# Finding Stress Patterns in Microprocessor Workloads

Frederik Vandeputte Lieven Eeckhout

Department of Electronics and Information Systems (ELIS), Ghent University, Belgium
fgvdeput,leeckhou@elis.UGent.be

**Abstract.** Power consumption has emerged as a key design concern across the entire computing range, from low-end embedded systems to high-end supercomputers. Understanding the power characteristics of a microprocessor under design requires a careful study using a variety of workloads. These workloads range from benchmarks that represent typical behavior up to hand-tuned stress benchmarks (so called stressmarks) that stress the microprocessor to its extreme power consumption.

This paper closes the gap between these two extremes by studying techniques for the automated identification of stress patterns (worst-case application behaviors) in typical workloads. For doing so, we borrow from sampled simulation theory and we provide two key insights. First, although representative sampling is slightly less effective in characterizing average behavior than statistical sampling, it is substantially more effective in finding stress patterns. Second, we find that threshold clustering is a better alternative than k-means clustering, which is typically used in representative sampling, for finding stress patterns. Overall, we can identify extreme energy and power behaviors in microprocessor workloads with a three orders of magnitude speedup with an error of a few percent on average.

## 1 Introduction

Energy, power, power density, thermal hotspots, voltage variation, and related design concerns have emerged as first-class microprocessor design issues over the past few years. And this is the case across the entire computing range, from low-end embedded systems to high-end supercomputers. A detailed understanding of these issues is of primary importance for designing energy-aware, power-aware and thermal-aware microprocessors, their power and thermal management strategies, their power supply unit, and thermal package.

Understanding the power, energy and thermal characteristics of a microprocessor under design requires appropriate benchmarking and simulation methodologies. At the one end of the spectrum, researchers and engineers consider average workload behavior. This is appropriate for studying a microprocessor's average power consumption or thermal map, however, it does not capture more extreme behaviors. At the other end of the spectrum, stressmarks are being used to explore a microprocessor's maximum power consumption [9,10], maximum thermal hotspots [27], and maximum dI/dt behavior [16]. These stressmarks are typically hand-tuned, and push the microprocessor to its extremes in order to understand the microprocessor's worst-case behavior. These

stress patterns are not expected to occur during typical operation, however, they *can* occur and therefore the microprocessor should be able to cope with them.

Microprocessors designed for maximum possible power consumption are not cost-effective though because of the large gap between maximum and typical power consumption. Dynamic thermal management (DTM) techniques [1,23] seek to exploit this gap: the microprocessor cooling apparatus is designed for a wattage less than the maximum power consumption, and a dynamic emergency procedure guarantees that this designed-for wattage level is never exceeded with minimal impact on overall performance. Gunther et al. [11] report that DTM techniques based on clock gating permitted a 20% reduction in the thermal design power for the Intel Pentium 4 processor. Developing and evaluating DTM mechanisms however requires adequate evaluation methodologies for quickly finding the extreme behaviors in typical workloads that are subject to DTM.

Therefore, this paper closes the gap between the two ends of the power benchmarking spectrum by studying ways of identifying *stress patterns* in typical workloads, also called 'worst-case execution behaviors' by Tiwari et al. [25]. More specifically, the goal of this work is to find stress patterns in typical workloads with the least possible simulation time. Identifying stress patterns in typical workloads is important because these stress patterns are expected to occur regularly in practice, much more often than the stress patterns represented by hand-tuned stressmarks. The stress patterns are the execution behaviors that DTM emergency procedures should adequately deal with.

We build on sampled simulation theory for identifying stress patterns in typical workloads. However, in contrast to sampled simulation for which the aim is to estimate *average* performance or power consumption by simulating a representative sample of the entire program execution, the goal in this paper is to leverage sampled simulation theory to find a sample of real program execution that includes stress patterns with *extreme* workload behavior, e.g., max power, max energy, etc. There are two common ways in sampled simulation, statistical sampling (as done in SMARTS [29]) and representative sampling (as done in SimPoint [22]). Our experimental results using the SPEC CPU2000 benchmarks confirm that statistical sampling is generally more accurate than representative sampling for estimating average behavior as shown in prior work [30], however, the new insight provided in this paper is that representative sampling is substantially more effective in identifying stress patterns in typical workloads. The intuitive explanation is that representative sampling uses knowledge about the program structure and execution to find representative sampling units, whereas statistical sampling is largely agnostic to any notion of program structure and execution. Sampling units selected through representative sampling therefore have a higher likelihood of including extreme workload behaviors. In addition, we find that threshold clustering is a better clustering method than k-means clustering (which is commonly used in representative sampling such as SimPoint) for identifying sampling units with extreme workload behavior. The end result is that we can estimate stress patterns in typical workloads with a three orders of magnitude simulation speedup compared to detailed simulation of entire workloads with an error of at most a few percent on average.

In this paper, we make the following contributions:

– We close the gap between sampled simulation focusing on average workload behavior and hand-crafted stressmarks focusing on extreme behavior by identifying stress patterns in typical workloads.
– We make the case that representative sampling is substantially more effective in finding extreme behaviors in microprocessor workloads than statistical sampling, although statistical sampling is (slightly) more effective in capturing average behavior.
– The results in this paper motivate changing current simulation practice. Not only does representative sampling using threshold clustering estimate average performance and power nearly as accurate as statistical sampling, it is substantially more accurate when it comes to estimating stress patterns. And although representative sampling may be more commonly used than statistical sampling in current simulation practice, this paper shows that threshold clustering is substantially more effective than k-means clustering (which is typically being used) for finding stress patterns. In other words, representative sampling with threshold clustering is both effective at estimating average performance as well as stress patterns, whereas prevalent techniques (representative sampling with k-means clustering and statistical sampling) are only effective for estimating average performance.
– We show that the proposed method can be used for finding many different flavors of extreme workload behaviors, such as high cache miss rate, low IPC, or low branch predictability behaviors. These behaviors may be useful for understanding program patterns that lead to these extremities.

We believe this work is timely as power is a primary design concern in today's computer systems, and we are in need for appropriate benchmarking and performance analysis methodologies. In addition, stress patterns will become even more relevant as we enter the multi-core era and the gap between average and peak power widens as the number of cores increases. Benchmarking consortia have also recognized the need for energy- and power-oriented benchmarks and associated benchmarking methodologies. For example, SPEC has developed the SPECpower_ssj2008 benchmark suite [24], which evaluates the performance and power characteristics of volume server class computers. Likewise, EEMBC has released the EnergyBench benchmark suite, which reports energy consumption while running performance benchmarks [18].

## 2 Sampled simulation

In sampled simulation, only a limited number of *sampling units* from a complete benchmark execution are simulated in full detail. We refer to the selected sampling units collectively as the *sample*. Sampled simulation only reports performance for the instructions in the sampling units, and discards the instructions in the pre-sampling units. And this is where the dramatic performance improvement comes from: only the sampling units, which account for only a small fraction of the total dynamic instruction count, are simulated in a cycle-by-cycle manner.

There are three major issues with sampling: (i) what sampling units to select, (ii) how to initialize a sampling unit's architecture starting image, and (iii) how to accurately estimate a sampling unit's microarchitecture starting image. This paper only con-

cerns the first issue because the other two issues can be handled easily by leveraging existing technology. For example, the architecture starting image (registers and memory state) can be set through fastforwarding or through checkpointing [26,28]; and the microarchitecture starting image (caches, branch predictors, etc.) can be estimated with microarchitecture state warmup techniques — there is a wealth of literature covering this area, see for example [5,8,12,19,26,28,29].

There are basically two major ways for determining what sampling units to select, namely (i) statistical sampling, and (ii) representative sampling. We now discuss both approaches.

### 2.1 Statistical Sampling

Statistical sampling takes a number of sampling units across the whole execution of the program. These sampling units are chosen randomly or periodically in an attempt to provide a representative cross-cut of the entire program execution.

Laha et al. [20] propose statistical sampling for evaluating cache performance. They select multiple sampling units by randomly picking intervals of execution.

Conte et al. [5] pioneered the use of statistical sampling in processor simulation. They made a distinction between sampling bias and non-sampling bias. Non-sampling bias results from improperly constructing the microarchitecture starting image prior to each sampling unit. Sampling bias refers to how accurate the sample is with respect to the overall average. Sampling bias is fundamental to the selection of sampling units.

The SMARTS (Sampling Microarchitecture Simulation) approach by Wunderlich et al. [29] proposes *systematic sampling*, which selects sampling units periodically across the entire program execution, i.e., the pre-sampling unit size is fixed, as opposed to random sampling. The potential pitfall of systematic or periodic sampling compared to random sampling is that the sampling units may give a skewed view in case the periodicity present in the program execution under measurement equals the sampling periodicity or its higher harmonics. This does not seem to be a concern in practice though as SMARTS achieves highly accurate performance estimates compared to detailed entire-program simulation. The important asset of statistical sampling compared to representative sampling, is that it builds on well-founded statistics theory, which enables computing confidence bounds at a given confidence level.

### 2.2 Representative Sampling

Representative sampling contrasts with statistical sampling in that it first analyzes the program execution to pick a representative sampling unit for each unique behavior. The most well known representative sampling approach is the SimPoint approach proposed by Sherwood et al. [22]. SimPoint picks a small number of sampling units that accurately create a representation of the complete execution of the program. To do so, they break an entire program execution into intervals — an *interval* is a contiguous sequence of instructions from the dynamic instruction stream — and for each interval they create a code signature. The code signature is a so called Basic Block Vector (BBV) that counts the number of times each basic block is executed in the interval, weighted with the number of instructions per basic block. After normalizing the BBVs

so that the BBV elements sum up to one, they then perform clustering to group intervals with similar code signatures (BBVs) into so called *phases*. BBV similarity is quantified by computing the Manhattan distance between two BBVs. The intuitive notion is that intervals of execution with similar code signatures have similar architectural behavior, and this has been shown to be the case by Lau et al. [21]. Therefore, only one interval from each phase needs to be simulated in order to recreate an accurate picture of the entire program execution. They then choose a representative sampling unit from each phase and perform detailed simulation on that representative unit. Taken together, these sampling units (along with their respective weights) represent the complete execution of a program. A sampling unit is called a *simulation point* in SimPoint terminology, and each simulation point is an interval with on the order of millions, or tens to hundreds of millions of instructions. The simulation points can be used across microarchitectures because the BBVs, based on which the simulation points are identified, are microarchitecture-independent.

The clustering step in the SimPoint approach is a crucial step as it classifies intervals into phases, with each phase representing distinct program behavior. There exist a number of clustering algorithms; here, we discuss k-means clustering (which is used by SimPoint) and threshold clustering (which we advocate in this paper for identifying stress patterns in typical workloads).

*K-means clustering.* K-means clustering produces exactly $k$ clusters and works as follows. Initially, $k$ cluster centers are randomly chosen. In each iteration, the distance is calculated for each interval to the center of each cluster, and the interval is assigned to its closest cluster. Subsequently, new cluster centers are computed based on the new cluster memberships. This algorithm is iterated until no more changes are observed in the cluster memberships. It is well known that the result of k-means clustering is dependent on the choice of the initial cluster centers. Therefore, SimPoint considers multiple randomly chosen cluster centers and uses the Bayesian Information Criterion (BIC) [22] to assess the quality of the clustering: the clustering with the highest BIC score is selected.

*Threshold clustering.* Classifying intervals into phases using threshold clustering can be done in two ways, using an iterative algorithm or using a non-iterative algorithm. The iterative algorithm selects an instruction interval as a cluster center and then computes the distance to all the other instruction intervals. If the distance measure is smaller than a given threshold $\theta$, the instruction interval is considered to be part of that cluster. Out of all remaining instruction intervals (not part of previously formed clusters), another interval is selected randomly as a cluster center and the above process is repeated. This iterative process continues until all instruction intervals are assigned to a cluster/phase. The $\theta$ threshold is expressed as a percentage of the maximum possible Manhattan distance between two intervals; the maximum Manhattan distance between two intervals is 2 assuming normalized BBVs, i.e., the sum across all BBV elements equals one.

The non-iterative algorithm scans all intervals from the beginning until the end of the dynamic instruction stream. If the interval is further away from any previously seen cluster center than a given threshold $\theta$, the interval is considered the center of a new cluster. If not, the interval is assigned to the closest cluster. The non-iterative algorithm

is computationally more efficient and performs well for our purpose — we therefore use the non-iterative approach in this paper.

The important advantage of threshold clustering is that, by construction, it builds phases for which its in-phase variability (in terms of BBV behavior) is limited to a threshold $\theta$. This is not the case for k-means clustering: the variability within a phase can vary across phases.

## 3 Experimental setup

### 3.1 Benchmarks and simulators

We use the SPEC CPU2000 benchmarks and all of their reference inputs in our experimental setup. These benchmarks were compiled and optimized for the Alpha ISA; the binaries were taken from the SimpleScalar website; all benchmarks are run to completion.

We use the SimpleScalar/Alpha v3.0 [3] superscalar out-of-order processor simulator. The processor model is configured along the lines of a typical four-wide superscalar microprocessor such as the Alpha EV7 (21364). Power is estimated using Wattch v1.02 [2] and HotLeakage [23] assuming a 70nm technology, 5.6GHz clock frequency and 1V supply voltage. We assume an aggressive clock gating mechanism.

### 3.2 Sampled simulation

For statistical sampling, we use periodic sampling, as done in SMARTS [29], i.e., we select a sampling unit every $n$ intervals. We will vary the sampling rate $1/n$ in the results presented in this paper.

For representative sampling, we use SimPoint v3.0 with its default settings. In short, SimPoint computes a BBV per interval, and subsequently performs k-means clustering on randomly projected 15-dimensional BBVs; SimPoint evaluates all values of $k$ between 1 and maxK and picks the best $k$ and random seed per $k$ based on the BIC score of the clustering. We will vary the sampling rate by varying the SimPoint maxK parameter. In the evaluation section of this paper, we will compare k-means clustering versus threshold clustering. For doing so, we replace the k-means clustering algorithm with the threshold clustering algorithm while leaving the rest of the SimPoint software untouched.

In this paper, for both statistical and representative sampling, the interval size is set to 1M ($2^{20}$) instructions unless mentioned otherwise, i.e., the stress patterns constitute of 1M dynamically executed instructions. This choice does not affect the general conclusions in this paper though — the methodology can be applied to other interval granularities as well. In fact, we experiment with larger interval sizes — not reported here because of space constraints — and obtain similar results as for the 1M-instruction interval granularity. However, for smaller interval granularities, there may be practical considerations that prohibit the use of representative sampling, the reason being that the clustering algorithm may become very time-consuming for a large number of intervals. Addressing the computational concerns of clustering large data sets is left for future work.

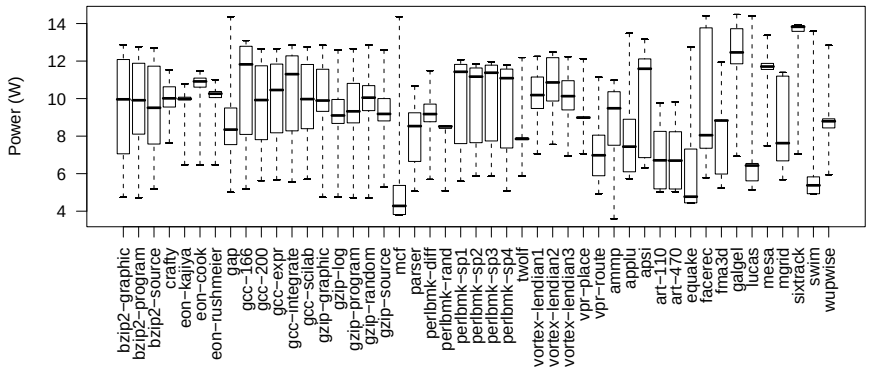


**Fig. 1.** Boxplots characterizing the distribution of power consumption at the 1M-instruction interval granularity; the boxes represent the 5% and 95% quartiles, and the thick horizontal line in each box represents the median.

## 4 Evaluation

In the evaluation section, we now compare statistical sampling against representative sampling for finding stress patterns in microprocessor workloads. This is done in a number of steps: we present per-benchmark max power stress patterns, as well as processor component power stress patterns; we also evaluate the error versus simulation speedup trade-off; and finally, we demonstrate the efficacy of the proposed technique for finding other flavors of extreme behavior, such as max CPI, max cache miss rate and max branch misprediction rate stress patterns.

### 4.1 Motivation

Before evaluating sampled simulation for identifying stress patterns in typical microprocessor workloads, we first further motivate the problem by showing that the variability over time in power consumption is significant within a single benchmark execution. We therefore compute the power consumption on an interval basis, i.e., we compute the power consumption per interval of 1M instructions in the dynamic instruction stream. This yields a distribution of power consumption numbers. Figure 1 represents this distribution as a boxplot per benchmark. The box represents the 5% and 95% quartiles, i.e., 90% of the data lies between these two markers, and thick horizontal line in the box represents the median power consumption across the entire program execution. The outliers are represented by the dashed lines that fall out of the box; the minimum and maximum values are represented by the bottom and top horizontal lines at the ends of the dashed lines, respectively.

The box plots clearly show that there is significant variability over time in power consumption, and, more importantly within the context of this paper, there is a large discrepancy in median versus max power consumption. In fact, for many benchmarks, the max power consumption is substantially higher than its median power consumption,

e.g., for mcf the max power consumption is more than three times as high as its median power consumption. And in addition, the bulk of the power consumption numbers falls far below the max power consumption. This illustrates that finding stress patterns for these benchmarks is challenging, i.e., we need to find one of the few intervals that cause max power consumption out of the numerous intervals that constitute the entire benchmark execution — there are typically tens or even hundreds of thousands of 1M-instruction intervals per benchmark.

### 4.2 Per-benchmark stress patterns

We now evaluate the efficacy of sampled simulation in finding stress patterns at the 1M-instruction interval granularity. For doing so, we assume a $1000\times$ simulation speedup for both statistical and representative sampling compared to the simulation of the entire program execution; we will consider other simulation speedups in Section 4.4. Simulation speedup in this paper is defined as the number of instructions in the entire benchmark execution divided by the number of instructions in the sample. This simulation speedup metric does not include the overhead of setting the architecture and microarchitecture starting images, as discussed in Section 2, however, state-of-the-art sampled simulation methods use checkpointing to initialize a sampling unit's starting image, for which the overhead only depends on the number of sampling units (to a first-order approximation). In other words, comparing sampling strategies in terms of simulation speedup can be done by simply comparing the number of sampling units (intervals) in the sample versus the entire program execution.

We simulate all sampling units selected by statistical and representative sampling, respectively, and retain the max power consumption of any of these sampling units. We then compare this sampled maximum against the max power consumption observed across the entire benchmark execution — this is done by simulating the complete benchmark execution while keeping track of the max power consumption at the 1M-instruction interval size. The percentage difference between the max power values is called the *error*, which is a smaller-is-better metric: the smaller the error score, the closer the stress pattern identified through sampled simulation reflects the real stress pattern observed across the entire benchmark execution. Figure 2 shows the error in estimating the maximum power consumption. We observe that statistical sampling is less effective in finding stress patterns than representative sampling, i.e., the error can be as high as 60% (and average error of 9.3%) for statistical sampling whereas representative sampling is much more effective. Representative sampling with k-means clustering achieves an average error of 3% (and 14% at most); representative sampling with threshold clustering is even more effective with an average error of 2.3% and a maximum error of at most 11%. The reason for the difference in efficacy between statistical sampling and representative sampling is that representative sampling selects sampling units based on the benchmark execution and structure (through the BBVs that are being collected for finding the distinct phase behaviors), whereas statistical sampling is largely agnostic to any notion of program structure and behavior. In other words, for statistical sampling, the likelihood of hitting upon a stress pattern is inverse proportional to the sampling rate, whereas representative sampling identifies distinct program behavior by looking into the code that is being executed.

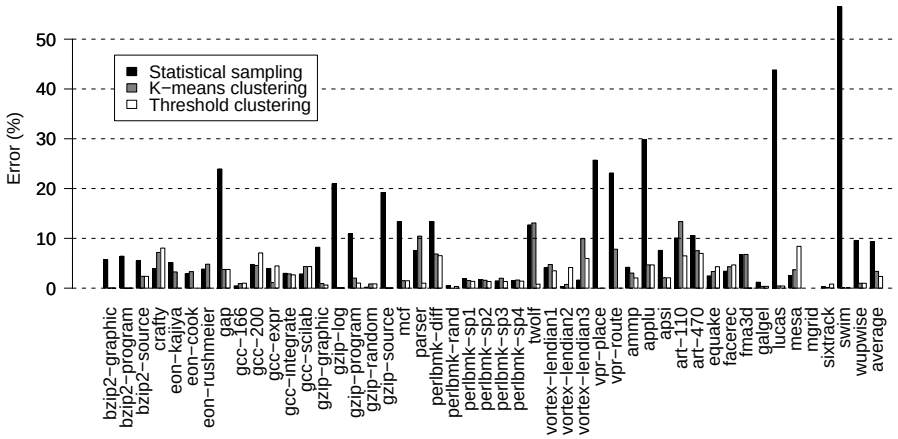


**Fig. 2.** Error in estimating max power stress patterns.

The reason why threshold clustering outperforms k-means clustering is that threshold clustering, by construction, bounds the amount of variability within a cluster, whereas k-means clustering does not. In other words, for a given simulation speedup, i.e., for a given number of clusters, threshold clustering will yield more sparsely populated clusters than k-means clustering; i.e., outliers in the data set will end up in separate clusters in contrast to k-means clustering, which may group those outliers with its closest, albeit relatively far away, cluster.

The end conclusion is that representative sampling with threshold clustering results in a simulation speedup of three orders of magnitude compared to entire benchmark simulation with an error of at most a few percent on average for finding stress patterns in the SPEC CPU2000 benchmarks. And in addition, respresentative sampling with threshold clustering is more effective than representative sampling with k-means clustering and statistical sampling.

### 4.3 Processor component stress patterns

In the previous section, the focus was on stress patterns for the entire processor. We now look into stress patterns for individual processor components, such as the instruction window, functional units, caches, branch predictor, etc. This, in conjunction with a microprocessor floorplan, could provide valuable information in terms of power density and thermal hotspots [23]. Figures 3 and 4 quantify the error in estimating average and maximum per-component power consumption, respectively. (We assume a $1000\times$ simulation speedup and present average results computed across all benchmarks.) The interesting observation from these graphs is that both statistical and representative sampling are very accurate in estimating average processor component power consumption (the average error is around 1% on average), however, representative sampling is by far more effective in capturing stress patterns. For representative sampling with threshold clustering, the processor component power error for the stress patterns is less than 5%, whereas representative sampling with k-means clustering and statistical sampling lead to an processor component power error of up to 10% and 20%, respectively.

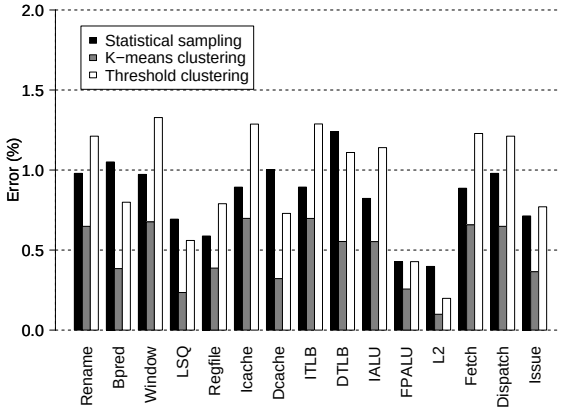


**Fig. 3.** Error in estimating average power consumption per processor component.

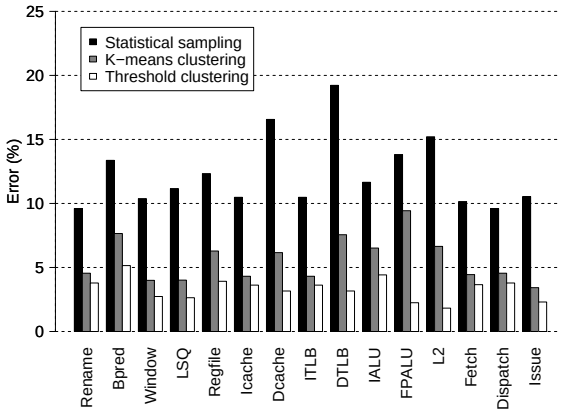


**Fig. 4.** Error in estimating max power consumption per processor component.

### 4.4 Error versus simulation speedup

The previously reported results assumed a simulation speedup of three orders of magnitude ($1000\times$). We now explore the trade-off between error and simulation speedup in more detail, see Figure 5, which shows two graphs, one for estimating average power consumption (left graph) and another one for estimating max power consumption (right graph) — these graphs show average results across all benchmarks. The vertical and horizontal axes show percentage error and simulation speedup with respect to simulating the entire benchmark, respectively. For computing these graphs, we simulate all sampling units; for the left graph, we then compute the average power consumption across all sampling units, and compare it against the true average power consumption computed by simulating the entire benchmark; for the right graph, we retain the largest power consumption number of any of the sampling units and compare it against the

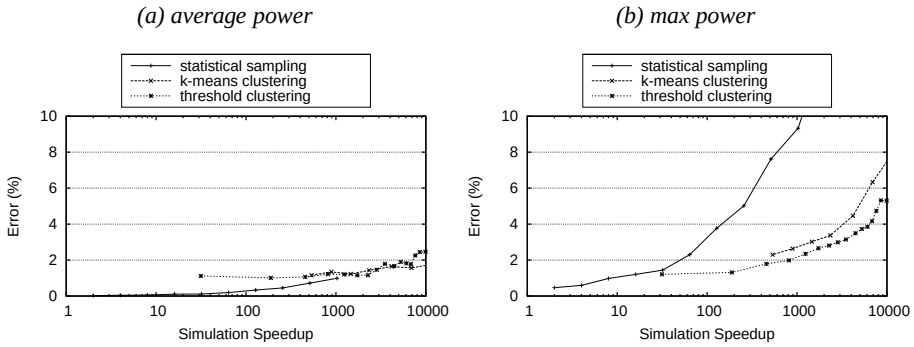


**Fig. 5.** Statistical sampling versus representative sampling: error as a function of simulation speedup for estimating average power consumption (left graph) and max power (right graph).

largest power consumption number observed across the entire program execution. For statistical sampling, one sampling unit is selected every $n$ intervals; this corresponds to a simulation speedup of a factor $n$. For representative sampling, we set a maxK parameter or $\theta$ threshold for the clustering yielding $n$ clusters or sampling units; this corresponds to a $n_{total}/n$ simulation speedup with $n_{total}$ the number of intervals in the entire program execution.

We observe that statistical sampling is more accurate than representative sampling for estimating average power consumption, see left graph Figure 5. The results in the left graph confirm the earlier findings by Yi et al. [30] who provide a detailed comparison of statistical and representative sampling for estimating average performance: they found that average performance is more accurately estimated through statistical sampling, however, representative sampling has a better speed versus accuracy tradeoff.

However, when it comes to estimating max power consumption, representative sampling is more effective, and threshold clustering is the most effective approach. In particular, representative sampling with threshold clustering finds an interval with a power consumption number around 2% on average of the max power number found through simulation of the entire benchmark at a simulation speedup of three orders of magnitude. For the same simulation speedup, statistical sampling achieves an error of 10% on average. Or, reversely, for an error of 2%, statistical sampling only achieves a simulation speedup around a factor of 40. In other words, representative sampling with threshold clustering is both faster and more effective in capturing max power stress patterns.

### 4.5 Other extreme behaviors

Representative sampling with threshold clustering is effective at finding other flavors of extreme behaviors as well, beyond power related stress patterns. Figure 6 shows four examples, namely max CPI, max L1 D-cache miss rate, max L2 cache miss rate and max branch misprediction rate stress patterns. In all four examples, representative sampling with threshold clustering is the most effective approach; this is especially the case for the CPI and cache miss rate extreme behaviors. These extreme behaviors can

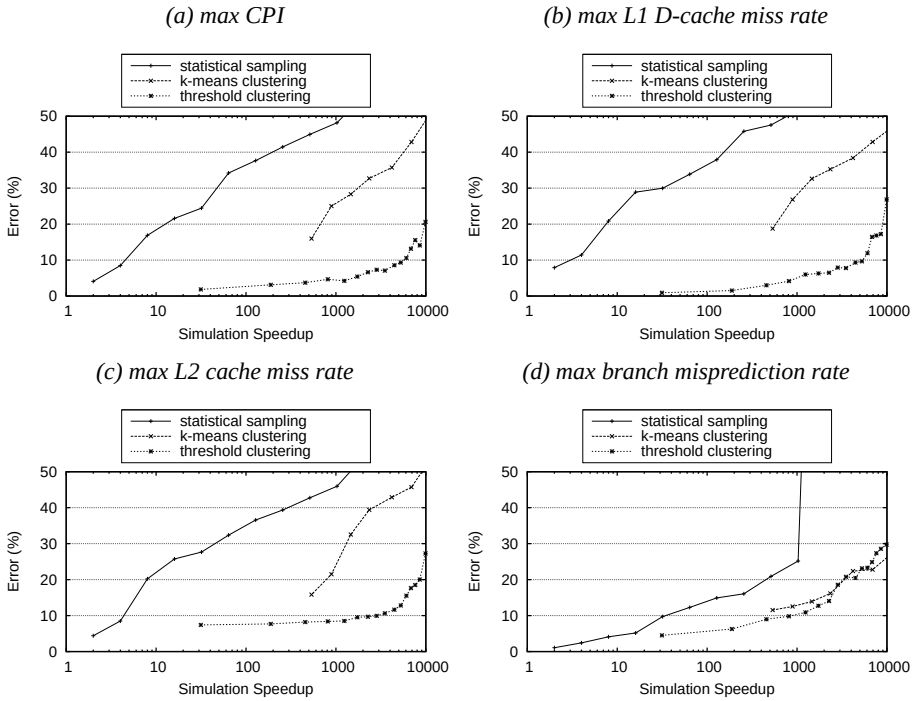


**Fig. 6.** Finding other flavors of stress patterns: max CPI (top left), max L1 D-cache miss rate (top right), max L2 cache miss rate (bottom left), and max branch misprediction rate (bottom right).

provide valuable insight and understanding about problematic program behaviors and patterns.

## 5 Related work

*Stress testing.* In VLSI circuit design, statistically generated test vectors are used to stress a circuit by inducing maximum switching activity [4]. At the microarchitectural level, engineers develop hand-crafted synthetic test cases, so called stressmarks, to estimate maximum power consumption of a microprocessor. This is common practice in industry, see for example [9,10,27]. Recent work by Joshi et al. [17] proposes a framework for automatically developing stressmarks by exploring the workload space using an abstract workload model.

*Power phase characterization.* A lot of work has been done on characterizing time-varying program behavior, and different authors have been proposing different ways for doing so, such as code working sets [6], BBVs [22], procedure calls [13], and performance data [7].

Isci and Martonosi [14] propose a methodology for tracking dynamic power phase behavior in real-life applications using a real hardware setup. They measure total pro-

cessor power consumption data using a digital multimeter and simultaneously collect raw performance counter data. They then use the performance counter data to estimate processor component power consumption numbers, which they subsequently use to identify power phase behavior at runtime using threshold clustering. Whereas the goal of the work by Isci and Martonosi is on tracking power consumption and power phase behavior at runtime, the focus of our work is on finding stress patterns to guide processor design under extreme workload behavior, which is a related but different problem.

In their follow-on work, Isci and Martonosi [15] compare clustering based on BBVs versus processor component power numbers, and found both approaches to be effective, but processor component power numbers to be more accurate for tracking power phase behavior. The downside of processor component power numbers though is that it requires that the entire benchmark be measured in terms of its power behavior, which may be costly in terms of equipment (in case of a real hardware setup) or which may be too time-consuming (in case of a simulation setup). In addition, processor component power numbers are specific to one particular microprocessor implementation. A BBV profile is both inexpensive and fast to measure through software instrumentation, and, in addition, is microarchitecture-independent, i.e., can be used across microarchitectures. Since our goal is to find stress patterns to be used during the design of a processor, we advocate the BBV approach because of its microarchitecture-independence, its low cost and its fast computation.

## 6 Conclusion and future work

Power consumption has emerged as a key design concern over the entire range of computing devices, from embedded systems up to large-scale data centers and supercomputers. Understanding the power characteristics of workloads and their interaction with the architecture however, is not trivial and requires an appropriate benchmarking methodology. Researchers and engineers currently use a range of workloads for gaining insight into the power characteristics of processor architectures. On the one side, typical workloads such as SPEC CPU and other commercial workloads are used to assess average power consumption. On the other side, hand-crafted stressmarks are being used to understand worst-case behavior in terms of a processor's max power consumption. This paper closed the gap between these two ends of the power benchmarking spectrum by finding stress patterns in typical microprocessor workloads.

In this paper, we advocated and studied sampled simulation as a means of finding these stress patterns efficiently. Although sampled simulation is a well studied and mature research area, the objective in this paper is completely different. While the goal of sampled simulation traditionally has been on estimating average performance, the problem addressed in this paper is on estimating worst-case performance rather than average performance, i.e., the goal is to find stress patterns in typical workloads without having to simulate the complete benchmark execution. We found that although statistical sampling is more effective than representative sampling for estimating average behavior, representative sampling is substantially more effective than statistical sampling when it comes to capturing extreme behavior. In addition, we found that threshold clustering is

substantially more effective than k-means clustering for finding stress patterns (which is a frequently used clustering technique for representative sampling). Our experimental results using the SPEC CPU2000 benchmarks demonstrate that stress patterns at a million-instruction granularity can be found with an error of a few percent on average at a simulation speedup of three orders of magnitude.

We believe that this work could lead to a new line of research towards finding stress patterns in microprocessor workloads. Sampled simulation, which was traditionally used for estimating average behavior, may benefit from specific enhancements towards stress pattern identification. One focus of future research may be to improve the computational requirements of the clustering algorithm in representative sampling so that larger data sets and thus smaller granularity stress patterns may become feasible in practice. One example of a stress pattern that requires a small granularity is a dI/dt stress pattern: stress patterns with large power swings over short periods of time are of interest for studying the dI/dt problem [16] as the associated current swings may lead to ripples on the voltage supply lines, which may introduce timing errors and/or cause circuits to fail. Existing clustering algorithms however are too time-consuming when applied to a large data set.

## Acknowledgements

We would like to thank the anonymous reviewers for their valuable comments. Lieven Eeckhout is a postdoctoral fellow with the Fund for Scientific Research in Flanders (Belgium) (FWO-Vlaanderen). Additional support is provided by the FWO projects G.0160.02 and G.0255.08.

## References

1. D. Brooks and M. Martonosi. Dynamic thermal management for high-performance microprocessors. In *HPCA*, pages 171–182, Jan. 2001.
2. D. Brooks, V. Tiwari, and M. Martonosi. Wattch: A framework for architectural-level power analysis and optimizations. In *ISCA*, pages 83–94, June 2000.
3. D. C. Burger and T. M. Austin. The SimpleScalar Tool Set. Computer Architecture News, 1997. See also `http://www.simplescalar.com` for more information.
4. T. Chou and K. Roy. Accurate power estimation of CMOS sequential circuits. *IEEE Transaction on VLSI Systems*, 4(3):369–380, Sept. 1996.
5. T. M. Conte, M. A. Hirsch, and K. N. Menezes. Reducing state loss for effective trace sampling of superscalar processors. In *ICCD*, pages 468–477, Oct. 1996.
6. A. Dhodapkar and J. E. Smith. Managing multi-configuration hardware via dynamic working set analysis. In *ISCA*, pages 233–244, May 2002.
7. E. Duesterwald, C. Cascaval, and S. Dwarkadas. Characterizing and predicting program behavior and its variability. In *PACT*, pages 220–231, Oct. 2003.
8. L. Eeckhout, Y. Luo, K. De Bosschere, and L. K. John. BLRL: Accurate and efficient warmup for sampled processor simulation. *The Computer Journal*, 48(4):451–459, May 2005.
9. W. Felter and T. Keller. Power measurement on the Apple Power Mac G5. Technical Report RC23276, IBM, 2004.

10. M. K. Gowan, L. L. Biro, and D. B. Jackson. Power considerations in the design of the Alpha 21264 microprocessor. In *Proceedings of the 35th Design Automation Conference (DAC)*, pages 726–731, June 1998.
11. S. H. Gunther, F. Binns, D. M. Carmean, and J. C. Hall. Managing the impact of increasing microprocessor power consumption. *Intel Journal of Technology*, 5(1), Feb. 2001.
12. J. W. Haskins Jr. and K. Skadron. Accelerated warmup for sampled microarchitecture simulation. *ACM Transactions on Architecture and Code Optimization (TACO)*, 2(1):78–108, Mar. 2005.
13. M. Huang, J. Renau, and J. Torrellas. Positional adaptation of processors: Application to energy reduction. In *ISCA*, pages 157–168, June 2003.
14. C. Isci and M. Martonosi. Runtime power monitoring in high-end processors: Methodology and empirical data. In *MICRO*, pages 93–104, Dec. 2003.
15. C. Isci and M. Martonosi. Phase characterization for power: Evaluating control-flow-based and event-counter-based techniques. In *HPCA*, pages 122–133, Feb. 2006.
16. R. Joseph, D. Brooks, and M. Martonosi. Control techniques to eliminate voltage emergencies in high performance processors. In *HPCA*, pages 79–90, Feb. 2003.
17. A. M. Joshi, L. Eeckhout, L. K. John, and C. Isen. Performance cloning: A technique for disseminating proprietary applications as benchmarks. In *HPCA*, pages 229–239, Feb. 2008.
18. D. Kanter. EEMBC energizes benchmarking. *Microprocessor Report*, July 2006.
19. S. Kluyskens and L. Eeckhout. Branch history matching: Branch predictor warmup for sampled simulation. In *HiPEAC*, pages 153–167, Jan. 2007.
20. S. Laha, J. H. Patel, and R. K. Iyer. Accurate low-cost methods for performance evaluation of cache memory systems. *IEEE Transactions on Computers*, 37(11):1325–1336, Nov. 1988.
21. J. Lau, J. Sampson, E. Perelman, G. Hamerly, and B. Calder. The strong correlation between code signatures and performance. In *ISPASS*, pages 236–247, Mar. 2005.
22. T. Sherwood, E. Perelman, G. Hamerly, and B. Calder. Automatically characterizing large scale program behavior. In *ASPLOS*, pages 45–57, Oct. 2002.
23. K. Skadron, M. R. Stan, W. Huang, S. Velusamy, K. Sankaranarayanan, and D. Tarjan. Temperature-aware microarchitecture. In *ISCA*, pages 2–13, June 2003.
24. SPEC. Specpower_ssj2008. `http://www.spec.org/power_ssj2008/`.
25. V. Tiwari, D. Singh, S. Rajgopal, G. Mehta, R. Patel, and F. Baez. Reducing power in high-performance microprocessors. In *DAC*, pages 732–737, June 1998.
26. M. Van Biesbrouck, L. Eeckhout, and B. Calder. Efficient sampling startup for sampled processor simulation. In *HiPEAC*, pages 47–67, Nov. 2005.
27. R. Vishmanath, V. Wakharkar, A. Watwe, and V. Lebonheur. Thermal performance challenges from silicon to systems. *Intel Technology Journal*, 4(3), Aug. 2000.
28. T. F. Wenisch, R. E. Wunderlich, B. Falsafi, and J. C. Hoe. Simulation sampling with livepoints. In *ISPASS*, pages 2–12, Mar. 2006.
29. R. E. Wunderlich, T. F. Wenisch, B. Falsafi, and J. C. Hoe. SMARTS: Accelerating microarchitecture simulation via rigorous statistical sampling. In *ISCA*, pages 84–95, June 2003.
30. J. J. Yi, S. V. Kodakara, R. Sendag, D. J. Lilja, and D. M. Hawkins. Characterizing and comparing prevailing simulation techniques. In *HPCA*, pages 266–277, Feb. 2005.