Many Benchmarks Stress the Same Bottlenecks

Hans Vandierendonck and Koen De Bosschere
Dept. of Electronics and Information Systems
Ghent University, Sint-Pietersnieuwstraat 41, 9000 Gent, Belgium
E-mail: {hvdieren,kdb}@elis.UGent.be

Abstract

The performance of a microprocessor is determined by many factors, including the memory hierarchy, clock frequency, organization, etc. A different trade-off between these factors is made in each microprocessor design. The performance is determined on the one hand by the bottlenecks of the machine (e.g., memory accesses, mispredicted branches, etc.) and their associated penalties and on the other hand by the frequency by which the bottlenecks occur, which is largely a property of the executed program. It is therefore important that a benchmark suite stresses all major bottlenecks in a microprocessor. If not, the benchmark suite gives a skewed view on performance. A method is presented to determine the most important bottlenecks stressed by benchmarks by analyzing their execution times. This method is applied to the SPEC CPU2000 benchmarks and it is shown that these benchmarks stress only about 4 important bottlenecks.

1 Introduction

There are many factors that determine the performance of a microprocessor, e.g., memory hierarchy, clock frequency, pipeline depth, issue width, etc. It is the task of the architect to make trade-offs between all of these factors in order to obtain a globally optimized design. The performance of the design is evaluated by means of benchmark programs. However, depending on the trade-offs made, different benchmarks will indicate a different speed for the machine, because the chosen trade-offs may turn out particularly favorable for one benchmark but not for another.

This paper analyzes the measurements of the SPEC CPU2000 benchmark suite running on 340 different machines as published by SPEC. We determine the factors that are actually measured or stressed by the CPU2000 benchmarks. Hereto, the execution time of a benchmark on a machine is modelled as a baseline execution time, increased by the number of cycles spent in each bottleneck. The importance of a bottleneck is determined by both the benchmark and the machine. The benchmark determines how many times the bottleneck is exercised (e.g., predictability of branches) while the machine design determines how large the influence is of the bottleneck on the execution time (e.g., branch mispredict penalty).

We use principal components analysis (PCA) to determine how many and which bottlenecks of the machines are actually stressed by the CPU2000 benchmarks. The bottlenecks obtained with this approach are expressed as the weighted sum of the execution times of the benchmarks. As such, the bottlenecks obtained with PCA are a mix of elementary bottlenecks (e.g., mispredicted branches, cache misses), just like the benchmarks themselves are a mix of bottlenecks. The principal components are “usage modes”, i.e., they describe how a benchmark uses a machine, i.e., how much each of the bottlenecks of the machine is stressed. The number of usage modes and their associated penalties are determined using PCA. Unfortunately, it is hard to interpret these usage modes in terms of the elementary bottlenecks.

Most bottlenecks are evidently exercised by all benchmarks (e.g., branch mispredictions and cache misses) but each benchmark typically exercises a different mix of bottlenecks. If two benchmarks do exercise a very similar mix of bottlenecks, then these benchmarks are redundant, as the same speed-up is measured. The principal components analysis allows us to identify groups of similar benchmarks. It is shown that most of the SPEC CPU2000 benchmarks reduce to 4 usage modes (i.e., all processor bottlenecks are exercised in only 4 different mixes). In other words, a benchmark suite with 4 benchmarks gives almost the same information on the investigated machines as the full suite containing 26 benchmarks.

1.1 Performance Model

Performance is measured as the execution time of a benchmark. Let us call T_{ij} the execution time of benchmark i on machine j. As we analyze the execution times of benchmarks on systems introduced over a period of 4 years, it is evident that technological progress has a major influence (e.g., clock frequency, memory bandwidth and latency). We reduce this effect by studying the number of cycles it takes to run a benchmark, as this more closely follows architectural improvements:

$$C_{ij} = T_{ij}\ freq_j \tag{1}$$

In principle, all instructions can be executed in a small number of cycles. This is sometimes called the baseline CPI (cycles per instruction). Some instructions require more cycles to execute than others, e.g., when there are branch mispredictions or cache misses [11]. For these instructions, the baseline CPI has to be increased by a certain amount, depending on how frequent these instructions are and on how many cycles it takes to handle them. Based on these observations, we model the cycle count as the sum of the cycle counts spent in each of the bottlenecks:

$$C_{ij} = \sum_{k=1}^{K} A_{ik}\, CPI_{kj} \tag{2}$$

The term CPI_{kj} describes the additional cycles that the kth bottleneck introduces when it occurs in machine j and the term A_{ik} describes how many times the bottleneck occurs in benchmark i. The problem faced in this paper is to identify the K bottlenecks and to determine the quantities A_{ik}.

We deduce the bottlenecks from the correlations between the cycle counts C_{ij}. Hereto, we apply principal components analysis [7]. Principal components analysis (PCA) transforms the 26 performance metrics (26 cycle counts, one per benchmark) into 26 new metrics, called principal components. Each of these new performance metrics corresponds to a bottleneck in Equation 2. The principal components are not correlated to each other and are sorted with decreasing variance. The principal components are a linear combination of the original performance metrics. Thus, machine j has a value for the kth principal component equal to

$$PC_{kj} = \sum_{i=1}^{26} a_{ki}\, C_{ij} \tag{3}$$

where the coefficients a_{ki} are determined by PCA. Interpreting the principal components as the bottlenecks of the machines (i.e., $PC_{kj} = CPI_{kj}$) learns that Equation 3 is the inverse of Equation 2. Consequently, the coefficients A_{ik} can be identified: the matrix A containing the coefficients A_{ik} is the inverse of the matrix a of coefficients a_{ki}.

In the above argument we have to assume that $K = 26$. However, as most principal components only have a small variance (i.e., they do not contain much information), it suffices to study only those principal components (bottlenecks) with the highest variance (impact on performance). This way, the number of important bottlenecks can be reduced to a manageable number (e.g., $K = 4$).

It is best to normalize the data before applying PCA, i.e., the mean cycle count for each benchmark should be zero and the standard deviation should be one. If the data is not normalized, then PCA will attach a higher weight to benchmarks that run longer. This is an undesirable effect, because it seems that SPEC did not intentionally choose to make some benchmarks run longer, but that the running times of the benchmarks is determined by the inputs that were available. Thus, PCA is applied to the quantities

$$C'_{ij} = (C_{ij} - \mu_i)/\sigma_i \tag{4}$$

where μ_i is the average execution time of benchmark i over all machines and σ_i is the standard deviation of the execution times of benchmark i.

1.2 Related Work

Workload characterization concerns itself with measuring abstract performance metrics, called workload characteristics [5]. These abstract metrics allow an architect to anticipate the effect of design decisions on performance. There are numerous abstract metrics that one can measure, each corresponding to one particular aspect of a processor [3, 12]. The CPI model is often used to separate the effect of the memory hierarchy from the remainder of the execution time [1, 11].

The abstract metrics typically do not change much when the input to the program is changed. This can be detected using principal components analysis or cluster analysis techniques [8, 10, 9].

Several papers have analyzed the suitability of the SPEC benchmarks for research purposes. Citron analyses how often the CPU benchmarks are subsetted and arguments that this is a bad idea [4]. He also shows that many authors still use old benchmark suites. Weicker also expresses his concerns about the use of old benchmark suites and of subsetting [14]. Mirghafori *et al.* [13] show that the compiler flags have a huge impact on the execution times, so the compilation mode should always be reported.

Dujmovic and Dujmovic [6] develop a method for the quantitative evaluation of benchmark suites that is similar to the model presented in this paper. They define metrics that measure the size, completeness and redundancy of the benchmark space. In the current work, machines are mapped in a space where the coordinates of the machines are determined by performance metrics (benchmark cycle counts) while Dujmovic and Dujmovic map benchmarks in a benchmark space where the coordinates are determined by execution times. Although the concept of a space of benchmarks is a useful one, it is questionable whether the coordinates of the benchmarks should be determined by the execution times of the benchmarks on a set of machines. This implies that the characteristics of the benchmarks are determined by the machines on which they are run. Consequently, two benchmarks may be very similar when the benchmark space is defined using only desktop processors, but when workstations and servers are also used to identify the benchmark space, then the benchmarks suddenly have different properties.

2 Experimental Setup

We analyze the CPU2000 data published on the SPEC website (`http://www.spec.org/cpu2000/results/`) before August 2003. As we are interested in both SPECint and SPECfp results for all machines, the analysis is limited to those machines for which both SPECint and SPECfp peak results have been submitted. As SPEC publishes these results in separate files, we have to match the SPECint and SPECfp results files that describe the same machine. Each file describes the tested machine and the running software in detail using a number of lines of the type "Compiler: This Company's C Compiler". Two files describe the same machine when there is a textual match on all descriptions of the hardware and the software.[1] Results that are incomplete because they are not in compliance with the SPEC rules are ignored. It is also required that the measurements are performed on the same date. Using these rules, we find that every SPECint result file is coupled to at most 1 SPECfp result file. This method finds 340 machines, representing 7 different architectures, for which both SPECint and SPECfp peak performance results are submitted (Table 1).

[1]Spaces and special symbols are stripped. The disk subsystem is not compared, as it should not have an influence. It is usually not possible to have a match on the compiler field, as SPECint requires a C and a C++ compiler and SPECfp requires a C and a Fortran compiler. Therefore, the compiler field is ignored and random checks are performed manually to verify that the same version of the C compiler is used.

Table 1. The architectures (ISA) and the generations of processors in this study.

Architecture	Generation	No.
Alpha AXP	21264	19
	21364	3
80x86	Pentium III	32
	Pentium 4	86
	Pentium M	1
	Xeon (Pentium III and 4)	37
	Athlon	27
	Opteron	6
Itanium	Itanium	1
	Itanium 2	5
PA-RISC	8600	3
	8700	8
POWER/PowerPC	PowerPC 604e	1
	RS64-III, RS64-IV	13
	POWER3	17
	POWER4	18
MIPS	R12000	4
	R14000	5
SPARC	UltraSPARC II/IIi	6
	UltraSPARC III	10
	UltraSPARC III Cu	14
	SPARC64GP, SPARC64V	24

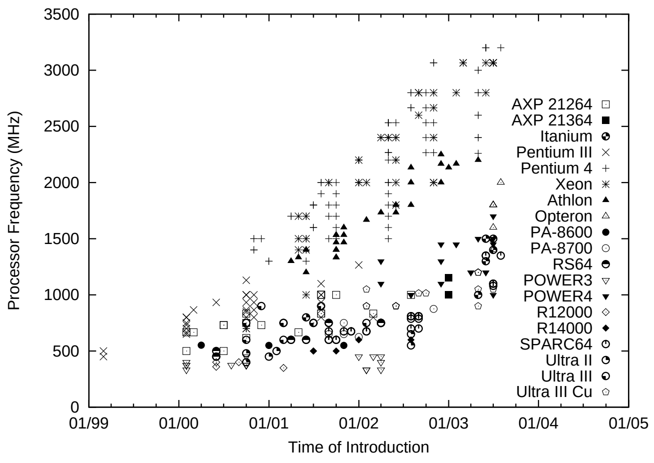

Figure 1. The clock frequency varying over time. The date shown is the time when both the tested hardware and software are available to the public.

3 Analysis

3.1 The Data

We first analyze the trends that are present in the SPEC performance measures. It is well known that,

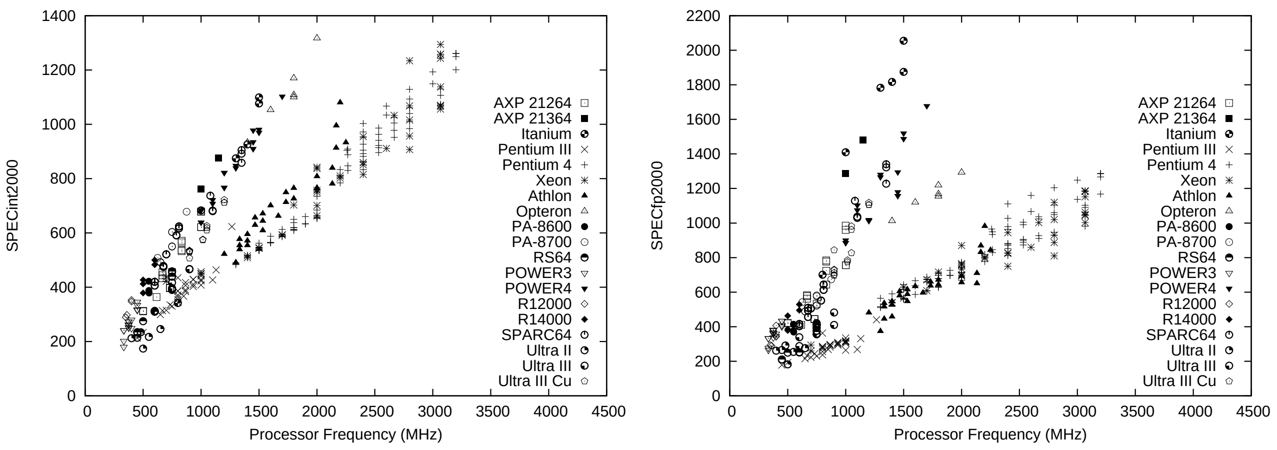

Figure 2. Peak performance for the integer (left) and floating-point benchmarks (right) as a function of the processor's clock frequency.

through transistor scaling and architectural choices, the processor clock frequency continually increases. At any time, machines are introduced covering a large range of different frequencies (Figure 1). These machines can be cataloged into either the "brainiac" processors, which try to achieve high IPC but have a limited frequency, and the "speed demons", which strive for high clock frequencies, whereby they can only achieve a limited amount of IPC. The graph shows that the Pentium III, Pentium 4, Xeon and Athlon processors are of the speed demon type. The frequency of the speed demons grows with about 800MHz per year, while the growth rate is about 300–350 MHz for the brainiacs. This number excludes the Opteron which was introduced only recently and has a clock frequency which is somewhere in between the brainiacs and the speed demons.

There is more to performance than the processor frequency. The summary performance metrics called SPECint2000 and SPECfp2000 are shown in Figure 2. These assume that peak optimization flags are used. The plot of SPECint2000 clearly shows the difference between "brainiacs" and "speed demons". The brainiacs try to achieve high IPC but have a limited frequency. The "speed demons" strive for high clock frequencies, whereby they can only achieve a limited amount of IPC. The brainiacs are clustered in the higher range, while the speed demons form the lower range. A similar trend can be detected in the SPECfp2000 metric. However, the points in the brainiac cluster are not so closely packed, especially in the range of 1 to 2 GHz.

The SPECint2000 metric correlates strongly with the clock frequency. However, one should not conclude from this that the SPECint2000 benchmarks only measure MHz! The processor frequency increases linearly over time mostly because of technological progress. The same progress drives the possibility to build bigger and more complex processors and to improve the performance and capacity of the memory subsystem. Hence, it is normal that any benchmark is executed faster as technology progresses, which is indicating by an increased frequency.

3.2 Not All Benchmarks See an Equal Speed-Up

This section shows that the design decisions made for a machine may lead to a strong improvement for some benchmarks, but not for others. The execution speed of benchmark i on machine j is defined as $S_{ij} = R_i/T_{ij}$ where R_i is the execution time of the benchmark on a reference machine, a Sun Ultra5-10 300 MHz. The SPECint2000 and SPECfp2000 metrics are the geometric mean of the S_{ij} values of the integer and floating-point benchmarks.

The execution speed of a small number of machines and benchmarks shows that the selected SPARC processors obtain a huge speed for art (Figure 3). These processors can execute galgel at a rate comparable to the other processors while the remaining benchmarks are executed rather slowly. Thus, when speed is measured using only art, then these SPARC processors will appear very fast, but when speed is measured using one of gap, crafty or mcf, then these processors are rather slow. With a similar reasoning, it follows that using galgel to measure speed shows that the POWER4 is very fast, while this machine is only moderately fast when the execution time of crafty is the measure of speed.

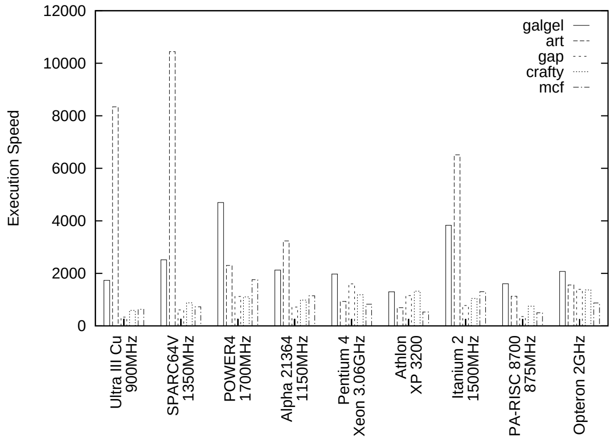

Figure 3. A sample of the results showing that different machines can obtain very different speed-ups, depending on the benchmark.

3.3 The Usage Modes

Let us now turn to the identification of the usage modes that are present in the SPEC CPU2000 benchmarks. Application of PCA on the 26 cycle counts delivers 26 principal components, each corresponding to a bottleneck. The following table shows the percentage of the total variance explained by each principal component, as well as the cumulative percentage of variance that is explained.

	PC1	PC2	PC3	PC4	PC5	PC6
%Var	64.36	14.77	5.84	3.39	2.75	1.76
Cum.	64.36	79.13	84.97	88.36	91.11	92.87

Only a few principal components have a large variance. How many PCs to include in the analysis is up to the user of PCA. Dunteman [7] advices to analyze all principal components that explain more information than a single variable, i.e., the variance is higher than (100%/26). We select 4 principal components, as PC4 explains slightly less information than a single benchmark but it still has a useful interpretation. Note that it is equally valid to state that the CPU2000 benchmarks stress 3 or 5 bottlenecks, but it is important to note that the 4th and 5th bottlenecks play a minor role. PC1–PC4 account for 88.36% of the information present in the data. In other words, the CPU2000 benchmarks exercise the machines in only 4 different ways, i.e., only 4 usage modes are present or the elementary bottlenecks are stressed in only 4 different mixes.

To make the interpretation of the results easier, the first 4 principal components are rotated using the varimax procedure [7]. Rotation redefines the principal components such that each principal component either does not load on a variable ($a_{ki} \approx 0$) or it loads strongly on a variable (a_{ki} differs from zero and all non-zero a_{ki} have approximately the same magnitude). Other values of a_{ki} make the interpretation difficult. The resulting components, called factors, span the same 4-dimensional space as PC1–PC4 and together they explain the same amount of variance. The factors are named F1 to F4. The factor loadings a_{ki} are presented in Table 2. Note that, due to the rotation, values in the range 0.10–0.20 are rare.

Table 2. The Factor Loadings (a_{ki})

Benchmark	F1	F2	F3	F4
gzip	**0.17**	0.07	**-0.33**	**0.20**
vpr	**0.35**	0.00	-0.06	0.07
gcc	**0.25**	0.06	0.11	**-0.14**
mcf	**0.38**	0.07	**0.24**	-0.06
crafty	**0.17**	-0.01	**-0.29**	0.04
parser	**0.16**	**-0.19**	-0.09	**-0.20**
eon	**-0.23**	0.08	**-0.54**	-0.11
perlbmk	0.01	-0.08	**-0.43**	-0.01
gap	-0.01	**0.67**	-0.05	**0.36**
vortex	**0.17**	-0.02	**-0.21**	-0.04
bzip2	**0.31**	-0.02	-0.06	-0.00
twolf	**0.34**	0.03	-0.04	0.06
wupwise	0.07	**0.23**	-0.02	**-0.15**
swim	**-0.17**	**0.20**	-0.01	**-0.35**
mgrid	-0.04	**0.20**	-0.02	**-0.29**
applu	-0.04	0.07	0.05	**-0.42**
mesa	-0.01	0.04	**-0.36**	-0.09
galgel	0.05	-0.02	-0.03	**-0.34**
art	**0.27**	-0.07	0.06	**-0.16**
equake	0.04	**0.35**	0.07	-0.05
facerec	0.05	0.04	-0.00	**-0.30**
ammp	**0.29**	-0.00	-0.06	-0.03
lucas	-0.02	**0.33**	0.12	**-0.20**
fma3d	0.05	**0.30**	-0.11	-0.05
sixtrack	0.05	-0.05	**-0.16**	**-0.26**
apsi	**0.29**	0.10	-0.05	0.02

The factors F1 through F4 model the number of cycles that are imposed as a penalty when bottleneck k occurs. Figure 4 shows scatter plots where each machine is plotted against the 4 factors (bottlenecks). Note that, by using PCA to determine these values, the presented numbers are the deviations of the cycle counts from the average over all machines, because the cycle counts are normalized (mean=0, std. dev.=1) before applying PCA (see Equation 4).

Machines of the same generation are strongly clustered in the scatter plots. Consequently, changing the clock frequency or memory system does not fundamen-

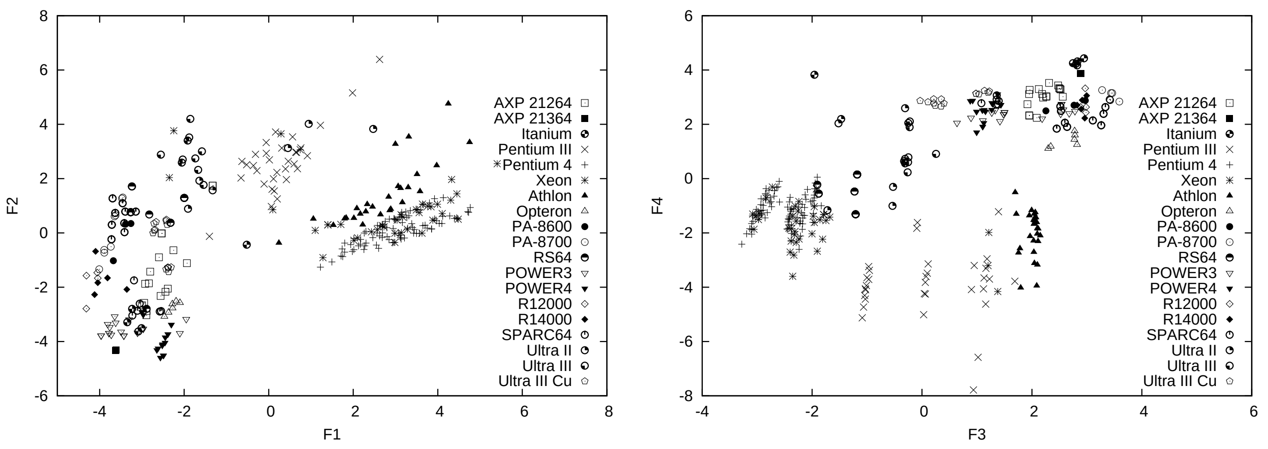

Figure 4. Scatter plots showing each machine as a point in the four-dimensional space of bottlenecks. The position of the point shows how sensitive the machine is to each of the bottlenecks.

tally change the behavior of the machine. If it performs better for a particular benchmark in one configuration, then it will remain better for that benchmark in another configuration.

There is a clear difference between how the brainiacs and the speed-demons achieve high performance. Brainiacs typically have F1< 0 and F4> 0. There are a few exceptions: the Ultra-II's have F1> 0 and the RS64's have F4< 0 because they do not have very powerful floating-point hardware (see below).

Even though the analysis works on cycle counts, F1 proves to correlate well with the clock frequency of the processor. The correlation coefficient is 0.74, so machines with higher frequencies have larger values for F1. If F1 loads positively on the cycle count of a benchmark, then, when increasing the clock frequency of the processor, the benchmark will need additional cycles. Consequently, these benchmarks resist optimizations by increasing the frequency (e.g., deep pipelining). Reasons for this could be the presence of many off-chip memory accesses and/or a limited branch prediction accuracy. On the other hand, if F1 loads negatively on a benchmark, then it is possible to speedup that benchmark by means of deeper pipelines. The factor loadings a_{ki} (Table 2) reveal that the benchmarks that resist increases in frequency are gzip to parser, vortex to twolf, art, ammp and apsi, while eon and swim profit from this. The other benchmarks are largely indifferent to the frequency, i.e., their cycle count is mostly determined by other factors.

The factor loadings show that factor F4 is mostly determined by floating-point benchmarks with strong negative loadings. Thus, F4 determines to a large extent how well the machine performs on floating-point benchmarks. Machines that perform well on these benchmarks (i.e., they have many floating-point units, exploit ILP and have large data caches) score positively on F4. Note that the RS64 processors of the POWER architecture score negatively on F4: these processors are optimized for a multi-threaded commercial workload and do not have very extensive floating-point hardware [2].

3.4 Similar Benchmarks

Each usage mode is expressed as a linear combination of the cycle counts of the benchmarks. A usage mode will load strongly on a group of benchmarks when the cycle counts of these benchmarks are correlated (i.e., if machine j executes benchmark i faster than machine j', then it will also execute benchmark i' faster). The usage mode loads negatively on a benchmark when its cycle count is negatively correlated with that of the other benchmarks. The factor loadings show which benchmarks are similar (Table 2). Factor loadings printed in bold are considered to be important and other values are considered to be noise. The threshold to distinguish between the two is set to 0.14.

Usage mode F1 states that the benchmarks gzip, vpr, gcc, mcf, crafty, parser, vortex, bzip2, twolf, art, ammp and apsi have a common aspect of behavior. The benchmarks eon and swim are a-typical examples of this behavior and techniques that would typically speed-up a benchmark with the properties exercised in usage mode F1 actually result in a slow-down for eon and swim. E.g., deeper pipelines could speed up the group of benchmarks with positively loadings for F1, while it would slow down the others.

Usage mode F2 says that the benchmarks gap, wupwise, swim, mgrid, equake, lucas and fma3d have a common property while parser is an a-typical example of this property. Usage mode F3 shows that gzip, crafty, eon, perlbmk, vortex, mesa and sixtrack are similar and that mcf has the opposite behavior. Finally, F4 states that parser, wupwise, swim, mgrid, applu, galgel, art, facerec, lucas and sixtrack have a common property while gzip and gap are a-typical. From the interpretation of F4 in the previous paragraph, we know that F4 measures floating-point performance. Thus, machines with good floating-point performance typically score worse on gzip and gap than machines which are not optimized for floating-point.

3.5 Reducing the Benchmark Suite

As CPU2000 contains only 4 usage modes, many of the benchmarks are redundant. We reduce CPU2000 to a smaller number of benchmarks that, when used instead of the full suite, leads to a good approximation of the overall performance metric. Note that it will not be possible to reduce the suite to 4 benchmarks, as none of the benchmarks will coincide exactly with one of the 4 usage modes. Thus, a few more benchmarks will be necessary.

Dunteman [7] describes a heuristic procedure to rank the benchmarks on which PCA is applied in order of decreasing interest. It is assumed that the benchmarks that have the highest factor scores in the first principal components cover the largest part of the total variance. Benchmarks are selected one at a time. In step k, the benchmark i that has the highest factor loading a_{ki} in PC_k is put in position k. If it has already been selected, then the variable with the second highest factor score is put in this position, etc.

The resulting ranking is presented in Table 3. The table also shows in row n what percentage of the total variance is explained by the first n benchmarks. The table shows that apsi alone explains almost 60% of the differences between the 340 machines. With 4 benchmarks, already 80% of the differences are explained and 9 benchmarks explain 90%. If an error of less than 5% is desired, then one needs 14 benchmarks, or just more than half of the CPU2000 suite. Note that leaving out vpr from CPU2000 has no impact on what machine is deemed to be the fastest.

The ranking shows that, if subsetting the suite is desired or necessary, then it is recommended to include at least apsi, lucas, mcf and gap. Note that, as the procedure is heuristic, there are other combinations of benchmarks which together explain the same amount of variance. In any case, the conclusion of this experiment is that one does not need 26 benchmarks to collect the same information that CPU2000 collects.

At the last ISCA conference, a discussion was held on whether subsetting the SPEC benchmarks for research purposes is good practice, or not [4]. The above results show that subsetting SPEC seems good practice, provided that the choice of subset is motivated. Note that, as one typically focuses on a particular part of a processor when doing research, the required subset will be dependent on the topic of investigation.

Table 3. A ranking of the CPU2000 benchmarks.

#	Bench.	% Var.	#	Bench.	% Var.
1	apsi	58.9%	14	gzip	95.4%
2	lucas	72.9%	15	perlbmk	96.3%
3	mcf	76.8%	16	applu	97.2%
4	gap	80.2%	17	galgel	97.5%
5	facerec	84.4%	18	wupwise	98.0%
6	mesa	86.4%	19	equake	99.0%
7	art	87.0%	20	mgrid	99.1%
8	eon	88.9%	21	gcc	99.4%
9	parser	91.1%	22	vortex	99.7%
10	fma3d	91.6%	23	bzip2	99.8%
11	swim	93.2%	24	sixtrack	99.9%
12	crafty	94.0%	25	ammp	100.0%
13	twolf	94.5%	26	vpr	100.0%

The notion that benchmarks measure largely the same factors of the performance of a machine is strengthened by measuring what percentage of the total variance is already explained by a single benchmark (Table 4, column %Var.). Most benchmarks measure between 27.4% and 58.9% of the variance of the whole suite. Gap is an exception as this benchmark alone captures only 10.6%, which implies that gap contains a small number of usage modes. We also measured the percentage of variance that is unique to each benchmark, i.e., a difference between two machines that only this benchmark can detect (column Uniq.). Every benchmark contains less than 1% of the total variance that is not measured by the other benchmarks. It follows that a single benchmark contains hardly any information that is not present in the other benchmarks. Or if it does, then it has no influence on the performance of the investigated machines.

4 Conclusion

A benchmark suite has to stress all important bottlenecks of a microprocessor (i.e., conditions or situations that hamper performance). Each benchmark typically stresses multiple bottlenecks at once but stresses some more than others. Ideally, each benchmark in a benchmark suite should stress a different mix of bottlenecks. We call such a mix of bottlenecks a usage mode. Bench-

Table 4. Percentage of variance explained by the benchmarks.

Bench.	%Var.	Uniq.	Bench.	%Var.	Uniq.	Bench.	%Var.	Uniq.	Bench.	%Var.	Uniq.
gzip	35.1%	0.25%	vpr	54.3%	0.02%	gcc	51.3%	0.21%	mcf	45.3%	0.16%
crafty	49.0%	0.07%	parser	39.8%	0.83%	eon	28.5%	0.38%	perlbmk	33.0%	0.28%
gap	10.6%	0.43%	vortex	48.8%	0.22%	bzip2	56.5%	0.02%	twolf	53.4%	0.11%
wupwise	51.2%	0.12%	swim	29.2%	0.24%	mgrid	46.0%	0.12%	applu	44.7%	0.14%
mesa	45.4%	0.29%	galgel	51.5%	0.23%	art	52.7%	0.14%	equake	27.4%	0.72%
facerec	47.8%	0.18%	ammp	57.9%	0.05%	lucas	31.1%	0.35%	fma3d	46.7%	0.19%
sixtrack	53.6%	0.10%	apsi	58.9%	0.06%						

marks that stress the same usage modes are mutually redundant.

Principal components analysis, a statistical data analysis technique, is applied to derive the most important usage modes from the cycle counts of the SPEC CPU2000 benchmarks. It is shown that the 26 CPU2000 benchmarks use a machine in only 4 different ways. Consequently the number of benchmarks can be reduced without loosing much information. When using fourteen benchmarks only 5% of the information in the benchmark suite is lost. Using 9 carefully selected benchmarks throws away 10% of the information that can be captured with the full suite.

Acknowledgements

Hans Vandierendonck is sponsored by the Flemish Institute for the Promotion of Scientific-Technological Research in the Industry (IWT). Many thanks to Bjorn De Sutter for proof-reading the paper. The authors are indebted to SPEC (`http://www.spec.org/`) for publishing the vendor's measurements and to the vendors for making the measurements. All interpretation of the data is that of the authors.

References

[1] A. Borg, R. E. Kessler, G. Lazana, and D. W. Wall. Long address traces from RISC machines: Generation and analysis. Technical report, Western Research Laboratory, Sept. 1989.

[2] J. Borkenhagen, R. J. Eickemeyer, R. Kalla, and S. Kunkel. A multi-threaded PowerPC processor for commercial servers. *IBM Journal of Research and Development*, 44(6):885–898, Nov. 2000.

[3] P. Bose and T. M. Conte. Performance analysis and its impact on design. *IEEE Computer*, 31(5):41–49, May 1998.

[4] D. Citron. The use and abuse of SPEC: An ISCA panel. *IEEE Micro*, 23(4):73–77, July 2003.

[5] T. M. Conte and W.-m. W. Hwu. Benchmark characterization. *IEEE Computer*, 24(1):48–56, Jan. 1991.

[6] J. J. Dujmovic and I. Dujmovic. Evolution and evaluation of SPEC benchmarks. *ACM SIGMETRICS Performance Evaluation Review*, 26(3):2–9, Dec. 1998.

[7] G. H. Dunteman. *Principal Components Analysis.* SAGE Publications, 1989.

[8] L. Eeckhout, H. Vandierendonck, and K. De Bosschere. How input data sets change program behaviour. In *Workshop on Computer-Architecture Evaluation Using Commercial Workloads, held in conjunction with HPCA-8*, Feb. 2002.

[9] L. Eeckhout, H. Vandierendonck, and K. De Bosschere. Designing computer architecture research workloads. *IEEE Computer*, 36(2):65–71, Feb. 2003.

[10] L. Eeckhout, H. Vandierendonck, and K. De Bosschere. Quantifying the impact of input data sets on program behavior and its applications. *Journal of Instruction-Level Parallelism*, 5:1–33, 2 2003.

[11] P. G. Emma. Understanding some simple processor-performance limits. *IBM Journal of Research and Development*, 41(3):215–231, May 1997.

[12] L. K. John, P. Vasudevan, and J. Sabarinathan. Workload characterization: Motivation, goals and methodology. In *Workload Characterization: Methodoloy and Case Studies*. IEEE Computer Society, 1999.

[13] N. Mirghafori, M. Jacoby, and D. Patterson. Truth in SPEC benchmarks. *ACM Computer Architecture News*, 23(5):34–42, Dec. 1995.

[14] R. Weicker. On the use of SPEC benchmarks in computer architecture research. *Computer Architecture News*, 25(1):19–22, 1997.