Early Design Phase Power/Performance Modeling through Statistical Simulation

Lieven Eeckhout* Koen De Bosschere
Department of Electronics and Information Systems (ELIS), Ghent University, Belgium
E-mail: {leeckhou,kdb}@elis.rug.ac.be

Abstract

Microprocessor design time and effort are getting impractical due to the huge number of simulations that need to be done to evaluate various processor configurations for various workloads. An early design stage methodology could be useful to efficiently cull huge design spaces to identify regions of interest to be further explored using more accurate simulations. In such an early design stage methodology, power consumption should be considered besides performance, since power consumption is becoming a key design issue for midrange and high-end microprocessor designs. In this paper, we propose to use statistical simulation as an early design stage methodology that considers both performance and power. We evaluate the applicability and the accuracy of this methodology and we show that statistical simulation is indeed capable of identifying a region of energy-efficient architectures. In addition, we demonstrate that this methodology can be used to explore workload design spaces in terms of power/performance by varying program characteristics that are hard to vary using real programs.

1 Introduction

The efforts to design future microprocessors are ever increasing as microprocessor designs and workloads are becoming more and more complex. This is reflected in the huge number of time-consuming simulations that need to be run in order to obtain a cost-effective and high-performance design. Indeed, numerous processor configurations need to be evaluated for various workloads, which results in a huge number of simulations to be done. Note that each of these simulations takes several hours for completion (a typical example: 12 hours for simulating one second of a real system). Exploring huge design spaces even through high-level architectural simulations is impractical if not impossible given time-to-market considerations. So, there is a need for methods to efficiently cull huge design spaces in early design stages. These early design stage methods should identify regions of interest that could be further explored in more detail by more accurate architectural simulations.

In addition to the need for early design stage methodologies for modeling performance, power consumption is becoming a key design issue when designing midrange and high-end microprocessors. This is due to the increased packaging and cooling cost for more complex designs. Note that it is important that power consumption is considered early in the design cycle so that computer engineers are not confronted with unexpected power consumptions in a late design phase. Recent work [3, 4, 7, 15] addresses this issue by integrating power modeling techniques in architectural simulators. In other words, power models of various processor structures are combined with counters that measure the activity at the architectural level of each of these structures to calculate the total power consumption of a microprocessor. Unfortunately, because of the fact that these power modeling methodologies are based on detailed architectural simulation, they equally suffer from long simulation times.

In this paper, we address this issue by presenting a methodology that incorporates power modeling one step earlier in the design cycle which allows us to efficiently identify a region of interest with desirable power/performance characteristics. The methodology presented in this paper is based on statistical simulation [8, 9, 12, 13]. In statistical simulation, a statistical profile or a set of statistical program characteristics is extracted from a program execution, e.g., the instruction mix, the distribution of the dependency distance between instructions, etc. This statistical profile is then used to generate a synthetic trace that is subsequently fed into a trace-driven simulator. In traditional statistical simulation methodologies, the trace-driven simulator only yields performance characteristics. In this paper, we have integrated an architectural power model, namely Wattch [4], into our trace-driven simulator so that both power and performance characteristics are be-

*Lieven Eeckhout is supported by a grant from the Flemish Institute for the Promotion of the Scientific-Technological Research in the Industry (IWT).

ing measured. We evaluate the accuracy of this approach by comparing power and performance estimates using real and synthetic traces and we find that statistical simulation is indeed capable of identifying a region of energy-efficient architectures. Note that identifying such a region can be done efficiently since statistical simulation is a fast simulation technique due to the fact that performance characteristics quickly converge. In addition, we show that statistical simulation allows computer designers to investigate the influence and the interaction of several program characteristics that are hard to vary using real programs, such as branch prediction accuracy and cache miss rate, on both performance and energy consumption per cycle.

Hsieh and Pedram [11] presented a comparable technique to estimate performance and power dissipation of a microprocessor. However, the statistical profile presented in [11] includes characteristics, such as pipeline stall rate and IPC, that make architecture and workload design space explorations impossible. The methodology presented in this paper on the other hand, makes architecture and workload design space explorations viable as is clearly demonstrated in this paper.

This paper is organized as follows. In section 2, we present the statistical simulation methodology. The basics of architectural-level power modeling and the integration of Wattch [4] in our statistical simulation tool are detailed in section 3. Section 4 discusses the benchmarks, the out-of-order architecture model and the metrics used. In section 5, we evaluate the usefulness of the statistical simulation methodology for power/performance modeling. In section 6, we investigate the relation and the interaction between program characteristics—such as cache miss rate and branch misprediction rate—versus performance and energy consumption per cycle. This is done using the statistical simulation methodology. Finally, we end with some conclusions.

2 Statistical Simulation

The statistical simulation methodology [9] works in three steps: statistical profiling, synthetic trace generation and trace-driven simulation.

2.1 Statistical Profiling

During the statistical profiling step, a real program trace is analyzed by a *microarchitecture-dependent* profiling tool and a *microarchitecture-independent* profiling tool. The complete set of statistics collected during statistical profiling is called a *statistical profile*. The microarchitecture-independent profiling tool extracts statistics concerning the instruction mix (we identify 19 instruction classes according to their semantics and the number of source registers), the age of the input register instances (the number of dynamic instructions between writing and reading a register instance; measured per instruction class and per source register; 22 distributions in total) and the age of memory instances.

The microarchitecture-dependent profiling tool extracts statistics concerning the branch and cache behaviour of the program trace for a specific branch predictor and a specific cache organization. The *branch statistics* are branch prediction accuracies per branch class (conditional, indirect, etc.). The *cache statistics* include two sets of distributions: the D-cache and the I-cache statistics. The D-cache statistics contain two probabilities for a load operation, namely the probability that a load needs to access the L2 cache (L1 cache miss) and main memory (L2 cache miss) to get its data; idem for store operations and the I-cache statistics.

A statistical profile can be computed from an actual trace but it is more convenient to compute it on-the-fly from either an instrumented functional simulator, or from an instrumented version of the benchmark program running on a real system which eliminates the need to store huge traces. A second note is that although computing a statistical profile might take a long time, it should be done only once for each benchmark with a given input. And since statistical simulation is a fast analysis technique, computing a statistical profile will be worthwhile. A third important note is that making a distinction between microarchitecture-dependent and -independent characteristics implies that statistical simulation cannot be used to study branch predictors or cache organizations; other microarchitectural parameters however, like window size, issue width, instruction latency, etc. can be varied freely.

2.2 Synthetic Trace Generation

Once a statistical profile is computed, a *synthetic trace* is generated by a synthetic trace generator using this statistical profile. This is done à la Monte Carlo: a random number is generated, that will determine a program characteristic using a cumulative distribution function.

The generation of a synthetic trace itself works on an operation-by-operation basis. Consider the generation of operation x in the synthetic instruction stream:

1. Determine the instruction type using the instruction-mix distribution. An approach for generating clusters of instruction types is given in [9].
2. For each source operand, determine the operation that creates this register instance using the age of register instances distribution. Notice that when a dependency is created in this step, we cannot assure that the operation that is the creator of that register instance, is neither a store nor a conditional branch operation[1]. The

[1]Relative jumps, indirect jumps and returns do not have destination

demand of syntactical correctness does not allow us for assigning a destination operand to a store and a conditional branch instruction. This problem is solved as follows: look for another creator instruction until we get one that is not a store nor a conditional branch. If after a certain maximum number of trials still no dependency is found that is not supposedly created by a store or a conditional branch instruction, the dependency is simply squashed.

3. If instruction x is a load operation, use the age of memory instances distribution to determine whether a store operation (before instruction x in the trace) accesses the same memory address.
4. If instruction x is a branch instruction, determine whether the branch and its target will be correctly predicted using the branch statistics.
5. If instruction x is a load or a store operation, determine whether the load or store will cause a L1 cache hit/miss or L2 cache hit/miss using the D-cache statistics.
6. Determine whether or not instruction x will cause an I-cache hit/miss at the L1 or L2 level.

2.3 Trace-driven simulation

The last phase of the statictical simulation method is the trace-driven simulation of the synthetic trace which yields IPC (number of instructions retired per cycle). In order to model resource contention due to instructions along misspeculated control flow paths, we apply the following strategy when a branch instruction was marked as misspeculated by the synthetic trace generator: inject instructions into the simulator (using our synthetic trace generator) and mark them coming from a misspeculated execution path. When the misspeculated branch is executed, the instructions of the misspeculated path are squashed and instructions are fetched from the right control flow path.

Simulating statistical cache behaviour is done as follows. In case of an I-cache miss, the processor stops fetching new instructions as long as the I-cache is unresolved. In case of a D-cache miss, the load that causes an L1 or an L2 D-cache miss will get an L2 D-cache access time or a main memory access time assigned, respectively.

2.4 Simulation Speed

Our experiments showed that statistical simulation is a fast simulation technique due to the statistical nature of the technique. The standard deviation on the power/performance characteristics is less than 1% after simulating 1 million synthetically generated instructions which leads to small margins of error on the data produced.

operands either. However, we will not mention them for the remainder of this paper although we take this into account.

3 Power Modeling

Several architectural-level power estimation models have been proposed in last few years, e.g., Wattch [4], SimplePower [15], PowerTimer [3] and TEM^{2}P^{2}EST [7]. In this study, we used Wattch as the power estimation model because of its public availability and its flexibility for architectural design space explorations; the power models included in Wattch are fully parameterizable which provides its high flexibility. Wattch also provides good *relative* accuracy which is required for doing architectural design space explorations. In addition, the power models in Wattch closely resemble the microarchitecture modeled in our simulator which is important for the validity of the results, according to the analysis done in [10].

Wattch [4] calculates the dynamic power consumption P of a processor unit (e.g., functional unit, I-cache, D-cache, register file, clock distribution, etc.) using the following formula: $P = CV^2af$, where C, V, a and f are the capacitance, the voltage, the activity factor ($0 < a < 1$) and the frequency, respectively. V and f are technology dependent; in this study, we assume $V = 2.5$V, $f = 600$MHz and a .35μm technology. Wattch estimates the capacitance C based on circuit and transistor sizing models. The activity factor a measures how often clock ticks lead to switching activity on average. In this study, we assumed a base activity factor of $1/2$ modeling random switching activity, which we believe is a reasonable approximation in an early design stage[2]. The activity factor a can be further lowered by clock-gating unneeded units. In this study, we assumed an aggressive conditional clocking scheme in which the power estimate is linearly scaled from the maximum power consumption with port or unit usage; unused units dissipate 10% of their maximum power. Measuring the unit usage is done by inserting so called *activity counters* in the simulator that keep track of the number of accesses per clock cycle to the various units.

For this study, we have integrated Wattch in our trace-driven simulator by inserting the Wattch activity counters in our simulator. The architecture modeled is based on an instruction window that serves as both reorder buffer and instruction queue, as is done with the register update unit (RUU) design of SimpleScalar's `sim-outorder` [5] on which Wattch is built. We also assume 3 extra pipeline stages between the fetch and the issue stage, as is done in Wattch. In addition, our simulator also models a load/store queue that is used in our case for dynamic memory disambiguation. Ghiasi and Grunwald [10] evaluated two architectural power models and concluded that the architecture

[2] An alternative approach would be to use a distribution of the data values produced in a program execution for generating synthetic data values in the synthetic trace. These synthetically generated data values could then be used to measure the switching activity in various structures.

instruction latencies	integer (1), load (3), multiply (8), FP (4) divide (18/31); fully pipelined except divide
branch predictor	8-bit gshare (4KB), 4KB bi-modal, 4KB meta 4-way 512-sets BTB, RAS 8 entries
caches	32KB DM L1 I$, 64KB 2WSA L1 D$, 256KB 4WSA L2; 10/80 cycles to L2/mem

Table 1. Out-of-order architecture.

modeled in the simulator should be the same as the one used by the power model to produce reliable results. The above discussion shows that this is the case in this study.

4 Methodology

4.1 Benchmarks

In this study we used the IBS benchmark suite [14] to evaluate statistical simulation for power/performance modeling. The IBS traces were generated on a MIPS-based DEC 3100 system running the Mach 3.0 operating system. The IBS traces are known to have a larger instruction footprint (due to the inclusion of significant amounts of operating system activity) and to stress the memory subsystem more than SPECint benchmarks do [14].

4.2 Out-of-order architecture

To validate our performance modeling methodology, we assumed wide-issue out-of-order superscalar architectures. The instruction window size is varied from 32 to 256 instructions; the issue width is varied from 4 to 12. The fetch bandwidth and the reorder bandwidth were chosen to be the same as the issue bandwidth. Dynamic memory disambiguation and selective re-execution to recover from misspeculated loads are modeled in our simulator. More details on the architectures simulated can be found in Table 1.

4.3 Metrics

To evaluate the usefulness of statistical simulation for power/performance modeling in an early design stage, we use several metrics. First, we measure the *IPC prediction error* which is the relative error between the IPC by simulating the real trace and the IPC by simulating a synthetic trace that was generated using the statistical profile of the corresponding real trace. A positive error means an IPC overestimation. Analogously, we measure the *power prediction error* which is the relative error between the average energy consumption per cycle of the real trace and its synthetic 'clone' trace. Second, we use two power/performance metrics that are useful for evaluating the power-performance efficiency of midrange and high-end microprocessor designs [1], namely $CPI^2 \times energy/cycle$ and $CPI^3 \times energy/cycle$. The $CPI^2 \times energy/cycle$ is also called the energy-delay product (EDP) and gives more emphasis on performance than the $CPI \times energy/cycle$ metric. The $CPI^3 \times energy/cycle$ or the ED^2P metric gives an even greater emphasis on performance which makes this metric useful for high-end microprocessors [3].

5 Evaluation

Figures 1 and 2 show the performance (on the left) and the power prediction errors (on the right) for the various benchmarks and for various architectural configurations. In Figure 1, the window size is fixed to 128 instructions and the issue width is varied; in Figure 2, various window sizes are considered for a fixed 8-issue width machine. Positive errors imply an IPC or power overestimation while negative errors imply underestimations. These graphs show that the IPC prediction error is generally smaller than 15% and the power prediction error is generally smaller than 10%, which is acceptable for an early design stage methodology. The IPC prediction errors are generally overestimations. This is due to the fact that long critical paths in computer programs are badly modeled through average dependencies leading to many short critical paths during synthetic trace generation.

Figures 3 and 4 show the two power/performance metrics, namely the EDP and ED^2P metrics, for both the real trace (labeled as 'real' metric) and the synthetic trace (labeled as 'estimated' metric). Figure 3 shows the EDP and the ED^2P as a function of the issue width for an out-of-order architecture with a 128-entry window. Figure 4 shows the EDP and the ED^2P as a function of the window size for an 8-issue machine. In both figures, the estimated metrics are smaller than the real metrics. This is due to the IPC overestimations, see Figures 1 and 2. Figures 3 and 4 show that statistical simulation is capable of accurately predicting the energy-efficiency trend. An architecture is called to be energy-efficient when the EDP or the ED^2P metric is minimal in its design point. For example, the 8-issue architecture with an instruction window of 48 to 96 instructions is identified (by simulating real traces) to be the most energy-efficient architecture when considering the ED^2P metric, see the graph on the right of Figure 4. Statistical simulation, identifies an instruction window of 48 to 128 instructions to be energy-efficient. In conclusion, these graphs show that statistical simulation is capable of identifying a region of energy-efficient organizations. The region of interest identified through statistical simulation can then be further analyzed through more accurate trace-driven or execution-driven simulation techniques based on real benchmarks.

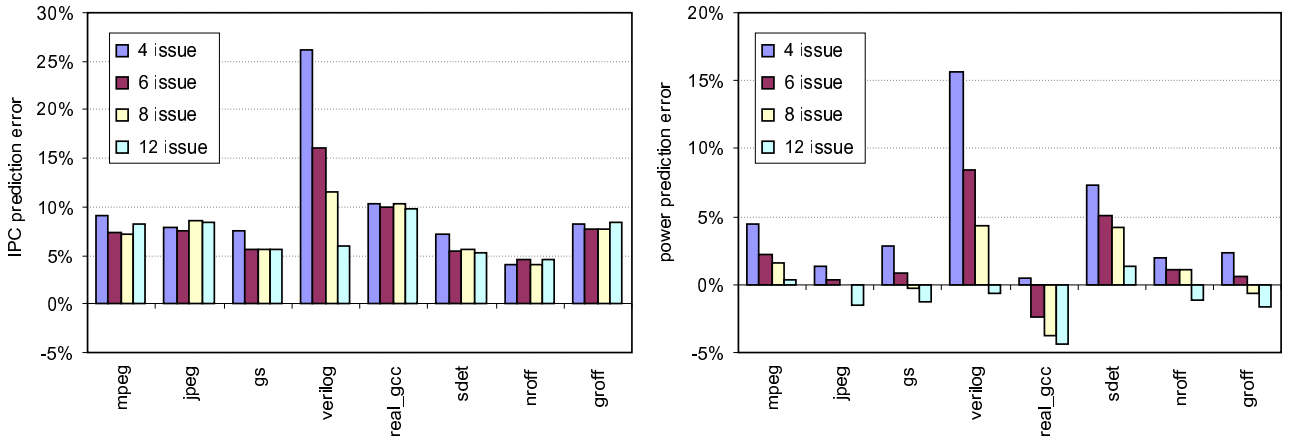

Figure 1. IPC and power prediction error for a processor with a 128-entry instruction window.

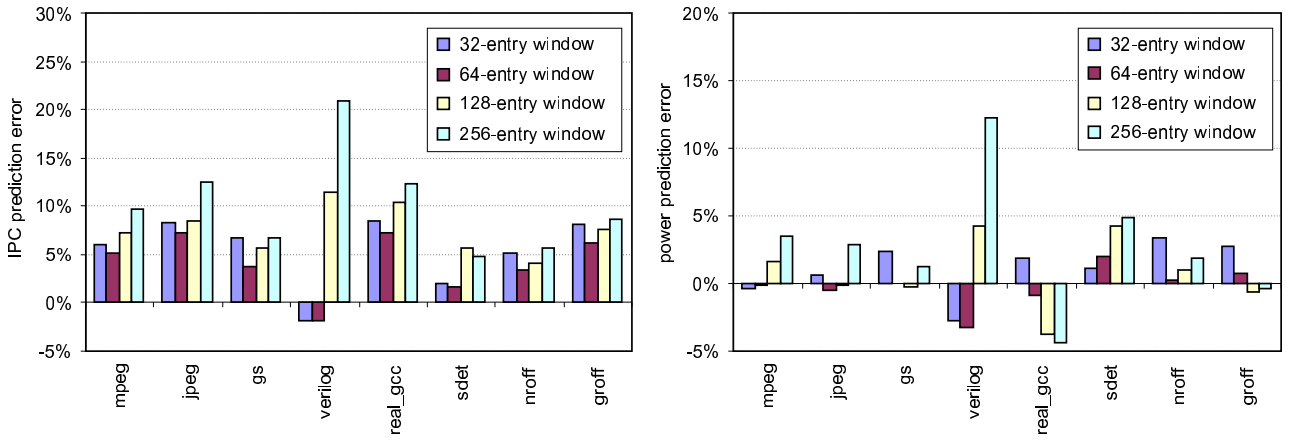

Figure 2. IPC and power prediction error for an 8-issue processor.

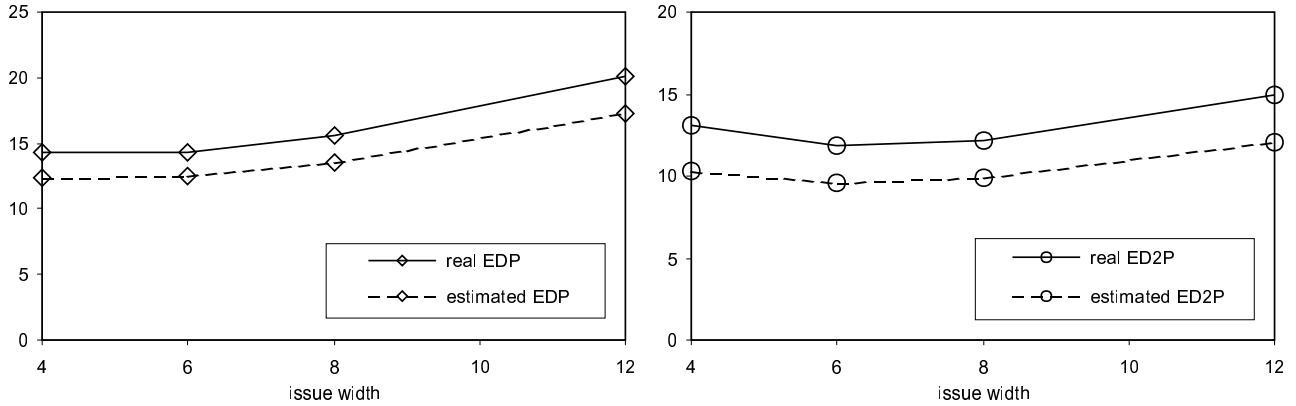

Figure 3. Real and estimated EDP (left) and ED^{2}P (right) metrics as a function of the issue width in a 128-entry window machine.

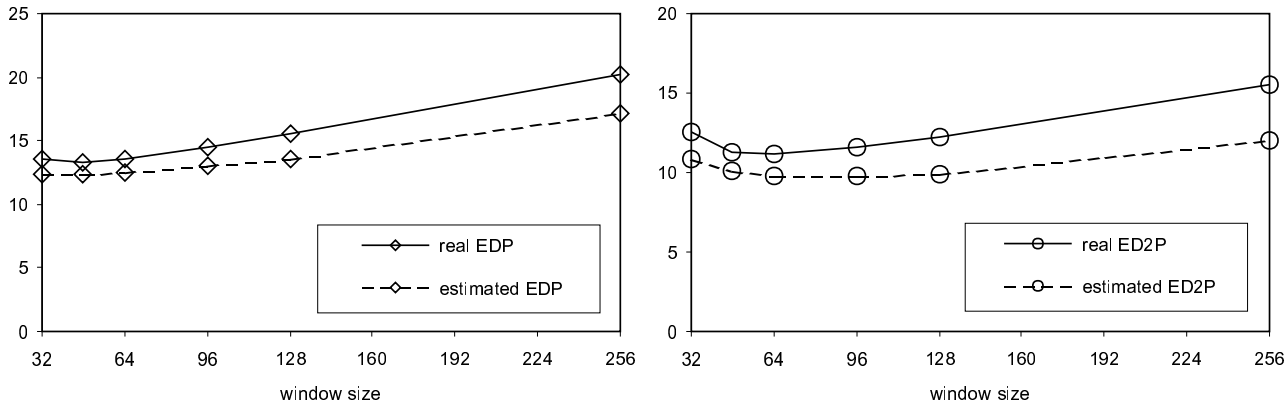

Figure 4. Real and estimated EDP (left) and ED^{2}P (right) metrics as a function of the window size in an 8-issue machine.

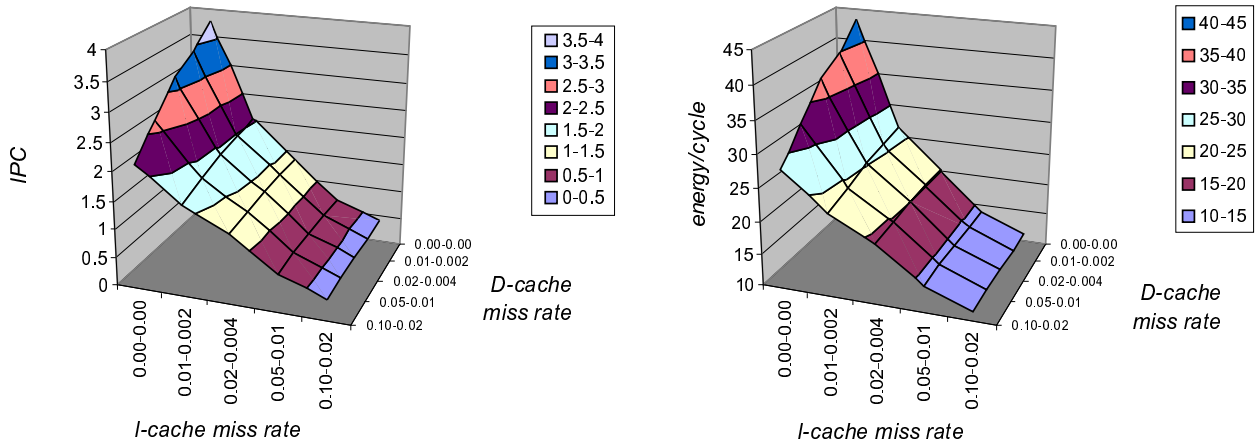

Figure 5. IPC (on the left) and energy consumption per cycle (on the right) as a function of I- and D-cache miss rate for a 6-issue architecture with a 128-entry window. The branch prediction accuracy was set to 95%. Note that '0.02-0.004' means a 2% L1 miss rate and a 0.4% global L2 miss rate.

6 Energy behaviour vs. program characteristics

Note that statistical simulation is maybe not the most suitable tool for exploring the most energy-efficient branch predictor or memory hierarchy. This is due to the distinction made between microarchitecture-dependent and microarchitecture-independent characteristics, see section 2. To evaluate two different memory hierarchies for example, cache miss rates have to be computed by simulating the real trace for both hierarchies. In this case, full blown architectural simulations using real benchmarks (simulating the architecture and the memory hierarchy simultaneously), although slower than simulating a memory hierarchy and subsequently doing a statistical simulation, are likely to be more appropriate due to their higher accuracy.

On the other hand, statistical simulation seems to be more suitable for investigating the energy consumption *per cycle* and its interaction with program characteristics. This addresses the question where in the execution of a computer program most energy is consumed. This is an important issue for techniques such as dynamic thermal management [2] and dynamic adaptation of hardware resources [6]. These techniques search at reducing the energy consumption per cycle (e.g., to control the temperature of the chip) without compromising performance too much. Statistical simulation combined with power modeling gives an excellent opportunity to investigate this relation since program characteristics can be varied freely and independently.

Figure 5 explores IPC and energy per cycle as a function of I- and D-cache miss rate for a 6-issue architecture with a 128-entry window. The branch prediction accuracy was set to 95%—the graphs for other branch prediction accuracies are very similar. We did these experiments because it

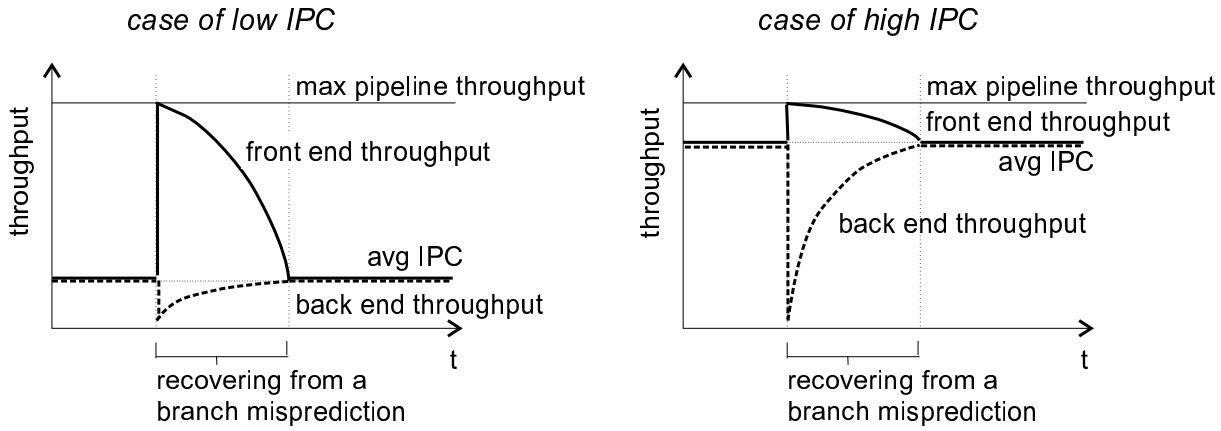

Figure 7. Front and back end activity on a branch misprediction.

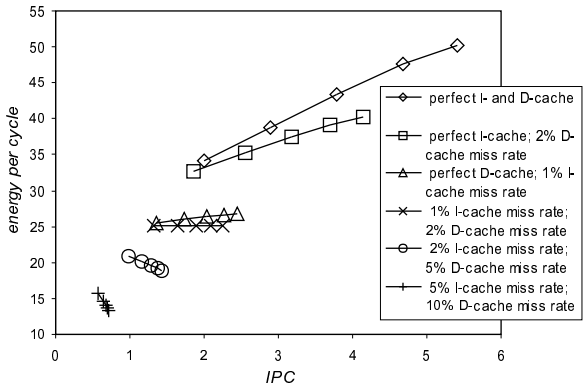

Figure 6. IPC and energy per cycle as a function of branch prediction accuracy for a 6-issue architecture with a 128-entry window. The different points on each line correspond to the branch prediction accuracy: 80%, 90%, 95%, 98% and 100% from left to right, respectively.

is unclear in advance how energy per cycle will vary with cache miss rate. On the one hand, instruction throughput will be lower with an increasing cache miss rate. Thus, we might expect an energy per cycle reduction with increasing cache miss rates. On the other hand, a higher cache miss rate results in more accesses to higher levels in the memory hierarchy[3] which might lead to higher energy consumptions since larger memory structures consume more energy per access than smaller memory structures. Therefore, it is unclear if higher cache miss rates will result in a higher or lower energy consumption per cycle. The data of Figure 5 reveal that generally energy consumption per cycle decreases with increasing miss rates. For low levels of ILP (or high I-cache miss rates) however, the energy per cycle slightly increases (not visible on the graph) with increasing D-cache miss rates. From these experiments, we can conclude that the impact on the energy consumption per cycle of the reduction in pipeline throughput is more significant than the increase in activity per cycle in the memory hierarchy with increasing cache miss rates. Note again that this conclusion is only true as far as the on-chip energy consumption is concerned since (off-chip) memory power consumption is not modeled in Wattch.

In the next set of experiments, we focus on the influence of branch mispredictions on the energy consumption per cycle. Figure 6 quantifies the impact of branch prediction accuracy and cache miss rates on IPC and energy per cycle. Several configurations were considered with varying cache miss rates, see the legend of Figure 6. The different points on each line correspond to different branch prediction accuracies: 80%, 90%, 95%, 98% and 100% from left to right, respectively. This figure reveals an interesting result, namely that energy per cycle is positively correlated with branch prediction accuracy for low cache miss rates. For high cache miss rates on the other hand, energy per cycle is negatively correlated with branch prediction accuracy. This can be explained as follows, see Figure 7. When a branch misprediction occurs, a significant part of the instruction window needs to be squashed and refilled with instructions from the correct control flow path. While refilling the instruction window, the fetch activity is increased compared to the steady state situation without branch mispredictions. The instruction execution activity on the other hand is decreased since fewer instructions reside in the instruction window. As a result, there is an increased activity in the front end of the pipeline and a decreased activity in the back end while recovering from a branch misprediction. In the case of high cache miss rates (and thus low average IPC), see Figure 7 on the left, the increase in activity in the front end is larger than the decrease in the back end. As a

[3]Note that only on-chip memories, i.e., L1 and L2 caches, are modeled in Wattch; energy consumption due to main memory accesses is not considered here.

net result, the energy consumption per cycle will be larger while recovering from a branch misprediction. So, the energy consumption per cycle increases with an increasing number of branch mispredictions. In other words, energy consumption per cycle decreases with higher branch prediction accuracies, see Figure 6. In case of low cache miss rates (and thus high average IPC) on the other hand, see Figure 7 on the right, the increase in activity in the front end is smaller than the decrease in the back end with a net decrease as result. So, the energy consumption per cycle increases with higher branch prediction accuracies, see Figure 6.

Next to providing insights in the power/performance behaviour of computer programs, this kind of experiments could be useful for identifying new microarchitectural techniques that reduce the energy per cycle consumption without sacrifying performance. For example, see Figure 6, increasing the branch prediction accuracy in a program with high average cache miss rates leads to an IPC increase and an energy per cycle decrease.

7 Conclusion

In current microprocessor designs, detailed performance evaluation and (only recently) power modeling are done through architectural simulations. Unfortunately, architectural simulations have an important drawback that they are impractical due to the huge number of simulations that need to be done for evaluating various processor configurations for various workloads. In this paper, we propose to incorporate power modeling in the statistical simulation methodology, a fast simulation technique previously only considered for performance evaluation. This methodology allows computer designers to efficiently identify a region of interest with desirable power/performance characteristics in an early design stage. This region can then be further analyzed through more accurate simulations using real benchmarks. In this paper, we have evaluated the accuracy and the applicability of this methodology and we have found that statistical simulation is indeed capable of identifying a region of energy-efficient architectures. In addition, we have investigated the interaction between program characteristics and power/performance using statistical simulation.

References

[1] D. Brooks, P. Bose, and et al. Power-aware microarchitecture: Design and modeling challenges for next-generation microprocessors. *IEEE Micro*, 20(6):26–44, November/December 2000.

[2] D. Brooks and M. Martonosi. Dynamic thermal management for high-performance microprocessors. In *Proceedings of the Seventh International Symposium on High-Performance Computer Architecture (HPCA-7)*, pages 171–182, Jan. 2001.

[3] D. Brooks, M. Martonosi, J.-D. Wellman, and P. Bose. Power-performance modeling and tradeoff analysis for a high end microprocessor. In *Proceedings of the Power-Aware Computer Systems (PACS'00) held in conjunction with ASPLOS-IX*, Nov. 2000.

[4] D. Brooks, V. Tiwari, and M. Martonosi. Wattch: A framework for architectural-level power analysis and optimizations. In *Proceedings of the 27th Annual International Symposium on Computer Architecture (ISCA-27)*, pages 83–94, June 2000.

[5] D. C. Burger and T. M. Austin. The SimpleScalar Tool Set. Computer Architecture News, 1997. Version 2.0.

[6] A. Buyuktosunoglu, S. Schuster, D. Brooks, P. Bose, P. Cook, and D. Albonesi. An adaptive issue queue for reduced power at high performance. In *Proceedings of the Power-Aware Computer Systems (PACS'00) held in conjunction with ASPLOS-IX*, Nov. 2000.

[7] A. Dhodapkar, C. H. Lim, G. Cai, and W. R. Daasch. TEM^{2}P^{2}EST: A thermal enabled multi-model power/performance estimator. In *Proceedings of the Power-Aware Computer Systems (PACS'00) held in conjunction with ASPLOS-IX*, Nov. 2000.

[8] L. Eeckhout and K. De Bosschere. Hybrid analytical-statistical modeling for efficiently exploring architecture and workload design spaces. In *Proceedings of the 2001 International Conference on Parallel Architectures and Compilation Techniques (PACT-2001)*, pages 25–34, Sept. 2001.

[9] L. Eeckhout, K. De Bosschere, and H. Neefs. Performance analysis through synthetic trace generation. In *The IEEE International Symposium on Performance Analysis of Systems and Software (ISPASS-2000)*, pages 1–6, Apr. 2000.

[10] S. Ghiasi and D. Grunwald. A comparison of two architectural power models. In *Proceedings of the Power-Aware Computer Systems (PACS'00) held in conjunction with ASPLOS-IX*, Nov. 2000.

[11] C. Hsieh and M. Pedram. Micro-processor power estimation using profile-driven program synthesis. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, 17(11):1080–1089, Nov. 1998.

[12] S. Nussbaum and J. E. Smith. Modeling superscalar processors via statistical simulation. In *Proceedings of the 2001 International Conference on Parallel Architectures and Compilation Techniques (PACT-2001)*, pages 15–24, Sept. 2001.

[13] M. Oskin, F. T. Chong, and M. Farrens. HLS: Combining statistical and symbolic simulation to guide microprocessor design. In *Proceedings of the 27th Annual International Symposium on Computer Architecture (ISCA-27)*, pages 71–82, June 2000.

[14] R. Uhlig, D. Nagle, T. Mudge, S. Sechrest, and J. Emer. Instruction fetching: Coping with code bloat. In *Proceedings of the 22nd Annual International Symposium on Computer Architecture (ISCA-22)*, pages 345–356, June 1995.

[15] N. Vijaykrishnan, M. Kandemir, M. J. Irwin, H. S. Kim, and W. Ye. Energy-driven integrated hardware-software optimizations using SimplePower. In *Proceedings of the 27th Annual International Symposium on Computer Architecture (ISCA-27)*, pages 95–106, June 2000.