

How Sensitive is Processor Customization to the Workload’s Input Datasets?

Maximilien Breughe[†] Zheng Li[‡] Yang Chen[‡] Stijn Eyerman[†]
Olivier Temam[‡] Chengyong Wu[‡] Lieven Eeckhout[†]

[†]Ghent University, Belgium

[‡]INRIA, France

[‡]ICT, Beijing, China

Abstract—Hardware customization is an effective approach for meeting application performance requirements while achieving high levels of energy efficiency. Application-specific processors achieve high performance at low energy by tailoring their designs towards a specific workload, i.e., an application or application domain of interest. A fundamental question that has remained unanswered so far though is to what extent processor customization is sensitive to the training workload’s input datasets. Current practice is to consider a single or only a few input datasets per workload during the processor design cycle — the reason being that simulation is prohibitively time-consuming which excludes considering a large number of datasets.

This paper addresses this fundamental question, for the first time. In order to perform the large number of runs required to address this question in a reasonable amount of time, we first propose a mechanistic analytical model, built from first principles, that is accurate within 3.6% on average across a broad design space. The analytical model is at least 4 orders of magnitude faster than detailed cycle-accurate simulation for design space exploration. Using the model, we are able to study the sensitivity of a workload’s input dataset on the optimum customized processor architecture. Considering MiBench benchmarks and 1000 datasets per benchmark, we conclude that processor customization is largely dataset-insensitive. This has an important implication in practice: a single or only a few datasets are sufficient for determining the optimum processor architecture when designing application-specific processors.

I. INTRODUCTION

Energy efficiency is a primary design goal in the embedded space. Embedded and mobile devices are typically battery operated, and hence battery lifetime is a key design goal. Along with this quest for improved energy efficiency comes the trend of ever more complex and diverse applications.

Hardware specialization, e.g., ASICs, is an effective approach for bridging the gap between ever more complex applications and the quest for improved energy efficiency. Although ASIC design typically yields the optimum performance-energy trade-off, it is very costly and hence it is only viable when shipped in large volumes. A more cost-effective approach is to consider a programmable processor that is optimized for the application of interest, a so-called application-specific processor. This specialized processor is optimized to achieve the best possible performance within a given energy envelope, or, vice versa, the processor is optimized to consume the least possible energy while achieving a given performance target;

e.g., for a soft real-time application this means reducing energy consumption while meeting most of the deadlines. Companies such as Tensilica (LX3), MIPS Technologies (CorExtend) and ARC (600 and 700) provide solutions along this line.

In order for a customized application-specific processor to be effective, its design methodology is obviously key. It needs the ability to identify the sweet spot in a huge design space that optimizes the processor along multiple criteria, such as performance, energy, and cost, among others. An important question that has remained unanswered though, to the best of our knowledge, relates to how the design methodology accounts for different input datasets. Typically, the design process of the customized processor involves the application of interest along with one or a couple input training datasets. The question that arises is whether a specialized processor that is optimized for a single (or a few) dataset(s) is going to yield an optimum trade-off when considering other datasets that potentially are (very) different from the ones considered during the design process.

The goal of this paper is to answer this fundamental question. Although posing the question is easy, answering it is not, for two reasons. First, it requires considering a large number of data sets; we consider 1000 data sets for each program in this study. Second, current practice of cycle-level simulation is too slow to answer this question in a reasonable amount of time. For example, for the design space (1024 design points) and the 1000 datasets that we consider in our setup, exhaustive enumeration would amount to more than 400 years of simulation.

In this paper, we consider analytical modeling to alleviate the simulation wall, which enables us to answer the above fundamental question for the first time. The mechanistic analytical model is built based on first principles and predicts performance and energy consumption within 3.6% on average for a pipelined processor across a set of 10 MiBench benchmarks. The model predicts performance and energy consumption for a large number of processor configurations from a single profiling run, which reduces the overall evaluation time by 4 orders of magnitude for our setup. By running a single profiling run for each of the 1000 datasets from KDataSets [2], we derive the optimum application/dataset-specific processor configuration. We conclude that the configuration of an application-specific processor is largely dataset-insensitive. In other words,

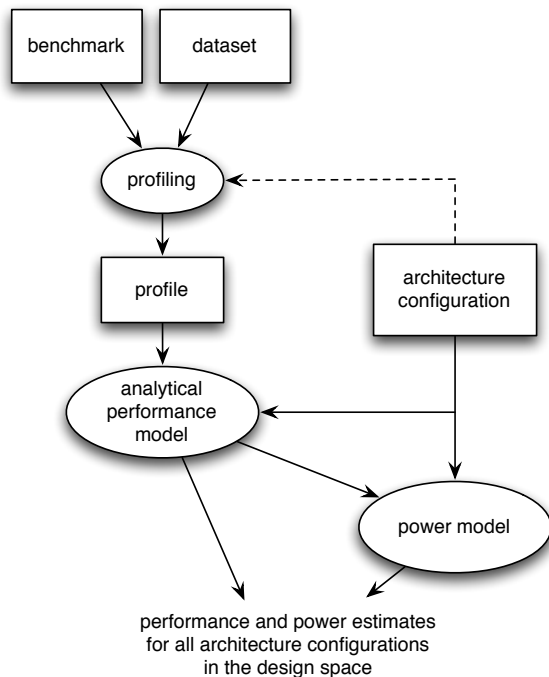


Fig. 1. Overall design space exploration framework.

picking a single or a few training datasets during design space exploration yields an optimum (or close to optimum) processor configuration.

This result has important practical consequences. It implies that building application-specific processors does not require a vast number of datasets; a single or limited number of datasets is sufficient to identify the *optimum* architecture configuration. Common intuition is that datasets would have a significant impact on the optimum processor architecture though. Consider for example how the size of a dataset is likely to affect the optimum cache and processor configuration. Our results contradict this intuition: the optimum processor architecture is largely dataset insensitive.

The remainder of this paper is organized as follows. We first describe our design space exploration approach leveraging analytical modeling (Section II). We subsequently describe our experimental setup (Section III). We then evaluate the sensitivity of customized processors to datasets (Section IV). Finally, we discuss related work (Section V) and conclude (Section VI).

II. DESIGN SPACE EXPLORATION

A. Framework

Figure 1 illustrates the overall framework for design space exploration. We run a single profiling for a given application and a given dataset. The profiling step is very fast as it essentially involves functional simulation, also referred to as instruction-set simulation. This profiling step also involves single-pass multi-cache simulation [15] which enables simulating multiple caches in a single run by exploiting the inclusion property of the LRU stack. We also simulate a selected number of branch predictors we are interested in during design space

exploration. This profiling run needs to be done only once per application and per dataset. It is relatively fast compared to detailed cycle-accurate simulation. Our profiler runs at a speed of 2.1 MIPS compared to 47 KIPS for detailed processor simulation using the M5 simulation infrastructure [1] (see later for a more detailed outline of our experimental setup).

The profile serves as input to the analytical performance model, along with the definition of the processor design space. The model then predicts performance for all the architecture configurations in the design space. This is done almost instantaneously because the analytical model involves computing a small number of equations only. The performance estimates along with the architecture configuration definition and event counts such as number of dynamically executed instructions, number of cache misses and branch mispredictions, etc., serve as input to the power model which then estimates power consumption for all microarchitectures in the design space. The performance model is a mechanistic analytical model, which we describe in the next section in more detail; the power model is based on McPAT [14], a recently proposed power model.

When put together, exploring the design space using the proposed framework is several orders of magnitude faster than detailed cycle-accurate timing simulation. For our setup, which involves a design space of 1024 design points, the proposed framework incurs an overall simulation time reduction of 4 orders of magnitude. The key reason for this huge saving is that profiling needs to be done only once while the analytical model predicts performance for all possible configurations in the design space. In contrast, the traditional approach on the other hand incurs detailed cycle-accurate simulation for every design point.

B. Performance modeling

The mechanistic analytical performance model is a central piece in our framework. The model is built on the previously proposed interval analysis [5], [9]. Whereas prior work in interval analysis focused on out-of-order processors, in this paper we extend interval analysis towards in-order processors. Counterintuitively, in-order processor performance is more complicated to model than out-of-order processor performance. The reason is that out-of-order processor are designed to hide instruction execution latencies and dependencies which means that these phenomena are not so important from a modeling perspective. An in-order processor on the other hand, cannot hide these phenomena, and hence instruction execution latencies and dependencies immediately translate in a performance impact. As a result, these phenomena require appropriate modeling in in-order processors.

Figure 2 illustrates how pipelined processor performance is viewed through interval analysis. The key observation in interval analysis is that the rate at which instructions flow through a processor pipeline is intermittently disrupted by miss events and hazards. Whereas out-of-order processor performance modeling focuses on miss events only — because hazards are largely hidden through out-of-order execution — in-order processor performance modeling also requires adequate

| Parameter | Description |
|----------------------------|--|
| N | number of dynamically executed insns |
| m_{Icache} | number of I-cache misses |
| m_{Dcache} | number of D-cache misses |
| ℓ_{MEM} | memory access time in cycles |
| N_{br}^T | number of taken branches |
| m_{br} | number of branch mispredictions |
| d | pipeline depth between fetch and execute |
| $N_{ld\ deps}$ | number of insns that depend on previous load |
| $N_{ld/st}$ | number of loads and stores |
| $\ell_{Dcache\ hit\ time}$ | D-cache hit time |

TABLE I
MODEL PARAMETERS.

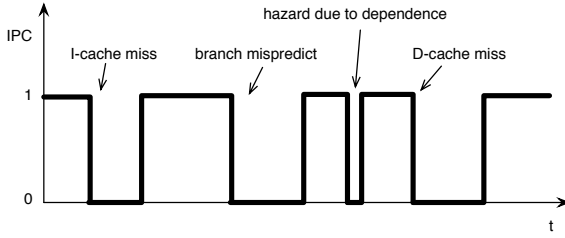


Fig. 2. Interval analysis.

modeling of the performance impact of hazards. Assuming a scalar pipelined architecture, instructions flow through the pipeline at a rate of one instruction per cycle. This smooth flow of instructions is intermittently disrupted by miss events (cache misses, TLB misses and branch mispredictions) and hazards (pipeline stalls due to dependencies). The analytical model estimates performance by counting the number of instructions executed and by keeping track of the miss events and hazards, i.e., a performance penalty is accounted for each miss event and hazard. We consider the following miss events and hazards:

- **Cache misses and TLB misses.** The penalty incurred by cache misses and TLB misses — at the instruction side as well as at the data side — equals their miss latency and is independent of the depth of the pipeline.
- **Branches and branch mispredictions.** Taken branches incur a one-cycle penalty (bubble) on architectures with the branch predictor access happening in the decode stage and not the fetch stage. Mispredicted branches incur a penalty that depends on pipeline depth and equals the number of cycles needed to refill the front-end pipeline, i.e., the number of pipeline stages between fetch and execute.
- **Hazards due to dependencies.** Assuming there is forwarding logic to enable back-to-back execution of dependent instructions, dependencies typically do not incur a performance penalty. However, an instruction that depends on a load instruction may incur a hazard, i.e., an instruction that depends on a load needs to be stalled in the pipeline until the load returns its data value. The penalty depends on pipeline depth and is a function of the data cache hit time.

Put together, the analytical model estimates performance

using the following formula:

$$T = N + m_{Icache} \cdot \ell_{MEM} + m_{Dcache} \cdot \ell_{MEM} + N_{br}^T + m_{br} \cdot d + N_{ld\ deps} + N_{ld/st} \cdot (\ell_{Dcache\ hit\ time} - 1). \quad (1)$$

See Table I for a description of the various symbols. To simplify the model’s formula, we omitted the terms that relate to TLB misses and BTB misses, because these are modeled in a similar way as the cache misses and branch mispredictions, respectively.

Note that some model parameters are a function of the microarchitecture only, such as memory access time, pipeline depth, and cache hit time. Other parameters are a function of the workload only, such as the number of dynamically executed instructions, the number of loads and stores, the number of taken branches and the number of instructions that depend on a previous load instruction. Hence, these parameters need to be measured only once for each workload. The remaining parameters are a function of both the workload and the processor architecture, such as the number of I-cache misses, D-cache misses and branch mispredictions. The fact that these parameters depend on both the workload and the microarchitecture, implies that we need to measure them for every combination of workload and microarchitecture. However, as mentioned before, we leverage a single-pass algorithm to collect miss rates for a wide range of cache configurations in a single profiling run.

The model captures first-order effects only, which assumes that miss events and hazards happen in isolation. Second-order effects, which happen as a result of miss event and hazard overlaps, are not captured by the model. An example second-order effect is when an instruction cache miss (partially) overlaps with a data cache miss or a hazard due to a data dependency. Second-order effects cause the penalty to be either partially or completely hidden. For example, a pipeline hazard due to a data dependency may be hidden underneath an instruction cache miss, and hence the model should not account for it. The reason why the model does not capture second-order effects is that we aim for a simple enough model because modeling second-order effects would require substantially more complicated profiling. Our current profiling approach evaluates each component (instruction cache, data cache, branch predictor, etc.) in isolation; modeling second-order effects would require evaluating all possible combinations of these components which would lead to an explosion in the number of evaluations and would quickly become intractable. As will be shown in the evaluation section, we found the first-order approach to be accurate enough for our purposes.

III. EXPERIMENTAL SETUP

A. Workloads

We use 10 benchmarks from the MiBench benchmark suite [6]. MiBench is suite of embedded benchmarks from different application domains, including automotive/industrial, consumer, office, network, security, and telecom. We limit ourselves to 10 benchmarks in total in order to limit simulation time during performance model validation while covering the

| Benchmark | Category |
|--------------|-----------------------|
| qsort | automotive/industrial |
| jpeg_c | consumer |
| jpeg_d | consumer |
| stringsearch | office |
| patricia | network |
| dijkstra | network |
| sha | security |
| gsm_c | telecom |
| adpcm_c | telecom |
| adpcm_d | telecom |

TABLE II
BENCHMARKS.

| Parameter | Range |
|------------------|--|
| I-cache | 8KB vs 32KB 2 vs 8 way set-assoc 32 vs 64 byte blocks |
| D-cache | 8KB vs 32KB 2 vs 8 way set-assoc 32 vs 64 byte blocks |
| pipeline depth | 5 – 7 – 9 – 11 stages 99 – 266 – 433 – 600 MHz |
| branch predictor | 2KB hybrid 10b local and 10b global history 5.5KB hybrid 10b local and 13b global history 7.6KB hybrid 13b local and 13b global history always-taken static branch prediction |

TABLE III
ARCHITECTURE DESIGN SPACE.

above application domains, see also Table II. For each of these benchmarks, we consider 1000 datasets from KDataSets [2]. These datasets were taken from various sources and represent a wide range of inputs. Moreover, as shown by Chen et al. [2], these 1000 datasets exhibit widely diverse behavior in terms of their cache/TLB behavior, branch behavior, IPC, etc. In particular, the data cache miss rates vary broadly across datasets; for instance, the miss rate of *dijkstra* varies by 4 orders of magnitude across datasets, with a cache miss rate up to 10% for some data sets.

B. Simulators

We use the M5 simulation framework [1]. We derive our profiler from M5’s functional simulator, and we validate our model against detailed cycle-accurate simulation with M5. We use McPAT [14] for our power estimates; we assume a 32nm chip technology.

C. Architecture design space

The architecture design space is depicted in Table III, which is loosely inspired by the Loongson/Godson processor [7], a 32-bit MIPS-compatible embedded core. We vary the size and the configuration of the caches; we vary the depth of the pipeline; and we consider different branch predictors. There are 1024 possible configurations in the design space that we consider in this study. The reason for limiting the number of design points is that we need to compute power numbers for each architecture configuration — there are 1024 configurations — and for each workload — there are 1000 datasets, for a total of one million runs for each of the 10

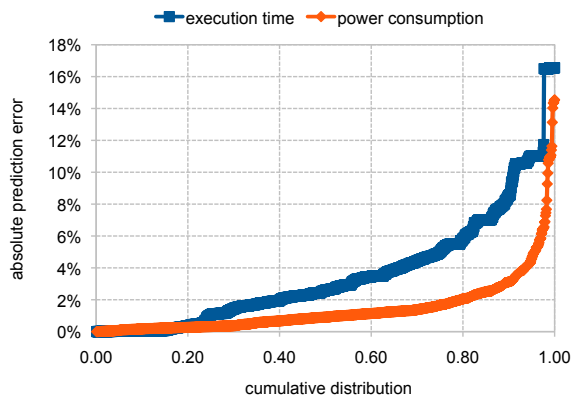


Fig. 3. Evaluating the model’s performance prediction accuracy: cumulative distribution of the absolute prediction error across the 10 MiBench benchmarks and 100 random architecture design points.

benchmarks. Although running McPAT is fairly fast — 4 seconds per run — this amounts to more than 470 compute-days, which reaches the limits of what we could possibly do given the infrastructure that we have.

The design space that we consider in this paper only varies parameters that relate to the processor’s microarchitecture; we do not vary the instruction-set architecture. However, some companies such as Tensilica offer customizable processors in which the instruction set can be tailored towards the application (domain) of interest. The design space by such application-specific instruction-set processor is even more complex than the one considered in this paper. The design space is not only larger, it is also involves incorporating the compiler into the optimization loop. In this paper we limit ourselves to evaluating dataset sensitivity for microarchitecture customization.

IV. EVALUATION

The evaluation is done in three major steps. We first validate the analytical performance model which is at the core of our methodology. We subsequently evaluate how sensitive processor customization is with respect to the workload’s input dataset.

A. Model accuracy

We first evaluate the performance model accuracy. We consider 100 random points across the design space that we explore, and compare detailed cycle-accurate simulation results against the model. Figure 3 shows the cumulative distribution of the performance (execution time) and power prediction error (vertical axis) across these 100 design points and the 10 benchmarks. This graph illustrates the accuracy of the model: 90% of the design points have a prediction error that is smaller than 8.5% and 3% for performance and power, respectively. The maximum error equals 16.5% and 14.5% with an average error of 3.6% and 1.5% for performance and power, respectively. Figure 4 shows the predicted and measured CPI numbers for a specific processor configuration with 32KB caches (8-way set-assoc and 64 byte line size), 7.6KB hybrid branch

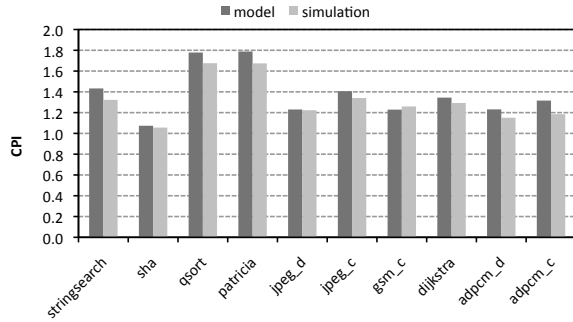


Fig. 4. Predicted and measured CPI numbers for the model versus detailed cycle-accurate simulation for a processor configuration with 32KB caches (8-way set-associative, 64 byte blocks), 7.6KB hybrid branch predictor and a 433MHz clock frequency.

predictor and a 433MHz clock frequency. We observe both positive and negative errors, ranging from -2.4% (gsm_c) to 11% (adpcm_c); the average absolute error equals 5.3% for this configuration. Overestimations are due to not modeling second-order effects for miss events (i.e., overlapping miss events are not modeled); underestimations are due to the approximate modeling of hazards (i.e., dependencies between subsequent instructions are modeled, however, dependencies among non-subsequent instructions may incur a hazard which is not modeled).

Although the average error is relatively small, it would be instructive to understand what the sources of error are. We therefore employ a Plackett and Burman design of experiment [17] in which we systematically vary the processor configuration along different dimensions in the design space. The Plackett and Burman design of experiment involves a small number of simulations — substantially fewer simulations than simulating all possible combinations of microarchitecture parameters — while still capturing the effects of each parameter and selected interactions. The Plackett-Burman design of experiment requires the simulation of extreme architectures with microarchitectural parameters set slightly outside the design space, and provides a way to determine the most significant sources of error. This analysis reveals that the foremost important source of error is pipeline depth. The reason is that as the pipeline gets deeper more second-order overlap effects take place which is not captured by the model.

B. Design space exploration

Although the model incurs some inaccuracies and although it does not model second-order effects, the model is sufficiently accurate for driving design space explorations. The key point is that in spite of its absolute error, the model is accurate enough for deriving relative performance/power differences between architecture configurations. To illustrate this, we consider 100 random design points, and we determine the performance and energy consumed for each of these points through both the analytical model and detailed cycle-accurate simulation. We then compute the Spearman rank correlation coefficient between the model versus simulation: this coefficient assesses how well the relationship between

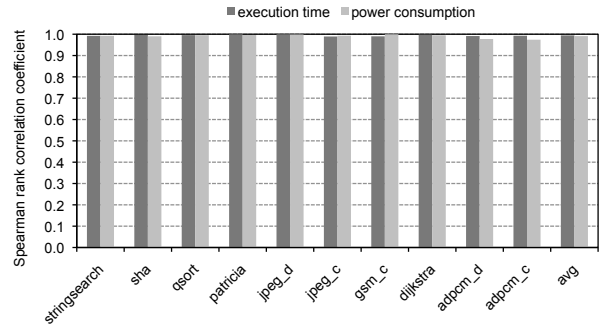


Fig. 5. Rank correlation coefficients for the model versus simulation across 100 random architecture design points.

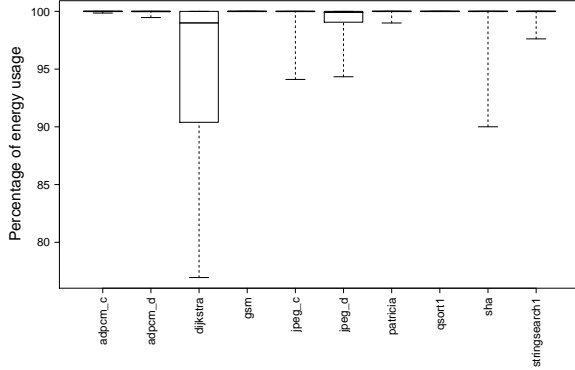
the model and simulation can be described by a monotonic function. In other words, it quantifies how well the model can predict the relative ranking of processor configurations relative to simulation. A correlation coefficient of ‘1’ implies a perfect monotonic function. Figure 5 reports the rank correlation coefficient is larger than 0.97 for power and larger than 0.98 for performance for all benchmarks. We conclude that the model is able to rank architecture configurations accurately compared to detailed cycle-accurate simulation.

C. Dataset sensitivity

Now that we have validated our framework for design space exploration purposes, we use it to study how sensitive processor customization is with respect to the training workload’s input datasets. We consider two design scenarios. The first scenario aims at finding the most energy-efficient processor architecture that achieves performance within 10% of the best possible performance observed across the design space. Figure 6(a) shows the delta in energy consumption if we determine the most energy-efficient processor for a single dataset versus for all datasets — the graph reports box plots across the 1000 datasets. The second scenario is dual to the first one and aims at finding the best performing processor architecture within 10% of the least energy consuming processor, see Figure 6(b). The key insight from these graphs is that the delta in energy and performance is relatively small if a processor is customized using a single dataset compared to all datasets. In fact, for most datasets and workloads, the delta is negligible; for a few workloads and datasets, the delta is higher but still limited to no more than 10% (except for one dataset for dijkstra for which the delta equals 33%).

This result suggests that customizing a processor architecture using one dataset leads to an optimum architecture configuration that it is likely to yield (close to) optimum performance/power for other datasets. In other words, one dataset is predictive for another dataset. We verify this insight through the following experiment. We rank all the processor configurations for a single dataset, and compare that ranking against the ranking obtained for the other datasets; we quantify how well the rankings match using the Spearman rank correlation coefficient. We repeat this for each dataset. Figure 7 shows boxplots of the rank correlation coefficients: the middle line in the box shows the median, the box boundaries show

(a) Determining most energy-efficient processor



(b) Determining best performing processor

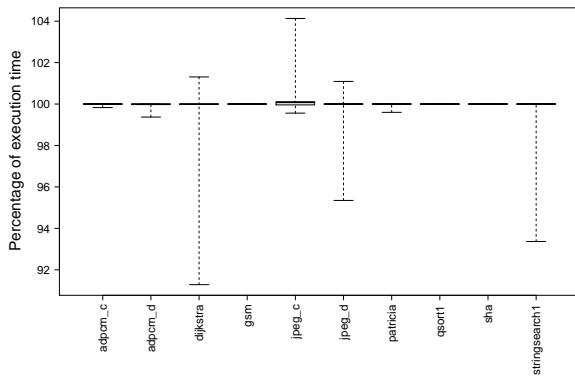


Fig. 6. Boxplots showing the delta in energy and performance for a processor customized using a single dataset versus all datasets: (a) the most energy-efficient processor within 10% of the best possible performance, and (b) the best performing processor within 10% of the most energy-efficient processor.

the first and third quartiles, and the outer lines represent the minimum and maximum observed across the 1000 datasets. The correlation coefficient is always larger than 0.92; for most data sets it is larger than 0.98. In conclusion, a dataset is predictive for other datasets, and hence, a single or a few datasets are sufficient for determining the optimum customized processor configuration.

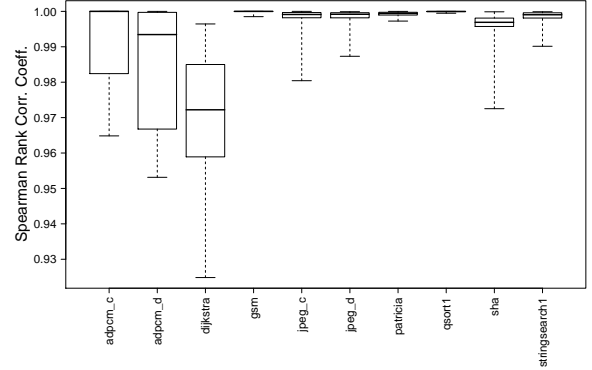
V. RELATED WORK

We now discuss related work in the areas of analytical performance modeling, design space exploration, dataset sensitivity and processor customization.

A. Analytical performance modeling

There are basically two approaches to analytical performance modeling, empirical modeling and mechanistic modeling. Empirical modeling treats the processor as a black box and typically uses regression [12] or neural networks [8] for building performance models from a set of training examples. Mechanistic modeling on the other hand models processor performance based on first principles, based on a profound understanding of the architecture. Interval analysis is a recently proposed mechanistic model for out-of-order processors [5],

(a) Execution time



(b) Power consumption

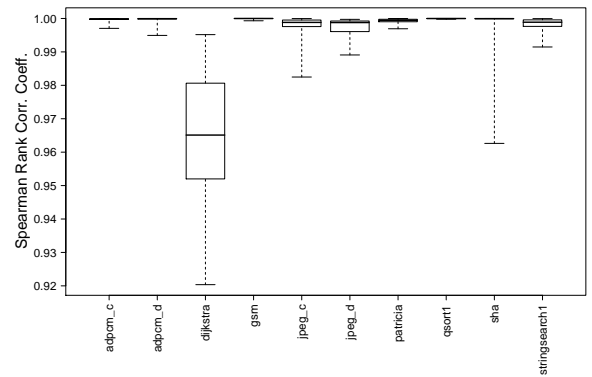


Fig. 7. Evaluating whether a dataset is predictive for other datasets for determining the relative rank ordering of processor configurations.

[9]. This paper extended the interval model towards in-order processors.

B. Design space exploration

Design space exploration is a very complicated endeavor for a number of reasons. For one, the design space is typically huge. Second, the evaluation of a single design point typically takes a very long time due to the slow simulation speeds compared to native hardware execution. Given the importance of the problem, a fair amount of research has been done on design space exploration. Yi et al. [17] use a Plackett-Burman design of experiment to identify the important axes in the design space in order to drive the design process. Eyerman et al. [4] leverage machine learning techniques to more quickly steer the search process to the optimum design point. Lee and Brooks [13] use empirical models to explore the design space of adaptive microarchitectures, while Karkhanis and Smith [10] use mechanistic model to guide the design of application-specific out-of-order processors.

C. Dataset sensitivity

Chen et al. [2] proposed KDataSets, a set of 1000 datasets for a broad set of the MiBench benchmarks. They used KDataSets to understand dataset sensitivity for iterative optimization which aims at finding the best set of compiler

optimizations for a given application. They concluded that iterative optimization is relatively dataset insensitive, i.e., a single combination of compiler optimizations achieves at least 86% of the best possible speedup across all datasets.

Eeckhout et al. [3] proposed a methodology based on Principal Component Analysis (PCA) to gauge the impact of input datasets on an application's behavioral execution characteristics. They found that while some applications exhibit limited sensitivity, others exhibit large sensitivity. They use the method to identify a small set of representative benchmarks and input datasets for architecture design space exploration.

D. Processor customization

Processor customization consists of either instruction-set and/or micro-architecture customization. LISATek is an example framework for generating custom processor instructions [11]. Several customizable architectures enable both instruction-set and microarchitecture customization. Microarchitecture customization includes the pipeline length (Tensilica LX3), the caches (Tensilica LX3, and ARC 600 and 800), the complexity of several logic blocks, such as multipliers (MIPS CorExtend), etc. Rowen and Leibson [16] provide a detailed overview and empirical evaluation of the customization capability of the Tensilica Xtensa processor, including the customization of the memory system (instruction and data cache size, associativity, line size, data cache write policy).

VI. CONCLUSION

This paper addressed the fundamental, but so-far unanswered, question whether processor customization is sensitive to the workload's input datasets. Using an accurate mechanistic analytical model, which allows for exploring a complex design space at least 4 orders of magnitude faster than detailed cycle-accurate simulation with an average prediction error of 3.6%, we found that processor customization is largely dataset insensitive. This has an important practical implication: a single or only a few datasets is sufficient for determining the optimum customized processor for a specific application or application domain of interest. We found this result to be surprising: in spite of the wide diversity across datasets, optimizing the processor architecture for a specific dataset typically leads to the same (or similar) optimum design point as for other datasets.

As part of our future work, we plan to explore an even larger design space including application-specific architectures with custom instruction-set architectures, along with combined architecture-compiler exploration.

ACKNOWLEDGEMENTS

We would like to thank the anonymous reviewers of this paper for their valuable feedback. Maximilien Brueghe and Lieven Eeckhout are supported by the FWO projects G.0255.08 and G.0179.10, the UGent-BOF projects 01J14407 and 01Z04109, and the European Research Council under

the European Commission's Seventh Framework Programme (FP7/2007-2013) / ERC Grant agreement No. 259295. Stijn Eyerman is supported through a postdoctoral fellowship by the Research Foundation-Flanders (FWO). Yang Chen and Chengyong Wu are supported by the National Natural Science Foundation of China under grant No. 60873057 and 60921002, and National Basic Research Program of China under grant No. 2011CB302504.

REFERENCES

- [1] N. L. Binkert, R. G. Dreslinski, L. R. Hsu, K. T. Lim, A. G. Saidi, and S. K. Reinhardt. The M5 simulator: Modeling networked systems. *IEEE Micro*, 26(4):52–60, 2006.
- [2] Y. Chen, Y. Huang, L. Eeckhout, G. Fursin, L. Peng, O. Temam, and C. Wu. Evaluating iterative optimization across 1000 datasets. In *PLDI*, pages 448–459, June 2010.
- [3] 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, Feb. 2003. <http://www.jilp.org/vol5>.
- [4] S. Eyerman, L. Eeckhout, and K. De Bosschere. Efficient design space exploration of high performance embedded out-of-order processors. In *DATE*, pages 351–356, Mar. 2006.
- [5] S. Eyerman, L. Eeckhout, T. Karkhanis, and J. E. Smith. A mechanistic performance model for superscalar out-of-order processors. *ACM Transactions on Computer Systems (TOCS)*, 27(2), May 2009.
- [6] M. R. Guthaus, J. S. Ringenberg, D. Ernst, T. M. Austin, T. Mudge, and R. B. Brown. MiBench: A free, commercially representative embedded benchmark suite. In *Proceedings of the IEEE 4th Annual Workshop on Workload Characterization (WWC)*, Dec. 2001.
- [7] W.-W. Hu, F.-X. Zhang, and Z.-S. Li. Microarchitecture of the godson-2 processor. *J. Comput. Sci. Technol.*, 20:243–249, March 2005.
- [8] E. Ipek, S. A. McKee, B. R. de Supinski, M. Schulz, and R. Caruana. Efficiently exploring architectural design spaces via predictive modeling. In *ASPLOS*, pages 195–206, Oct. 2006.
- [9] T. Karkhanis and J. E. Smith. A first-order superscalar processor model. In *ISCA*, pages 338–349, June 2004.
- [10] T. Karkhanis and J. E. Smith. Automated design of application specific superscalar processors: An analytical approach. In *ISCA*, pages 402–411, June 2007.
- [11] K. Karuri, M. A. Al Faruque, S. Kraemer, R. Leupers, G. Ascheid, and H. Meyr. Fine-grained application source code profiling for ASIP design. In *DAC*, June 2005.
- [12] B. Lee and D. Brooks. Accurate and efficient regression modeling for microarchitectural performance and power prediction. In *ASPLOS*, pages 185–194, Oct. 2006.
- [13] B. Lee and D. Brooks. Efficiency trends and limits from comprehensive microarchitectural adaptivity. In *ASPLOS*, pages 36–47, Mar. 2008.
- [14] S. Li, J. H. Ahn, R. D. Strong, J. B. Brockman, D. M. Tullsen, and N. P. Jouppi. McPAT: An integrated power, area, and timing modeling framework for multicore and manycore architectures. In *MICRO*, pages 469–480, Dec. 2009.
- [15] R. L. Mattson, J. Gecsei, D. R. Slutz, and I. L. Traiger. Evaluation techniques for storage hierarchies. *IBM Systems Journal*, 9(2):78–117, June 1970.
- [16] C. Rowen and S. Leibson. Flexible architectures for engineering successful SOCs. In *DAC*, June 2004.
- [17] J. J. Yi, D. J. Lilja, and D. M. Hawkins. A statistically rigorous approach for improving simulation methodology. In *HPCA*, pages 281–291, Feb. 2003.