Efficient System Design using the Statistical Analysis of Architectural Bottlenecks Methodology

Manish Arora*

Feng Wang[†]

Bob Rychlik[†]

Dean M. Tullsen*

* Department of Computer Science and Engineering
University of California San Diego, La Jolla, CA, U.S.A.

{marora, tullsen}@cs.ucsd.edu

[†] Qualcomm Incorporated

5775 Morehouse Drive, San Diego, CA, U.S.A.

{fwang, brychlik}@qualcomm.com

Abstract—CPU processor design involves a large set of increasingly complex design decisions. Doing full, accurate simulation of all possible designs is typically not feasible. Prior techniques for sensitivity analysis seek to identify the most critical design parameters, but also struggle to handle the increasing design space well. They can be overly sensitive to the starting fixed point of the design, can still require a large number of simulations, and do not necessarily account for the cost of each design parameter.

The Statistical Analysis of Architectural Bottlenecks (SAAB) methodology simultaneously analyzes multiple parameters and requires a small number of experiments. SAAB leverages the Plackett and Burman analysis method, but builds upon the technique in two specific ways. It allows a parameter to take multiple values and replaces the unit-less impact factor with a cost-proportional impact value. This paper applies the SAAB methodology to the design of a mobile processor sub-system. It considers area and power cost models for the design.

Index Terms—Plackett and Burman, Bottleneck Analysis, Cost Optimized Design.

I. INTRODUCTION

The architecture of modern CPU processors involves increasingly complex design decisions. The introduction of multicore processors significantly exacerbates the design space exploration problem, because it allows the architect to not only vary the number of cores, but also to have much more flexibility in the size of the target cores. Accurate simulation of all design points is often not feasible for commercial design decisions; as a result, sensitivity analysis is a commonly used technique in architectural exploration. This method varies a single parameter value at a time (or a small set) across its design space while keeping all other parameter values fixed, in order to measure the effect of the varying parameter. However, sensitivity analysis suffers from a variety of problems.

First, if design parameters are not independent, the reported importance is sensitive to the choice of the fixed point of reference. For example, the parameters of the L2 cache may appear critical when the L1 cache is small and unimportant when it is large. This makes results dependent on the specific fixed choice of parameters.

Statistical techniques such as the Plackett and Burman (P&B) method [23] have been proposed to address the shortcomings of sensitivity analysis [28], [29]. The P&B method takes in a low value and high value for each parameter in the system. It then runs a defined set of experiments utilizing

those endpoints for all of the parameters simultaneously and calculates an impact factor for each. For each parameter of the design space analysis, the P&B impact factor represents the percentage contribution to performance for that parameter, which is obtained by simulating at both the low and high values for the parameter. P&B is able to provide this information from only $O(N)$ experiments, where N is the number of varying parameters in the system. The brute force full factorial design methods require exponential experiments.

We were attracted to the P&B method for two reasons: 1) Ability to analyze the design space using a linear set of experiments. 2) The method is not iterative in nature. We execute independent experiments in parallel and the total simulation time is bound by the longest experiment. Since simulation times for each experiment can be several days, any iterative scheme such as [22], [25] quickly increases our design time from days to months. Thus we started by applying the most recently proposed P&B methodology [28], [29] to guide the design of our mobile processor sub-system. However, we found the existing proposals to have key shortcomings:

(1) There is little direction in choosing P&B +1 (max) and -1 (min) values. Literature suggests using values greater than the nominal range, but the reported impact factors can be highly skewed by these choices – selecting too wide a range will typically overstate its importance.

(2) P&B reports impact values for variations from the low value to the high value, but no intermediate values. This assumes a linear relation, but in fact this is not typically the case. Many parameters (e.g., cache size) are most sensitive at small sizes. These parameters often have a “knee” above which marginal gains are reduced.

(3) The impact factors are essentially “unitless”, meaning that they do not account for the cost of varying the parameter. In fact, it is dangerous to directly compare impact factors. For example, if we find that the L2 cache size has an impact factor of 20% vs 10% for the L1 cache, that does not necessarily tell us how to achieve the highest gain given constrained resources, since doubling the L2 size costs substantially more area than doubling the L1 cache.

In this work, we describe our design space and performance evaluation methodology which we call Statistical Analysis of Architectural Bottlenecks, or SAAB. SAAB uses the Plackett

CPUType	CPUFreq	L2Size	L2Freq	MEMFreq	MEMInt	INTFreq	Execution Time (in msec)
+1	+1	+1	-1	+1	-1	-1	425.7
-1	+1	+1	+1	-1	+1	-1	448.7
-1	-1	+1	+1	+1	-1	+1	304.3
+1	-1	-1	+1	+1	+1	-1	635.2
-1	+1	-1	-1	+1	+1	+1	465.4
+1	-1	+1	-1	-1	+1	+1	545.1
+1	+1	-1	+1	-1	-1	+1	776.8
-1	-1	-1	-1	-1	-1	-1	982.9
-1	-1	-1	+1	-1	+1	+1	807.2
+1	-1	-1	-1	+1	-1	+1	510.0
+1	+1	-1	-1	-1	+1	-1	960.7
-1	+1	+1	-1	-1	-1	+1	492.2
+1	-1	+1	+1	-1	-1	-1	494.0
-1	+1	-1	+1	+1	-1	-1	605.9
-1	-1	+1	-1	+1	+1	-1	481.2
+1	+1	+1	+1	+1	+1	+1	245.4
5251	339103	2307347	545701	1834481	2941	887939	Abs Impact
0.1	5.7	38.9	9.2	31.0	0.1	15.0	Impact (%)

TABLE I

P&B DESIGN MATRIX EXAMPLE DESIGN MATRIX FOR 7 PARAMETERS.

and Burman method at its core and retains most of the advantages that method offers, such as the simultaneous varying of all parameters without bias toward a fixed point, and of course fewer experiments. However, we have introduced several key improvements:

(1) SAAB allows us to examine intermediate points and identify the knee of the performance impact curve.

(2) SAAB incorporates cost models into the design space evaluation process, allowing architects to discern cost effective design choices. In this work, we introduce both area and power cost models.

(3) SAAB uses at worst $O(KN^2)$ experiments, with the typical number being $O(KN)$, where N is the number of parameters and K is the maximum number of choices for any parameter.

The rest of this paper is organized as follows. Section 2 explains the P&B method in detail. Section 3 provides the experimental verification methodology and verifies the applicability of the P&B method to our system. Section 4 explains the components of our mobile processor system under design including the benchmarks used for the study. Section 5 explains the SAAB methodology for changing P&B bounds. Section 6 details the incorporation of cost models into the system and system design results. In section 7 we discuss related work followed by conclusions in section 8.

II. PLACKETT AND BURMAN METHOD

In this section, we introduce P&B and show its applicability to our domain – CPU design space exploration. In the next section we will verify the P&B method by comparing it to a full factorial design. P&B is a method for finding the dependence of some quantity to a set of independent variables, using a minimal number of experiments. It varies N parameters, with two values each, over $2N$ simulations, where N is a multiple of 4. We use the P&B design with foldover [20].

A. P&B Design Matrix and Evaluating Impact

Table I shows an example of a P&B design consisting of 7 parameters, whose effects are analyzed using a set of 16 experiments. It is simple to obtain design matrices for other sizes [23] [28]. For the system under test, the architect chooses a low end value (-1) and high end value (+1) for each parameter. Table II describes our parameters, and gives

Parameter	Description	Low Value (-1)	High Value (+1)	Mid Value (0)
CPUType	CPU model configurations	Config A	Config B	-
CPUFreq	CPU frequency	1GHz	2GHz	1.5GHz
L2Size	Shared L2 cache size	256KB	1MB	512KB
L2Freq	Shared L2 cache frequency	1GHz	2GHz	1.5GHz
MEMFreq	Memory system frequency	266MHz	800MHz	533MHz
MEMInt	Memory interleaving schemes	Scheme 1	Scheme 2	-
INTFreq	Shared interconnect frequency	200MHz	800MHz	500MHz

TABLE II

P&B PARAMETER VALUES P&B REQUIRES THE SELECTION OF HIGH (+1) AND LOW (-1) VALUES (MID VALUES ARE USED IN THE VERIFICATION OF RESULTS AS IN SECTION 3.1).

Benchmark	Description	Suite
Art	Image recognition	SPEC 2000 [3] MinneSPEC input set [14]
Mcf	Combinatorial optimization	SPEC 2000 [3] MinneSPEC input set [14]
Bzip2	Compression decompression	SPEC 2000 [3] [14]
Nat	Network address translation	EEMBC [2]
IPReassembly	IP packet reassembly	EEMBC [2]
Pktcheck	IP header validation	EEMBC [2]
Samurai	Mobile gaming (CPU part only)	3D Mobile Mark 2006 [1]
Proxycon	Mobile gaming (CPU part only)	3D Mobile Mark 2006 [1]
MemCS	Memory copy and set	Internal Microbenchmark
MemRW	Memory read and write	Internal Microbenchmark
Tree	Tree traversal	Internal Microbenchmark
Memcpy	Optimized memory copy	Internal Microbenchmark

TABLE III

BENCHMARK DESCRIPTION WE USED 12 BENCHMARKS FROM VARIOUS SUITES AND INTERNAL MICROBENCHMARKS TO VERIFY P&B.

an example of low end and high end values. Using the chosen +1 and -1 values, and experiments specified by the design matrix, measurements are collected for each benchmark. We use execution time as our metric. The rightmost column in Table I shows the execution time for one specific benchmark. We use an in-house hardware-verified, cycle-accurate, SOC system simulator for our simulations.

After performing the simulations, the effect of parameters is computed by multiplying each parameter's P&B value by the measurement result and summing up the products across all configurations. For example, for the parameter CPUType the absolute impact can be calculated as:

$$\text{Impact} = \text{abs}([+1 \times 425.7] + [-1 \times 448.7] + \dots + [+1 \times 245.4]) = 5251$$

Once we have the impact values for all the 7 parameters we can find the percentage contributions of impact by summing impacts across parameters and dividing the absolute impact of each parameter by the sum of all impacts:

$$\text{Impact}(\text{CPUType}) = 100 \times 5251 / (5251 + 339103 + \dots) = 0.1\%$$

We show this in the last row of Table I. The percentage contribution of each impact will be referred to simply as *Impact* in this paper.

B. Application of the P&B Design Method

We apply the P&B procedure over the 12 benchmarks listed in Table III and the results are shown in Figure 1. The y-axis in the figure is the impact for each parameter. CPUFreq has high impact on the compute intensive benchmarks Bzip2 and Tree, whereas memory system microbenchmarks MemCS, MemRW, and Memcpy see a higher impact from MEMFreq, L2Freq, L2Size, and INTFreq. As seen from the figure, the parameter CPUFreq has the highest average impact across all benchmarks, followed by MEMFreq and so on.

The P&B method produces results with only 16 experiments per benchmark, rather than 2^7 needed for all combinations of parameters. It does so by simultaneously varying all parameters, rather than varying parameters from a single fixed point,

Fig. 1. P&B results for 7 parameter system.

and thus is not overly influenced by behavior around that fixed point. The benefit of reduced experimentation only increases as the system under test becomes more complex. The reduced set of experiments allows us to perform more rigorous analysis or more accurate simulations, since we are severely limited by the speed of our simulation infrastructure.

III. VERIFYING P&B RESULTS

P&B assumes independence of variables, and thus the small number of experiments cannot capture all of the interactions between variables when that assumption is not valid. The assumption of independence is questionable in the case of CPU design exploration – in fact, it is known to be not true for some parameters. The only way to fully capture all possible interactions is to exhaustively run all combinations of design parameters and compare these with P&B results to measure the loss of accuracy.

In doing so, we also increase the number of design choices for some of the parameters to 3 choices (low, mid, and high values). This allows us to perform three validation steps (for the low-mid, mid-high, and low-high ranges). However, we will focus on the low-high validation in this paper to conserve space. Results for low-mid and mid-high evaluations were similar to the low-high validations. Table II shows the values we use. For each of the 12 benchmarks, we run experiments consisting of all possible combinations of parameters ($2 \times 3 \times 3 \times 3 \times 3 \times 3 \times 2 = 972$ experiments per benchmark).

A. Verification using Slopes of Lines Method

To create an equivalent metric to the P&B impact factor for our exhaustive experiments, we have created the *relative slopes of lines* method. To calculate this, we first capture a slope for each parameter which captures the sensitivity of performance to that parameter. To do this, we bin all experiments for a single parameter with respect to the possible values for that parameter, then compute the average value for each bin and draw a line through the values. For example, consider the variations due to L2Size from our 972 experiments, one third of which have cache size of 256KB, 512KB and 1MB respectively. Figure 2 shows all the experiments for the Art

Fig. 2. Performance of L2Size and CPUFreq over 972 experiments.

Parameter	Slope	Impact Using Slopes	Impact Using P&B	Difference (in %)
CPUFreq	20.6	6.1	6.1	0
MEMFreq	81.1	24.0	21.3	2.7
L2Size	114.9	34.0	39.1	5.1
L2Freq	41.7	12.3	11.8	0.5
INTFreq	76.5	22.6	19.6	3.0
CPUType	0.2	0.1	0.3	0.2
MEMInt	3.3	1.0	1.9	0.9

TABLE IV
EXAMPLE OF USING SLOPES TO ESTIMATE IMPACT IMPACT USING SLOPES IS ESTIMATED AS $100.0 \times \text{SLOPE} / (\text{SUM OF ALL SLOPES})$.

benchmark distributed as bins for the 3 choices each of L2Size and CPUFreq.

As we can see from the figure, L2Size has higher impact on performance than CPUFreq (as seen by the larger absolute slope of the lines). The slope shown is just the line through the average values at the endpoints. This is shown by the equations in figure 2. We have only shown the slopes for the lines low - high in the figure. To calculate a measure of relative impact we compare slopes across the various parameters, e.g. slope of CPUFreq line vs that for L2Size, in a manner that is analogous to how P&B derives the impact factor. We calculate the percentage contributions of a particular parameter by finding the proportion of its slope to the sum of slopes of all parameters.

Table IV gives an example of calculating slopes and relative contributions of slopes for a particular benchmark. We also show the P&B impact and absolute difference between the P&B impact and percentage impact found using the slopes method. In this example, we found the maximum difference to be 5.1% (for L2Size) for the single benchmark.

Figure 3 shows the absolute difference over all benchmarks between P&B impact and the impact obtained using the relative slope of lines method. We use 11664 (972×12) simulations for the exhaustive method vs 192 (16×12) simulations for the P&B method. We found a maximum difference for the P&B method of 6.0% and an average difference of 1.1%. The maximum difference observed was for the L2Size parameter. This is not unexpected, as the L2 cache performance gain is both non-linear with size, and likely to be correlated with several factors.

Overall, we find the baseline P&B method to be impressively accurate for processor design space exploration, given the low number of experiments and the known lack of independence of the parameters.

Fig. 3. Error between P&B and contributions of slopes method.

IV. SYSTEM UNDER DESIGN

This section describes the system we evaluate with our design space analysis tools. We apply the SAAB system to a mobile multi-core processor system optimized for web browsing workloads. We could not perform P&B validation on all of these workloads, because of the large number of long-running experiments for the exhaustive method. The infeasibility of doing that type of analysis with the full simulation emphasizes the importance of having a design exploration methodology that minimizes experimentation cost.

The full design space includes four more parameters, many more intermediate points (for later parts of our design study), and an altered set of benchmarks that tend to run longer and are more focused on the target market for this processor. We also introduce multi-threaded benchmarks into the picture. Thus, we will apply SAAB to a design space consisting of 11 parameters over 12 single threaded and multi-threaded browser workloads. Figure 4 shows a block diagram of our system.

Table V shows the parameters of our system. We have added core count, since we are now considering multi-threaded benchmarks. We have added MEMWidth, which is the width of the memory system connected via the memory controller. We have also added some other new memory subsystem parameters, specifically the interconnect queue size and the memory controller queue size. Lastly, we add L2 cache associativity, and consider varying it from 8 to 32.

Table VI gives a description of the benchmarks used going forward in this paper. We used benchmarks that test a web browsing system such as network packet processing via EEMBC [2], ASP and Javascript performance via IBench [17], and an internal Javascript benchmark. We also considered web performance of popular websites. These websites exhibit a variety of features such as varied use of advanced scripting, cookies, and forms. Although named similarly, these benchmarks stress different portions of the system. The websites were run on an internally developed web browser.

Figure 5 shows the P&B impact values for our system. As seen, for the single-threaded benchmarks, CPUFreq and L2Size have the highest impact. However, for the multi-threaded benchmarks, CPUFreq followed by number of CPUs

Fig. 4. Block diagram of system under test.

Fig. 5. P&B results for system under test.

(NCPU) have the highest impact. If we consider the average, CPUFreq is the most important parameter, followed by L2Size and NCPU. Queue size related parameters INTOut and MEMQSize were found to have relatively low impact. L2 associativity has little impact as the min value (8-way) appears to be sufficient.

These results are instructive, but fall short of giving concrete direction to arrive at a final design, or even to narrow in on a likely region for the best design. Issues include potentially misleading impact (due to oversensitivity to range of values

Parameter	Description	Low Value (-1)	High Value (+1)	Intermediate Values
CPUFreq	CPU frequency	1GHz	2GHz	100 MHz Steps
L2Size	Shared L2 cache size	256KB	1MB	512KB
NCPU	Number of Cores	1	4	2
INTFreq	Shared interconnect frequency	200MHz	800MHz	100MHz Steps
L2Freq	Shared L2 cache frequency	1GHz	2GHz	100MHz Steps
MEMFreq	Memory system frequency	266MHz	800MHz	133MHz Steps
MEMWidth	Memory width	32 bits	64bits	-
INTOut	Interconnect queue size	4	32	-
MEMQSize	Memory controller queue size	4	16	-
MEMChan	Memory system channels	1	2	-
L2Asso	L2 cache associativity	8	32	-

TABLE V
P&B PARAMETER VALUES FOR OUR SYSTEM UNDER TEST

Benchmark	Description	Suite
IBench1	ASP/Javascript Browser Benchmark	IBench [17]
IBench2	ASP/Javascript Browser Benchmark	IBench [17]
IBench3	ASP/Javascript Browser Benchmark	IBench [17]
IBench4	ASP/Javascript Browser Benchmark	IBench [17]
Nat	Network address translation	EEMBC [2]
IPReassembly	IP packet reassembly	EEMBC [2]
Pktcheck	IP header validation	EEMBC [2]
JSEngine	JavaScript Engine	Internal benchmark
Brow1	Popular webcontent stored on flash	Internal benchmark
Brow2	Popular webcontent stored on flash	Internal benchmark
Brow3	Popular webcontent stored on flash	Internal benchmark
Brow4	Popular webcontent stored on flash	Internal benchmark

TABLE VI
EXPANDED BENCHMARK SET DESCRIPTION

	Art	Mcf	Bzip2	Nat	Pktcheck	Samurai	Proxycon	CPUFreq -1/+1 Value
CPUFreq	5.7	14.8	87.3	22.0	37.1	39.8	40.1	1GHz–2GHz
MEMFreq	31.0	19.2	3.0	14.6	11.8	9.1	16.1	
L2Size	39.0	37.1	3.4	41.8	32.1	19.3	18.0	
CPUFreq	2.9	7.8	75.8	11.1	21.3	23.3	23.4	1.25GHz–1.75GHz
MEMFreq	31.7	20.7	5.8	16.6	14.8	11.5	20.5	
L2Size	40.3	40.4	6.5	47.4	40.4	24.6	23.0	

TABLE VII
DEPENDENCE OF IMPACT ON +1/-1 VALUES.

chosen), the lack of information about intermediate values, and the absence of correlation with the cost of parameters. The following sections describe extensions to the P&B method to overcome these limitations. These extensions, incorporated on top of our simulation infrastructure, constitute SAAB.

V. CHANGING BOUNDS AND KNEE ANALYSIS

The P&B method allows us to understand the influence of various factors on the performance of the system. The method lays out a set of experiments that vary each parameter between the upper bound (+1) and lower bound (-1) values. However, it leaves the selection of the +1 / -1 values up to the architect. The specific choice of values can make a significant difference in the results we obtain from the method.

To better understand the influence of the choice of range, we run new experiments where we alter the range of just one parameter. Table VII shows the impact values under two different scenarios. In the first, CPU frequency specifies the original range of 1-2 GHz. In the second, we specify a narrower range. Not only does the impact factor for CPU frequency change significantly, but the other factors move as well. In four of the seven benchmarks, we see that the relative order of these three parameters’ impact factors is changed by varying the range of CPU frequency. While the general guidance is that values slightly higher and lower than the nominal bounds are a good fit for the +1 and -1 values, this still leaves room for interpretation and the potential to influence results. It can be instructive, then, to vary the bounds and rerun the Plackett and Burman analysis. It is typical that the lower bounds are more constrained or better understood than the upper bounds. Hence we primarily consider variations on the upper bounds.

Another concern with P&B in this context is that it provides impact for the parameter over the entire range. If the impact is completely linear, that is sufficient, but for most design parameters, that is not the case. An architect typically looks for the proverbial “knee” of the curve – if most of the marginal gain can be achieved by small increases in a parameter, we can forego the cost of fully provisioning the resource, and instead seek gains from other resources. We can address this problem the same way we propose to explore the upper bound sensitivity. We set the original upper bound to an intermediate value and rerun the tool. This way we can identify the incremental impact of a parameter over various ranges. This now gives us the ability to track non-linear relationships and identify knees. Without this, an architect using the P&B analysis would always increase the factor with the highest overall impact to its maximum value before moving to the next parameter. With incremental impact factors, we increase the highest impact factor only until its marginal impact decreases

below some other parameter. While doing so, we still preserve the key benefits of P&B analysis – varying multiple parameters together using linear order experiments.

Let us consider the number of experiments that are needed if we are to add a different +1 bound for a single parameter. The P&B design with fold-over consists of a set of $2N$ experiments where for each of the individual parameters, N experiments use the +1 value and N experiments use the -1 value. This can be observed in Table I. If we are to do an extra P&B experiment for a different max value of CPUFreq, for example, we will need to perform N extra experiments. For the specific case of our experiments, 12 experiments are needed for every new design point. This can be generalized. If we are to add K design points for one parameter, we will need to perform $2N + KN$ experiments. If we decide to add K design points for all N parameters we would require a total of $2N + KN^2$ or $O(KN^2)$ experiments. If some parameters have the maximum k design choices, the typical number of experiments will be $O(KN)$. The worst case and typical are far less than an exhaustive search, which requires $O(K^N)$ experiments.

The SAAB system allows us to analyze variations of all parameters at intermediate steps as discussed above. We have added up to 9 intermediate steps in the analysis of different parameters as shown in the last column of Table V. Although the worst case number of experiments required are of the order of $O(KN^2)$ experiments, in a typical search for the optimal design we will run far fewer experiment. For our system, we need a total of 360 experiments for a representative design space search, with 11 parameters and 28 intermediate design points. All the experiments were run in parallel on a cluster. We could have run experiments partially and iterated based on their results but the total number of experiments needed were small enough to be all run in parallel and remove the need for an iterative analysis. If the number of design choices are much higher than available parallelism, we could perform incremental search through the design space, where we select the parameter with the highest impact factor at each step to be increased to continue the search. Next we show results and design choices that can be obtained using this analysis.

A. Evaluating Intermediate Impact Trends

This section gives experiment results for a full set of intermediate values, and shows ways to identify some target processor designs with this data, following the methodology described in the prior section.

The first experiment takes each of our 6 highest overall impact parameters, and examines the impact value for the range from the first intermediate value to maximum value. In all cases, the baseline machine is all parameters at their minimum. The intermediate values are those given in Table V. This experiment gives significant insight into the impact of each parameter across the design space, yet requires only a total of 360 experiments (vs. over 400,000 required to explore the complete design space).

Figure 6 shows the variation in impact values for six parameters. CPUFreq continues to rapidly gain impact with increasing +1 values, but the results vary across benchmarks in

Fig. 6. Variation in Impact with changing +1 values for important parameters.

two dimensions. First, the absolute impacts vary considerably in some cases. Also, we see differences in where the impact factor tends to tail off.

We see a couple of other important, albeit expected, results. First, all impact factors grow as the range, and in particular the top of the range, grows. Second, we see that impact factors grow less than linearly (negative second derivative) as the range grows. This means that the marginal impact values (between intermediate points) are shrinking as we move to larger parameter values. This validates the possibility of a design search starting with the lowest values. Because incremental impact values always shrink, we can increase parameter values until they fall below some threshold without worrying that we are missing an important design point with a higher (marginal) impact at a higher parameter value.

For CPUFreq, once the new +1 value reaches about 1.6GHz, we observe a slowdown in continuing impact for some benchmarks such as IBench4. However, for other benchmarks such as JSEngine, increasing the CPUFreq has continued gains. A similar trend is observed for the L2Size parameter as shown in figure 6(b). With the exception of the Pktcheck benchmark, all benchmarks gain from increased cache sizes.

B. Utilizing Impact Trends

With this set of experiments, we can determine not only what design parameters have high impact, but also what value or size of that parameter is required to get arbitrarily close to the full impact (which represents one definition of the “knee” in our curves). For example, in figure 6(c) we see that adding

a second core has a high impact value of 15%, but going to four cores increases impact by only less than 3%. This allows us to find design points that do not waste area or resources on parameters that do not have impact, but also do not grow high-impact parameters beyond their useful size.

Thus, we might identify a resource-efficient design by establishing a performance threshold (e.g., within 1%, 3%, or 5% of the maximum impact), and setting the high-impact parameters at the lowest parameter that meets that threshold. We will set the low-impact parameters at their lowest value. The 1% point might be suitable for a high end system where we wish to be always within 1% of the maximum impact for each parameter. A 3% operating point provides a more balanced approach providing more opportunities to trade off resources for performance. Lastly, a 5% system might be suitable for a resource constrained low end system. Figure 7 shows the results of our analysis.

Figure 7 shows the resulting design points varying by benchmark and within the 1/3/5% thresholds. We see that the 1% design needs most parameters at or near their highest point, but the 3% and 5% designs have quite a bit of room for resource savings. We see that L2 cache size is not negotiable in most designs, but most of the frequency factors can be relaxed. The 5% design never needs more than two cores.

Based on this analysis, we define a set of “interesting” design points in Table VIII. These configurations represent one possible approach to utilizing impact trends. The configurations are (A) all parameters at highest values, (B) six high impact factors set to highest value, low impact factors set to

Fig. 7. Operating points within a certain percentage of Maximum Impact.

Configuration	CPUFreq	L2Size	NCPU	INTFreq	L2Freq	MEMFreq	MEMWidth	INTOut	MEMQSize	MEMChan	L2Asso	Description
A	2GHz	1MB	4	800MHz	2GHz	800MHz	64bits	32	16	2	32	Highest end system
B	2GHz	1MB	4	800MHz	2GHz	800MHz	32bits	4	4	1	8	Low impact -1 value
C	1.8GHz	1MB	2	533MHz	1.6GHz	600MHz	32bits	4	4	1	8	Low impact -1 rest -3% value
D	1.8GHz	1MB	4	533MHz	1.6GHz	600MHz	32bits	4	4	1	8	+ NCPU highest end
E	2GHz	1MB	4	533MHz	1.6GHz	600MHz	32bits	4	4	1	8	+ CPUFreq/NCPU highest end
F	1GHz	256KB	1	200MHz	1GHz	266MHz	32bits	4	4	1	8	Lowest end system

TABLE VIII
DESIGN CONFIGURATIONS

Description	IBench1	IBench2	IBench3	IBench4	Nat	IPReassembly	Pktcheck	JSEngine	BROW 1	BROW 2	BROW 3	BROW 4	Average
Config B vs A (% slowdown vs A)	+1.3	+5.2	+3.8	+3.4	+0.0	+0.0	+0.2	+0.6	+3.4	+0.0	+1.2	+1.3	+1.7
Config C vs A (% slowdown vs A)	+7.0	+30.8	+35.4	+30.2	+16.2	+16.3	+15.3	+16.1	+24.3	+26.5	+24.4	+41.2	+23.6
Config D vs A (% slowdown vs A)	+6.7	+19.1	+26.4	+24.9	+11.6	+11.3	+7.9	+8.8	+21.0	+18.5	+18.3	+23.8	+16.5
Config E vs A (% slowdown vs A)	+6.7	+9.5	+23.8	+17.2	+11.6	+11.3	+7.9	+7.2	+14.1	+14.6	+10.1	+14.0	+12.4
Config F vs A (% slowdown vs A)	+245.0	+351.6	+356.3	+376.3	+217.7	+209.3	+175.1	+150.1	+268.8	+266.3	+248.6	+290.8	+263.0

TABLE IX
EVALUATING RUNTIME PERFORMANCE OF CONFIGURATIONS

min value (this is the design, for example, that would come out of P&B analysis alone), (C) six high impact factors set by the 3% threshold, (D) similar to C, but the highest impact factor (NCPU) not already maximized is also set to its highest value, (E) the next highest impact factor (CPUFreq) is also set at max, and (F) all parameters set to minimum.

We simulate each of these configurations in Table VIII, and the results are in Table IX across all benchmarks. All performance results are relative to the execution time of the highest end system. We see that the low-end system is 2.5X slower than the high end system. Configuration B suffers only a 1.7% runtime increase versus the highest end system. This is a system we might derive directly from the base P&B formulation – it gives good performance, but does not allow us to compromise on resources for the high-end

parameters. But with nearly half the parameters set at their low values, it represents a strong trade-off. Cutting resources more aggressively is possible with our SAAB methodology. Config C targets the 3% operating points for the high-impact parameters. It is still within 24% of the best design, despite 5 resources being at their minimum value, and 5 of the other 6 being below their max value. For this design, then, the parameter values are on average closer to the minimum design than the maximum design, yet performance is far closer to the maximum design than the minimum design. Thus, configuration C offers significant area and power advantages over the highest end system, with only moderate sacrifice in performance. Configurations D and E add back in the full suite of cores and maximize clock frequency, respectively. They get up to within 17% and 13% of the max configuration. But those

Parameter	Metric	Choice 1	Choice 2	Choice 3	Choice 4	Choice 5	Choice 6	Choice 7	Choice 8	Choice 9	Choice 10	Description
Baseline	Power(mWatt)	442	-	-	-	-	-	-	-	-	-	Lowest end system
	Area (mm^2)	5.8	-	-	-	-	-	-	-	-	-	Lowest end system
CPUFreq	Marginal Impact	6.7	4.9	3.5	3.0	2.3	1.8	1.6	1.6	1.1	1.6	1.1GHz - 2GHz
	Marginal Power	14.2	14.2	14.2	14.2	14.2	14.2	14.2	14.2	14.2	14.2	in mWatt
	Power norm Impact	47.0	34.4	24.5	21.1	16.2	12.6	11.2	11.2	7.7	11.2	in % per mWatt
	Marginal Area	0	0	0	0	0	0	0	0	0	0	in mm^2
	Area norm Impact	large	large	large	large	large	large	large	large	large	large	in % per mm^2
L2Size	Marginal Impact	14.0	7.6	-	-	-	-	-	-	-	-	512KB - 1MB
	Marginal Power	44.3	51.1	-	-	-	-	-	-	-	-	in mWatt
	Power norm Impact	31.6	14.9	-	-	-	-	-	-	-	-	in % per mWatt
	Marginal Area	0.66735	.924	-	-	-	-	-	-	-	-	in mm^2
	Area norm Impact	21.0	8.2	-	-	-	-	-	-	-	-	in % per mm^2
NCPU	Marginal Impact	11.5	2.1	-	-	-	-	-	-	-	-	2 - 4
	Marginal Power	251.8	503.6	-	-	-	-	-	-	-	-	in mWatt
	Power norm Impact	4.6	0.4	-	-	-	-	-	-	-	-	in % per mWatt
	Marginal Area	3.89554	7.791	-	-	-	-	-	-	-	-	in mm^2
	Area norm Impact	2.9	0.3	-	-	-	-	-	-	-	-	in % per mm^2
INTFreq	Marginal Impact	5.6	1.5	1.5	1.0	0.9	0.8	-	-	-	-	200MHz - 800MHz
	Marginal Power	8.3	8.3	8.3	8.3	8.3	8.3	-	-	-	-	in mWatt
	Power norm Impact	67.5	18.1	18.1	12.0	10.8	9.6	-	-	-	-	in % per mWatt
	Marginal Area	0	0	0	0	0	0	0	0	0	0	in mm^2
	Area norm Impact	large	large	large	large	large	large	large	large	large	large	in % per mm^2
L2Freq	Marginal Impact	2.3	1.7	1.4	1.4	1.0	1.0	0.8	0.6	0.6	0.4	1.1GHz - 2GHz
	Marginal Power	5.2	5.2	5.2	5.2	5.2	5.2	5.2	5.2	5.2	5.2	in mWatt
	Power norm Impact	44.5	32.9	27.1	27.1	19.4	19.4	15.5	11.6	11.6	7.7	in % per mWatt
	Marginal Area	0	0	0	0	0	0	0	0	0	0	in mm^2
	Area norm Impact	large	large	large	large	large	large	large	large	large	large	in % per mm^2
MEMFreq	Marginal Impact	4.1	2.1	1.5	1.5	-	-	-	-	-	-	400MHz - 800MHz
	Marginal Power	11.3	15.2	19.4	23.2	-	-	-	-	-	-	in mWatt
	Power norm Impact	36.4	13.8	7.7	6.5	-	-	-	-	-	-	in % per mWatt
	Marginal Area	0	0	0	0	-	-	-	-	-	-	in mm^2
	Area norm Impact	large	large	large	large	-	-	-	-	-	-	in % per mm^2

TABLE X

MARGINAL IMPACT, POWER AND AREA COSTS. NORMALIZED METRICS HAVE BEEN MULTIPLIED BY CONSTANT FACTORS FOR EASIER COMPARISONS.

are both high prices to pay (at least in terms of power), so may be less energy efficient than configuration C, despite the increased performance.

SAAB allows us to navigate a very complex and dense design space with a minimal number of experiments (in this case, less than .1% of all designs), and arrive at a set of designs that provide high resource efficiency. Those designs sacrifice little in performance, yet provide significant area and energy advantages over the maximal system. While this methodology provides a mechanism to identify a design that is arbitrarily far from the maximal design and likely to be area and energy efficient, it does not provide a methodology to maximize performance per cost. The next section extends the technique further to create even more targeted designs.

VI. INCORPORATING COST MODELS

The previous section describes how SAAB can be used to arrive at designs that use processor resources efficiently, but fails to fully empower the architect to make the best performance per unit cost decisions. The impact factors from P&B are essentially unitless, and are difficult to compare between parameters because they do not associate impact with the cost required to achieve that impact.

We have seen that L2 cache size has a higher impact than L2 cache associativity. But this does not necessarily imply that we should increase L2 size first. The area cost of doubling the size of the L2 cache is quite high, for example, while the cost of doubling the associativity is low. Thus, the latter may still provide the best “bang for the buck” in a resource-constrained system. Nearly all CPU designs, in fact, are resource-constrained systems – some are area constrained, some are power and energy constrained, some are all of

the above. In this section, we will describe a mechanism to evaluate the impact of parameters and their operating points with respect to area and power costs incurred. We then propose algorithms to arrive at system configurations in resource or performance constrained environments.

To evaluate parameters with respect to resource usage we define *cost normalized marginal impact*. Cost normalized marginal impact is defined as the P&B impact gained by moving from one design choice to another, divided by the increment of resources used at the new design. For example, we gain an average impact of 11.5% while moving from a 1-core to a 2-core system and incur an area cost of $3.9mm^2$ to add another core. Hence, the area normalized marginal impact of NCPU operating at 2 cores is $11.5\%/3.9mm^2$. We can similarly express power normalized marginal impact.

In this way, we replace the unitless P&B impact factor (unitless because they assume a uniform and cost-blind -1 to +1 range to all parameters). We essentially replace the -1 to +1 range in the P&B calculations with a range that reflects the actual cost of spanning that range.

Notice that this also allows our methodology to become much more tolerant of poor range choices in this way. If we assume that doubling the size of a cache doubles the impact, then the P&B impact factor will double when the upper limit is changed, while the cost normalized impact factor will remain the same. In fact, in the special case of linear effects, this completely solves the range choice problem.

In order to arrive at cost normalized parameters, we first estimate the costs associated with each parameter and its operating point. In order to derive costs we use McPAT [16], an architectural power, area, and timing estimation tool. We derive power and timing information for the various design choices


```

1 Start from the highest end system
2 Set design choice for each parameter to the highest
3 Cost = Sum of all baseline and marginal costs
4 While (Cost > Budget or (No More Design Choices))
{
    Find parameter with minimum (Marginal Impact / Marginal Cost)
    Set design choice for that parameter to one step lower
    Cost = Cost - Marginal Cost going one step lower
}
5 Set current design choices of all parameters as final design
that fits within the Cost Budget

```

Fig. 8. **Maximizing Impact for a fixed area or power cost.**

of our system. We start by using an out-of-order processor design. We then modify size structures and parameters as per our design options. Parameters not under design, such as L1 size, were set to publically available information [4] [18] whenever available. We have intentionally not verified these values against our internal designs, for intellectual security reasons, but the tool itself has been validated against several real designs. Since we are primarily interested in mobile-processor systems we focus on 22nm, 28nm, and 32nm designs built using low operating power (LOP) transistors.

Upon deriving the power and area estimates we use marginal impact values averaged over all benchmarks to arrive at cost normalized marginal impact values for both area and power metrics. Table X shows the values obtained for the various metrics. We also show marginal impact, marginal power and area costs, and cost normalized marginal impact for our various design choices. We show values only for the 22nm technology node to save space. We found area and power to scale similarly across different technology nodes.

From these results we see that there are significant insights available beyond the original methodology. We will focus on the power results, since we can compute power increments for every parameter (unlike area). For example, while P&B would indicate that CPUFreq has the highest impact, we see that INTFreq (interconnect frequency) is actually the most important factor initially (with a power normalized impact of 67.5). The importance of that first step from 200 MHz to 300 MHz is huge, but obscured in the P&B analysis by the fact that the overall gain across the entire range is not much larger than the initial step. After increasing the interconnect frequency, our next priority is CPU frequency. But by having incremental information, we will not necessarily increase the CPU frequency to its maximum value. Instead, once CPU frequency has been bumped up a step to 1.1 GHz, the marginal gains are decreased, and we would want to increase L2Freq and MEMFreq before revisiting the CPU frequency. While prior techniques, give general direction to the architect regarding important design trends, the full SAAB methodology directly guides the architect toward the optimal design space.

A. Deriving Cost Optimized Designs

The goal of the cost normalized designs is to do efficient system design in constrained environments – this is the environment in which almost all processor architecture is done.

The system is typically constrained in two specific manners. First, we could be constrained on resources, for example, we have a limited area budget which is exceeded by the highest end system. The goal of an optimization technique is to derive the system with the highest performance while fitting within the area budget. Similarly, it is possible that for a system where we are performance constrained, we need to arrive at the system with minimum area while maintaining a fixed performance budget. Or similarly, we could be seeking the best performance within a given power budget, or a design that meets a performance goal at minimum power.

We formalize an algorithm for finding a design within constraints in Figure 8. This reverses the discussion above (which started with the smallest system and worked up), starting with the largest system and removing resources until it meets the area or power budget. This algorithm attempts to minimize the loss of impact while reducing costs. We use cost normalized marginal impact to compare and select design choices. At every step, we select for decrement the parameter with the minimum cost normalized marginal impact.

Table XI shows results for this design space exploration exercise. We expect that this algorithm gives good results, given the straightforward shape of our impact curves (e.g., unlikely to find local maxima or minima of significant effect). Possible minor sources of inefficiency include boundary conditions near the thresholds. We used our methodology to discover designs for four different area constraints and four different power constraints. The results are expressed relative to the area and power of the largest design.

We see from these results that our design algorithm, powered by the SAAB results, does indeed find highly resource-efficient designs. For example, we have a design that consumes 70% of the power, but performs within 5.6% of the full design. Even at 40% of the area budget, we are able to find a design that still maintains 84% of the maximum performance.

VII. RELATED WORK

Accelerating design space exploration is a well studied topic with two main approaches: 1) Speeding up individual simulations, and 2) Reducing the number of search points. Gries [8] et al. provide an excellent overview of design space exploration techniques. One of the methods to speed up individual simulations is to use analytical models instead of cycle accurate simulation as suggested by Karkhanis et al. [13]. Lee et al. [15] use a regression model to derive performance models. Ipek et al. [10] consider the use of predictive modeling using neural networks for design. In [6] [12], first a sampling of the design space is carried out and then an analytical model is fit on to the obtained data. Another method to reduce the speed of individual simulation is to perform benchmark sub-setting as suggested in [24] [27] [14].

Methods for design space pruning involve sensitivity analysis [9], constraint analysis [19], hill-climbing [26], tabu search [5] and genetic search [22] [25]. Most of these methods involve picking a starting point, measuring performance of that point and its neighbors, and then deciding a search direction.

Optimization Metric	Budget	Impact	Area	Power	Slowdown vs Highest End	CPUFreq	L2Size	NCPU	INTFreq	L2Freq	MEMFreq	MEMWidth	INTOut	MEMQSize	MEMChan	L2Asso
Area	70%	97.7%	60.3%	68.5%	5.5%	2GHz	1MB	2	800MHz	2GHz	800MHz	64	32	16	2	32
Area	60%	86.2%	41.0%	52.7%	15.8%	2GHz	1MB	1	800MHz	2GHz	800MHz	64	32	16	2	32
Area	50%	86.2%	41.0%	49.1%	15.8%	2GHz	1MB	1	800MHz	2GHz	800MHz	64	32	16	2	32
Area	40%	85.5%	39.7%	39.6%	15.9%	2GHz	1MB	1	800MHz	2GHz	800MHz	64	32	16	1	32
Power	70%	97.0%	59.5%	70.7%	5.6%	2GHz	1MB	2	800MHz	2GHz	800MHz	64	32	16	1	32
Power	60%	84.9%	39.2%	56.0%	16.1%	2GHz	1MB	1	800MHz	2GHz	800MHz	64	32	16	1	8
Power	50%	81.1%	38.0%	56.0%	18.9%	2GHz	1MB	1	800MHz	2GHz	667MHz	64	4	4	1	8
Power	40%	64.3%	37.9%	53.8%	39.0%	1.5GHz	1MB	1	500MHz	1.7GHz	400Mhz	32	4	4	1	8

TABLE XI
COST CONSTRAINED SYSTEM RESULTS

Yi et al. [28] propose the use of P&B for reducing the processor design search space. Our work uses their work as a basis and builds upon it. In [29] Yi et al. use P&B to find benchmark similarity in SPEC benchmarks. They use P&B to analyze bottlenecks in the system and cluster various benchmarks based on impact of various system parameters. Joseph et al. [11] consider the use of regression models focused purely on performance. In [7], Fields et al. tackle the issue of interacting parameters by defining interaction cost, which can be zero (independent, no interaction), positive (parallel), or negative (serial). They illustrate the value of using interaction costs in processor design and optimization. Nookala et al. [21] directly apply the P&B method to the floor-planning problem.

VIII. CONCLUSION

This work describes the SAAB methodology. It is a technique for identifying near-optimal processor design, in the face of a significant number of possibly interacting design parameters, with multiple intermediate points and non-linear impact on performance. It applies the Plackett and Burman methodology to identify high impact parameters, but supplements that methodology in several ways to make it more useful in this domain. SAAB extracts the impact for multiple intermediate points. There is an increase in the number of experiments, but still few enough to be highly practical. It is shown that with these intermediate points, we can effectively choose incremental points in the design space that get close to full performance with a significant decrease in resources.

This paper also shows that by incorporating a cost model into SAAB, we can extract the impact per cost of each feature. Between these two optimizations, then, SAAB produces results that can lead the architect directly to an architecture that sets the intermediate value of each parameter so as to (for example) maximize performance for a given cost constraint. This design methodology navigates a complex design space and select highly resource-efficient designs. One design achieves 94% of maximal performance at 70% power. Another achieves 84% performance at 40% of the area.

ACKNOWLEDGMENT

This work was funded in part by NSF grant CCF-1018356.

REFERENCES

- [1] 3d mobile mark 2006. <http://www.futuremark.com>.
- [2] EEMBC benchmark suite. <http://www.eembc.org>.
- [3] SPEC CPU 2000 benchmark specifications.
- [4] Anandtech. <http://www.anandtech.com>.
- [5] J. Axelsson. Architecture synthesis and partitioning of real-time systems: A comparison of three heuristic search strategies. In *Proceedings of the International Workshop on Hardware/Software Co-Design*, 1997.
- [6] C. Dubach, T. Jones, and M. O'Boyle. Microarchitectural design space exploration using an architecture-centric approach. In *Proceedings of the International Symposium on Microarchitecture*, 2007.
- [7] B. A. Fields, R. Bodík, M. D. Hill, and C. J. Newburn. Using interaction costs for microarchitectural bottleneck analysis. In *Proceedings of the International Symposium on Microarchitecture*, 2003.
- [8] M. Gries. Methods for evaluating and covering the design space during early design development. *Integration, the VLSI Journal*, 2004.
- [9] G. D. L. Hei. Trimedia cpu64 design space exploration. In *Proceedings of the International Conference on Computer Design*, 1999.
- [10] E. İpek, S. A. McKee, R. Caruana, B. R. de Supinski, and M. Schulz. Efficiently exploring architectural design spaces via predictive modeling. *SIGARCH Computer Architecture News*, 2006.
- [11] P. Joseph, K. Vaswani, and M. Thazhuthaveetil. Construction and use of linear regression models for processor performance analysis. In *Proceedings of the International Symposium on High-Performance Computer Architecture*, 2006.
- [12] P. J. Joseph, K. Vaswani, and M. J. Thazhuthaveetil. A predictive performance model for superscalar processors. In *Proceedings of the International Symposium on Microarchitecture*, 2006.
- [13] T. S. Karkhanis and J. E. Smith. A first-order superscalar processor model. In *Proceedings of the International Symposium on Computer Architecture*, 2004.
- [14] A. KleinOswski and D. J. Lilja. Minnespec: A new SPEC benchmark workload for simulation-based computer architecture research. *Computer Architecture Letters*, 2002.
- [15] B. C. Lee and D. M. Brooks. Accurate and efficient regression modeling for microarchitectural performance and power prediction. In *Proceedings of the International Conference on Architectural Support for Programming Languages and Operating Systems*, 2006.
- [16] 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 *Proceedings of the International Symposium on Microarchitecture*, 2009.
- [17] Lionbridge, VeriTest and PC Magazine. IBench web browser suite written in asp and javascript. <http://en.wikipedia.org/wiki/IBench>.
- [18] Microprocessor Report free online articles. <http://www.mpronline.com>.
- [19] S. Mohanty, V. K. Prasanna, S. Neema, and J. Davis. Rapid design space exploration of heterogeneous embedded systems using symbolic search and multi-granular simulation. In *Proceedings of the Joint Conference on Languages, Compilers and Tools for Embedded systems: Software and Compilers for Embedded Systems*, 2002.
- [20] D. C. Montgomery. *Design and Analysis of Experiments*. John Wiley & Sons, 2006.
- [21] V. Nookala, Y. Chen, D. J. Lilja, and S. S. Sapatnekar. Microarchitecture-aware floorplanning using a statistical design of experiments approach. In *Proceedings of the Design Automation Conference*, 2005.
- [22] M. Palesi and T. Givargis. Multi-objective design space exploration using genetic algorithms. In *Proceedings of the International Symposium on Hardware/Software Codesign*, 2002.
- [23] R. L. Plackett and J. P. Burman. The design of optimum multifactorial experiments. *Biometrika*, 1946.
- [24] T. Sherwood, E. Perelman, G. Hamerly, and B. Calder. Automatically characterizing large scale program behavior. In *Proceedings of the International Conference on Architectural Support for Programming Languages and Operating Systems*, 2002.
- [25] V. Srinivasan, S. Radhakrishnan, and R. Vemuri. Hardware/software partitioning with integrated hardware design space exploration. In *Proceedings of the Conference on Design, Automation and Test in Europe*, 1998.
- [26] E. C. Systems and G. Snider. Spacewalker: Automated design space exploration for. Technical report, Compiler and Architecture Research Program, Hewlett Packard Laboratories, 2001.
- [27] R. E. Wunderlich, T. F. Wenisch, B. Falsafi, and J. C. Hoe. Smarts: accelerating microarchitecture simulation via rigorous statistical sampling. In *Proceedings of the Annual International Symposium on Computer Architecture*, 2003.
- [28] J. Yi, D. Lilja, and D. Hawkins. A statistically rigorous approach for improving simulation methodology. In *Proceedings of the International Symposium on High-Performance Computer Architecture*, 2003.
- [29] J. J. Yi, A. Joshi, R. Sendag, L. Eeckhout, and D. J. Lilja. Analyzing the processor bottlenecks in SPEC CPU2000. In *Proceedings of the 2006 SPEC Benchmark Workshop*.