

# Rethinking Processor Design: Parameter Correlations

Nana B. Sam, Sally A. McKee  
Computer Systems Lab  
Cornell University  
Ithaca, NY 14853, USA  
sam@csl.cornell.edu

Prabhakar Kudva  
IBM TJ Watson Research Center  
Yorktown Heights, NY 15098, USA  
kudva@us.ibm.com

**Abstract**—Computer architects rely heavily on simulation to explore increasingly complex design spaces. Keeping simulations within tractable limits forces architects to evaluate only subsets of design parameters, which must be carefully chosen to yield accurate reflections of the benefits and costs of a new architecture. As a first step in formulating a methodology to appropriately evaluate processor design parameters for simulation, we develop an approach based on statistical correlation analysis, which measures the interaction between processor parameters. The resulting *parameter correlation matrix (PCM)* allows architects to determine how parameters affect each other, which parameter variations introduce bottlenecks and where, and what tradeoffs can be made to maintain performance while optimizing power consumption. This approach can be used to constrain a simulation model, narrowing the design space for optimization. As an initial demonstration, we develop and evaluate a preliminary analytic model for the instruction fetch unit (IFU).

## I. INTRODUCTION

Forecasting expected benefits and costs is critical in microprocessor design: without reliable predictions, poor decisions may result. Accuracy requires identifying all design parameters contributing to a design’s performance and costs, as well as assessing which have greatest influence. Values of these critical parameters dictate a design’s success or failure: changing values can considerably affect expected gains, turning a viable design into a non-viable one. To avoid building a research prototype for each design, designers use detailed simulations to explore design spaces when developing new processors or improving existing ones.

Much effort has been invested in building microarchitecture models that closely mimic hardware. Accuracy (or relative accuracy with respect to alternative designs) is essential [1][2], but so is the manner in which models are used to evaluate designs. Microprocessors are increasing in complexity due to aggressive microarchitectures, power and thermal limits, reliability, and the trend towards multicore chips. The diverse programs used in architectural simulation represent many

billions of dynamically executed instructions. Detailed simulators run many orders of magnitude slower than real systems: even a single experiment takes days to weeks.

Using reduced input sets can reduce experiment time, but cannot replace representative modeling of full workloads for final decisions: behavior varies significantly across input sets for many standard benchmark applications [10][6]. To compensate, designers often model only portions of programs. Selecting representative portions is crucial, since program execution usually consists of several phases with diverse behaviors [10]. Other approaches exploit the repetitive nature of applications [3][7].

Statistical simulation [4] estimates IPC based on statistically generated synthetic workloads. Computation time is several orders of magnitude less than full benchmark simulation, but models a lower level of detail, and has not yet been embraced by industry. Yi et al. [12] rank parameter importance via Plackett Burman analysis to reduce the number of simulations in a sensitivity study, but they do not quantify and interpret specific interactions. Fields et al.’s [5] use of higher-order models to consider multiple parameter interactions based on critical path estimates is most closely related to what we propose here. In contrast, our analytic approach requires no additional hardware support, and it complements all these approaches for reducing large design spaces or reducing time per experiment. Furthermore, its application is straightforward.

To address design space complexity, we develop a methodology for measuring parameter interaction based on statistical correlation analysis, and demonstrate how parameter correlations can guide optimization by developing and evaluating an analytic model for the instruction fetch unit (IFU).

## II. A MOTIVATING EXAMPLE

To meet time constraints, architects usually evaluate their techniques by varying a handful of parameters directly affecting the system component being modified.

TABLE IV. IPC FOR VARYING I-BUFFER SIZES AND NUMBER OF BTB ENTRIES

I-buf size	BTB entries = 16			BTB entries = 32			BTB entries = 64		
	BHT size ( $\log_2$ )			BHT size ( $\log_2$ )			BHT size ( $\log_2$ )		
	12	14	16	12	14	16	12	14	16
16	0.97350	0.98014	0.98201	0.97471	0.98136	0.98323	0.97487	0.98153	0.98340
32	0.97555	0.98185	0.98371	0.97651	<b>0.98310</b>	0.98497	0.97666	<b>0.98326</b>	0.98513
64	<b>0.97583</b>	<b>0.98203</b>	<b>0.98378</b>	<b>0.97710</b>	0.98298	<b>0.98525</b>	<b>0.97726</b>	0.98318	<b>0.98543</b>

TABLE V. EFFECTS OF PARAMETER INTERACTIONS FOR DIFFERENT AVERAGE FETCH RATES

	I-cache	BP	I-buf	prefetch	I-cache	BP	I-buf	prefetch	I-cache	BP	I-buf	prefetch
	avg fetch rate ~ 2 IPC, sample = 1404				avg fetch rate ~ 3 IPC, sample = 1404				avg fetch rate ~ 4 IPC, sample = 2808			
	BP	0.017				0.010				-0.004		
I-buf	-0.740	-0.642			-0.746	-0.634			-0.886	-0.393		
prefetch	-0.860	-0.009	0.723		0.860	-0.005	0.717		-0.862	0.004	0.845	

“One-at-a-time” experimental design, in which only one parameter is varied per experiment while others are held constant, can mask important effects due to parameter interactions. Furthermore, a constant parameter at an extreme value may overshadow effects of parameters under investigation. To illustrate the problem, Table I shows geometric mean IPC of 26 SPECcpu2000 programs for varying branch target table entries (BTB), branch history table entries (BHT), and I-buffer entries (I-buffer). Bold values indicate the knee of the curve. For a BTB of size 16, IPC begins to saturate at an I-buffer of 64, regardless of BHT size. For BTBs of size 32 and 64, when the BHT is of size 14, optimal IPC results from an I-buffer of 32. Assuming a designer is investigating appropriate I-buffer size for optimum performance, the choice could be 32 or 64, depending on the baseline. For 42% of the SPEC programs a larger I-buffer is only beneficial at a larger BHT. If designers independently optimize the I-buffer and the BHT with “one-at-a-time” experiments, local optima may not coincide with the global optimum.

The designer could vary all parameters simultaneously, but this is computationally prohibitive: e.g., measuring effects of 50 parameters, each of which assumes only three values, requires  $3^{50}$  simulations. The magnitude of modern design spaces combined with increasing design complexity (which leads to slow models) calls for methodologies to assess parameter interactions and exploit them in design evaluations.

### III. PARAMETER CORRELATIONS

To assess potential benefit from enhancing a design, architects spend considerable time performing simulations and varying parameters. Parameters are often varied in a single direction due to time constraints. Costs of performance enhancements are quickly exceeding benefits, thus architects must improve modeling methodologies to avoid mispredicting effects of an

enhancement. Architects know intuitively that some parameters interact, but relatively little work has characterized these interactions to guide simulation or adapt architectural techniques. We provide a measure of these interactions in an easily interpretable manner via a processor *parameter correlation matrix* (PCM).

Examining all interactions within the microprocessor quickly becomes intractable. Step one in our approach to managing this complexity divides the processor into decoupled units, examining interactions within each. One can design metrics to characterize performance for a well decoupled unit. For instance, average fetch rate is a well understood metric characterizing the IFU. Step two identifies *subunits*: the IFU is comprised of instruction cache (I-cache), prefetch buffer, I-buffer, branch predictor (BP), and instruction TLB (I-TLB). Step three identifies *submetrics* defining performance and costs of subunits. For the IFU example, these could be I-cache miss rate, prefetch miss rate, I-buffer stall rate, BP miss rate, and I-TLB miss rate. Cost metrics could be power consumed, subunit delay, and area.

Understanding how parameters interact requires understanding how submetrics correlate. Since submetrics are directly impacted by parameters, *it is reasonable to expect correlations to exist between submetrics in the same manner as between the corresponding parameters*. Our submetrics are continuous variables, thus it is easier to establish statistical correlations at that level.

We first perform a multifactorial simulation based on parameters of interest to obtain submetric values at different design points. Then we determine the Pearson Product-Moment Correlation Coefficient,  $r$ , between all submetric pairs. This widely used measure of correlation shows the degree of relationship between two variables when the dependent variable is at the interval or ratio level. The correlation coefficient ranges from  $[-1,1]$

(perfect negative correlation to perfect positive correlation), and is calculated by summing the products of the deviations of the scores from the mean:

$$r = (\sum(X-\mu_x)(Y-\mu_y))/(N\sigma_x\sigma_y). \quad (1)$$

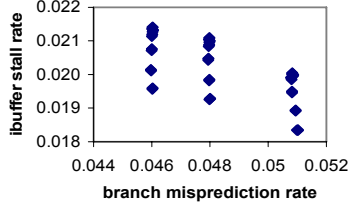


Figure 1. Branch misprediction and I-buf stall correlations at  $IPC \approx 1$

Fig. 1 shows a scattergram of branch misprediction against I-buffer stall rates for an average fetch rate of  $IPC \approx 1$  for all SPEC programs. The submetrics correlate negatively ( $r = -0.642$ ): PCMs represent such relationships. Table II shows correlation matrices between instruction cache miss, branch misprediction rate, I-buffer stall, and prefetch miss rates for three I-fetch rates. Note that correlations are fairly consistent for different average fetch rates. Even though we consider a wide range of parameter values, we find that two or three values per parameter suffice to establish submetric correlations. This reinforces the notion that finding correlations at the submetrics level suffices.

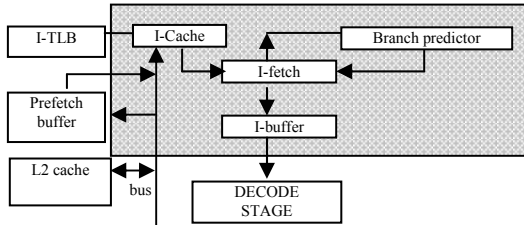


Figure 2. Instruction fetch microarchitecture (shaded section encompasses analytic model).

#### IV. ANALYTIC INSTRUCTION FETCH MODEL

In this section we present an IFU analytic model for optimization and tradeoff experiments during early stage design. Fig. 2 shows a block diagram of the model used in this analysis. Wallace and Bagherzadeh [11] describe the fetch rate of a simple I-cache with line width  $n$  as:

$$ICFR = n / (P(br)n + 1 - P(br)). \quad (2)$$

$P(br)$  represents the probability of a control transfer instruction, and cache line fills take one cycle. Equation (2) holds for the IFU if fetch width equals line width, with no stalls. However, the fetch unit may stall due to branch mispredictions, a full I-buffer, I-cache or TLB misses, or reaching the limit on in-flight instructions.

TABLE IV. INPUTS TO (3)

$i$	I-cache	Branch predictor	I-buffer
$P_i(\text{access})$	1	$P(br)$	1
$t_i(\text{access})$	Cache access Latency	Predictor access latency	Buffer access latency
$P_i(\text{stall})$	Cache miss rate	Branch misprediction rate	Probability of full buffer
$t_i(\text{stall})$	$\sum_j P_j(\text{hit}) * t_j(\text{hit})$ for $j$ in {prefetch buffer, L2 cache, memory}	Branch misprediction penalty	Expected stall time

Let the IFU fetch period,  $T_{\text{fetch\_period}}$  be the time between an I-cache access and when the instruction is placed in the I-buffer. The time to complete a fetch is:

$$T_{\text{fetch\_period}} = \sum P_i(\text{access})(t_i(\text{access}) + P_i(\text{stall})t_i(\text{stall})) \quad \text{for } i \text{ in \{I-cache, branch predictor, I-buffer\}. \quad (3)$$

$P_i(\text{access})$  is the probability of accessing unit  $i$  in a given fetch cycle,  $t_i(\text{access})$  is the time to access unit  $i$ ,  $P_i(\text{stall})$  is the probability of a stall caused by unit  $i$ , and  $t_i(\text{stall})$  is the expected time to service such a stall. Table III gives inputs to (3).  $P_j(\text{hit})$  is the probability of a hit in unit  $j$ , and  $t_j(\text{hit})$  is the hit latency of unit  $j$ . The model can be expanded by expressing cache miss rate as a function of cache parameters (size and associativity), branch misprediction rate as a function of BHT and BTB sizes, and I-buffer stall rate as a function of buffer size and instruction dispatch or completion rate. The rate of the instruction fetch unit,  $IFU\_FR$ , is:

$$IFU\_FR = ICFR / T_{\text{fetch\_period}} \quad (4)$$

Several other analytic models have been developed to improve early stage microprocessor design. For instance, Michaud et al. [9] develop a model to express ILP as a function of window size in superscalar processors. They seek insight into the relationship between issue and fetch widths. Our model also covers the front end, and, unlike theirs, it includes the branch predictor and I-cache.

#### A. Evaluating the Model

We use Turandot, a parameterized, out-of-order, eight-way superscalar simulator [8]. The baseline architecture has a 64-entry I-buffer, with 150 maximum in-flight instructions, 128-entry two-way L1 and 1024-entry four-way L2 instruction and data TLBs, a 64KB direct-mapped L1 I-cache, and a 2MB four-way L2 cache. Cache lines are 128B. There are three integer, one floating-point, two memory, and one branch units. The 16K-entry branch predictor has a 16-entry return address stack, and a 32-entry direct-mapped BTB. Mispredictions take three cycles.

TABLE IV. PREDICTED AVERAGE FETCH RATE OF ANALYTIC MODEL

Program	Configuration A		Configuration B	
	Predicted fetch rate	% error	Predicted fetch rate	% error
ammp	1.546	-4.256	4.735	1.974
applu	1.904	5.461	7.974	-2.474
apsi	1.722	8.449	5.868	6.283
art	2.242	-5.315	7.642	-1.083
equake	1.938	-5.337	7.004	-3.409
facerec	1.285	25.146	3.434	40.844
fma3d	2.019	-3.342	7.440	-0.350
galgel	1.711	29.906	6.663	9.485
lucas	2.424	-14.544	8.421	-6.738
mesa	1.231	7.960	3.423	-3.563
mgrid	1.888	9.989	7.881	-0.536
sixtrack	1.293	31.566	3.233	59.470
swim	1.663	28.346	7.914	-1.186
wupwise	1.586	4.061	5.018	9.056
bzip2	1.057	-3.548	2.621	-6.907
crafty	1.142	-0.574	3.467	-3.125
eon	1.161	5.390	3.487	6.000
gap	1.330	-12.740	3.044	-8.877
gcc	1.227	-2.430	2.512	9.618
gzip	1.162	-5.225	2.632	-12.622
mcf	1.153	-34.224	2.092	-32.860
parser	1.020	-20.591	1.791	-25.065
perlbmk	1.123	-9.420	2.200	-7.036
twolf	0.872	7.363	3.044	-12.803
vortex	1.405	14.207	2.744	19.778
vpr	1.064	-6.864	2.426	6.429

Table IV compares average fetch rate predicted by our model to that obtained from simulation. Configuration A is the baseline with a 16-entry I-buffer and a 4K-entry branch predictor with a 16-entry branch target buffer. Configuration B is identical to A but with a 64-entry I-buffer. 65% and 73% of the programs have error rates  $\leq 10\%$  for each configuration, respectively.

### B. Optimizing the Model

When a model is “trustworthy”, it can be used to find parameter values that optimize the model’s results. Since we require our model to behave as close to the simulator as possible (since it will be used to cull the simulation design space), we must introduce a representation of the simulator’s real behavior into the analytic model itself. We propose to use the parameter correlation matrix as such a representation. Note that the PCM provides the real relationship between the submetrics (with respect to the simulator), and these are also inputs ( $P_i(stall)$ ) to the analytic model. The correlation matrix not only bounds the design space, but causes the model to approach the behavior of the simulator. The optimization procedure is:

1. Consider a set of unit parameters.
2. Derive submetrics.
3. If the submetrics meet conditions set by the correlation matrix, add parameters to *parameter set*. Otherwise, discard.
4. Iterate 1–3 until stopping conditions are met.
5. Optimize model within limits of *parameter set*.

Applying this algorithm to the IFU for the mcf application reduced error rate by 5.8% after only a few iterations. The parameter set of interest guides the architect in determining stopping conditions. Solving this optimization problem is beyond the scope of this preliminary work.

## V. CONCLUSIONS AND FUTURE WORK

We describe a methodology for identifying correlations between microarchitectural parameters and presenting them in a parameter correlation matrix (PCM). Knowing parameter correlations allows designers to identify where potential bottlenecks may be introduced as parameters vary or to economically reduce sizes of non-bottleneck resources, saving area and energy. Correlation information also benefits design optimization: analytic models constrained by correlations can quickly reduce large design spaces. Detailed analyses can then be performed within the smaller spaces.

This paper presents a preliminary analytic model for the IFU and suggests how the PCM can be used to improve convergence in finding optimum design points for this unit.

## REFERENCES

- [1] P. Bose, T.M. Conte, T. M. Austin, ed., “Identifying design bugs: processor modeling and validation”, IEEE Micro special issue, vol. 19, no. 3, May/June 1999
- [2] R. Desikan, D. Burger, S. Keckler, “Measuring experimental error in microprocessor simulation”, Int’l. Symp. on Computer Architecture, July 2001, pp. 266-277.
- [3] E. Duesterwald, C. Casççaval, S. Dwarkadas. “Characterizing and predicting program behavior and its variability”, Int’l. Conf. on Parallel Architectures and Compilation Techniques, September 2003, pp. 220-231.
- [4] L. Eeckhout, S. Nussbaum, J.E. Smith, K. De Boschere, “Statistical simulation: Adding efficiency to the computer designer’s toolbox”, IEEE Micro, vol. 23, no.5, September/October 2003, pp. 26-38.
- [5] B.A. Fields, R. Bodik, M.D. Hill, C.J. Newburn, “Using iteration costs for microarchitectural bottleneck analysis”, Int’l. Symp. on Microarchitecture, December 2003, pages 228-242.
- [6] A. KleinOowski and D. Lilja, “MinneSPEC: A new SPEC benchmark workload for simulation-based computer architecture research”, *Computer Architecture Letters*, 1, June 2002.
- [7] W. Liu, M. C. Huang. “EXPERT: Expedited simulation exploiting program behavior repetition”, Int’l. Conf. on Supercomputing, June 2004, pp. 126-135.
- [8] M. Moudgill, P. Bose, J. Moreno, “Validation of Turandot, a fast processor model for microarchitecture exploration”, Int’l. Performance, Computing, and Communications Conf., February 1999, pp. 452-457.
- [9] P. Michaud, A. Sez nec, S. Jourdan, “An exploration of instruction fetch requirement in out-of-order superscalar processors”, Int’l. J. of Parallel Programming, vol. 29, no. 1, February 2001, pages 35-58.
- [10] T. Sherwood, E. Perelman, G. Hamerly, B. Calder, “Automatically characterizing large scale program behavior”, Int’l. Conf. on Architectural Support for Programming Languages and Operating systems, October 2002, pp. 89-97.
- [11] S. Wallace, N. Bagherzadeh, “Modeled and measured instruction fetching performance for superscalar microprocessors”, IEEE Trans. On Parallel and Distributed Systems, vol. 9, no. 6, June 1998, pp. 570-578.
- [12] J.J. Yi, D.J. Lilja, D.M. Hawkins. “A statistically rigorous approach for improving simulation methodology”, IEEE Int’l. Symp. on High Performance Computer Architecture, February 2003, pp. 281-292.