0.2	0.1	0.0	0.0	0.0	0.3	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.1
I-TLB Size	0.0	0.1	0.1	0.0	0.1	0.1	0.1	0.3	0.1	0.0	0.2	0.0	0.0	0.1	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.1	0.1	0.1	0.0	0.0	0.0	0.0	0.0	0.0	0.1	0.1	0.0	0.0	0.0	0.0	0.1	0.2	0.2	0.2	0.0	0.0	0.0	0.0	0.0	0.1
BTB Associativity	0.0	0.1	0.2	0.0	0.1	0.0	0.1	0.2	0.1	0.1	0.1	0.1	0.1	0.0	0.1	0.0	0.0	0.1	0.1	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.1	0.0	0.0	0.0	0.0	0.1	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
Speculative Branch Update	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.2	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.1	0.0	0.0	0.0	0.0	0.2	0.0	0.0	0.0	0.0	0.1	0.0	0.0	0.0	0.2	0.2	0.0	0.0	0.1	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0

Table 3. Plackett and Burman design results for all energy consumption bottlenecks; sorted in descending order of the percentage of the total variation accounted for by each bottleneck.

Parameter	*gzip-graphic*	*gzip-log*	*gzip-program*	*gzip-random*	*gzip-source*	*wupwise*	*swim*	*mgrid*	*applu*	*vpr-route*	*gcc-166*	*gcc-200*	*gcc-expr*	*gcc-integrate*	*gcc-scilab*	*mesa*	*galgel*	*art-110*	*art-470*	*mcf*	*equake*	*crafty*	*facerec*	*ammp*	*lucas*	*fma3d*	*parser*	*sixtrack*	*eon-cook*	*eon-kajiya*	*eon-rushmeier*	*perlbmk-diffmail*	*perlbmk-makerand*	*perlbmk-splitmail 535*	*perlbmk- splitmail 704*	*perlbmk- splitmail 850*	*perlbmk-s splitmail 957*	*gap*	*vortex-1*	*vortex-2*	*vortex-3*	*bzip2-graphic*	*bzip2-program*	*bzip2-source*	*apsi*	*twolf*	*Average*
BTB Associativity	13.6	16.8	14.6	15.4	15.1	16.7	9.1	12.6	10.3	13.9	10.3	17.0	15.7	11.5	16.7	12.5	10.3	6.2	6.3	8.0	11.6	11.6	13.4	11.8	7.2	14.4	16.8	13.2	13.3	12.7	13.9	15.7	13.3	13.0	13.9	15.4	16.3	16.1	17.1	16.3	17.0	14.0	14.5	15.5	14.6	15.2	13.5
BTB Entries	14.3	17.8	16.3	15.6	16.3	11.6	5.4	8.1	4.9	11.9	5.3	14.3	11.8	7.0	13.3	13.5	9.8	4.5	4.6	6.8	11.4	13.2	11.6	10.1	4.1	12.7	15.6	13.4	14.5	14.2	14.5	16.0	14.1	14.0	14.9	15.8	16.9	15.8	17.6	17.1	17.4	12.7	13.7	14.6	13.1	15.5	12.6
BPred Type	16.1	9.6	22.2	11.2	17.6	3.6	0.3	0.6	0.5	7.0	2.2	7.7	6.7	3.8	7.7	1.4	0.0	0.0	0.0	0.7	0.3	2.9	2.3	1.2	0.8	0.5	11.5	0.2	4.5	5.3	5.8	11.5	5.9	14.0	13.6	15.9	14.8	6.6	7.6	6.7	7.3	13.8	14.3	9.3	2.8	0.1	6.5
Memory Latency First	0.0	0.1	0.0	0.1	0.0	7.2	23.3	10.4	17.0	7.1	6.0	3.2	3.1	4.7	2.9	0.3	3.5	8.2	8.2	18.3	24.0	0.5	4.6	8.5	5.9	12.1	1.8	0.1	0.1	0.1	0.2	0.2	0.0	0.3	0.3	0.2	0.3	2.8	1.4	1.1	1.4	1.3	1.1	2.0	8.2	6.3	4.5
L2 Cache Latency	0.5	1.8	1.3	0.7	1.6	3.0	0.5	7.7	2.1	1.9	1.8	4.5	4.4	2.4	5.3	12.5	2.5	3.4	3.4	2.0	1.6	13.7	2.0	2.7	0.3	9.1	2.3	6.3	10.7	11.1	10.2	6.2	13.1	0.5	0.9	1.2	2.6	7.5	6.1	7.4	6.3	0.8	0.9	1.4	6.8	6.1	4.4
L1 I-Cache Size	11.9	11.4	9.1	11.4	10.2	3.0	2.2	1.6	2.4	3.8	3.3	1.8	1.4	3.1	0.9	2.6	3.5	1.0	1.0	0.8	1.9	4.6	6.8	3.8	4.0	0.5	7.0	2.8	0.3	0.5	0.1	0.0	1.6	3.5	2.4	4.3	2.1	0.1	0.0	0.2	0.0	10.4	10.2	8.6	0.2	0.5	3.5
L2 Cache Size	5.3	5.3	4.1	5.5	4.7	2.3	1.0	0.3	0.7	0.2	2.4	0.0	0.0	0.9	0.0	3.5	0.3	12.3	12.2	13.5	2.2	2.6	0.2	3.1	0.0	2.3	0.3	8.0	6.0	5.8	5.9	4.3	5.1	5.0	4.6	4.8	4.7	4.3	1.7	1.8	1.5	0.5	0.7	0.0	3.5	0.2	3.3
ROB Entries	0.4	0.6	0.0	0.8	0.1	5.1	3.9	6.8	5.0	0.8	0.2	0.4	0.2	0.2	0.2	1.7	16.4	8.2	8.2	1.9	5.5	0.1	9.2	9.0	0.9	4.7	0.5	16.0	1.6	0.9	1.1	0.5	0.2	0.4	0.3	0.6	0.5	0.1	0.4	0.2	0.4	1.1	1.1	2.4	1.3	11.0	2.9
L1 D-Cache Size	2.4	0.2	0.0	2.2	0.0	6.3	5.7	0.9	5.3	1.5	7.2	3.8	5.2	6.9	4.4	1.7	1.5	1.2	1.2	1.5	2.8	0.1	4.4	2.5	2.9	2.9	1.6	2.0	1.3	1.4	1.6	1.2	1.2	2.5	2.9	4.3	3.3	3.7	1.8	1.9	1.7	4.3	3.6	3.0	1.9	1.4	2.6
L1 D-Cache Block Size	0.1	0.1	0.1	0.1	0.0	1.4	5.0	3.0	5.3	0.0	5.5	1.8	3.2	5.5	2.2	0.0	1.2	0.3	0.3	0.4	2.0	0.1	1.0	1.2	3.6	1.1	0.5	0.4	0.0	0.0	0.0	0.0	0.1	0.1	0.1	0.0	0.0	0.5	0.0	0.0	0.0	0.1	0.1	0.1	0.5	0.6	1.0
Memory Bandwidth	0.1	0.1	0.0	0.1	0.1	1.6	3.2	1.6	2.8	1.7	1.5	0.9	0.9	1.3	0.8	0.1	1.3	3.0	3.0	4.0	2.5	0.0	2.0	2.3	2.2	1.7	0.6	0.1	0.0	0.0	0.0	0.1	0.0	0.1	0.1	0.1	0.2	0.6	0.6	0.4	0.6	0.6	0.5	0.9	1.8	1.9	1.0
Int ALUs	3.1	2.5	0.9	4.2	1.5	0.8	0.1	0.1	0.0	0.0	0.2	0.8	0.8	0.5	0.6	0.2	0.1	0.0	0.0	0.1	0.3	0.3	0.7	0.0	0.1	0.0	0.9	0.1	0.5	0.4	0.5	1.4	0.1	0.9	0.9	2.8	2.3	0.4	1.4	0.9	1.4	2.3	2.3	2.2	0.1	0.1	0.9
FP ALUs	1.3	1.0	0.7	1.3	0.8	0.9	1.5	1.0	1.3	1.0	2.5	1.3	1.7	2.3	1.4	0.0	1.8	0.5	0.5	0.6	0.7	0.1	1.2	1.5	2.3	0.1	1.4	0.0	0.0	0.0	0.0	0.3	0.4	0.4	0.3	0.7	0.6	0.6	0.3	0.2	0.3	1.3	1.2	1.1	0.7	0.5	0.9
L1 I-Cache Block Size	0.0	0.0	0.0	0.0	0.0	0.7	0.0	1.0	0.3	0.0	0.5	0.6	0.9	0.6	1.1	3.6	0.0	0.2	0.2	0.0	0.2	3.6	0.0	0.0	0.2	2.9	0.1	1.4	3.0	3.5	3.0	1.1	2.6	1.0	1.2	0.4	0.6	1.9	0.7	1.0	0.7	0.0	0.0	0.0	0.4	0.2	0.9
L1 D-Cache Latency	0.6	0.7	0.5	0.6	0.5	1.0	0.6	0.8	0.7	0.4	1.3	1.5	1.8	1.6	1.6	0.8	0.5	0.0	0.0	0.0	0.3	1.0	0.3	0.3	0.2	1.1	0.7	0.4	1.0	1.2	1.1	1.2	1.0	1.0	1.1	1.2	1.2	2.2	1.7	1.6	1.7	0.6	0.7	0.6	0.6	0.3	0.9
D-TLB Size	0.5	0.2	0.1	0.7	0.2	1.1	1.0	2.0	1.2	2.9	0.8	0.4	0.5	0.8	0.4	0.2	0.3	0.4	0.4	0.2	1.1	0.1	0.4	0.6	0.5	0.2	0.5	0.0	0.0	0.0	0.0	0.0	0.3	2.5	2.1	0.3	0.2	0.2	0.1	0.2	0.1	0.9	0.6	0.6	0.7	0.3	0.6
I-TLB Page Size	0.1	0.2	0.1	0.1	0.3	0.2	0.4	0.0	0.4	4.6	0.6	0.0	0.2	0.6	0.0	0.5	4.1	0.6	0.6	0.5	0.1	0.6	0.0	0.2	3.9	0.0	0.1	0.1	0.1	0.1	0.1	0.5	0.2	0.9	0.7	0.0	0.2	0.0	0.6	0.7	0.6	0.4	0.2	0.5	0.5	0.6	0.6
FP Multiply Latency	0.3	0.2	0.3	0.3	0.3	0.2	0.0	0.4	0.3	1.0	0.4	0.7	0.8	0.6	0.8	0.9	0.3	0.2	0.1	0.1	0.0	1.0	0.1	0.4	2.1	0.6	0.3	2.2	0.8	0.8	0.8	0.7	0.4	0.0	0.0	0.2	0.4	0.1	0.5	0.7	0.6	0.3	0.3	0.4	0.6	0.6	0.5
L2 Cache Block Size	0.2	0.3	0.1	0.2	0.3	0.0	1.0	0.0	0.1	1.0	0.3	0.4	0.5	0.4	0.4	0.1	0.2	0.6	0.5	0.3	4.3	0.2	0.5	0.6	0.5	0.1	0.2	0.0	0.4	0.4	0.4	0.4	0.2	0.5	0.4	0.4	0.3	0.0	0.4	0.3	0.4	0.8	0.7	0.8	0.4	0.6	0.5
LSU Entries	0.0	0.0	0.2	0.0	0.0	0.1	0.1	0.0	0.0	0.3	0.0	0.1	0.0	0.0	0.1	0.9	0.8	1.4	1.3	0.1	0.4	0.4	0.8	0.2	0.2	0.4	0.0	0.5	1.3	1.3	1.3	0.7	0.5	0.3	0.4	0.5	0.6	0.1	0.4	0.5	0.4	0.4	0.2	0.3	0.0	1.0	0.4
L2 Cache Associativity	0.4	0.5	0.3	0.5	0.4	0.8	1.1	0.1	0.7	0.1	1.1	0.7	0.9	1.1	0.8	0.3	0.1	0.1	0.1	0.0	0.0	0.1	0.2	0.3	1.6	0.2	0.5	0.1	0.0	0.0	0.0	0.3	0.1	0.9	0.8	0.5	0.2	0.1	0.1	0.1	0.2	0.3	0.3	0.2	0.0	0.1	0.4
FP Mult/Div	0.0	0.1	0.1	0.0	0.1	0.5	1.1	0.4	1.1	0.1	1.4	0.5	0.7	1.2	0.5	0.0	0.5	0.1	0.1	0.5	0.3	0.0	0.6	0.7	0.0	0.3	0.1	0.3	0.0	0.0	0.0	0.0	0.2	0.7	0.7	0.1	0.0	0.0	0.1	0.0	0.1	0.0	0.1	0.1	0.3	0.9	0.3
L1 D-Cache Associativity	0.3	0.2	0.1	0.4	0.1	0.9	0.9	2.5	1.3	0.0	0.3	0.0	0.2	0.4	0.1	0.1	0.2	1.0	1.0	0.9	0.4	0.1	0.0	0.0	0.8	0.1	0.1	0.0	0.0	0.0	0.0	0.0	0.2	0.3	0.4	0.0	0.0	0.1	0.1	0.1	0.0	0.1	0.1	0.0	0.2	0.2	0.3
L1 I-Cache Associativity	0.7	0.8	0.5	0.7	0.7	0.0	0.0	0.0	0.0	0.2	0.0	0.0	0.0	0.0	0.0	0.4	0.2	0.3	0.3	0.1	0.1	0.2	0.6	0.2	0.3	0.1	0.8	0.0	0.0	0.0	0.0	0.7	0.0	0.3	0.2	0.5	0.3	0.1	0.3	0.2	0.3	0.9	0.9	1.0	0.0	0.1	0.3
BPred Misprediction Penalty	0.4	0.1	0.4	0.2	0.2	0.2	0.2	1.3	0.8	0.1	0.3	0.0	0.1	0.1	0.0	0.4	0.7	0.5	0.4	0.1	0.1	0.2	0.0	0.3	1.5	0.7	0.2	1.0	0.1	0.1	0.1	0.0	0.0	0.1	0.1	0.2	0.1	0.0	0.0	0.0	0.0	0.1	0.1	0.0	0.1	0.5	0.3
D-TLB Associativity	0.0	0.0	0.0	0.0	0.0	0.1	0.2	0.1	0.4	0.1	0.1	0.0	0.0	0.1	0.0	0.1	0.0	0.2	0.2	0.0	0.1	0.0	0.0	0.0	4.0	0.1	0.0	0.0	0.0	0.0	0.0	0.2	0.1	2.5	2.2	0.3	0.2	0.0	0.1	0.1	0.1	0.1	0.1	0.1	0.0	0.0	0.3
Placeholder #1	0.0	0.0	0.1	0.0	0.1	0.3	1.5	1.1	1.4	0.3	0.7	0.2	0.5	0.7	0.3	0.1	0.0	0.4	0.4	0.1	0.4	0.1	0.1	0.2	0.5	0.8	0.0	0.0	0.1	0.1	0.1	0.0	0.0	0.1	0.1	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.2	0.5	0.3
Int Divide Latency	0.2	0.1	0.1	0.2	0.1	0.2	0.1	0.3	0.2	0.3	0.6	0.1	0.1	0.4	0.1	0.3	0.4	0.2	0.2	0.4	0.3	0.8	0.4	0.4	0.1	0.1	0.3	0.1	0.7	0.8	0.7	0.1	0.1	0.1	0.2	0.0	0.0	0.0	0.0	0.1	0.0	0.4	0.4	0.4	0.1	0.0	0.2
Instruction Fetch Queue Entries	0.1	0.2	0.1	0.1	0.1	0.0	0.0	0.2	0.2	0.0	1.2	0.9	1.1	1.1	0.9	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.1	0.1	0.0	0.0	0.7	0.4	0.0	0.0	0.0	0.2	0.1	0.2	0.2	0.2	0.2	0.3	0.4	0.3	0.4	0.1	0.2	0.1	0.1	0.1	0.2
FP ALU Latencies	0.0	0.0	0.0	0.0	0.0	0.1	0.4	0.0	0.3	0.3	1.2	0.3	0.6	1.0	0.4	0.3	0.0	0.1	0.1	0.1	0.1	0.1	0.0	0.0	0.1	0.0	0.0	2.6	0.4	0.4	0.3	0.0	0.4	0.1	0.1	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.1	0.2
Speculative Branch Update	0.0	0.0	0.0	0.0	0.0	0.2	0.5	0.3	0.6	0.4	0.3	0.1	0.1	0.2	0.1	0.0	0.3	0.6	0.6	0.6	0.4	0.0	0.3	0.4	1.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.1	0.0	0.1	0.2	0.3	0.2
FP Square Root Latency	0.0	0.0	0.1	0.0	0.0	0.2	0.5	0.6	0.6	0.2	0.2	0.0	0.0	0.2	0.0	0.1	0.2	0.4	0.4	0.6	0.0	0.0	0.2	0.1	0.0	0.0	0.1	0.0	0.1	0.1	0.1	0.2	0.0	0.0	0.0	0.1	0.1	0.0	0.2	0.3	0.2	0.2	0.2	0.3	0.2	0.2	0.2
Int Multiply Latency	0.0	0.0	0.1	0.0	0.1	0.2	0.4	0.3	0.5	0.0	0.2	0.0	0.0	0.1	0.0	0.2	0.1	0.3	0.3	0.6	0.4	0.4	0.0	0.4	0.6	0.1	0.0	0.0	0.3	0.3	0.3	0.1	0.0	0.2	0.2	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.1	0.0	0.2
I-TLB Latency	0.0	0.0	0.0	0.0	0.0	0.0	0.1	0.0	0.1	1.0	0.1	0.0	0.0	0.1	0.0	0.1	1.4	0.0	0.0	0.1	0.0	0.1	0.0	0.1	1.3	0.0	0.0	0.1	0.1	0.1	0.2	0.1	0.2	0.6	0.6	0.0	0.0	0.0	0.0	0.1	0.0	0.0	0.0	0.1	0.1	0.1	0.2
Int Mult/Div	0.0	0.0	0.0	0.0	0.0	0.1	0.7	0.9	0.7	0.0	0.3	0.1	0.2	0.3	0.1	0.1	0.0	0.5	0.5	0.1	0.0	0.0	0.2	0.0	0.4	0.2	0.0	0.0	0.0	0.0	0.0	0.0	0.1	0.3	0.3	0.0	0.0	0.0	0.0	0.0	0.0	0.1	0.1	0.2	0.0	0.0	0.2
Int ALU Latencies	0.5	0.7	0.8	0.3	0.7	0.0	0.1	0.2	0.4	0.0	0.1	0.0	0.0	0.0	0.0	0.0	0.2	0.3	0.2	0.1	0.0	0.0	0.0	0.0	0.0	0.1	0.3	0.0	0.0	0.0	0.0	0.1	0.2	0.0	0.0	0.3	0.3	0.1	0.0	0.1	0.0	0.1	0.2	0.0	0.0	0.3	0.1
Return Address Stack Entries	0.0	0.1	0.0	0.0	0.0	0.1	0.3	0.4	0.4	0.0	0.5	0.3	0.3	0.4	0.3	0.0	0.1	0.0	0.0	0.0	0.0	0.1	0.0	0.0	0.9	0.1	0.1	0.0	0.1	0.0	0.1	0.1	0.0	0.0	0.1	0.0	0.1	0.2	0.1	0.1	0.1	0.0	0.0	0.0	0.0	0.0	0.1
Placeholder #2	0.2	0.2	0.2	0.2	0.2	0.2	0.2	0.2	0.3	0.1	0.4	0.2	0.3	0.4	0.2	0.0	0.1	0.0	0.0	0.0	0.0	0.0	0.1	0.1	0.8	0.0	0.1	0.0	0.0	0.0	0.0	0.0	0.1	0.2	0.0	0.0	0.1	0.1	0.0	0.1	0.1	0.1	0.2	0.2	0.0	0.0	0.1
I-TLB Size	0.1	0.1	0.2	0.0	0.2	0.0	0.1	0.5	0.2	0.1	0.2	0.0	0.0	0.1	0.0	0.0	0.0	0.1	0.1	0.1	0.1	0.1	0.0	0.1	0.2	0.0	0.0	0.0	0.1	0.1	0.1	0.3	0.0	0.1	0.1	0.1	0.1	0.0	0.4	0.3	0.4	0.0	0.0	0.0	0.0	0.0	0.1
Memory Ports	0.0	0.0	0.0	0.0	0.0	0.0	0.3	0.0	0.2	0.6	0.1	0.0	0.0	0.0	0.0	0.1	0.1	0.2	0.1	0.5	0.1	0.0	0.0	0.2	0.3	0.0	0.0	0.1	0.1	0.1	0.1	0.0	0.2	0.1	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.3	0.1	0.1
L1 I-Cache Latency	0.0	0.1	0.1	0.0	0.1	0.1	0.4	0.0	0.4	0.0	0.3	0.1	0.2	0.4	0.1	0.1	0.2	0.1	0.1	0.0	0.0	0.0	0.0	0.0	0.3	0.0	0.0	0.1	0.0	0.0	0.0	0.0	0.2	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.1
FP Divide Latency	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.1	0.0	0.1	0.1	0.1	0.1	0.1	0.1	0.0	0.1	0.1	0.1	0.1	0.0	0.0	0.0	0.0	1.0	0.0	0.0	0.1	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.1	0.1	0.1	0.0	0.0	0.0	0.0	1.0	0.1
I-TLB Associativity	0.1	0.0	0.0	0.1	0.0	0.2	0.0	0.0	0.1	0.0	0.2	0.1	0.1	0.2	0.1	0.1	0.0	0.2	0.2	0.0	0.0	0.0	0.0	0.0	0.1	0.0	0.0	0.1	0.0	0.0	0.0	0.0	0.0	0.2	0.0	0.1	0.0	0.2	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.1

Table 2 also clearly shows the effect that each benchmark has on the processor. The "effect" that a benchmark has on the processor can be defined as the set of bottlenecks that the benchmark induces in the processor. For example, for a compute-intensive benchmark, the number of functional units and the branch prediction accuracy will probably be significant performance bottlenecks. On the other hand, for a memory-intensive benchmark, the size of the L2 cache and the memory latency may be the significant bottlenecks. An example of a compute-intensive benchmark is *gzip-graphic* since combined percentages of the number of integer ALUs and the branch predictor type (*i.e.*, branch prediction accuracy) accounts for over 33% of the total variation in the CPI. An example of a memory-intensive benchmark is *mcf* since the combined percentage of the L2 cache size and the memory latency is over 63%.

5.2 Processor Energy Consumption Bottlenecks

From Table 3, we can see that the most significant performance and energy consumption bottlenecks are quite similar. However, although the size and the associativity of the branch target buffer (BTB) are the two most significant energy consumption bottlenecks, both are insignificant as performance bottlenecks. Also, the number of ROB entries, which is the most significant performance bottleneck, is only the eighth most significant energy bottleneck. While BTB size and associativity accounts for the highest average percentage of the total power consumption variation, its true significance is not as dramatic as Table 3 would indicate. The reason for this apparent discrepancy is that the high value for the BTB associativity (fully-associative) forces the BTB to dissipate a disproportionately high amount of power, while not significantly affecting the performance. In other words, while using large fully-associative BTB inflates the importance of the BTB as an energy consumption bottleneck, the BTB associativity does not affect the significance of the BTB as a performance bottleneck.

The other significant parameters for performance and energy consumption are similar; five of the seven most significant parameters for performance and energy are the same (branch predictor type, memory latency, L2 cache latency, L2 cache size, and L1 I-cache size). Although bottlenecks such as the memory and L2 cache latency do not directly dissipate power, they still are important energy consumption bottlenecks since the processor still dissipates static power when servicing memory accesses. Therefore, higher latencies increase the overall energy consumption by increasing the amount of static energy consumption.

6 Similarity of SPEC CPU 2000 Bottlenecks

Understanding the similarity between benchmarks is extremely important when constructing a benchmark suite or when selecting a representative subset from a benchmark suite. Obviously, selecting benchmarks that are not very distinct may overestimate or underestimate the performance of an optimization and lead to misleading conclusions. On the other hand, simulating redundant benchmarks will significantly increase the time and effort required for performance evaluation.

To quantify the similarity between benchmarks, a computer architect can using several different criteria, such as the execution time, instruction mix, and cache miss rate. In this section, we measure the similarity between benchmarks in SPEC CPU 2000 suite based on the degree to which they stress the same processor bottlenecks. In other words, we consider two benchmarks to be similar if they stress the same performance and energy consumption bottlenecks to the same degree. In order to include the combined effect of performance and energy consumption, we use the energy-delay product (EDP), which is an energy-efficiency metric that combines performance with energy consumption [1]. EDP is the product of the CPI and EPI.

After calculating the P&B magnitudes for each benchmark based on their EDPs, we rank each bottleneck based on its P&B magnitude, where the bottleneck with the largest magnitude is assigned a rank of 1 and the smallest is assigned a rank of 43. Then, we vectorize their ranks such that a vector with 43 elements corresponding to the ranks of the EDP bottlenecks represents each benchmark. (Our previous experience indicated that using ranks instead of magnitude does not distort the results.) Since the dimensionality of the rank vector is very large, it is difficult to look at the data and draw meaningful conclusions from it. Therefore, we use PCA to reduce the dimensionality of the data set and remove correlation while retaining most of the original information. PCA computes new variables, called principal components, which are linear combinations of the original variables, such that the principal components are uncorrelated [2]. We retain the principal components that together account for at least 80% variance of the original data.

After using PCA, we use the K-means clustering algorithm [9] to cluster the benchmarks based on the similarity of their EDP bottlenecks. The K-means clustering algorithm groups the benchmarks into K distinct clusters. Since not all values of K fit the input data set well, we explore various values of K in order to find the optimal clustering for the given data set. Additionally, since the quality of the K-means clustering results depend on the initial placement of cluster centers, we use 100 different initial cluster starting points for each value of K.

Table 4: Optimal groups of clusters for the 46 benchmark-input pairs based on their similarity of energy-delay product bottlenecks.

Cluster	Benchmarks
1	*mesa, crafty, eon-cook,* ***eon-kajiya****, eon-rushmeier, perlbmk-makerand*
2	***perlbmk-splitmail_535****, perlbmk-splitmail_704*
3	*perlbmk-diffmail,* ***vortex-1****, vortex-2, vortex-3*
4	***wupwise****, swim, mgrid, equake, fma3d, sixtrack, gap*
5	*applu,* ***gcc-166****, gcc-integrate*
6	*gzip-graphic, gzip-program, gzip-random,* ***gzip-source****, perlbmk-splitmail_850, perlbmk-splitmail_957*
7	*gcc-200, gcc-expr,* ***gcc-scilab***
8	*gzip-log, parser, bzip2-graphic,* ***bzip2-program****, bzip2-source*
9	*mcf, facerec,* ***ammp****, twolf, apsi*
10	*vpr-route, galgel, art-110,* ***art-470***
11	***lucas***

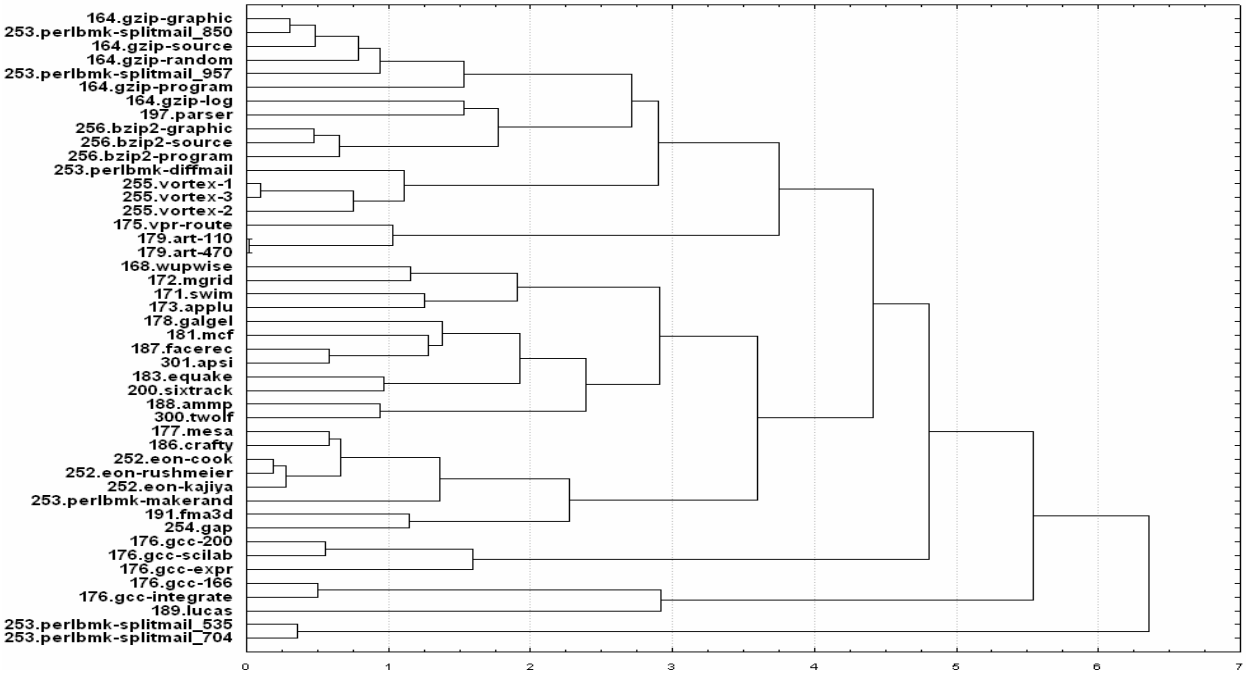

Figure 1. Dendrogram showing complete linkage distance for the 46 benchmark-input pairs based on their similarity of energy-delay product bottlenecks.

After clustering, we use the Bayesian Information Criterion (BIC) to determine the K-value with the best fit. For a value of K, the BIC score indicates the probability that the data belongs to K different normal distributions. Since a higher BIC score indicates a greater probability of a good fit for that value of K, we select the result that yields the highest BIC score as the optimal value of K [14]. After applying the aforementioned steps, we found that K=11, or 11 benchmark clusters, was the best fit for the bottleneck characterization data.

In order to visualize the relative positions of the benchmarks in the workload space and the distance between them, we also present a tree, or dendrogram, using hierarchical clustering. The vertical scale of the dendrogram lists the benchmark, while the horizontal scale corresponds to the linkage distance obtained from hierarchical clustering analysis. The shorter the linkage distance the closer the benchmarks are to each other in the workload space.

Table 4 shows this clustering while Figure 1 presents the dendrogram of this clustering. In Table 4, the benchmarks in boldfaced font are the ones that are closest to the center of their respective cluster. Note

that selecting the boldfaced benchmark from each cluster forms a representative subset of the SPEC CPU 2000 benchmarks based on their EDP bottlenecks. Computer architects can use this subset in lieu of the entire suite to reduce the simulation time.

In order to understand how the benchmarks are similar/dissimilar, we first categorized the bottlenecks into four different categories related to (1) Data memory (*e.g.*, L1 D-cache size, L1 D-cache latency, *etc.*), (2) Control flow (*e.g.*, Branch misprediction penalty, number of BTB entries, *etc.*), (3) Instruction memory (*e.g.*, L1 I-cache size, L1 I-cache latency, *etc.*), and (4) Processor core pipeline (Number of Integer ALUs, number of ROB entries, *etc.*), and then clustered the benchmarks considering only the bottlenecks from one category at a time.

Due to space constraints, we do not present the results for each of these categories. However, we use this information to explain why specific benchmarks were clustered together using the overall benchmark characteristics. From these results, we make a number of interesting observations related to the similarity between the input sets of the benchmarks, across benchmarks, and the benchmarks that emerge as outliers in terms of how they stress the performance and power bottlenecks with respect to the EDP efficiency metric.

6.1 Similarity Between Input Sets

In this section we discuss the similarity between the processor bottlenecks that various input sets invoke from a particular benchmark.

All the 3 inputs sets for *vortex* stress the data memory, control flow, and instruction memory bottlenecks almost identically. However, *vortex-2* stresses the processor core pipeline bottlenecks slightly differently (higher sensitivity to ALU latencies) as compared to the other two input sets. However, in general, the three different input sets for *vortex* exhibit very similar behavior, as do the three input sets of *bzip2*. Similarly, for *art*, the ranks of bottlenecks for the both input sets are almost identical. The same is true for the three input sets of *eon*. From these observations we can conclude that having more than one input set for *vortex, bzip2, art,* and *eon* does not expose a different bottlenecks that are fundamentally different.

On the other hand, the behavior of *gzip, gcc, and perlbmk* is heavily dependent on the input set. The {*graphic*, *program*, *random*, *source*} input sets for *gzip* exhibit similar behavior, whereas the *log* input set invokes a different behavior. More specifically, the *log* input does not stress the branch predictor as much as the other four input sets. *gcc* stresses the processor bottlenecks in two different ways. For the {*166*, *integrate*} input sets, the L1 I-cache size bottleneck is very significant, while the {*200*, *expr*, *scilab*} input sets are more sensitive to the branch predictor accuracy bottleneck. The bottlenecks stressed by *perlbmk* are also highly dependent on the input set. The *splitmail_850* and *splitmail_957* input sets are similar to each other while the same is true for the *splitmail_535* and *splitmail_704* input sets. The *splitmail_850* and *splitmail_957* input sets stress the ALU latencies and L2 cache related bottlenecks more than the *splitmail_535* and *splitmail_704* input sets. The other two input sets, *diffmail* and *makerand*, differ in how they stress the data memory bottlenecks; the *makerand* input set stresses the L1 D-cache size and L2 cache latency bottlenecks, while for *diffmail* L2 cache size is one of the bottlenecks that affects performance the least. The *diffmail* and *makerand* input stress the L1 I-cache bottlenecks significantly more than the other input sets of *perlbmk*. Therefore, depending on the input set, the input sets for *perlbmk* exposes 4 different sets of bottlenecks.

6.2 Similarity Between Benchmarks

The clusters in Table 4 show the similarity between various benchmarks across all workload characteristics. However, benchmarks can be more similar for a particular set of characteristics than others. In this section, we explain why a group of benchmarks are clustered together or appear in different clusters.

art, *mcf*, *ammp*, and *twolf* are similar to each other based on the data memory bottlenecks. These are the 4 benchmarks in SPEC CPU 2000 that heavily stress the data memory bottlenecks. Surprisingly, *art* and *mcf* – the benchmarks with the highest L1 D-cache miss rates in the entire suite – do not appear in the same cluster. While *art* and *mcf* stress data memory bottlenecks and control flow related bottlenecks similarly, they stress the instruction memory and processor core pipeline bottlenecks differently. Therefore, these two benchmarks appear in different clusters when all bottlenecks are used for clustering.

gzip-log stresses the bottlenecks in the same way as all input sets of *bzip2*. While *gzip* and *bzip2* are both compression algorithms, the other four input sets of *gzip*, {*graphic*, *program*, *random*, *source*}, and *bzip2* stress the bottlenecks differently enough that they belong to different clusters.

mesa and *crafty* form a very tight cluster and show very similar behavior across the 4 different categories of bottlenecks. *gzip*, *parser*, *perlbmk*, *vortex* and *bzip2* have similar behavior for the control flow bottlenecks, in that they heavily stress the branch predictor bottlenecks in the processor.

The benchmarks *mesa*, *crafty*, *fma3d*, *eon*, *perlbmk*, *gap*, and *vortex* all stress the instruction memory hierarchy, but end up in different groups

depending on whether they are sensitive to L1 I-cache latency, L1 I-cache size, *etc.*

6.3 Outlier Benchmark s

From the clustering results in Table 4, we observe that compared to the other benchmarks, *lucas* is very unique as it is the sole member of Cluster 11. Unlike other benchmarks, *lucas* only stresses the number of FP ALUs and the FP Multiply latency bottlenecks. All other processor core pipeline bottlenecks do have much impact on the performance and energy consumption. In stark contrast to the other benchmarks, the number of ROB entries is insignificant for *lucas*, while it is usually of the most significant bottlenecks for the other benchmarks. For the control flow bottlenecks, the branch misprediction penalty is important, but the number of BTB entries is not. Finally, the memory latency bottleneck is more important than the L2 cache latency bottleneck, which suggests that the L1 cache misses also result in L2 cache misses, thus making the memory latency a significant bottleneck.

Although *wupwise* appears in the same cluster as *swim*, *mgrid*, *equake*, *fma3d*, *sixtrack*, and *gap* when clustered based on all bottlenecks, it is an outlier when only considering the processor core pipeline bottlenecks. The reason for this is that the Integer and FP ALU bottlenecks are the highest ranked bottlenecks (have almost no impact on performance), which makes it an aberration compared to other benchmarks.

In conclusion, the results in this section show that characterizing and clustering benchmarks based on their bottlenecks can help computer architects gain a better understanding of the benchmark behavior and its similarity to other benchmarks. Computer architects can use this information when constructing benchmark suites and/or finding a representative subset of an existing suite.

7 Summary

One the key characteristics of a benchmark are the bottlenecks that it exposes when running on a processor. Since the performance of the processor is limited by its performance bottlenecks, and likewise for the energy consumption bottlenecks, determining which bottlenecks a benchmark exposes is important.

In this paper, we use the Plackett and Burman design to efficiently determine the performance and energy consumption bottlenecks in the SPEC CPU 2000 benchmark suite. Our results show that the number of ROB entries is the most important performance bottleneck, while the L2 cache size, L1 I-cache size, and memory latency are also significant. With the exception of the number of BTB entries and associativity, the energy consumption bottlenecks are very similar to that of the performance bottlenecks. The large "+1" value of the number of BTB entries and BTB associativity overinflates the importance of these two parameters as energy consumption bottlenecks.

Our clustering results show that *lucas* is the most unique benchmark based on its performance bottlenecks, which makes it a valuable addition to the SPEC CPU 2000 benchmark suite. Also, *gzip*, *gcc*, and *perlbmk* are the only benchmarks in the SPEC CPU2000 benchmark suite that expose dramatically different sets of performance bottlenecks depending on the input sets used. Finally, we provide subset of benchmarks that are representative of the performance and energy consumption bottlenecks stressed by the entire benchmark suite. Computer architects and researchers can use this as a guideline to subset the benchmark suite if the time required to simulate all the benchmarks is prohibitively high.

Acknowledgements

This work was supported in part by IBM, Intel, the University of Minnesota Digital Technology Center, the Minnesota Supercomputing Institute, NSF grant 0429806, IBM Corporation through a CAS award, Ghent University, and the European HiPEAC network of excellence. Lieven Eeckhout is a Postdoctoral Fellow of the Fund for Scientific Research – Flanders (Belgium) (F.W.O. Vlaanderen).

References

[1] D. Brooks, P. Cook, P. Bose, S. Schuster, H. Jacobson, P. Kudva, A. Buyuktosunoglu, J. Wellman, V. Zyuban, and M. Gupta, "Power-Aware Microarchitecture: Design and Modeling Challenges for Next-Generation Microprocessors," IEEE Micro, Vol. 20, No. 6, November/December 2000, pp. 26-44.

[2] G. Dunteman, "Principal Component Analysis," Sage Publications, 1999.

[3] L. Eeckhout, H. Vandierendonck, and K. De Bosschere, "How Input Data Sets Change Program Behaviour," Workshop on Computer Architecture Evaluation using Commercial Workloads, 2002.

[4] L. Eeckhout, H. Vandierendonck, and K. De Bosschere, "Workload Design: Selecting Representative Program-Input Pairs," International Conference on Parallel Architectures and Compilation Techniques, 2002.

[5] L. Eeckhout, R. Sundareswara, J. Yi, D. Lilja, and P. Schrater, "Accurate Statistical Approaches for Generating Representative Workload Compositions," International Symposium on Workload Characterization, 2005.

[6] B. Fields, S. Rubin, R. Bodik, "Focusing Processor Policies via Critical-Path Prediction,"

International Symposium on Computer Architecture, 2001.

[7] B. Fields, R. Bodik, and M. Hill, "Slack: Maximizing Performance under Technological Constraints," International Symposium on Computer Architecture, 2002.

[8] B. Fields, R. Bodik, M. Hill, and C. Newburn, "Using Interaction Cost for Microarchitectural Bottleneck Analysis," International Symposium on Microarchitecture, 2003.

[9] A. Jain and R. Dubes, "Algorithms for Clustering Data," Prentice Hall, 1988.

[10] D. Lilja, "Measuring Computer Performance," Cambridge University Press, 2000.

[11] D. Montgomery, "Design and Analysis of Experiments," Third Edition, Wiley 1991.

[12] A. Phansalkar, A. Joshi, L. Eeckhout, L. John, "Measuring Program Similarity: Experiments with SPEC CPU Benchmark Suites," International Symposium on Performance Analysis of Systems and Software, 2005.

[13] R. Plackett and J. Burman, "The Design of Optimum Multifactorial Experiments," Biometrika, Vol. 33, Issue 4, June 1946, Pages 305-325.

[14] T. Sherwood, E. Perelman, G. Hamerly, and B. Calder, "Automatically Characterizing Large Scale Program Behavior," International Conference on Architectural Support for Programming Languages and Operating Systems, October 2002.

[15] E. Tune, D. Liang, B. Calder, and D. Tullsen "Dynamic Prediction of the Critical Performance Path," International Symposium on High-Performance Computer Architecture, 2001.

[16] E. Tune, D. Tullsen, and B. Calder "Quantifying Instruction Criticality," International Conference on Parallel Architectures and Compilation Techniques, 2002.

[17] H. Vandierendonck and K. De Bosschere, "Many Benchmarks Stress the Same Bottlenecks," Workshop on Computer Architecture Evaluation using Commercial Workloads, 2004.

[18] http://www.eecs.umich.edu/~chriswea/ benchmarks/SPEC2000.html

[19] R. Wunderlich, T. Wenisch, B. Falsafi, and J. Hoe, "SMARTS: Accelerating Microarchitectural Simulation via Rigorous Statistical Sampling," International Symposium on Computer Architecture, 2003.

[20] J. Yi, D. Lilja, and D. Hawkins, "A Statistically Rigorous Approach for Improving Simulation Methodology," International Symposium on High-Performance Computer Architecture, 2003.