Area-Efficiency in CMP Core Design: Co-Optimization of Microarchitecture and Physical Design

Omid Azizi¹, Aqeel Mahesri², Sanjay J. Patel², Mark Horowitz¹

¹Department of Electrical Engineering, Stanford University
²Department of Electrical and Computer Engineering, University of Illinois at Urbana-Champaign

Abstract

In this paper, we examine the area-performance design space of a processing core for a chip multiprocessor (CMP), considering both the architectural design space and the tradeoffs of the physical design on which the architecture relies. We first propose a methodology for performing an integrated optimization of both the micro-architecture and the physical circuit design of a microprocessor. In our approach, we use statistical and convex fitting methods to capture a large micro-architectural design space. We then characterize the area-delay tradeoffs of the underlying circuits through RTL synthesis. Finally, we establish the relationship between the architecture and the circuits in an integrative model, which we use to optimize the processor. As a case study, we apply this methodology to explore the performance-area tradeoffs in a highly parallel accelerator architecture for visual computing applications. Our results indicate that, for our set of benchmarks, two separate designs are performance/area optimal: a simpler single-issue, 2-way multithreaded core running at high-frequency, and a more aggressively tuned dual-issue 4-way multithreaded design running at a lower frequency.

1. Introduction

Computer design is an optimization problem. Historically, it was to maximize performance given a certain number of gates; today, it is to maximize performance given an area or power budget for high-performance processors, or to minimize area or power subject to some performance requirement for low power embedded processors. While the exact design objective and constraints may change, the design optimization process consists of examining the cost-benefit tradeoffs of different design choices to produce a design that achieves the goal in the most efficient manner possible.

In the past, this tradeoff analysis and optimization has been performed in a somewhat ad hoc fashion. Often times, the designer would pick a parameter of interest and explore how it affects performance and power. While this approach leads to more efficient designs, the design space exploration is slow and tedious, and is unable account for interactions between more than a few design space parameters parameters at a time.

More recently, there have been advances in more systematic approaches to design optimization. The use of statistical approaches to modeling large multi-dimensional spaces has resulted in more thorough characterizations of the architectural design space [7, 10]; this, in turn, has opened the door to interesting design space exploration studies.

Advancements have not been limited to the field of computer architecture. At the circuit-design level, there have been similar advances, and tools now exist that can quickly optimize circuits to produce a large energy-delay trade-off space. Moreover, the popularization of mathematical optimization tools such as convex optimizers has facilitated the optimization of designs, particularly for circuit design [15].

While there has been progress in the modeling and optimization of both the architecture and the circuits, a systematic optimization methodology that co-optimizes across both domains is an area that still needs to be addressed. In this paper, we examine this problem, and we propose a framework for performing global processor optimization by simultaneously optimizing across the architectural and circuit implementation spaces. Although the framework we present is not specific to any particular cost function, in this paper we focus on the area-performance optimization of a chip multiprocessor core.

In particular, we focus on the design space for a highly parallel architecture meant to accelerate visual computing applications. We evaluate architectures built around arrays of in-order cores. We perform a preliminary evaluation of superscalar issue and multithreading. We vary cache sizes and branch predictors, and we also look at the latencies of various pipeline components, from the fetch, decode, and execute stages. Finally, we tie all these architectural parameters to circuit-level design trade-offs, and the perform the optimization. The methodology is detailed in Section 3, and the full design space is detailed in Section 4.

Our preliminary results show how considering the interaction between circuits and architecture provides a more holistic view of design, and how this can affect design
choices. The results are discussed in Section 5

2. Optimization Problem

In a processor, the design of the architecture and the design of the underlying circuits are not independent problems. The amount of logic in a pipe stage together with the clock cycle time (CCT) define how aggressively a circuit has to be designed, and this has an impact on the area- or energy-efficiency of a design. Because of the close interaction between the two domains, one cannot simply evaluate architectural features by their improvement on the IPC; physical design is also affected. If cycle time is extended to allow for a new architectural feature, then the IPC improvement is offset by a frequency degradation. If, on the other hand, the cycle time is maintained, then circuits will need to be designed to be faster at the cost of increased area and energy. Finally, if a new pipe stage is introduced to include the additional logic, then there is again an IPC loss whenever there are instruction dependencies that need to be resolved.

More formally, the problem is that the true performance metric that we need to optimize for (assuming a fixed ISA), is time per instruction (TPI), which is defined as

\[ \text{TPI} = \frac{\text{CCT}}{\text{IPC}}. \]  

(1)

Changing part of the architecture can change both CCT and IPC. It also requires changes in the underlying circuit implementations which, in turn, affect area and energy consumption, ultimately affecting the area- or energy-efficiency of the whole system.

To optimize such a system, we break the analysis into the traditional architectural and circuit design parts, and then introduce the necessary link between the two.

2.1 Architectural Design Space

We partition architectural parameters into two groups. **Macro-architectural design choices** essentially change the processing ‘algorithm’ of the architecture, and include superscalar processing, multi-threading, out-of-order execution, the implementation of dedicated units for processing complex or SIMD instructions (ISA extensions), and various forms of speculative execution.

In contrast, **micro-architectural design choices** affect performance, energy and area, but do not change the basic order of instructions. Micro-architectural parameters include sizing of caches, queues and other structures, the setting of pipeline latencies for different units, and the choice of clock frequency.

We classify these architectural parameters into these two sets to signify how they can be explored and optimized. The micro-architectural parameters, being more numerical in nature, are more amenable to mathematical optimization techniques. The macro-architectural parameters, on the other hand, are more discrete in nature. Since they represent very different architectural topologies, we generate individual models for each of these architectures and optimize them separately.

2.2 Circuit and Logic-Level Optimization

At the circuit/logic level, there is also a large trade-off space between energy, area and delay. This trade-off space needs to be characterized as it can have a considerable impact on the area and performance characteristics of the processor as a whole.

There are many design choices at the logic-level. Particular pieces of logic can be implemented in different ways, resulting in different delay and area characteristics. For example, a ripple-carry adder is small and consumes little energy, while a carry-lookahead adder is much faster, but requires more area and energy consumption. Logic functions throughout the microprocessor can be implemented in varying ways to trade off energy, delay, and area.

The choice of circuit style is another parameter in the implementation design space. Static CMOS circuits save energy, but are slower than dynamic and differential logic styles.

Circuit sizing is yet another low-level optimization. We can save area by using minimally-sized gates, at the cost of higher delay. Alternatively, we can size gates to minimize delay using techniques such as logical effort. This results in a much faster circuit, but comes at the cost of added area and energy consumption.

2.3 Optimal Design Points

For a uniprocessor design, the optimization process is a search in a two-dimensional design space, with performance

![Figure 1. Area-performance design space. Each point is a possible design. Those points which lie on the frontier are optimal (area- or energy-efficient) designs.](image-url)
on one axis and the cost function (energy or area) on the other axis (Figure 1). In this space, there is a Pareto-optimal curve consisting of all the design points that maximize performance for any given cost. Equivalently this can be viewed as the set of points that minimize cost for any given performance level. In the design process, the design of choice should be the point along this Pareto curve that is closest to our cost budget (in terms of area or power).

In a parallel architecture, the optimization criteria is the same. However, when our workload has sufficient parallelism, we can make a simplifying observation. As long as we are not constrained by overheads or by resources outside the processing core (for instance, memory bandwidth), then performance will scale linearly with the number of cores. In this case, we can optimize the design of the cores by looking at the performance versus area or power tradeoffs of a single core.

Naturally, this simplifying assumption does not always apply. Nevertheless, it is instructive to look at the performance and area tradeoffs of a single core as part of the design process for a parallel architecture. This performance versus area tradeoff is the analysis we perform in the rest of this paper.

3 Co-optimization Methodology

To perform a design space exploration that spans both the architecture and circuit design, we first model each of the spaces independently, and we then establish the appropriate relationships between the two design spaces to create a higher-level model.

The architectural design space is a large multi-dimensional space; this design space grows very quickly and making it impractical to explore fully through simulation. To resolve this issue, we use statistical data fitting methods over a relatively small number of design space samples to create a model of the design space.

To determine the circuit tradeoffs of the underlying units, we rely on a combination of RTL synthesis and CACTI to model circuit area-delay tradeoffs. We finally integrate the architectural models with the circuit tradeoff data in a high-level model. This integrative model makes the connections between circuit delays, architectural cycles and the clock cycle time to characterize the full design space. Figure 3 provides an overview of the entire optimization framework.

3.1 Performance Model

For our workloads, we use VISBench, a set of visual computing applications representing a variety of existing and emerging applications related to the display and processing of visual information. These applications serve as a proxy for the class of applications targeted by future GPUs and accelerator architectures. The benchmarks are summarized in Table 1. The VISBench applications are further described in [12]. We use hand parallelized versions of the applications. Parallel loops are annotated with dummy function calls that mark the boundaries of parallel execution.

We run our annotated binaries sequentially through a functional simulation front-end that simulates x86 code. This front-end fast-forwards the sequential code to reach the parallel portions that would run on our accelerator. The front-end detects the dummy function calls that serve as thread boundaries and generates an instruction trace for each thread. A cycle-accurate timing model simulates the performance of all cores. The submodel for each core executes the instruction traces of the threads it has been assigned.

We simulate a 64-core multiprocessor. In order to focus on the architecture of the core, we overprovision the shared resources on the chip. We simulate private L1 caches and a shared L2 cache of 8MB, banked 32 ways to allow up to 32 simultaneous accesses, a number sufficient to minimize contention. The latency of the L2 cache is 18 cycles with no contention. We simulate a memory system that can transfer up to 128 bytes from memory to the global cache per cycle, a bandwidth comparable to high-end GPUs. The latency is a minimum of 50 cycles with no contention, and may be higher if many cores access the same memory channel. Again, the memory bandwidth, while achievable in a real processor, is overprovisioned so as not to be a bottleneck.

We use this simulation environment to generate an analytical form of the architectural design space. We use statistical approaches similar to those proposed in several works [7, 10]. The general approach constitutes randomly sampling the design space through simulation, and then using mathematical data fits to characterize the effect of the parameters on the performance. By using these statistical approaches, we can limit the number of simulations that are required to map a design space. While previous work used either cubic splines or artificial neural networks, we chose to apply a convex fit instead. The convex fits were able to provide us with good fits—cross-validation errors of lower than 5% on average—while allowing us to leverage convex optimization techniques during the optimization phase.

We capture the lower-level micro-architectural parame-
Figure 2. Overview of the optimization framework. The architectural simulator and model fitting generate the architectural design space. The circuit tradeoffs define the circuit design space. The link is made through a higher level model which then feeds to the optimizer.

3.2 Circuit Models

To characterize the area-delay tradeoffs of circuits, we rely on a mixed approach of using verilog models for logic portions of the processor, and CACTI for memory units. For logic units, such as the ALUs and the decode stage, we use verilog implementations which we synthesize for different delay targets using Synopsys Design Compiler and a 90nm technology. Design Compiler tries to meet each target delay while attempting to minimize area, yielding circuit implementations with varying topologies and gate sizings. By sweeping the delay target and creating designs with different delay and area characteristics, we are able to map out the tradeoff space between the area and delay of circuits.

We should note that parameters such as supply voltage, Vt or circuit style are not included in the circuit design tradeoff space we currently generate—only logic synthesis and gate sizing are circuit parameters. This restriction is only due to Design Compilers abilities. If we had more design data points that included designs of different circuit styles or with different threshold voltages, these could easily be incorporated into the circuit tradeoffs.

For the memory units, we use CACTI 6.0 to explore the SRAM design tradeoff space. CACTI internally performs an exhaustive exploration of several important SRAM design parameters. Because we are interested in the area-delay tradeoffs, we modify CACTI to dump area and access latency information for all the configurations it explores during its search. We then analyze this data to generate the Pareto-optimal tradeoff curve of access time versus area for the desired SRAM.

Since cache size is one of our micro-architectural optimization parameters, we also need to include cache size in the circuit tradeoff space for caches. In these cases, therefore, we run CACTI for different cache sizes, gathering area and delay information for each size. We then compile this data into a 3-dimensional delay-size-area tradeoff. This way we essentially have a function that defines the area cost of a cache as a joint function of its size and delay. The optimizer can then use this information to jointly optimize cache size and delay.

3.3 Architecture to Circuit Link

We have described an architectural performance model and the circuit-level area-delay tradeoffs. As a final step before optimization, we must link these two models together to form the complete area-performance design space.

The architectural latencies indicated by the architectural simulator are linked to the physical delays of the circuit tradeoffs through the clock cycle time parameter. Ignoring any overheads caused by pipeline registers, the number of pipe stages is the delay of the circuit divided by the clock period, simply cutting the logic into stages. If we add delay overheads for pipeline registers we get following relation-
use multithreading to hide memory latency. We compare single threaded cores versus 2 and 4 threads per core. The additional cost of multithreading is additional complexity in the front end of the pipeline. To allow instructions from a thread to continue processing even when other threads are stalled, we need to introduce a different set of pipeline registers for each thread. We also need muxes to select a thread for each pipeline stage. Another significant cost is that we need a separate register file for each thread. The addition of multiple register files not only increases the area through replication, but the additional decoders required to access the register files also increases the physical access time.

At the micro-architectural level, we optimize several parameters. We explore L1 I-cache sizes from 2KB to 32KB. D-cache size is independently optimized and can vary from 4KB to 64KB. We also explore the latencies of different architectural units. Fetch latency, dcache latency and decode latency are allowed to vary from 1 to 3 cycles. Integer ALU latency can vary from 1 to 4 cycles, and FP ALU latency from 3 to 12 cycles. Hence, the total integer pipeline varies from 5 stages to 14 stages.

Tables 2 and 3 summarize the design space. We simulated the performance of approximately 750 randomly chosen microarchitectural configurations for each macro-architecture to create convex models of the design space.

5 Results and Analysis

After creating the architectural models, and characterizing the underlying circuit tradeoff space, we link the data
Figure 3. Performance-area tradeoff space for based on an average of all six benchmarks. Area is in \( \mu \text{m}^2 \). Four area-efficient designs are annotated, for which we detail the micro-architectural and circuit parameters in Table 5.

<table>
<thead>
<tr>
<th></th>
<th>Design 1</th>
<th>Design 2</th>
<th>Design 3</th>
<th>Design 4</th>
</tr>
</thead>
<tbody>
<tr>
<td>performance</td>
<td>0.069</td>
<td>0.316</td>
<td>0.553</td>
<td>0.685</td>
</tr>
<tr>
<td>issue width</td>
<td>1</td>
<td>1</td>
<td>2</td>
<td>2</td>
</tr>
<tr>
<td>threads</td>
<td>1</td>
<td>2</td>
<td>4</td>
<td>4</td>
</tr>
<tr>
<td>clock freq</td>
<td>265 MHz</td>
<td>1.7 GHz</td>
<td>1.4 GHz</td>
<td>1.4 GHz</td>
</tr>
<tr>
<td>icache</td>
<td>2.0 KB</td>
<td>2.0 KB</td>
<td>2.0 KB</td>
<td>4.6 KB</td>
</tr>
<tr>
<td>dcache</td>
<td>4.0 KB</td>
<td>4.7 KB</td>
<td>9.6 KB</td>
<td>27.1 KB</td>
</tr>
<tr>
<td>FP lat</td>
<td>9.3 ns</td>
<td>3.5 ns</td>
<td>3.0 ns</td>
<td>2.8 ns</td>
</tr>
<tr>
<td>IntALU lat</td>
<td>3.0 ns</td>
<td>1.17 ns</td>
<td>1.5 ns</td>
<td>1.17 ns</td>
</tr>
<tr>
<td>Fetch lat</td>
<td>0.95 ns</td>
<td>0.87 ns</td>
<td>0.71 ns</td>
<td>0.71 ns</td>
</tr>
</tbody>
</table>

Table 4. Optimal configuration of four design points on the area-efficient frontier.

Examining Figure 3, we can see that as our area budget increases from about 0.1 \( \mu \text{m}^2 \) to 1 \( \mu \text{m}^2 \), the most efficient architecture changes. For very area-constrained designs, the simplest single-threaded (ST), single-issue machine is the only architecture that can meet the objective. As we increase the area budget, the optimal architecture gradually changes, first moving to a single-issue 2-way multi-threaded (MT2), then to a dual-issue superscalar MT2 design, and finally to a dual-issue 4-way multi-threaded machine (MT4).

These results indicate that multithreading is initially a more cost-effective design choice in comparison to superscalar issue. This is expected, since the amount of latency and area overhead due to multi-threading is lower than for superscalar design, and yet it is an easy source of additional work for the processor which can be used to hide stalls in individual threads.

Neither the single-issue MT4 architecture, nor the dual-issue ST architecture is ever an area-efficient choice. This result is not completely unexpected either, as we expect an efficient design to be balanced. Both of these architectures allocate a lot of area resources to improving either superscalar issue or multi-threading, but ignore the other. This results in an imbalanced result which leaves more cost-effective sources of performance unexploited.

In Table 5, we examine some of the underlying micro-architectural and circuit parameters in more detail. We see the clock frequency rise rapidly as we push for more performance. There is, of course, an area cost in terms additional pipeline registers, but the benefits of frequency scaling seem to easily overshadow these costs, as is indicated by the immediate jump from 265 MHz to 1.7 GHz from Design 1 to Design 2. In Designs 3 and 4, which are superscalar MT4 architectures, physical limitations of more complex circuits limit frequency to 1.4 GHz, a number lower than the simpler single-issue Design 2.

Other things to note include the minimal amount of area allocated to the I-cache in most cases versus increasing amounts of area spent on the D-cache. The small I-caches are the result of the small instruction footprints of these ar-
applications on average. We also see a gradually decreasing FP circuit delay. In some cases, we see circuits that have reached their physical delay limit, as is the case with the 1.17 ns integer ALU.

So far, we have examined the area-performance design tradeoffs for an individual core. In parallel architectures, however, core count is variable. If we assume linear performance scaling with the number of cores, then the optimal configuration that we should choose should be the one which maximizes the ratio of performance to area. In the figures, a line is drawn from the origin through this optimal configure, which shows the area-efficient frontier for the whole system under these assumptions. Designs on the line are efficient, while those points far above the line have poor performance to area ratios, and are hence poor choices for an accelerator architecture.

Interestingly, these results indicate that for our set of benchmarks, both the single-issue MT2 architecture and the dual-issue MT4 architecture are good choices. While the single-issue MT2 core provides less performance, its lower area allows for more cores to be fit on die. On the other hand, the larger area of the dual-issue MT4 core means less cores can be included on the chip, but it makes up for this through better individual core performance.

While the optimization of the architecture needs to be done for the set of all benchmarks (as we have done), looking at individual optimizations of each the benchmarks can be instructive on the internal workings of the optimization as well. Figure 4 shows these results. Each of the optimizations favor different aspects of the architecture—a reflection of the different behaviors of the benchmark applications. For example, the MRI benchmark has a lot of dependent FP operations. The FP unit being a longer latency function, MRI favors a high degree of multi-threading in both the single-issue and dual-issue cases to hide stalls that would otherwise degrade performance. As another example, MRI, Facedetect and H.264 put virtually none of their area resources into the I-cache size, POVRay and ODE allocate area to I-cache size when performance is pushed hard, while Blender depends most heavily on larger I-cache sizes to achieve higher performances. This reflects how well the instruction footprint of each of the applications fits in the I-cache, and thus how critical the I-cache size is to performance.

5.1 Limitations

The work we have presented in this paper is preliminary, and the results still need to be validated against real designs. We are confident in the general trends indicated through our design space exploration example, but with more validation and some polishing of the models we are using, we expect there could be some changes to the results. Below, we mention some the specific limitations of our current study. Most of these limitations are not intrinsic to the methodology, and can be resolved with more accurate modeling.

We begin with limitations/inaccuracies in architecture modeling. In the performance simulator, we do not model overheads due to synchronization at barriers. We assume that we can implement a fast barrier mechanism, and with our shortest threads running around 20K instructions, the relative overhead of barriers should be small. Another limitation is that we do not vary the parameters of our shared resources, such as L2 cache latency, and these parameters are not included in the architectural models. Consequently, the optimizer cannot properly account for changes in the L2 access time penalty as core clock speed is varied. This problem could be solved adding the L2 cache access latency into the architectural models. Despite these limitations, we believe that our analysis is reasonable since we are focusing on the performance at the core level.

There are also limitations on our area model. We are naturally limited to the accuracy of the estimates made through our synthesis flow and CACTI for the logic and SRAM components respectively. These methods cannot capture the full range of the physical design space. Furthermore, we synthesize and verify pipeline components rather than the entire processor, and this could have an effect on the synthesis results. Finally, our current model of pipelining assumes linear growth of pipeline registers, which is not completely accurate. Some of our experiments seem to indicate that the sensitivity of our results to pipeline register growth is not too high.

There is some degree of mismatch between the performance and area models. In the performance model, we assume an ISA consisting of RISC micro-operations derived from x86, while in the verilog area-delay models, we assume an Alpha ISA. This may skew the absolute value of performance/area ratio of a particular design point, but we believe that it only minimally affects the relative ratios of different design points.

The final limitation, which is the only limitation of the actual methodology in its current state, is that the optimization procedure produces results that are continuous in nature. This is not a serious issue for parameters like cache size, but has to be managed for parameters such as the number pipe stages in a functional unit. Some post-optimization snapping of the results to discrete values is required. We expect that through this ‘discretization’ step, we may lose some design efficiency, but that the general conclusions indicated by the optimization will still hold.

6 Related Work

There has been a large body of work on microprocessor optimization, both in the architecture and circuit design domains. Some recent works in large-scale architecture optimization focus on modeling the high-dimensional architecture design space through statistical sampling and model
Figure 4. Performance-area tradeoff space for each individual benchmark. Area is in $\mu m^2$. 
focusing on area-efficiency there have been a lot of works as well. An early example of area-efficient architecture was the RISC project. The motivation for RISC was to put a whole processor onto a single CMOS chip [16]. Likewise, the motivation for early CMP work was to put a multiprocessor onto a single chip, thus achieving high-throughput on multiple applications [14].

A number of CMP studies have focused on area-tradeoffs for maximizing throughput/area or throughput/watt. In one early study, Huh et al. compare fixed-area CMPs made up of either in-order or out-of-order cores, and find that on SPEC workloads the out-of-order configurations with fewer cores still provide higher throughput [6]. Kumar et al. examine the microarchitectural optimization of cores [8] and on-chip interconnects [9], again using SPEC workloads. There have been other studies examining on-chip communication networks [1], the impact of core count, cache hierarchy, and interconnects on CMP power consumption [13], and the cache design space for many-core CMPs [5]. Finally, Li et al. perform a design space exploration with core count and core complexity under various power and area constraints, using analytical models to derive area costs for a large set of design points from a much smaller set [11].

7 Conclusions and Future Work

We have developed an optimization framework that allows for the systematic co-optimization of trade-offs in the architecture and circuit design spheres. By incorporating the physical design trade-offs in the architectural analysis, this framework helps guide the overall microprocessor design process by providing insight into the true impacts and costs of architectural design decisions.

While our optimization framework can be used to analyze both area and energy costs, in this paper we applied our methodology to perform an area-performance design space study. The design space we examined was the core design of a highly parallel accelerator architecture. For our set of benchmarks, we mapped out the area-efficient frontier, and found that as we increase the area budget of a single core, the design changes from a single-issue single-threaded design to a multi-threaded design and then to a dual-issue multi-threaded design.

Analyzing the results for maximum performance/area, we found two architectures that were equally effective: a high frequency, single-issue 2-way multi-threaded core, and a lower frequency, but architecturally aggressive dual-issue 4-way multi-threaded core. The result demonstrates how the interaction between the architecture and the circuits can affect the results, as the more advanced architecture had to scale back frequency to account for the more complex logic in its stages.

As future work, we plan to include energy consumption into our study to generate an analysis of energy-efficiency in conjunction with area-efficiency. Moreover, we intend to expand this work to explore large, out-of-order general purpose microprocessors as well as simple cores for accelerators. We hope to examine the contrast between the design space optimization for accelerator and general purpose architectures, as well as to examine the tradeoffs between core complexity and parallelism within parallel architectures.

References


