Microarchitectural Design Space Exploration Using An Architecture-Centric Approach

Christophe Dubach, Timothy M. Jones and Michael F.P. O'Boyle
Member of HiPEAC
School of Informatics
University of Edinburgh
christophe.dubach@ed.ac.uk, {tjones1,mob}@inf.ed.ac.uk

Abstract

The microarchitectural design space of a new processor is too large for an architect to evaluate in its entirety. Even with the use of statistical simulation, evaluation of a single configuration can take excessive time due to the need to run a set of benchmarks with realistic workloads.

This paper proposes a novel machine learning model that can quickly and accurately predict the performance and energy consumption of any set *of programs on any microarchitectural configuration. This* architecture-centric *approach uses prior knowledge from off-line training and applies it across benchmarks. This allows our model to predict the performance of any new program across the entire microarchitecture configuration space with just 32 further simulations.*

We compare our approach to a state-of-the-art program-specific *predictor and show that we significantly reduce prediction error. We reduce the average error when predicting performance from 24% to just 7% and increase the correlation coefficient from 0.55 to 0.95. We then show that this predictor can be used to guide the search of the design space, selecting the best configuration for energy-delay in just 3 further simulations, reducing it to 0.85. We also evaluate the cost of off-line learning and show that we can still achieve a high level of accuracy when using just 5 benchmarks to train. Finally, we analyse our design space and show how different microarchitectural parameters can affect the cycles, energy and energy-delay of the architectural configurations.*

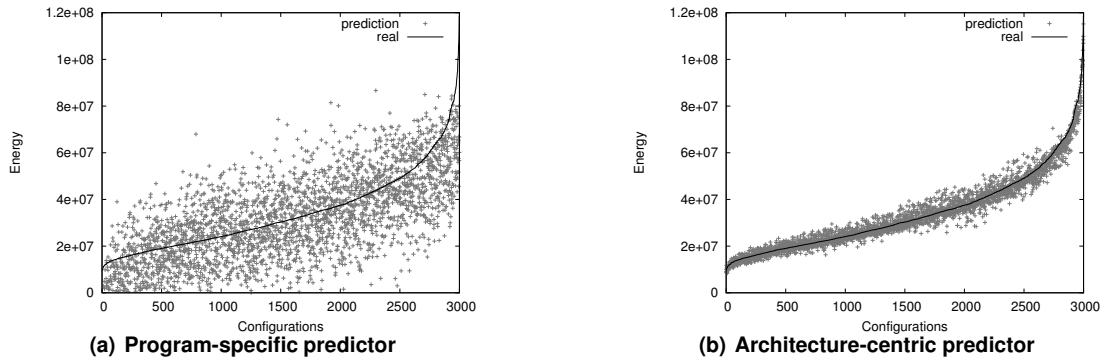
1. Introduction

Architects use cycle-accurate simulators to explore the design space of new processors. However, in superscalar processors the number of different variables and the range of values they can take makes the design space too large to be completely evaluated. This is coupled with the fact that cycle-accurate simulation can be slow due to the need for detailed modelling of the microarchitecture and the desire to simulate many benchmarks with realistic workloads.

Recently, several techniques based on statistical sampling have been developed to reduce the time taken for simulation, such as SimPoint [23] and SMARTS [27]. However, although these schemes increase the number of simulations possible within a given time frame, given the huge size of the design space to be explored, a full evaluation remains unrealistic.

Several studies have proposed the use of machine learning to help evaluate this massive space [11, 12, 16, 17, 18]. These schemes require a number of simulations of a benchmark to be run, the results from which are used to train a predictor. This can then be used to determine the rest of the design space without the need for further simulation. However, existing techniques suffer from several major drawbacks.

- Whenever a new program is considered, a new predictor must be trained and built, meaning there is a large overhead even if the designer just wants to compile with a different optimisation level [26]. Our approach learns across programs and captures the behaviour of the architecture rather than the program itself;
- A large number of training simulations are needed to use these existing predictors, offsetting the benefits of the schemes. In our approach, having previously trained off-line on a small number of programs, we only need a few simulations, called a signature, in order to characterise each new program we want to predict. We show that, in fact, this can be as low as just 32 simulations to maintain a high level of accuracy;
- Existing works only give the error of their models. No information is given on how the models are used in



(a) Program-specific predictor

(b) Architecture-centric predictor

Figure 1. The design space of applu when considering energy. We show the predictions given by a program-specific predictor and by our architecture-centric approach. Both models are given the same 32 simulations from this program, with the architecture-centric predictor having also been trained off-line on different benchmarks.

practice to select good configurations by searching the design space. We show how our model can be used to search for the best configuration in terms of cycles, energy and energy-delay (ED).

This paper presents a new and different approach to design space exploration using machine learning. We use existing knowledge to predict a new program on any given architecture configuration, something no other research has successfully attempted. We train our *architecture-centric* model off-line (at the factory) on a number of benchmark programs, then, using a completely new program never seen before, we run just 32 simulations of the new program. We can then predict the rest of the design space of 18 billion configurations. This means that encountering a new program, or simply exploring the compiler optimisation space, can be done efficiently, at the same time as microarchitectural design space exploration, with low overhead. This is an order of magnitude less than the current state-of-the-art, *program-specific* approaches [11, 12, 16, 17, 18] and shows the ability to learn across programs, using prior knowledge to predict new programs.

Although absolute error is an important metric to evaluate the accuracy of our predictor, in this setting of design space exploration, correlation is equally important. This shows how the model can follow the trend of the space. Hence we show the rmae and correlation coefficient for our architecture-centric predictor and prove that it is better than other, existing approaches. We then use our model to perform a search of the architectural design space and compare it against an existing program-specific predictor [11]. We show that after training we can find the best point in 3000 randomly selected configurations by performing just 3 further simulations per benchmark, compared to the program-specific predictor needing a further 103 simulations per benchmark.

One reasonable criticism of our work could be that the cost of off-line training is too high. We address this by considering the use of just 5 factory programs and show that even with exactly the same number of total simulations our model has an error 1.5 times smaller than program-specific predictors. We thus show that even if all off-line training were considered part of the overall training budget, our approach still outperforms existing techniques.

Finally, we analyse the design space and provide a statistical characterisation of it. We show how different architectural parameters affect cycles, energy and ED and characterise good design points.

The rest of this paper is structured as follows. Section 2 provides a simple example showing the accuracy of our predictor. We describe our design space in section 3 and then show the use of our model to predict the performance of an architecture configuration on any new program in section 4. Section 5 evaluates the use of our model in searching the whole space, then section 6 addresses the cost of off-line training. We characterise our design space in section 7 and describe work related to ours, especially the program-specific predictor that we compare against throughout this work, in section 8. Finally, we conclude this paper in section 9.

2. Motivation

This section provides a simple motivating example, illustrating the superior accuracy of our scheme in predicting architecture performance.

Figure 1 shows the energy design space for a typical application, *applu*. We show the resulting prediction for a program-specific predictor and our scheme. The program-specific model has been trained with 32 simulations from this benchmark, whereas our architecture-centric model has

been trained off-line at the factory with other benchmarks and given the same 32 simulations as a signature for this new program.

In each graph the different microarchitectural configurations are plotted along the x-axis in order of increasing energy which is plotted along the y-axis. We show the real value of energy as a line in each graph and each model then provides a prediction for that configuration which is plotted as a point. The closer the point is vertically to the line then the more accurate the prediction is.

Figure 1(a) shows how the existing program-specific technique performs when predicting this space and figure 1(b) shows how our architecture-centric scheme performs. For the same number of simulations from this program the program-specific predictor has a high error rate and cannot determine the trend within the design space. Our architecture-centric model, however, can apply prior knowledge from previously seen benchmarks and has a low error rate, accurately following the shape of the space. From these graphs it is clear that our model provides more accurate predictions than the existing scheme and can be used (as we show in section 5) to effectively search for the best microarchitectural configuration within this massive space.

3. Experimental Setup

This section describes the design space of microarchitectural configurations that we explore in this paper. We also present our simulation environment and the metrics used to evaluate predictor performance in the rest of the paper.

3.1. Microarchitecture Design Space

This paper proposes schemes to quickly and accurately determine the best microarchitectural configuration within a large design space. We chose to vary 13 different parameters in a superscalar simulator to give an overall total of 63 billion different configurations of the processor core. With a design space as large as this it would be impossible to simulate every configuration.

The parameters we varied, shown in table 1, are similar to those other researchers have looked at [11, 16] which allows meaningful comparisons with previous work. The left-hand column describes the parameter and the middle column gives the range of values the parameter can take along with the step size between the minimum and maximum. The right-hand column gives the number of different values that this range gives. For example, the reorder buffer (ROB) has a minimum size of 32 entries and a maximum size of 160 entries varied in steps of 8, meaning 17 different design points.

Table 2(a) describes the processor parameters that remained constant in all of our simulations. Table 2(b) describes the functional units which varied according to the width of the processor. So, for a 4-way machine we used 4 integer ALUs, 2 integer multipliers, 2 floating point ALUs and 1 floating point multiplier/divider.

Table 1. Microarchitectural design parameters that we varied with their range, steps and the number of different values they can take.

Parameter	Value Range	Number
Machine width	2, 4, 6, 8	4
ROB size	$32 \rightarrow 160 : 8+$	17
IQ size	$8 \rightarrow 80 : 8+$	10
LSQ size	$8 \rightarrow 80 : 8+$	10
RF sizes	$40 \rightarrow 160 : 8+$	16
RF read ports	$2 \rightarrow 16 : 2+$	8
RF write ports	$1 \rightarrow 8 : 1+$	8
Gshare size	$1K \rightarrow 32K : 2*$	6
BTB size	1K, 2K, 4K	3
Branches allowed	8, 16, 24, 32	4
L1 Icache size	$8KB \rightarrow 128KB : 2*$	5
L1 Dcache size	$8KB \rightarrow 128KB : 2*$	5
L2 Ucache size	$0.25MB \rightarrow 4MB : 2*$	5
Total		$6.3 * 10^{10}$

Within our design space of 63 billion points, we filtered out configurations that did not make architectural sense. So, for example, we did not consider configurations where the reorder buffer was smaller than the issue queue. This reduced the total design space to 18 billion points.

3.2. Simulator And Benchmarks

For our experiments we used the SPEC 2000 benchmark suite [24] compiled with the highest optimisation level. We used all programs apart from *ammp* which would not run correctly in our environment. We used the *reference* input set for each benchmark and ran each simulation for 100 million instructions.

We simulated representative traces of instructions through the use of SimPoint [23], to be confident that we were accurately representing all phases of each benchmark. SimPoint picked a trace of 100 million instructions for each program, after profiling, that captures the behaviour of each whole benchmark. We then fast-forwarded 100 million instructions before this point, warmed the cache and branch predictor and then performed the simulation as usual.

Our simulator is based on Wattch [4] (an extension to SimpleScalar [1]) and Cacti [25] and contains detailed models of energy for each structure within the processor. We accurately modelled the latencies of microarchitecture components using the Cacti timing information to make our simulations as realistic as possible.

In the following sections we use cycles as a metric for

Table 2. Microarchitectural design parameters that were not explicitly varied, either remaining constant or varying according to the width of the machine.

(a) Constant

Parameter	Configuration
BTB	4-way
L1 Icache	32B line, 4-way
L1 Dcache	32B line, 4-way
L2 Ucache	64B line, 8-way
FU Latencies (cycles)	IntALU 1, IntMul 3, FPALU 2, FPMulDiv (4/12)

(b) Related to machine width

Parameter	Number			
Machine width	2	4	6	8
IALU	2	4	5	6
IMul	1	2	2	3
FPALU	1	2	3	4
FPMulDiv	1	1	2	2

program performance and energy consumption (in nJ) as gained from Cacti and Wattch. We also show the energy-delay (ED) product to determine the trade-off between performance and energy consumption, or efficiency. This is an important metric in microarchitecture design because it indicates how efficient the processor is at converting energy into speed of operation, the lower the value the better [9]. The ED product implies that there is an equal trade-off between energy consumption and delay.

3.3. Evaluation Methodology

In order to evaluate the accuracy of our predictors, we use the *relative mean absolute error* (rmae) defined as $rmae = \left|\frac{\textbf{predicted value}-\textbf{real value}}{\textbf{real value}}\right| \cdot 100\%$ This metric tells us how much error there is between the predicted and actual values. For example, an rmae of 100% would mean that the model, on average, would be predicting a value that was double the real value.

Although the rmae is an important metric, it is not a good measure of how accurately the model is predicting the shape or trend of the space. Since we want to use our predictor to distinguish between good and bad architectures (*i.e.* low or high cycles, energy or ED), we need a metric that describes how accurately the predictor models the shape of the space.

To analyse the quality of our models, we therefore use the correlation coefficient. The correlation between two variables is defined as $corr = \frac{\textbf{cov}(\mathbf{X},\mathbf{Y})}{\sigma_X \cdot \sigma_Y}$, where σ_X and σ_Y represent the standard deviation of variable X and Y respectively, and $\textbf{cov}(X, Y)$ is the covariance of variable X and Y. The correlation coefficient only takes values between -1 and 1. At the extreme, a correlation coefficient of 1 means that the predictor perfectly models the shape of the real space. A correlation coefficient of 0 means there is no linear relation between the predictions and the actual space.

Unless otherwise stated, all our predictors are validated using 3000 configurations selected from our total design space (18 billion configurations) by uniform random sampling.

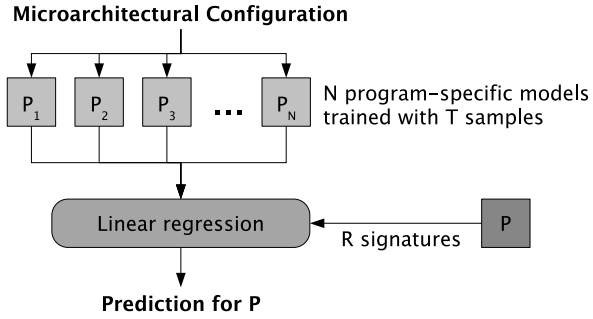


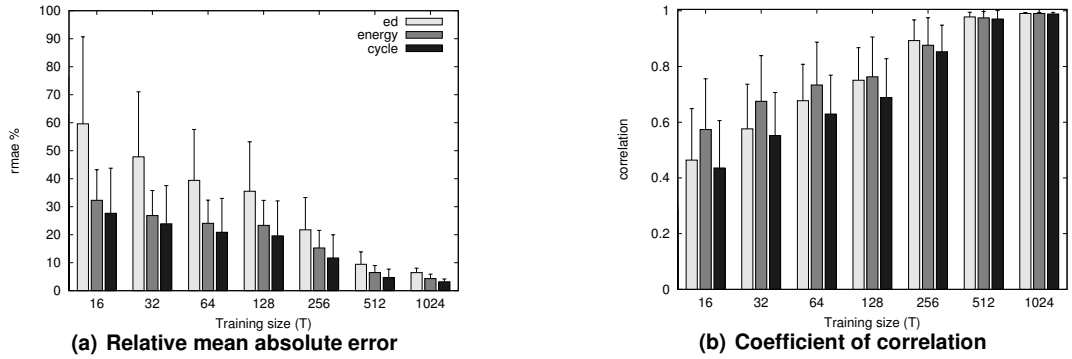
Figure 2. Our architecture-centric model. We train N program-specific predictors off-line with a number T of training simulations. The results are fed into a linear regressor along with a signature consisting of R simulations from a new program P to provide a prediction for any configuration in the microarchitectural design space.

4. Predicting A New Program

This section describes our scheme where we use prior information about a number of previously seen programs to quickly and accurately predict the number of cycles, energy and ED product of any new program within the microarchitectural design space. Our model is based on a simple linear combination of the design space of several individual programs from the training set. Given this linear combination we can accurately model the space of any new program. Figure 2 gives an overview of how our model works. Determining the best linear combination of previously trained models relies on a few simulations (known as the signature) of the new program and is described in section 4.2.

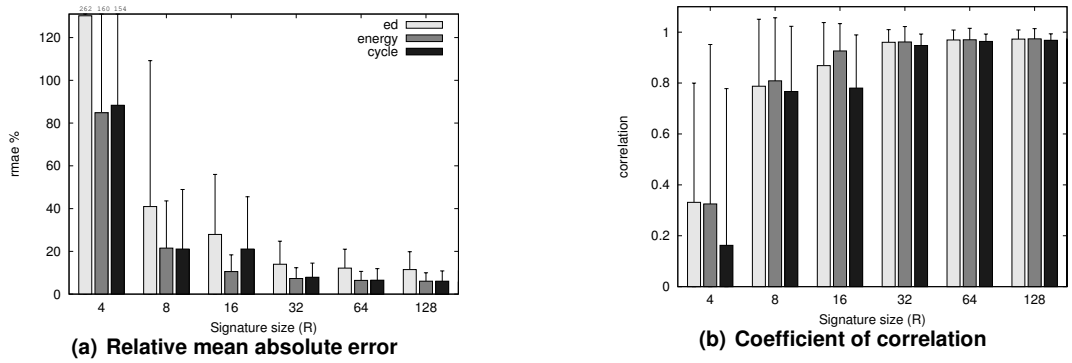
4.1. Program-Specific Predictors

Our scheme builds on top of program-specific predictors. We use artificial neural networks [3] to build these predic-



(a) Relative mean absolute error (b) Coefficient of correlation

Figure 3. The rmae and correlation (along with standard deviation) of the program-specific predictors when using varying numbers of training configurations. We average across all programs and show results for ED, cycles and energy. This shows that 512 is a good trade-off in terms of correlation and accuracy against the number of training configurations required.



(a) Relative mean absolute error (b) Coefficient of correlation

Figure 4. The rmae and correlation (with standard deviation) of our architecture-centric predictor when varying the size of the signature, from the new program. In these graphs we have fixed the number of training configurations to 512. We average across all programs and show results for ED, cycles and energy. This shows that beyond a size of 32 we do not get significant further improvement hence we fix the signature to be 32 simulations.

tors similar to those used in [11], although we could have used any other related approach [12, 13, 16, 17]. They consist of a multi-layer perceptron with 1 hidden layer of 10 neurons and use the sigmoid activation function for the input and hidden layers and the linear function for the output layer. We train each predictor off-line on a number of simulations from the training programs.

Figure 3 shows the rmae and correlation coefficient when varying the number of training simulations (per training program) to use for our program-specific predictors to predict ED, energy and cycles. We selected the training simulations from the design space of 18 billion configurations using uniform random sampling. In figure 3(a) we can see that, as expected, the rmae decreases as the size of the training set increases. The same is shown in figure 3(b) for the correlation coefficient, *i.e.* as you increase the size of the training data, the error gets smaller and the correlation increases.

From the graphs in figure 3 we can conclude that we should use 512 configurations per training program as input to our model, since this provides low rmae and high correlation for ED, cycles and energy. Increasing the number of configurations per training program gives only minor improvement.

4.2. Architecture-Centric Predictor

Our technique for predicting a new program using prior knowledge is based on off-line training of the program-specific predictors combined with a small number of sim-

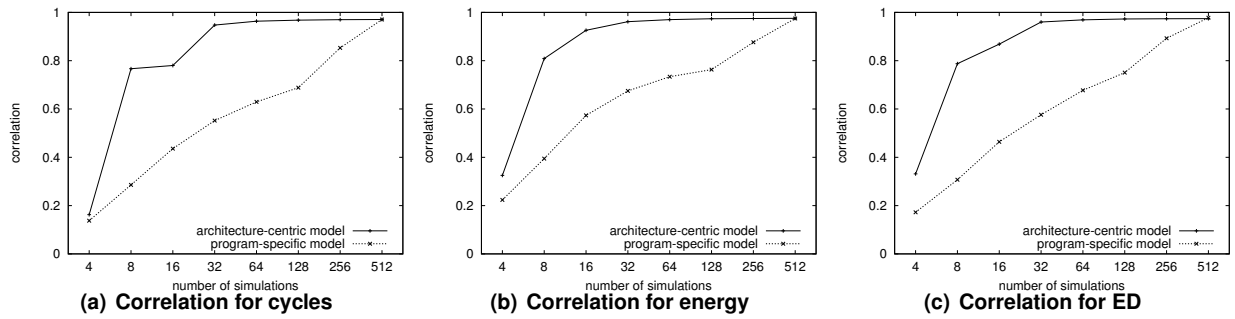


Figure 5. Correlation when varying the number of simulations for both the program-specific model and our architecture-centric approach. In the program-specific model these simulations are used as training data; in our model they are used as a signature for the new program. We average across all programs and show results for cycles, energy and ED.

ulations from the design space of the new program. We call these small number of simulations the signature.

The architecture-centric model is a simple linear regressor and is shown in figure 2. In effect, the behaviour of the architecture space on a new program can be modelled as a linear combination of their behaviour on previously seen programs. The small number of simulations (signature) are used to find weights which determine the right combination of previously seen models that best capture the behaviour of the new program. This surprisingly simple approach is actually highly accurate, as we show in this section.

We train a number of program-specific predictors off-line on a set of training configurations, as in section 4.1. To predict a new program we combine these predictors with a linear regressor which we also supply with a signature from the new program. This can then give us the prediction for any microarchitectural configuration within the design space.

Section 4.1 determined the optimum number of training configurations for the program-specific predictors was 512. We now wish to find the optimum size of the new program's signature to complete our architecture-centric model. Figure 4 shows the rmae and correlation coefficient for ED, cycles and energy when varying the size of the signature from the new program when all other benchmarks have been used as training with 512 configurations.

It is immediately obvious, looking at both figure 4(a) and figure 4(b) that using a signature size beyond 32 does not bring further benefits in terms of either rmae or correlation coefficient. Using a signature size of less than 32, however, increases the rmae by at least a factor of two for ED. Using size 32 we can get a correlation coefficient of 0.95 for cycles, energy and ED and an rmae of 7%, 7% and 14% for cycles, energy and ED respectively. Hence, we fix the size of the signature to be 32, along with the number of training configurations which we have already fixed at 512. We thus show that in our space of 18 billion configurations we only need 32 simulations from any new program to form a signature and enable us to accurately predict the entire design space for the new program.

4.3. Comparison

Given an equal number of simulations from a new program, we are interested to see how our scheme performs compared with the program-specific predictor proposed by İpek *et al.* [11]. Figure 5 shows the correlation for cycles, energy and ED when varying the number of simulations required. For the program-specific model the simulations are used as training data whereas in our model they are used as the signature for the new program. As it can be seen, for each metric our scheme is more correlated to the actual data than the program-specific approach, because our scheme can apply knowledge from the programs it has previously been trained on.

Figure 6 shows a detailed comparison between the program-specific model and our architecture-centric approach for each of the programs in our benchmark suite when predicting ED. In our scheme we trained off-line using all programs apart from the one shown by each bar, then ran 32 simulations of the new program as a signature. In the program-specific model we used these same 32 simulations as training data for the predictor.

As we can see, given the same number of simulations from the program of interest our model is able to outperform the program-specific model for each of the programs. On average, the rmae of the program-specific model for ED is 47% whereas for our approach it is only 14%. With both models, the maximum error is achieved on the same program (*art*). For the architecture-centric model, the maximum error is 50% while for the program-specific model it is 95%.

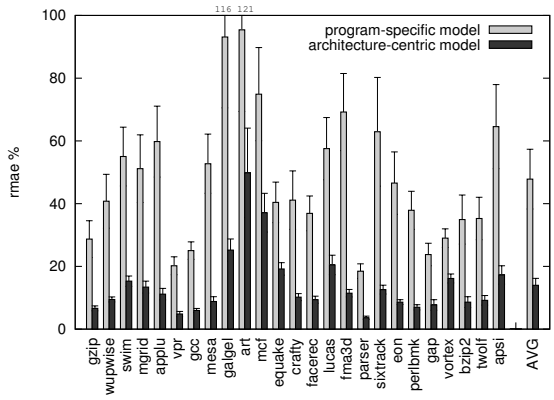


Figure 6. The rmae (with standard deviation) for ED on a per program basis for the program-specific model and our architecture-centric approach. Each model uses 32 training points from the new program which are used as training data for the program-specific predictor and as a signature from the new program for our scheme.

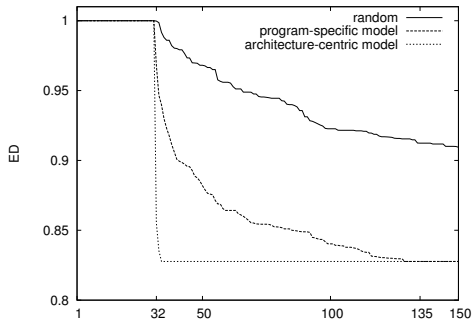


Figure 7. Searching the space for minimum ED using a program-specific model and our architecture-centric model. An initial 32 evaluations are used as training data and a signature respectively. After just 3 further simulations per benchmark we find the minimum point for ED in a randomly-selected space of 3000 configurations which has an ED of 0.85. The program-specific predictor, on the other hand, takes a further 103 simulations per benchmark to get to this point.

5. Searching The Space

In the section 4 we looked at the accuracy of our predictor using different metrics such as rmae and correlation coefficient. While those metrics are important to measure and compare the efficiency of our models, it is also important to keep in mind the reason for creating these models: exploring the design space. In this section we evaluate how search is performed using both our architecture-centric predictor and the program-specific approach.

5.1. Minimising ED

Figure 7 shows how our model performs when searching the design space for the best microarchitectural configuration in terms of the ED metric. We chose this metric because cycles and energy by themselves are intuitively easier to predict (large structures generally mean good performance, for example). ED is important to help designers consider the trade-offs between performance and energy. Of course, we could also have searched for the best configuration in terms of cycles or energy just as accurately.

First, we build our architecture-centric predictor. We selected 32 configurations per benchmark from the total design space using uniform random sampling and simulated them to extract the signature for our approach.

Ideally, we would like to search the entire space using this predictor to find the best configuration. However, in order to validate its prediction, we would then have to simulate the entire space, which is clearly infeasible, Instead we selected a further 3000 configurations from the total design space, again using uniform random sampling, and used our predictor to rank these configurations in order.

If our predictor were perfect then the highest ranked prediction would be the best configuration. Due to inaccuracies in the model, however, the best predicted configuration is not always the actual best. Instead, our predictor always finds the best configuration verified by simulation) within the predicted top 3. Hence we need only 3 further simulations per benchmark to find the best point in this space of 3000 configurations.

We repeated this experiment using the program-specific predictor and a random search (using the same 3000 randomly-selected configurations). The program-specific model needs a further 103 simulations per benchmark to find the best configuration, which we found with only 3 per benchmark. Knowing how far our models are from random search is important since it gives another indication about the quality of the predictions. Indeed random search is directly associated with the distribution of the space. In our case, both models outperform random search quickly; random actually finds the minimum after approximately 1500 further simulations per benchmark. Figure 7 summarises these results. In order to calculate the values for ED we selected the best configuration found in the initial 32 signature simulations as the baseline, (i.e. ED = 1.0).

The performance of the program-specific model, trained only with 32 simulations, is interesting. As we saw earlier, this predictor exhibits an rmae of 48% which seems poor compared to the same predictor trained with 512 sim-

Table 3. The microarchitectural configuration giving the minimum ED and the median configuration in our randomly-selected space of 3000 points.

Config	Best	Median
Width	4	6
ROB / IQ / LSQ	152 / 32 / 48	96 / 56 / 72
RF / Read / Write	136 / 4 / 1	72 / 4 / 3
Bpred / BTB / Br	16K / 2K / 8	1K / 2K / 24
I/D/U cache	64K / 8K / 4M	16K / 128K / 1M

ulations that achieves around 10%. However, using only 32 training simulations per benchmark allows it to find the minimum after a total of 135 simulations per benchmark. If we had considered using the predictor trained with 512 simulations per benchmark, it would have required at least $512 \cdot 25$ simulations in total. This clearly shows that it is important to keep in mind the overall goal of building predictors which is finding good architecture configurations rather than focusing narrowly on a metric such as rmae.

5.2. Best Configuration Found

The microarchitecture configuration giving the minimum ED in our 3000 points is shown in table 3, along with the median configuration in this space. The configuration giving the minimum ED has a larger reorder buffer than the median, a larger register file, branch predictor and unified L2 cache. These structures aid performance by enabling the extraction of ILP. It also has a narrower pipeline width and fewer read ports because these parameters affect the energy consumption of the processor: smaller is less power-hungry.

6. Reducing The Training Costs

The training of our architecture-centric model is performed off-line and, as such, is not taken into account when evaluating new programs. One criticism could be that this training in the factory does not come for free, so we have considered the extent to which we can reduce it and its effect on the accuracy of prediction for new programs.

6.1. Using Only Five Training Programs

Even though off-line training at the factory only has to take place once for all future users of the simulator, we wish to consider the view that any training, whether off-line or not, should be considered part of the overall training budget.

We therefore decided to allow a total budget of simulations available for prediction across all programs in our benchmark suite. This budget was fixed at 3200 simulations

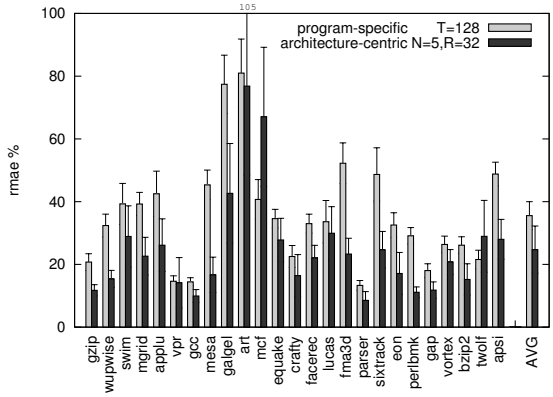


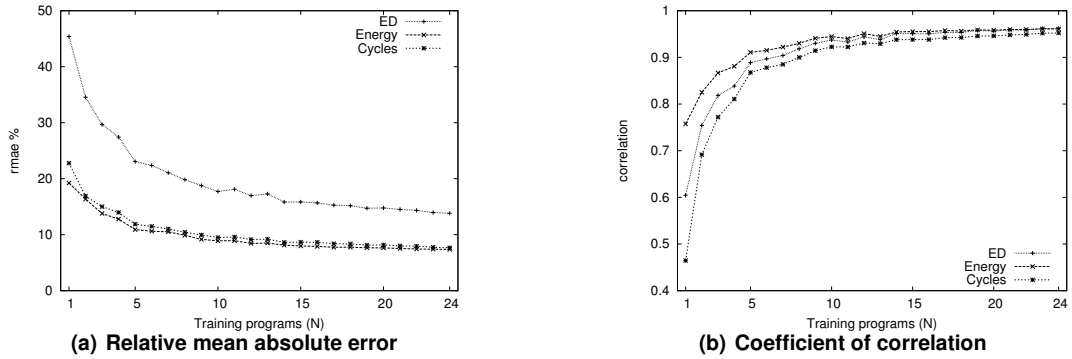
Figure 8. The rmae (with standard deviation) for ED with a fixed total number of simulations (training included) for both models. The program-specific predictors are trained on-line using the simulation budget whereas the architecture-centric approach is trained off-line with a portion of the simulations and then on-line with the remainder.

for ease of comparison. In the case of the program-specific model the budget was evenly spread over each program, *i.e.* we used 128 samples for training each of the 25 models needed to be built ($3200/25 = 128$). For our architecture-centric approach we trained using just 5 programs. We then ran 512 simulations on each of these 5 programs to build our predictor ($5 \cdot 512$). Then for each of the remaining 20 programs we extracted a signature requiring just 32 simulations ($20 \cdot 32$), giving a total budget of $5 \cdot 512 + 20 \cdot 32 = 3200$ simulations. Figure 8 shows how both models perform for ED when given this budget of simulations. As it can be seen, the error of the program-specific model is 36% while our model achieves 22%.

The cost of off-line training is reduced as we have more programs to predict. Predicting a new program with our model requires just 32 additional runs whereas the program-specific predictor needs 128 or more depending on the required precision. As we wish to get a more rounded view of a candidate architecture configuration, running more extensive benchmark suites with different compiler options, training many different program-specific models becomes impossible due to the high training cost. On the other hand, our model scales well, since it only needs the extraction of a few simulations per new program to create a signature

6.2. Varying Training Program Numbers

As seen in the previous section, we can reduce the cost of off-line training needed by our model by using fewer programs to train with. In this section we show the effect of



(a) Relative mean absolute error **(b) Coefficient of correlation**

Figure 9. The rmae and correlation for our architecture-centric predictor when varying the number of off-line training programs. The x-axis shows the number of off-line training programs and the y-axis shows the results when predicting on the remaining benchmarks. We show results for ED, energy and cycles. Each training program is run 512 times and the new program to be predicted given a signature size of 32.

the number of training programs on rmae and correlation coefficient. Figure 9 shows the accuracy of our model when varying the number of programs in the training set using a signature of size 32. As the number of programs increases, both the rmae and the correlation coefficient tend to improve. With 15 programs, both metrics reach a plateau and adding more programs makes only a negligible difference.

6.3. Summary

This section has shown that the one-off cost of training on between 5 and 15 programs results in a highly accurate predictor, capable of predicting the performance of all architecture configurations on any new program with just 32 further simulations. If the "factory" cannot afford the one-off cost of building the predictor, a designer using our approach can still build a more accurate predictor (with strictly the same training budget) than can be achieved using program-specific approaches. Furthermore, this architecture-centric predictor could be used in the future for any new program with just a nominal 32 further simulations.

7. Characteristics Of The Space

This section describes characteristics of the design space we have explored. There are 18 billion design points in our space which is obviously too many to simulate in total. Therefore we used uniform random sampling to pick 3000 architectural configurations and simulated these for each benchmark (3000 configurations $\cdot$ 25 programs = 75000 SimPoint simulations). Using uniform random sampling means that we have a fair and representative sample of the total design space.

Figure 10 shows the characteristics of this space on a per-program basis for cycles, energy and ED. In each graph we show the maximum of the space for each benchmark, then the 75% quartile, median, 25% quartile and minimum. Note that the y-axis in each graph is on a logarithmic scale.

As can be seen in figure 10(a), the number of cycles taken for each simulation varies considerably between programs ranging from the longest , $2*10^{10}$ cycles, to the shortest, $3*10^{7}$ cycles. Some programs vary enormously, for example *art* which varies between $7*10^{7}$ and $1*10^{9}$ cycles. Other programs, such as *parser*, vary only slightly (between $2*10^{8}$ and $3.5*10^{8}$).

Figure 11 shows some of the parameters that we vary and how they influence the number of cycles required. Each parameter design point is shown on the x-axis. The y-axis represents the frequency with which a design point occurs. These diagrams show how much influence the parameters have on the cycles and energy of each configuration. Figures 11(a) to 11(f) show the top 1% for cycles (so best performance) and figures 11(g) to 11(l) show the worst 1%.

From these diagrams we can see that the parameter having the greatest impact on performance is the size of the register file (figures 11(c) and 11(i)). This parameter is highly correlated with overall performance. In the worst-performing 1% a small register file is common (in 81% of them it has just 40 registers). This confirms the widely-known fact that register files are critical components in the microarchitecture. However, the best performing 1% of configurations have register files ranging from mid-sized to large. This suggests that a small register file is a bottleneck to performance, but a large register file does not necessarily mean high performance. The best configurations for cycles tend to have a wide pipeline (6 or 8 instructions per cycle, figure 11(a)), have a large reorder buffer (figure 11(b)),

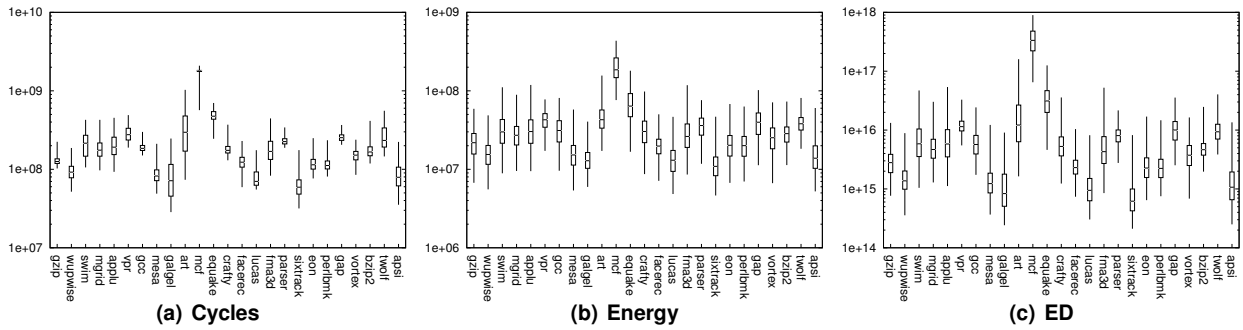


Figure 10. Characteristics of our design space for cycles, energy and ED. ED=1 is the value of the median configuration in the randomly-selected space. Each graph shows the median, quartiles for 25% and 75%, minimum and maximum values for each benchmark. There is large variation across different programs (the y-axis is logarithmic) and on average. This makes it difficult to easily pick a good configuration for all programs, justifying the need for a predictor to aid search.

branch predictor (figure 11(f)) and second level cache (figure 11(e)). This intuitively makes sense because the first two allow the extraction of ILP through branch speculation.

In terms of energy (figures 10(b) and 12) the characterisation of the space is more clear cut with large differences between the minimum and maximum of the space. The configurations with the highest energy consumption have a wide pipeline (figure 12(g)), small register file (figure 12(i)) and large second level cache (figure 12(k)). The low energy configurations tend to have a pipeline only a couple of instructions wide (figure 12(a)), only a few register file read ports (figure 12(d)) and a small L2 cache (figure 12(e)). However, they have moderately-sized register files with only a few read and write ports into them, and average-to-large sized branch predictors, etc. These configurations trade off dynamic energy consumption for static energy savings. Were the structures smaller then performance would drop and static energy consumption would rise, outweighing the benefits of lower dynamic energy consumption.

Recall that in section 5 we found the microarchitecture configuration with the minimum ED across all programs. This had width 4 which is good for both energy (figure 12(a)) and performance (figure 11(a)). It had a large register file with only 4 read ports to it (good for energy, figure 12(d)) and large branch predictor. However, it also had a large L2 cache which should be bad for energy (figure 12(k)) but in this case it has had to make a trade-off between higher energy and higher performance (figure 11(e)).

8. Related Work

In order to reduce simulation cost whilst maintaining enough accuracy for design space exploration, analytic models have been proposed [14, 19]. Unfortunately these approaches require a large amount of knowledge about the microarchitecture they attempt to explore and, furthermore, they need to be built by hand. When major changes are made to the microarchitecture design the models need to be updated [20]. In contrast, our technique builds such a predictor automatically and can easily accommodate any future microarchitecture design changes.

Sampling techniques [23, 27] approach the problem from a different angle by reducing the number of instructions needing to be simulated. This dramatically decreases the time required for simulation. A statistical analysis of the program's trace is performed to combine results from the instructions actually run. This approach is orthogonal to our technique and, in fact, we do use SimPoint [23] to speed up the simulations we perform both to train our predictor and to verify it.

Statistical simulation [6, 21, 22] is based on the same idea as sampling but goes a step further. The simulator is modified so that it symbolically executes the instructions based on a statistical model. This technique requires the extraction of program characteristics that are related to the microarchitecture under consideration. Thus, if any major changes occur, new features could be needed to continue to characterise the program. Eyerman *et al.* [7] focus their two-phase search using statistical simulation. Search techniques such as this can be used seamlessly with our approach.

There have been many recently proposed schemes for microarchitecture design space exploration based on linear regressors [12], artificial neural networks [10, 11], radial basis functions [13] and spline functions [16, 17]. The linear regressor approach [12] is, in fact, simply used to identify the key parameters in the design space. No measurements are given as to its accuracy and as such it can only be used to give hints to the designer. This is similar to the character-

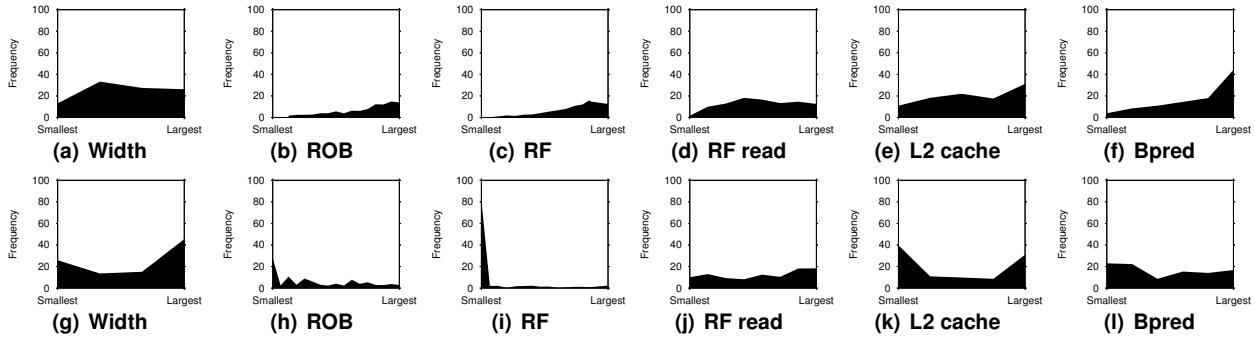


Figure 11. The frequency each parameter design point occurs in the 1% of configurations for each benchmark having the best (a-f) and worst (g-l) number of cycles.

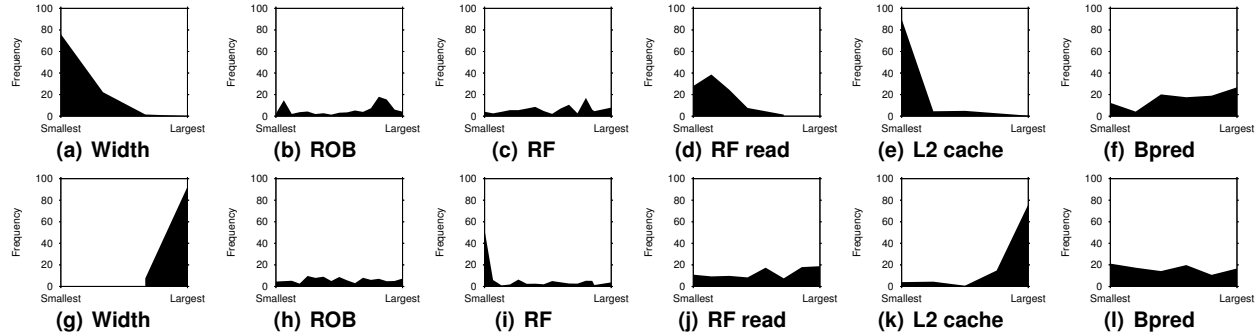


Figure 12. The frequency each parameter design point occurs in the 1% of configurations for each benchmark having the best (a-f) and worst (g-l) energy.

isation of the space that we conduct in section 7. The other schemes are similar in terms of accuracy [18, 26]. However, none of these papers characterise the space that they are exploring by showing the correlation of microarchitectural parameters to the best configurations. Furthermore, none of these papers show how to use their predictors to find good microarchitectural configurations by searching the design space, so we do not know how useful they are.

A fundamental difference between our scheme and these performance predictors resides in the fact that our model characterises the architecture space independently of the programs being simulated, rather than modelling the program-specific architectural space. This enables our approach to predict new programs with low overhead and high accuracy.

An interesting approach taken by Hoste *et al.* [8] uses a linear model to combine program design spaces, clustering benchmarks based on program features. We do not use features in our approach because they can be difficult to identify and might vary depending on the architecture under consideration. Hence our scheme is more versatile since it can be applied to any program and architecture.

Close to our work is using the cross-program learning approach to predict the performance of a new program on an unseen architecture [15]. Their model, however, has to be retrained when a new program is considered and no comparison is made with existing approaches. In addition, their predictor only achieves a 2% improvement over simply predicting the average of the training data, thus showing little cross-program learning. Other work has applied this type of learning to the software optimisation space for learning across programs [5].

Finally, Bird *et al.* [2] characterise the SPEC CPU 2006 benchmark suite in terms of how the programs stress the branch predictor, caches and features specific to the processor they use. Our design space characterisation in section 7 differs from theirs in that we vary the microarchitectural parameters and evaluate the effect on different microarchitectural structures.

9. Conclusions

This paper has proposed a novel approach to design space exploration using prior knowledge to predict energy, cycles or ED. We showed, when predicting performance, in terms of cycles, our architecture-centric model has a rela-

tive mean absolute error of just 7% and a correlation coefficient of 0.95, significantly out-performing a recently proposed program-specific predictor which has an rmae of 24% and correlation coefficient of 0.55. Using our model, we can also accurately predict the best microarchitectural configuration for ED across a range of programs in our randomly-selected space after only a further 3 simulations per benchmark, compared to a further 103 simulations per benchmark for the program-specific predictor. We address the cost of the off-line training our model requires and show that given the exact same training budget our predictor still has a better rmae than the program-specific predictor. Finally, we characterise the design space we have considered, showing how the varying microarchitectural parameters impact on cycles, energy and ED.

In conclusion, our architecture-centric approach can accurately predict the performance, energy or ED of a range of programs within a massive microarchitectural design space, requiring just 32 simulations as a signature from any new program and out-performing all other approaches.

References

[1] T. Austin. The simplescalar toolset. http://www.simplescalar.com.

[2] S. Bird, A. Phansalkar, L. K. John, A. Mericas, and R. Indukuru. Performance characterization of SPEC CPU benchmarks on intel's core microarchitecture based processor. In *SPEC Benchmark Workshop*, 2007.

[3] C. Bishop. *Neural Networks for Pattern Recognition*. Oxford University Press, 2005.

[4] D. Brooks, V. Tiwari, and M. Martonosi. Wattch: A framework for architectural-level power analysis and optimizations. In *ISCA-27*, 2000.

[5] J. Cavazos, C. Dubach, F. Agakov, E. Bonilla, M. F. P. O'Boyle, G. Fursin, and O. Temam. Automatic performance model construction for the fast software exploration of new hardware designs. In *CASES*, 2006.

[6] L. Eeckhout, R. H. B. Jr., B. Stougie, K. D. Bosschere, and L. K. John. Control flow modeling in statistical simulation for accurate and efficient processor design studies. In *ISCA-31*, 2004.

[7] S. Eyerman, L. Eeckhout, and K. D. Bosschere. Efficient design space exploration of high performance embedded out-of-order processors. In *DATE*, 2006.

[8] K. Hoste, A. Phansalkar, L. Eeckhout, A. Georges, L. K. John, and K. D. Bosschere. Performance prediction based on inherent program similarity. In *PACT*, 2006.

[9] C.-H. Hsu, W. chun Feng, and J. S. Archuleta. Towards efficient supercomputing: A quest for the right metric. In *Proceedings of the High-Performance Power-Aware Computing Workshop*, 2005.

[10] E. İpek, B. R. de Supinski, M. Schulz, and S. A. McKee. An approach to performance prediction for parallel applications. In *Euro-Par*, 2005.

[11] E. İpek, S. A. McKee, R. Caruana, B. R. de Supinski, and M. Schulz. Efficiently exploring architectural design spaces via predictive modeling. In *ASPLOS-XII*, 2006.

[12] P. J. Joseph, K. Vaswani, and M. J. Thazhuthaveetil. Construction and use of linear regression models for processor performance analysis. In *HPCA-12*, February 2006.

[13] P. J. Joseph, K. Vaswani, and M. J. Thazhuthaveetil. A predictve performance model for superscalar processors. In *MICRO-39*, 2006.

[14] T. S. Karkhanis and J. E. Smith. A first-order superscalar processor model. In *ISCA-31*, 2004.

[15] S. Khan, P. Xekalakis, J. Cavazos, and M. Cintra. Using predictive modeling for cross-program design space exploration in multicore systems. In *PACT*, 2007.

[16] B. C. Lee and D. M. Brooks. Accurate and efficient regression modeling for microarchitectural performance and power prediction. In *ASPLOS-XII*, 2006.

[17] B. C. Lee and D. M. Brooks. Illustrative design space studies with microarchitectural regression models. In *HPCA-13*, 2007.

[18] B. C. Lee, D. M. Brooks, B. R. de Supinski, M. Schulz, K. Singh, and S. A. McKee. Methods of inference and learning for performance modeling of parallel applications. In *PPoPP-12*, 2007.

[19] D. B. Noonburg and J. P. Shen. Theoretical modeling of superscalar processor performance. In *MICRO-27*, 1994.

[20] D. Ofelt and J. L. Hennessy. Efficient performance prediction for modern microprocessors. In *SIGMETRICS*, 2000.

[21] M. Oskin, F. T. Chong, and M. Farrens. Hls: combining statistical and symbolic simulation to guide microprocessor designs. In *ISCA-27*, 2000.

[22] R. Rao, M. Oskin, and F. T. Chong. Hlspower: Hybrid statistical modeling of the superscalar power-performance design space. In *HiPC*, 2002.

[23] T. Sherwood, E. Perelman, G. Hamerly, and B. Calder. Automatically characterizing large scale program behavior. In *ASPLOS-X*, 2002.

[24] The Standard Performance Evaluation Corporation (SPEC) CPU 2000 Benchmark Suite. http://www.spec.org/cpu2000/.

[25] D. Tarjan, S. Thoziyoor, and N. P. Jouppi. Cacti 4.0. Technical Report HPL-2006-86, HP Laboratories Palo Alto, 2006.

[26] K. Vaswani, M. J. Thazhuthaveetil, Y. N. Srikant, and P. J. Joseph. Microarchitecture sensitive empirical models for compiler optimizations. In *CGO*, 2007.

[27] R. E. Wunderlich, T. F. Wenisch, B. Falsafi, and J. C. Hoe. Smarts: Accelerating microarchitecture simulation via rigorous statistical sampling. In *ISCA-30*, 2003.