Energy-Performance Tradeoffs in Processor Architecture and Circuit Design: A Marginal Cost Analysis

Omid Azizi* Aqeel Mahesri† Benjamin C. Lee* Sanjay J. Patel‡ Mark Horowitz*

*Dept of Electrical Engineering
Stanford University
Stanford, CA
{oazizi,bcclee,horowitz}@stanford.edu

†NVIDIA Corporation
Santa Clara, CA
amahesri@nvidia.com

‡Dept of Electrical and Computer Engineering
University of Illinois at Urbana-Champaign
Urbana, IL
sjp@illinois.edu

ABSTRACT

Power consumption has become a major constraint in the design of processors today. To optimize a processor for energy-efficiency requires an examination of energy-performance trade-offs in all aspects of the processor design space, including both architectural and circuit design choices. In this paper, we apply an integrated architecture-circuit optimization framework to map out energy-performance trade-offs of several different high-level processor architectures. We show how the joint architecture-circuit space provides a trade-off range of approximately 6.5x in performance for 4x energy, and we identify the optimal architectures for different design objectives. We then show that many of the designs in this space come at very high marginal costs. Our results show that, for a large range of design objectives, voltage scaling is effective in efficiently trading off performance and energy, and that the choice of optimal architecture and circuits does not change much during voltage scaling. Finally, we show that with only two designs—a dual-issue in-order design and a dual-issue out-of-order design, both properly optimized—a large part of the energy-performance trade-off space can be covered within 3% of the optimal energy-efficiency.

Categories and Subject Descriptors

C.4 [**Performance of Systems**]: *design studies, modeling techniques*; C.1.0 [**Processor Architectures**]: General

General Terms

Design, Performance

Keywords

Microarchitecture, Energy efficiency, Design trade-offs, Optimization, Design space exploration, Co-optimization

ISCA'10, June 19–23, 2010, Saint-Malo, France.

1. INTRODUCTION

Technology scaling has historically been a driving force behind microprocessor performance. As the semiconductor industry has scaled into very small feature sizes, however, the need to control leakage current has prevented further threshold and supply voltage scaling. This break from Dennard scaling [7] has led to rapid increases in power density, and power consumption is now a primary constraint in all microprocessor design [14]. Not only does power dissipation impact battery-life in embedded devices, it also constrains achievable performance in server architectures.

To optimize a microprocessor for energy-efficiency, a designer must consider the cost-benefit trade-offs of all design options, choosing those features and parameter values that offer the best return in terms of performance per unit energy. While this strategy is straightforward in theory, in practice, it has been difficult to automate since the space of processor design options is often extremely large, and one needs to consider trade-offs in the architecture, the circuit design and potentially the technology.

Fortunately, previous research has already created many of the pieces needed to create such an optimization framework. Recent advances in microarchitectural design space modeling through statistical sampling and regression techniques have opened the door to large-scale architectural design space exploration and evaluation [18, 15]. At the circuit level, there also exist numerous tools that can characterize energy-delay trade-offs for a given circuit [22, 6]. Moreover, work over the past few decades has created extremely efficient methods to find the optimal solutions to problems with convex optimization objective functions [3]. We leverage these prior works to characterize a joint circuit-architecture design space; by creating architectural models and circuit trade-off characterizations that have log-convex[1] forms, we use geometric program solvers to optimize our model over this joint design space to map out the overall energy performance trade-off space. We have found that joint architecture and circuit optimization can save between 15% to 150% of the required energy, depending on the design objectives.

In this paper, we use this framework to explore a broad space of designs ranging from a simple single-issue in-order design to an aggressive quad-issue out-of-order machine. The results of the optimization determine the choice of underly-

[1]log-convex functions are convex after applying log transformation.

ing microarchitectural parameters, circuit implementations, and operating voltage. Our results show that, without voltage scaling, the architectural and circuit trade-offs yielded a performance range that was modest: the resulting optimal energy-performance curves tend to have rapidly changing marginal energy costs. Enhancing architecture design parameters to improve performance quickly requires large increases in energy. Conversely, trying to save energy quickly leads to rapid losses in performance. Voltage scaling has more slowly diminishing marginal costs and we find, with voltage scaling, a small subspace of architectures and circuits are efficient for a broad range of performance and energy targets.

2. OPTIMIZATION FRAMEWORK

To evaluate energy-performance trade-offs in the processor design space, we use a circuit-aware architecture optimization framework. In this approach, we first create architectural models using design space sampling and statistical inference, capturing a large multi-dimensional space of microarchitectural parameters. Next, we characterize the energy-delay trade-offs of each of the underlying circuit blocks that are found in processor designs, which we store in a circuit library. The architecture and circuit spaces are then joined together to form an integrated design space model. In this joint model, the architectural models are made aware of the energy-delay trade-offs of the underlying circuits, and any constraints that these circuits may place on the architecture are also enforced. This joint architecture-circuit design space is finally sent to an optimization/exploration engine, which, given an optimization objective and resource budgets, searches the space to find the most efficient design configuration. Figure 1 shows an overview of this co-optimization framework.

The combination of these techniques creates a framework that is both powerful and general. It is powerful since it enables a rigorous study of marginal performance benefits and energy costs of any design decision; with this framework we can identify the parameter values that yield the most energy efficient design for a performance target, or the highest possible performance design for a given energy target. It is general because we can construct architectural models for any system simply by extracting simulation samples from the designer's simulator of choice, and we can include circuit trade-offs from a broad range of tools. In this framework, building the architectural models and circuit libraries are generally one-time costs, unless new architectures or circuits are being explored. The design space exploration engine, because it uses a convex optimizer, can produce optimized designs in under 30 seconds.

In the following subsections, we provide a brief discussion of each component of this optimization framework. For a more in-depth examination of this framework, we refer the reader to [1].

2.1 Architectural Modeling

At the architectural level, we require models that predict how the architectural performance (CPI) of the overall system changes as we change the underlying parameters. The parameters at this level range from the choice of high-level architecture to the tuning of microarchitectural knobs such as cache sizes and functional unit latencies.

Traditionally, such modeling is done through simulation [2].

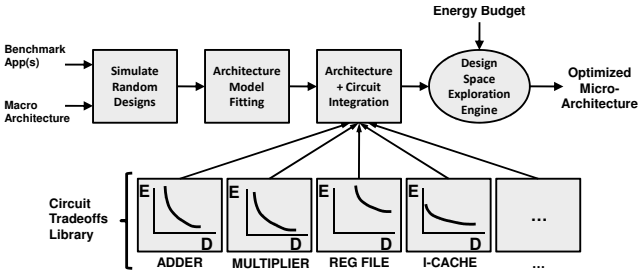


Figure 1: *Overview of the optimization framework. Architectural models are generated using sampling and fitting methods. Energy-delay (E-D) trade-off curves are characterized for each circuit. These design spaces are integrated, and the design space exploration engine finds the optimized design.*

Long run times, however, make simulation intractable for exploring large design spaces. To overcome this restriction, recent works have proposed the use of statistical sampling and inference to create fitted models from a relatively small sample of design points [18, 15, 8]. In these approaches, a small number of random design configurations (with randomly selected design parameter values) are simulated in the traditional manner; the simulation results are then used to infer how the different design parameters interact and affect overall performance. The inference process essentially fits a model to the data through regressions. These approaches produce predictive mathematical models which can be quickly evaluated to estimate the performance in any part of the captured space, and are powerful tools for design space exploration. In previous works, Lee and Brooks used cubic splines [18], whereas Ipek et al. used artificial neural networks [15]; both were shown to produce good fits.

We use similar statistical techniques to create architectural models in our framework. However, instead of the previously suggested functional forms, we use posynomial functions, which are mathematical functions consisting of the sum of any number of positive monomial terms[2] to produce the fits. Using posynomial functions offers the advantage that, as log-convex functions, they can be optimized very efficiently. The potential drawback is that posynomial functions are less general than other forms such as cubic splines, and can have difficulty capturing spaces with complex behaviors involving hills and valleys. Fortunately, the design knobs in the architectural performance space that we are trying to capture usually have a monotonic, diminishing returns profile: for example, reducing a unit's latency or increasing a cache size in the simulator typically only improves performance. It is not expected that turning such knobs would produce hills, valleys or plateaus which would otherwise cause problems. We have compared our fits to cubic spline fits and found them to be of comparable accuracy, thus suggesting that the use of posynomial functions is not generally a restricting factor in the models we produce.

When using a fitting approach to performance modeling, there are still limitations of how much of the design space we can effectively capture. The use of a smooth function such as a posynomial lends itself well to capturing the tunable parameters in the design space. Thus, parameters such as la-

[2]A monomial is a product of powers of variables. For example, $kx^a y^b z^c$ is a monomial in the variables x, y and z with k, a, b, and c as constants.

tencies of units and sizes of structures are often modeled well through posynomial functions. On the other hand, discrete changes to an architecture, such as in-order vs out-of-order execution, or use of centralized vs distributed instruction windows, are less natural to fit with continuous functions. For these these latter design choices—which we refer to as *macro*-architectural design choices—we generate individual models that we optimize separately. Thus, the models we generate predict CPI as a function of the latencies of units and sizes of structures like caches, buffers and queues:

$$CPI = f(..., latency_i, ..., size_j, ...) \quad (1)$$

Since each application behaves differently, we generate separate models for each benchmark using this approach. The fitted models for each benchmark can then be composed together to produce an overall CPI for a suite of benchmarks.

The number of design space samples required to generate a fit depends on the complexity of the system being modeled. For example, we have found 200 samples often enough for simple in-order processors, and 500 samples to be sufficient for a complex superscalar out-of-order processor. We set aside a fraction of these samples to perform validation. To measure model error, we use the same metric as in [18]: $error = |predicted - actual|/actual$. The average of median errors over different benchmarks range from less than 1% to 6%; more complex macro-architectures such as out-of-order processors tend to be harder to fit.

Figure 2 show three sample fits: a very accurate fit, a typical fit, and a worse fit. Even in the worst case, which is for a particular application running on a complex quad-issue out-of-order processor, the median error is less than 10%. The CDFs of these three fits are also shown to give the reader a sense of the distribution of errors in each of these generated models.

2.2 Circuit Trade-offs Library

At the circuit/logic level, there is also a large space of design options. A given circuit can be implemented in various ways that trade-off energy and delay. The design space at this level includes the choice of circuit topology (e.g. ripple-carry adders, carry-lookahead adders, etc.), logic synthesis mappings, circuit styles (e.g. static, dynamic, etc.) and the sizing of gates. Because the energy and delay characteristics of these circuits can affect the energy and performance of the higher-level system, we need to explore and characterize this trade-off space for each circuit block.

There are many tools that can help explore the circuit design space [6, 10, 22]. Given a circuit topology, many of these tools can automatically generate energy-delay trade-off curves. By trying different discrete circuit topologies and circuit styles with these tools, one can create a large trade-off space for a circuit [21]. Our optimization framework is not dependent on the particular tool used to explore the circuit design space; it only needs energy-delay points for each circuit. These design points can be annotated so that once a certain circuit energy-delay point is selected, we can back-reference the annotation to determine the specific circuit implementation.

Given a set of design points, our goal is to create a model that predicts the energy cost of a unit per use. For purely logic units, the trade-off is between energy and delay only. For storage structures such as caches, buffers and queues, the delay and energy also depend on their size. Thus, the

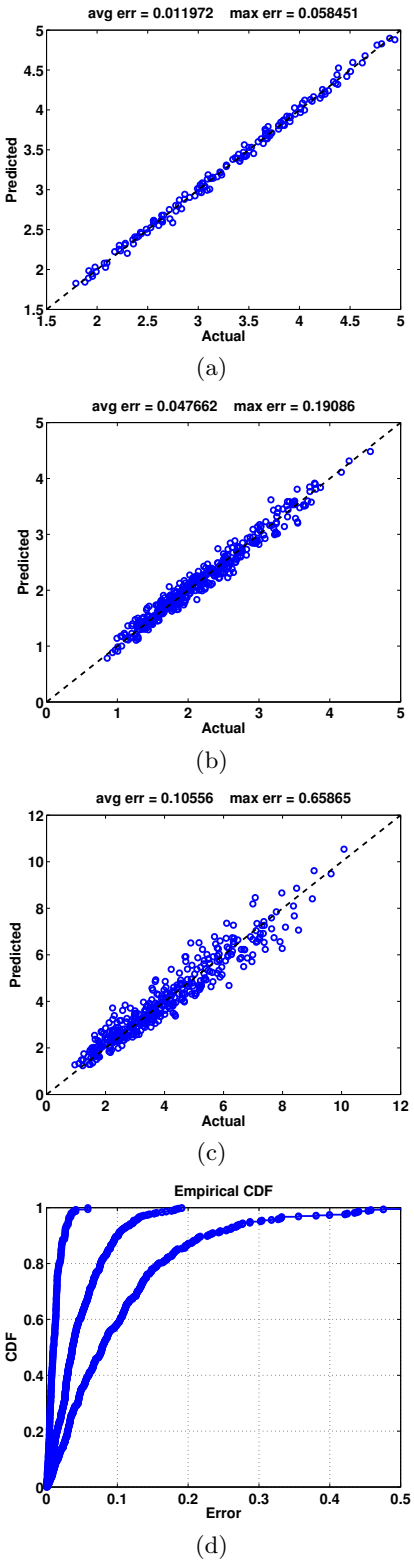


Figure 2: *Validation of three architectural models generated through posynomial design space fitting. (a)-(c) compare model predictions to the results of the simulator. (a) is a very accurate fit, (b) is a typical fit, and (c) is a worse fit. (d) shows the cumulative distribution of errors for these three models. Even in the worst case, the median error is less than 10% for a performance range that spans 10x.*

trade-off characterizations that we produce generally take the form of:

$$Energy = f(Delay), \text{ for logic units} \quad (2)$$

$$Energy = f(Delay, Size), \text{ for memory structures} \quad (3)$$

Just as with the architectural models, we use posynomial functions for $f()$ to produce these characterizations.

We characterize circuits for unpipelined logic. We do this intentionally because cycle time and pipeline depth are parameters in the optimization design space; we account for the energy and delay overheads of inserting pipeline registers during design space integration.

2.3 Design Space Integration

To build the full system model, we need to integrate the architectural and circuit design spaces. This requires making the appropriate links between circuit delays and the pipeline latencies, and then computing the total energy cost from the underlying circuits. For example, one issue with architectural models is that a pipe stage can contain arbitrary amounts of logic. In our integrated model, we make the architectural model aware of the real delays of logic units. We link the physical delays, D_i, of units in the circuit library to the cycle-based latency, N_i, in the architectural model through the cycle time T_{cyc}. The number of pipe stages in a unit is, therefore, the delay of the underlying circuit divided by the clock period T_{cyc} (i.e. cutting the logic into stages). Adding delay overheads for pipeline registers, we get following relationship

$$N_i = D_i/(T_{cyc} - T_{ff}). \quad (4)$$

where T_{ff} includes register setup, clock-to-q times and clock skew overheads.

The optimization framework treats the cycle time, T_{cyc}, as a design space variable (along with D_i and N_i), so it can also explore different pipeline depths for a unit by changing T_{cyc} while holding D_i constant. There is, of course, an energy cost associated with deeper pipelines that comes with an increased number of pipeline registers. This is accounted for in the energy models below.

There may also be certain units that cannot be pipelined. Control units such as the next-PC logic, a processor's instruction scheduler or stall logic may need to compute once a cycle. This is an important consideration as it can place constraints on the cycle time, affecting total performance. In such cases, our framework allows us to specify that these critical logic units cannot be pipelined, accounting for the effects of such circuits on system performance.

We model total energy per instruction (EPI) as the sum of three components: energy spent in logic blocks, in pipeline registers, and in the the clock network. For the logic component, total energy depends on the average energy consumed per use of each circuit, E_i, times its activity rate α_i:

$$EPI_{logic} = \sum_i (\alpha_i \times E_i) \quad (5)$$

The energy cost E_i links directly to the circuit trade-offs. Thus, choosing to use faster circuits will naturally cause an increase in total logic energy.

The energy spent in the registers and the clock network both depend on the number of pipeline registers. We approximate the number of pipeline registers in each unit, R_i, as a function of the number stages, N_i, and the average logic width, W_i. Then, total register energy is the average energy cost of a single register, E_{ff}, times the total number of pipeline registers in that unit, R_i, times the unit's activity factor, α_i. Total clock energy is the total number of registers times the average clock energy, E_{ffclk}, times the cycles per instruction, CPI, to convert it to a per instruction basis.

$$R_i = (N_i)^\eta \times W_i \quad (6)$$

$$EPI_{regs} = \sum_i (\alpha_i \times R_i \times E_{ff}) \quad (7)$$

$$EPI_{clk} = \sum_i (R_i) \times E_{ffclk} \times CPI \quad (8)$$

Here, the parameter η allows for modeling super-linear pipeline register growth, and can be unique for each architectural block. E_{ff} and E_{ffclk} are extracted from real designs. Using these models means that increasing pipeline depth will improve performance, but at an increased energy because of a larger R_i.

For most units, the activity factor α_i, which represents the number of times a unit is used per instruction, can be extracted from the application instruction mix and is constant. With caches, however, the number of times a higher level memory is accessed depends on the miss rate of the cache below it. Thus, we also characterize the miss rate as a function of cache size and use this data in the optimization framework to account for the activity factors of higher level memories as lower level caches change.

3. EXPERIMENTAL METHODOLOGY

We use the optimization framework of Section 2 to explore energy-performance trade-offs in the processor design space. For this study, we examine six different processor architectures: single-issue, dual-issue and quad-issue designs with both in-order and out-of-order execution. This covers a large range of the traditional architecture space, from a simple lower-energy, low-performance single-issue in-order processor to an aggressive higher-energy, high-performance quad-issue out-of-order processor. We note that this set of architectures builds in scope on those examined by previous works [19, 15, 8], all of which looked at superscalar, out-of-order designs only.

For each of these high-level architectures, there are also many tunable microarchitectural parameters that trade-off energy and performance. Table 1 lists these parameters for the design space we explore. This microarchitectural space includes billions of possible design configurations, without even taking into account the circuit design space we explore.

We use a large 8MB L2 cache with a fixed access time in this study. Because the clock cycle time of the core is an optimization parameter, the relative access latency in cycles can still vary, and so the L2 access latency is included in the design space. The DRAM latency is likewise included in the design space because the core frequency can change. The large L2 cache was used because it reduces costly accesses (both energy and delay-wise) to the main memory and lets us focus on the energy-efficiency of the processing core.

We use a YAGS branch predictor [9] in all architectures except the single-issue in-order design, where we use simple 2-bit counters instead. We tried both predictors on all architectures. From an energy-performance perspective, the performance return of the YAGS predictor was usually worth the small increase in total energy. We found the 2-bit coun-

Table 1: *Micro-architectural design space parameters.*

Parameter	Range
Branch predictor	0-1024 entries
BTB size	0-1024 entries
I-cache (2-way) size	2-32KB
D-cache (4-way) size	4-64KB
Fetch latency	1-3 cycles
Decode/Reg File/Rename lat.	1-3 cycles
Retire latency	1-3 cycles
Integer ALU latency	1-4 cycles
FP ALU latency	3-12 cycles
L1 D-cache latency	1-3 cycles
ROB size	4-32 entries
IW (centralized) size	2-32 entries
LSQ size	1-16 entries
L2 cache latency	8-64 cycles
DRAM latency	50-200 cycles
Cycle Time	unrestricted
Supply Voltage	0.7-1.4 V

ters were still useful for very low energy budgets, so we use this simpler predictor for the single-issue in-order design. The lower CPI of these simple designs also means that more aggressive predictors are not as important.

As benchmark suites, we use a subset of SPEC CPU benchmarks. We simulate 500 randomly generated design configurations per benchmark, from which we then generate our architectural models through statistical inference. We set aside a percentage of these these samples for validating our models. Median fitting errors are listed in Table 2; on average, our models have errors of 6.0% or less.

To create the circuit energy-delay characterizations for our circuit libraries, we use a mixed approach. For logic units and small memory structures (queues, register files, etc.), we build Verilog implementations, and use Synopsys Design Compiler to synthesize each of these blocks. The synthesis is based on a CMOS 90nm technology standard cell library. By sweeping the timing constraint on these blocks, Design Compiler produces different logic topologies, synthesis mappings and gate sizings that trade-off energy and delay. Designs with tighter delay constraints use more aggressive mappings and larger gates, resulting in higher energy per use.

For larger memories such as the memory caches and the BTB, we use CACTI 6.0 [20] to characterize the energy-delay trade-off space. CACTI searches the space of possible SRAM memory organizations to evaluate access time and power characteristics of design points. We use CACTI to extract all energy-delay points in this search space, which we then use to construct energy-delay trade-off models.

We use this approach to characterize the energy-delay trade-offs for all the major blocks in the processor: the ALUs, the caches, the reorder buffer, the instruction window, etc. While we have taken care to include all major components in a processor, there are often numerous smaller units and state registers that are present in commercial designs that we are not including. Moreover, while characterizing energy-delay trade-offs for individual circuit blocks is straightforward, accounting for the communication and control in a processor is more difficult, and is often done with empirical data. We have created first-order models of these effects, but we expect others, with more data to draw on, to improve our models in the future. While the detailed results will change as the underlying models improve, we believe that the general trends and conclusions in our study will still hold true.

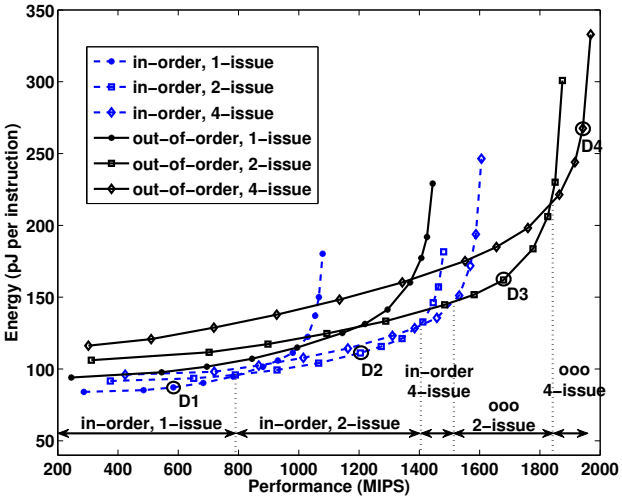


Figure 3: *Overall energy-performance trade-offs of our six macro-architectures for a 90nm CMOS technology, produced by jointly optimizing microarchitectural and circuit parameters. As the performance is pushed, the optimal choice of macro-architecture changes to progressively more aggressive machines. Design details for the circled design points are shown in Table 3.*

4. DESIGN SPACE OPTIMIZATION

We apply the optimization framework to each of our high-level architectures. The resulting energy-performance trade-offs for these architectures are shown in Figure 3. These Pareto-optimal curves show the entire range of trade-offs. As performance is pushed, each architecture uses more aggressive structures and circuits, causing the energy consumed per instruction to increase. Given an energy budget or performance target, a designer can use these curves to identify the most appropriate design.

We note that these Pareto-optimal trade-off curves between performance and energy are more general than commonly used metrics like ED or ED^2. ED^n metrics essentially set an exchange ratio between energy and performance, with higher powers of n favoring more performance; in this sense, they can be somewhat arbitrary. Graphically, ED^n metrics define a set of iso-cost contours in the energy-performance space. For example, in a log/log energy-performance plot ED would correspond to slope 1 lines, and ED^2 would correspond to slope 2 lines. Minimizing for either metric would produce the point on the Pareto-optimal curve where its slope matches the slope of the chosen metric. Thus, optimizing for ED^n with a particular value of n would correspond to a particular point on the Pareto-optimal curve. Since one generally wants to design for a specific performance target or energy budget, neither of these points is necessarily the desired answer. Representing the results as a trade-off curve between energy per operation and performance provides a more complete picture of the design space to designers.

The overall trade-off space spans approximately 6.5x in performance—from about 300 MIPS to 1950 MIPS—and 4x in energy—from about 80 pJ/op to 320 pJ/op. The various architectures contribute different segments to the overall energy-efficient frontier. As one would expect, the single-

Table 2: *Errors of architectural models generated through statistical inference.*

	1-issue in-order	2-issue in-order	4-issue in-order	1-issue out-of-order	2-issue out-of-order	4-issue out-of-order
bzip2	0.0049	0.0055	0.0041	0.0376	0.0443	0.0466
crafty	0.0618	0.0670	0.0825	0.0656	0.0774	0.0777
eon	0.0393	0.0925	0.0794	0.0637	0.0750	0.0727
gap	0.0110	0.0113	0.0130	0.0458	0.0501	0.0477
gcc	0.0445	0.0231	0.0658	0.0493	0.0630	0.0554
gzip	0.0125	0.0124	0.0625	0.0348	0.0410	0.0424
mcf	0.0233	0.0569	0.0578	0.0583	0.0642	0.0899
parser	0.0063	0.0139	0.0087	0.0341	0.0413	0.0380
perlbmk	0.0337	0.0207	0.0362	0.0562	0.0738	0.0598
twolf	0.0236	0.0340	0.0314	0.0408	0.0597	0.0527
average	**0.0261**	**0.0337**	**0.0442**	**0.0486**	**0.0590**	**0.0583**

Table 3: *Design Configuration Details For Selected Design Points.*

	D1	D2	D3	D4
In-order vs out-of-order	in-order	in-order	out-of-order	out-of-order
Issue width	1-issue	2-issue	2-issue	4-issue
Cycle time (FO4)	27.5	16.9	17.2	16.3
Branch pred size (entries)	264	600	1024	870
BTB size (entries)	64	90	554	1024
I-cache size (KB)	21	32	32	32
D-cache size (KB)	8	11	14	42
Fetch latency	1.0	1.6	2.2	2.1
Decode/Rename latency	1.0	1.7	2.4	3.0
Retire latency	N/A	N/A	2.0	2.2
Integer ALU latency	1.0	1.0	1.0	1.0
FP ALU latency	3.0	4.0	3.9	4.1
L1 D-cache latency	1.0	1.1	1.1	1.1
ROB size	N/A	N/A	22	32
IW size	N/A	N/A	11	9
LSQ size	N/A	N/A	16	16

issue in-order architecture is appropriate for very low energy design points, while the quad-issue out-of-order is only appropriate at very high performance points. In between these two extremes, we find that the dual-issue in-order and out-of-order processors are efficient for large parts of the design space. Thus, the order in which high-level architectural features should be considered from a basic single-issue design is, first, superscalar issue, and then, if more performance is still needed, out-of-order processing. From the perspective of marginal energy per unit performance, the move to a superscalar design is cheaper than investing in out-of-order processing. The quad-issue in-order design is only efficient for a small performance range, not being as energy-efficient as the dual-issue in-order design at lower energy points, and being outmatched at high performance points by the dual-issue out-of-order design. The single-issue out-of-order design is never efficient and does not contribute to the overall efficient frontier. This architecture represents a design that is out of balance. Being able to issue only a single instruction becomes a bottleneck to the out-of-order processor, resulting in wasted effort.

We can also examine how the various underlying parameters are changing throughout the design space. In Table 3, we examine these parameters for design points D1 through D4 as marked on Figure 3. Not surprisingly, as we push for more performance, the frequency and structure sizes generally increase, while latencies generally decrease. Some of the latencies show fractional values which would need to be snapped to discrete values, although techniques such as time borrowing and register retiming can also be used to work with the results. We highlight a few points from these results. First, both the I-cache and D-cache tend to stay away from small sizes, even when targeting lower-performance points. Across the design points, the D-cache reaches a minimum of 8KB even though a 4KB cache is available, and the I-cache never goes below 20KB. Although a smaller cache is less expensive to access, a larger cache potentially saves energy by reducing the number of misses that incur a more expensive access to higher level caches. Thus, for lower power design points, the optimizer determines that the marginal savings it can achieve by reducing misses outweighs the access cost of the larger caches, ultimately finding the right balance and settling on the chosen values. In these results, the I-cache tends to have larger sizes than the D-cache; the I-cache has higher hit rates which means that the marginal cost of increasing its size (per unit performance offered) is lower. Generally, these results show the importance of caches in energy-efficient designs as a way to both save energy and increase performance.

Secondly, we note that the instruction window (IW) is relatively small compared to the maximum available IW of size 32. In this case, a larger IW increases the complexity (and delay) of the instruction dispatch logic. Since, in this machine, the dispatch logic must execute every cycle, the delay of the dispatch circuitry can adversely affect the clock frequency. The optimizer realizes this trade-off and finds the right balance between architectural performance through a higher CPI and pipeline performance through a higher cycle time. Moving from design point D3 to D4, the instruction window size backs off to accommodate frequency scaling. We see a similar effect in the branch predictor size, another structure that needs to execute once per cycle.

Finally, we note that the delays of units such as the integer ALU and the D-cache are critical to resolving data dependencies. It is usually worth the energy cost to ensure these units fit into one clock cycle, so the optimizer always ensures that this is the case. Of course, this is not a surprising fact, and is confirmed by current design practices. It is important to note, however, that the delays of these units are changing with the cycle time, so it is not the case

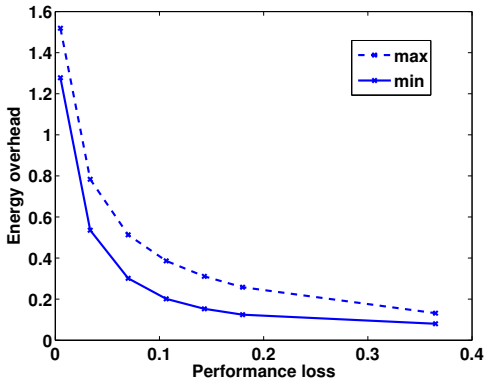


Figure 4: *Plot showing inefficiency caused by using fixed circuit data. Fixed circuit libraries both reduce the achievable peak performance and require more energy than a jointly optimized design. Each fixed library is placed along the x-axis by the performance loss of its fastest solution, so a circuit library at 1% off the minimum delay is located at near 0 performance loss and circuits at 50% off from the minimum delay cause the overall system to slow down 37%. For each library we then find the energy overheads compared to the jointly optimized design. The dotted line is the max overhead throughout the architectural design space, while the solid line is the min overhead. No fixed circuits provide good performance with low energy overhead.*

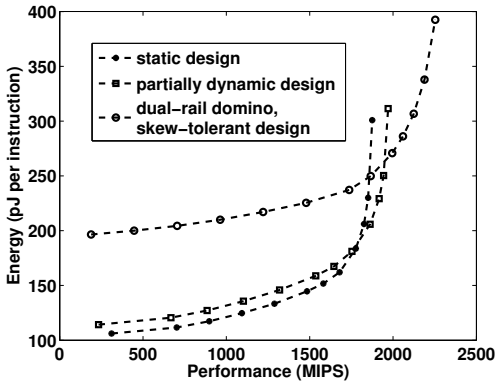


Figure 5: *Trade-offs of dynamic circuits. A partially dynamic design (with a dynamic ALU and issue logic) increases maximum performance by enabling shorter cycle times. A fully dynamic, skew-tolerant design offers even greater performance by removing clocking overheads. Both designs, however, come at significant energy costs which place them on a steep part of the trade-off curve.*

that the same implementations are being used throughout the design space. Machines with more aggressive cycle times use faster, higher energy versions of these circuits, whereas the lower power design points use lower-energy circuit implementations.

4.1 Circuit Trade-offs

The circuit-aware approach that we use integrates delay and energy information into the optimization, exposing different energy-delay design points to the design space exploration. This improves upon most architectural design space tools and studies which typically use fixed energy costs for each circuit (e.g. Wattch [5], others [26, 17]).

This additional fidelity allows us to trade-off the energy and delay within a circuit to find the optimal circuit operating point. As a result, the optimizer can choose to slow down a circuit to save energy when it does not need to run as fast, or it can allocate more energy to a circuit to speed it up if it finds that circuit to be critical to the system performance. This is an important consideration in the optimization space, especially when we consider that different circuits will be optimal at different performance targets.

To evaluate the advantage of exposing the circuit trade-off space, we compare our approach to a fixed energy cost approach. We restrict our circuit libraries to single energy-delay points, sweeping the fixed points to be at 1, 5, 10, 15, 20, 25 and 50% of the minimum delay. For each of these 'fixed' circuit libraries, we then run a complete architectural optimization, and compare the energy required to the energy consumed in the joint circuit/architecture optimization. Figure 4 shows the resulting inefficiency of using these "fixed" circuit points.

The results show that running all the circuits at near maximum speeds (left-most point, circuits at 1% of their min delay) allows the machine to run at near-peak performance, but comes with significant energy overheads of 130% or more. These large overheads are the result of uniformly running all circuits very fast. Only performance critical units should run at their maximum speed; other units should be slowed down to save energy. If, on the other hand, slower circuits (right-most points) are used in an effort to reduce energy, energy overheads reduce to about 15%, but maximum performance is sacrificed. There is no single circuit design point that can be optimal at both the high performance and low energy points. Circuit energy-delay trade-offs need to be included to yield accurate optimization results.

4.2 Dynamic Circuits

Using our optimization framework, we can also explore the effect of using different circuit styles. Our base circuit libraries use static CMOS energy-delay trade-offs which include the space of gate sizings and logic synthesis mappings. Dynamic circuits, however, were once commonly used in high-performance processors and can be significantly faster that static CMOS circuits. However, because of their clock power, dynamic circuits also come at a significant energy cost.

To characterize the performance gains and energy overheads of using dynamic circuits, we compare a dynamic dual-rail domino adder to a static implementation using a circuit optimization tool [21]; these results indicate that the dual-rail domino circuit achieves 0.67x the delay at 4x the energy. Since we do not have a complete dynamic circuit library, we use this adder scaling data as a proxy for all circuits (except memories[3]) to generate dynamic circuit trade-offs.

We examine two dynamic designs using these new libraries. In the first, we use dynamic circuits for certain performance-critical components, speeding up the integer ALU and the out-of-order issue logic. While the ALU does not strictly stand in the way of the cycle time—the ALU is pipelinable—the performance loss of pipelining the ALU due to data de-

[3]Most memories already use some dynamic circuits internally, so we do not change their performance in this experiment.

pendency stalls means it is usually not an energy-efficient design choice. The optimizer generally prefers a 1-cycle ALU with a longer cycle time over a higher frequency design with a multi-cycle ALU. In the case of the issue logic, it cannot be pipelined in our design, and actually limits the use of shorter cycles times.

Dynamic logic has a second potential advantage. Since every dynamic gate already has a clock, it is possible to build an entire system without including any explicit flops or latches. Furthermore, the overall design can be constructed to be tolerant of skew on the clock lines [11, 24]. This type of design essentially removes all clocking overheads that are found in conventional designs. For this method to work, all logic must be a monotonic function of its input (domino logic), so this generally requires one to create dual-rail gates, which compute both true and complement outputs from true and complement inputs. Thus, for our second design, we also explore the performance trade-offs of a complete dual-rail design[4].

Figure 5 shows the results of using these circuit styles on our dual-issue out-of-order architecture. Also shown is the original static version. As expected, both dynamic designs push performance to new limits, but come with some added energy overheads. The partially dynamic design provides more performance because it can now achieve higher cycle times. The faster issue logic now also allows for larger instruction windows of up to 20 entries; this in contrast to the small 8 entry instruction windows we saw in the static design. These performance benefits, of course, come at a somewhat high energy cost: the transition from the static design to the partially dynamic design comes at a marginal cost of 2.3% in energy for 1% in performance. The fully dual-rail, skew-tolerant design offers an even larger performance gain; it virtually eliminates clocking overheads. However, it comes with an even larger energy cost since the entire machine must be implemented in dynamic logic. This design option represents a more expensive choice at about 2.7% in energy for 1% in performance.

While neither of these options are cheap, a designer may be willing to pay the cost if the added performance is truly needed. As we will see in the next section, however, voltage scaling can often offer better marginal costs and should be considered first.

5. VOLTAGE AND MARGINAL COSTS

It is well-known that an important consideration in energy-efficient design is the choice of operating voltage. Figure 6(a) shows the energy and delay scaling characteristics of circuits as a function of voltage as obtained through SPICE simulations. The energy curve follows an expected Vdd^2 profile; the delay shows an inverse relationship proportional to $\frac{1}{Vdd^{3.325}} + 1$ (empirical fit). Composing these two relationships, we get the energy-delay scaling trade-offs of the supply voltage parameter in Figure 6(b).

This data shows that, by itself, voltage tuning from 0.7V to 1.4V provides a range of about 3x in performance and 4x in energy. More importantly, the profile of the energy-performance curve is relatively shallow throughout this entire range. This means that the marginal cost of increasing performance through voltage scaling does not change much as we continue to increase the voltage parameter. At low voltages, the marginal cost is at about 0.80% in energy for 1% in performance; at the high end, this marginal cost reaches 2.3% in energy for 1% in performance.

[4]While the memories would need to be modified to work in this system, the changes would be small and would not cause major changes in memory power or delay.

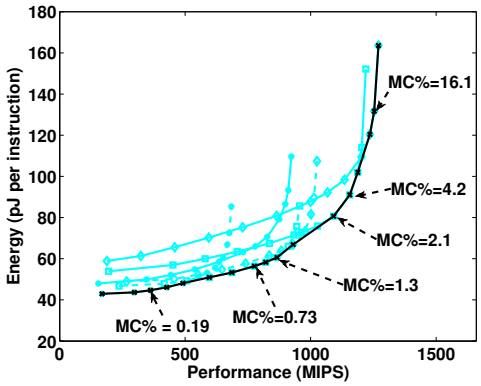
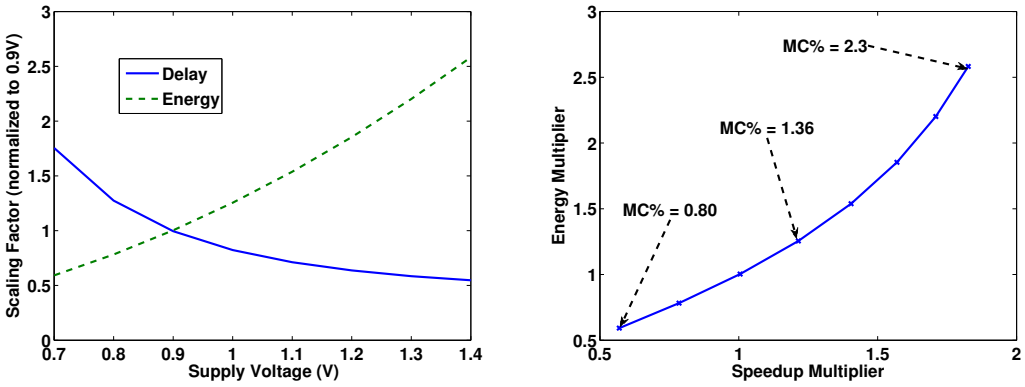


Figure 7: *Marginal costs of the joint architecture and circuit design space. In light are the energy-performance curves of our six architectures. In dark is the overall, composite energy-performance frontier. Data is normalized to 0.9V to allow for comparison to the voltage marginal costs. Marginal costs in the architecture/circuit space vary considerably more than voltage marginal costs. For most practical design objectives the optimal architecture should be selected from the narrow band of designs with a marginal cost of 0.80%-2.3% to match voltage marginal costs.*

We can contrast this marginal cost profile against the marginal cost profile of achieving performance through circuits and architecture, shown in Figure 7. This space shows a much larger range of marginal costs. At the low performance points, the marginal costs are very cheap, while at the high performance points, the marginal costs are very expensive.

We recall that to optimize a design, the marginal costs of all parameters should be equal. If this were not the case, then an arbitrage opportunity would exist, and the more expensive parameters could be exchanged for cheaper parameters: selling the expensive parameter would cause some performance loss, but this performance could be recovered at a lower cost through the cheaper parameter. Comparing the marginal costs of voltage versus architectural parameters, this suggests that, unless we are trying to achieve the very extremes of performance or low power, the optimal set of designs should lie in the range of marginal costs from 0.80% to 2.3% in order to match the marginal costs of voltage scaling. This results in a narrow band of architectural and circuit designs being optimal when the voltage scaling parameter is available.

Figure 8 shows the optimization results when the supply voltage parameter is included in the design space. Confirming the marginal cost analysis, we see that a smaller set of architectures cover a larger part of the energy-efficient frontier. The dual-issue out-of-order processor is energy-efficient for a large part of the design space. At low performance targets, the dual-issue in-order processor takes over, although the dual-issue out-of-order processor is still not overly inefficient. Only at the very extremes, when the voltage knob becomes capped, do the single-issue in-order and quad-issue

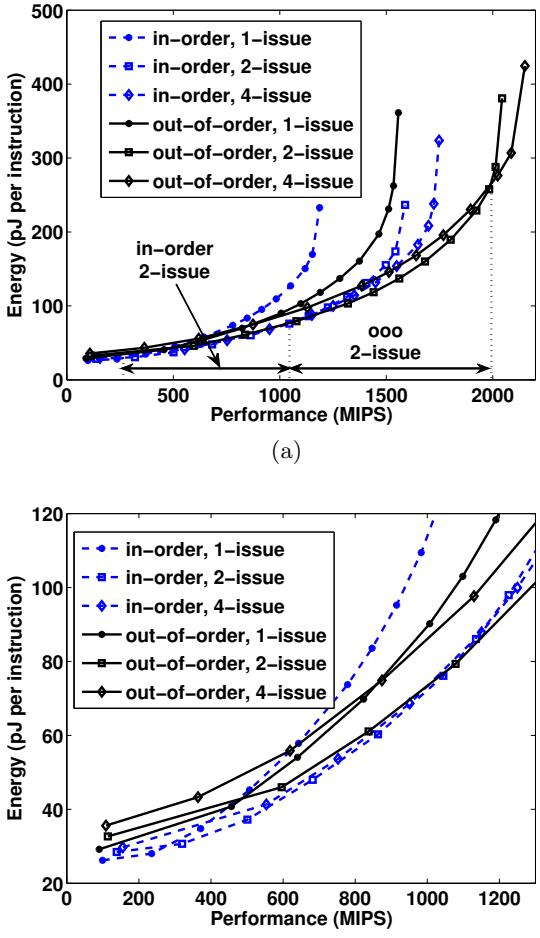


(a) (b)

Figure 6: *Effect of voltage on delay and energy. (a) shows the delay and energy scaling as a function of supply voltage (normalized to 0.9V). (b) shows the corresponding energy-performance trade-off curve. Percentage marginal costs (MC%) is the percentage energy cost required to increase performance by 1%. For a wide range of energy and performance, marginal costs do not change much, making voltage a powerful knob for energy-efficiency.*

out-of-order designs play a role, and these represent designs with very low and very high marginal costs respectively.

This result suggests that a small number of properly tuned designs can cover most of the overall energy-performance frontier at near optimal efficiencies simply by voltage and frequency scaling. We pick one dual-issue in-order processor and one dual-issue out-of-order processor with fixed microarchitectural and circuit parameters, and evaluate these fixed designs under voltage and frequency scaling. Figure 9 shows the energy overheads of scaling these fixed designs as compared to the fully optimized designs which also tune the architecture and circuits. Because design parameters are fixed, we see some inefficiency; the lines deviate from the normalized optimal value of 1. This result is expected because the marginal costs of all parameters in the system are no longer equal. Yet, we see the resulting inefficiency is small—under 3%. Of course, this result requires that we start with the right two designs in the architecture/circuit space sweet spot. Thus, with two carefully selected designs and voltage scaling, we can operate at near optimal energy-efficiencies over a broad performance range.



(a)

(b)

Figure 8: *(a) Energy-performance trade-offs for the processor design space with voltage scaling. The dual-issue out-of-order design now dominates an even larger part of the design space; the dual-issue in-order design is optimal at low energy points. (b) Same results zoomed in on low energy points.*

6. DISCUSSION

Since power has been an issue for many years, there exist a number of tools that help guide architects toward energy-efficient designs. Simulation based tools such as Wattch [5] and PowerTimer [4] have enabled power-performance studies, exploring various aspects of the power-performance space in processor designs. For instance, using these or similar tools people have evaluated power-performance trade-offs in different areas of the design space such as pipeline depth [26, 12, 25]. Applying statistical inference techniques to these tools, Lee and Brooks have evaluated power-performance trade-offs in an even larger microarchitectural design space [19]. Others, like Karkhanis and Smith have used an analytical model to explore trade-offs in the architecture design space [16]. There has even been some work trying to connect circuit and architectural optimization using metrics such as hardware intensity [27, 23].

We extend this prior work in two important ways. First, we integrate circuit trade-offs into the architectural design space analysis and show how it changes the optimization

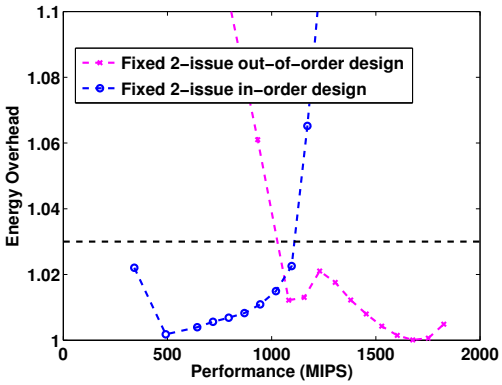


Figure 9: *By using two carefully selected designs—a fixed dual-issue in-order design and a fixed dual-issue out-of-order design—voltage scaling can be used to cover a large performance range within 3% of the optimal energy-efficiency.*

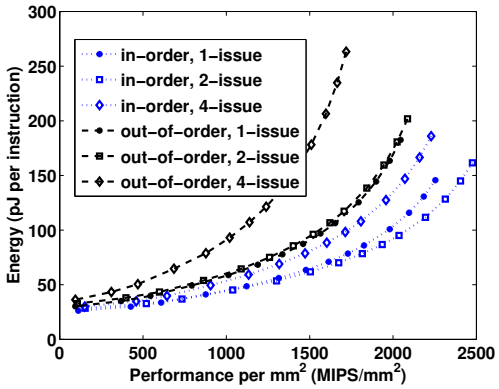


Figure 10: *Energy per op vs performance/mm^2 for a highly parallel workload. The area overheads of out-of-order processing make it a less attractive option.*

results. Second, by using posynomials to model the design space, we can explore the space very quickly—finding an optimal design point in only about 30 seconds. This integration allows us to better understand metrics like hardware intensity (which is simply the relative marginal performance vs. energy cost) and provides a practical framework to optimize this joint design space. Our tool creates the Pareto-optimal curve, which contains the best possible design points for different performance targets or energy budgets, and is not dependent on ED^n metrics.

Using this design framework, we extended the space of prior statistical performance-energy modeling efforts to a wide range of high-level architectures. While this paper has presented the results of optimally tuned processors, the real advantage of this type of design framework is the insight it can give a designer. For example, we initially had each of our macro-architectures fetch instructions according to the width of the machine (e.g. one word for the single-issue machines, etc.). This led to wider machines being more energy efficient than single issue machines even at low performance. Clearly, because of high instruction locality, it makes sense to have all macro-architectures fetch multiple instructions at a time to amortize the cost of going to cache. We are currently looking at our results to see what architectural changes can lead to more efficient cores.

The results presented in this paper have also focused on performance-energy trade-offs, and not considered area (die cost) or what happens for threaded or data parallel applications. It is easy to include these effects using our framework. For example, in the case of multi-core designs for highly parallel workloads, we need to change the performance objective. The number of cores that we can fit on a die is critical to performance, and we must consider both the performance and area of the cores. If we assume infinite parallelism, then we want to optimize the product of the performance per core and the number of cores that will fit in a given area. Thus the performance metric is no longer processor performance, but performance per mm^2. Figure 10 shows optimization results under this new design objective. In this case, the area overheads of implementing out-of-order processors outweigh their performance benefit, and so the dual-issue in-order design is always optimal. To account for workloads with more realistic amounts of parallelism, one just needs to change how the performance scales with the number of cores [13], which will just change the performance/area function that needs to be optimized. In analogous ways we can optimize whole systems that contain SIMD units, external high-power accelerators (i.e. GPUs) or other components that affect energy or performance.

7. CONCLUSION

We have created a general energy-performance optimization tool that will be essential to create the energy efficient designs that the future requires. By constructing a first order circuit library for processors we have been able to rapidly explore a very large processor design space, from a simple single-issue processor, to a wide quad-issue out-of-order machine. While there is clearly more work that can be done using this tool, our initial results have been quite interesting. First, it shows that it is very easy to build a poor machine which is far from the optimal frontier. In fact, hitting the frontier will be very difficult unless you jointly optimize circuits and architecture. More surprising, however, was the marginal cost profile of the optimized performance-energy trade-off space without voltage scaling. While the trade-off spanned a reasonably sized space of 6.5x in performance and 4x in energy, the marginal costs throughout this range were rapidly changing, with very expensive marginal costs for the quad-issue out-of-order design and very low marginal costs for the single-issue in-order design. Since voltage scaling spanned about the same range with smaller changes in marginal performance, we found voltage scaling to be a more effective knob for efficiently trading off energy and performance. With voltage, we found the dual-issue in-order and out-of-order machines were optimal over a large part of the design space. Moreover, by using two machines with the right fixed set of microarchitectural/circuit parameters and only voltage and frequency scaling, we showed that the loss in efficiency compared to the overall optimal machine was only 3%. Thus, by carefully optimizing the circuits and architecture, one can build a machine that can use voltage scaling to efficiently operate at a range performance targets.

In the end, the more important use of this tool will be to generate information and provide insights for the design of new methods that will further reduce the system power. The results can be used to identify both performance and energy limitations of the system, and can direct a designer

to focus their attention on these facets of the design. We have already begun to use it in this way, and are starting to apply it to other performance/energy constrained designs.

8. ACKNOWLEDGMENTS

The authors would like to thank the Chip Generator group at Stanford University for their support and feedback. This work was funded in part by the FCRP Focus Center for Circuit & System Solutions (C2S2), under contract 2003-CT-888. This material is also based upon work supported by the National Science Foundation under Grant #0937060 to the Computing Research Association for the CIFellows Project.

9. REFERENCES

[1] O. Azizi, A. Mahesri, J. P. Stevenson, S. Patel, and M. Horowitz. An integrated framework for joint design space exploration of microarchitecture and circuits. In *DATE '10: Proceedings of the conference on Design, automation and test in Europe*, pages 250–255, 2010.

[2] P. Bose and T. M. Conte. Performance analysis and its impact on design. *Computer*, 31(5):41–49, 1998.

[3] S. Boyd and L. Vandenberghe. *Convex Optimization*. Cambridge University Press, March 2004.

[4] D. Brooks, P. Bose, V. Srinivasan, M. K. Gschwind, P. G. Emma, and M. G. Rosenfield. New methodology for early-stage, microarchitecture-level power-performance analysis of microprocessors. *IBM J. Res. Dev.*, 47(5-6):653–670, 2003.

[5] D. Brooks, V. Tiwari, and M. Martonosi. Wattch: a framework for architectural-level power analysis and optimizations. *SIGARCH Comput. Archit. News*, 28(2):83–94, 2000.

[6] A. R. Conn, I. M. Elfadel, J. W. W. Molzen, P. R. O'Brien, P. N. Strenski, C. Visweswariah, and C. B. Whan. Gradient-based optimization of custom circuits using a static-timing formulation. In *DAC '99: Proceedings of the 36th ACM/IEEE conference on Design automation*, pages 452–459, New York, NY, USA, 1999. ACM.

[7] R. Dennard, F. Gaensslen, V. Rideout, E. Bassous, and A. LeBlanc. Design of ion-implanted mosfet's with very small physical dimensions. *Solid-State Circuits, IEEE Journal of*, 9(5):256–268, Oct 1974.

[8] C. Dubach, T. Jones, and M. O'Boyle. Microarchitectural design space exploration using an architecture-centric approach. In *MICRO '07: Proceedings of the 40th Annual IEEE/ACM International Symposium on Microarchitecture*, pages 262–271, Washington, DC, USA, 2007. IEEE Computer Society.

[9] A. N. Eden and T. Mudge. The yags branch prediction scheme. In *MICRO 31: Proceedings of the 31st annual ACM/IEEE international symposium on Microarchitecture*, pages 69–77, Los Alamitos, CA, USA, 1998. IEEE Computer Society Press.

[10] J. P. Fishburn and A. E. Dunlop. Tilos: A posynomial programming approach to transistor sizing. In *IEEE Int. Conf. Computer-Aided Design*, pages 326–328, 1985.

[11] D. Harris. *Skew-tolerant circuit design*. Morgan Kaufmann Publishers Inc., San Francisco, CA, USA, 2001.

[12] A. Hartstein and T. R. Puzak. The optimum pipeline depth considering both power and performance. *ACM Trans. Archit. Code Optim.*, 1(4):369–388, 2004.

[13] M. D. Hill and M. R. Marty. Amdahl's law in the multicore era. *Computer*, 41(7):33–38, 2008.

[14] M. Horowitz, E. Alon, D. Patil, S. Naffziger, R. Kumar, and K. Bernstein. Scaling, power, and the future of cmos. In *Electron Devices Meeting, 2005. IEDM Technical Digest. IEEE International*, pages 7 pp.–15, Dec. 2005.

[15] E. Ïpek, S. A. McKee, R. Caruana, B. R. de Supinski, and M. Schulz. Efficiently exploring architectural design spaces via predictive modeling. *SIGARCH Comput. Archit. News*, 34(5):195–206, 2006.

[16] T. S. Karkhanis and J. E. Smith. A first-order superscalar processor model. *SIGARCH Comput. Archit. News*, 32(2):338, 2004.

[17] T. S. Karkhanis and J. E. Smith. Automated design of application specific superscalar processors: an analytical approach. In *ISCA '07: Proceedings of the 34th annual international symposium on Computer architecture*, pages 402–411, New York, NY, USA, 2007. ACM.

[18] B. C. Lee and D. M. Brooks. Accurate and efficient regression modeling for microarchitectural performance and power prediction. *SIGARCH Comput. Archit. News*, 34(5):185–194, 2006.

[19] B. C. Lee and D. M. Brooks. Illustrative design space studies with microarchitectural regression models. In *Proceedings of the 13th International Symposium on High Performance Computer Architecture*, 2007.

[20] N. Muralimanohar, R. Balasubramonian, and N. Jouppi. Optimizing NUCA Organizations and Wiring Alternatives for Large Caches With CACTI 6.0. In *Proceedings of the 40th Annual International Symposium on Microarchitecture*, December 2007.

[21] D. Patil, O. Azizi, M. Horowitz, R. Ho, and R. Ananthraman. Robust energy-efficient adder topologies. In *ARITH '07: Proceedings of the 18th IEEE Symposium on Computer Arithmetic*, pages 16–28, Washington, DC, USA, 2007. IEEE Computer Society.

[22] D. Patil, S. J. Kim, and M. Horowitz. Joint supply, threshold voltage and sizing optimization for design of robust digital circuits. Technical report, Department of Electrical Engineering, Stanford University.

[23] Z. J. Qi, M. Ziegler, S. V. Kosonocky, J. M. Rabaey, and M. R. Stan. Multi-dimensional circuit and micro-architecture level optimization. In *ISQED '07: Proceedings of the 8th International Symposium on Quality Electronic Design*, pages 275–280, Washington, DC, USA, 2007. IEEE Computer Society.

[24] J. Silberman, N. Aoki, D. Boerstler, J. Burns, S. Dhong, A. Essbaum, U. Ghoshal, D. Heidel, P. Hofstee, K. T. Lee, D. Meltzer, H. Ngo, K. Nowka, S. Posluszny, O. Takahashi, I. Vo, and B. Zoric. A 1.0-ghz single-issue 64-bit powerpc integer processor. *Solid-State Circuits, IEEE Journal of*, 33(11):1600–1608, Nov 1998.

[25] V. Srinivasan, D. Brooks, M. Gschwind, P. Bose, V. Zyuban, P. N. Strenski, and P. G. Emma. Optimizing pipelines for power and performance. In *MICRO 35: Proceedings of the 35th annual ACM/IEEE international symposium on Microarchitecture*, pages 333–344, Los Alamitos, CA, USA, 2002. IEEE Computer Society Press.

[26] V. Zyuban, D. Brooks, V. Srinivasan, M. Gschwind, P. Bose, P. Strenski, and P. Emma. Integrated analysis of power and performance for pipelined microprocessors. *Computers, IEEE Transactions on*, 53(8):1004–1016, Aug. 2004.

[27] V. Zyuban and P. Strenski. Unified methodology for resolving power-performance tradeoffs at the microarchitectural and circuit levels. In *ISLPED '02: Proceedings of the 2002 international symposium on Low power electronics and design*, pages 166–171, New York, NY, USA, 2002. ACM.