Quantifying Acceleration: Power/Performance Trade-Offs of Application Kernels in Hardware

Brandon Reagen, Yakun Sophia Shao, Gu-Yeon Wei, David Brooks
Harvard University, Cambridge, MA, USA
{reagen, shao, guyeon, dbrooks}@eecs.harvard.edu

***Abstract*—As the traditional performance gains of technology scaling diminish, one of the most promising directions is building special purpose fixed function hardware blocks, commonly referred to as accelerators. Accelerators have become prevalent in industrial SoC designs for their low power, high performance potential. In this work we explore thousands of implementations of classical software workloads in hardware. This thorough, detailed design space search of hardware accelerators gives architects a quantitative way to reason about the differences in implementations. The exploration presented in this work shows that the space is full of poor design choices. By thoroughly analyzing each benchmark, we show which provide the most performance when implemented in hardware given a fixed power budget and explain which design techniques work best for each workload.**

***Index Terms*—Accelerator, Design Space Exploration, Power Performance Trade-offs**

I. Introduction

Over the past decade, power has increasingly constrained performance in modern general purpose cores. In light of diminishing performance gains from process scaling, a paradigm shift is necessary to sustain performance improvements in future technology generations. Much of the high power/performance cost associated with computing on general purpose cores is due to control flow and data movement overhead. Specialization, which eliminates the majority of these costs, offers a viable solution.

Hardware accelerators, a type of specialization in the form of fixed, custom circuits designed for specific tasks, are already used across several computing domains. Currently accelerators are designed by hand. This approach is very time consuming and doesn't reveal the higher level trade-offs of a design. Thus to optimize and understand accelerator power/performance, a larger design space must be considered.

Transitioning to a quantitative accelerator design approach, this paper presents extensive design space explorations, considering multiple architectural and circuit parameters for each benchmark. Our analysis pinpoints how properties of each benchmark affect power/performance trade-offs in the design space. Through an understanding of these properties, we show that the design space is substantial and should be understood before optimizing a particular implementation.

Specifically, we quantify the power/performance trade-offs of accelerators compared to an extremely low-power general purpose processor. Our results show that accelerators demonstrate impressive energy efficiency relative to general purpose cores across all benchmarks. Moreover, we observe that even optimal designs along Pareto frontiers exhibit large variations in their power/performance - which we credit to workload intrinsic characteristics. A true understanding of power/performance trade-offs is further developed by breaking down each workload in terms of instruction composition, accelerator area, and energy consumption. Our analytical view of accelerator design shows and makes sense of the large, diverse power/performance space.

The contributions of this work are:

1) We perform a thorough design space exploration for accelerators sweeping both architecture- and circuit-level parameters.
2) We compare different benchmarks' power/performance characteristics along Pareto frontiers and demonstrate how unique workload characteristics can lead to substantially different accelerator designs.
3) We show relative power/performance trade-offs by analyzing the Pareto optimal designs and demonstrating which workloads benefit the most from hardware acceleration.

The remainder of the paper is organized as follows: Section II discusses related work on hardware accelerators and design space exploration. Section III details the workflow and benchmarks used to build the design space. Section IV presents our analysis of the accelerator design space. Section V summarizes the takeaways of this work.

II. Related Work

General purpose computing offers the flexibility to run arbitrary workloads but often exhibits less performance and energy efficiency than specialized hardware [1]. The term accelerator, a type of specialization, is often a bit ambiguous. As the efficiency of specialization proves an effective remedy for these general purpose inefficiencies, many innovative implementations of accelerators have emerged. In [6] [9] [5] different approaches are taken to integrate specialized functional units of various granularity directly into a general purpose core. In [3] [8] [7] accelerators are considered as SoC components where data must be offloaded to the accelerator's own memory before the computation can begin. We refer to these types of accelerators as tightly- and loosely-coupled respectively [2].

The advantage of a tightly-coupled design is that there is minimal data communication and no invocation overhead. Loosely-coupled accelerators are advantageous in that special memories can be implemented to better suit a workload's needs, and designs have fewer restrictions as they need not be implemented with the core directly. In this work we focus on loosely-coupled accelerators for the large design space, assuming an ideal memory subsystem.

In [8], Lyons et al. propose an architecture to support systems with tens to hundreds of accelerators. They find accelerator area is dominated by SRAM and through the Accelerator Store they develop a way to share underutilized SRAM for little to no

978-1-4799-1235-3/13/$31.00 ©2013 IEEE

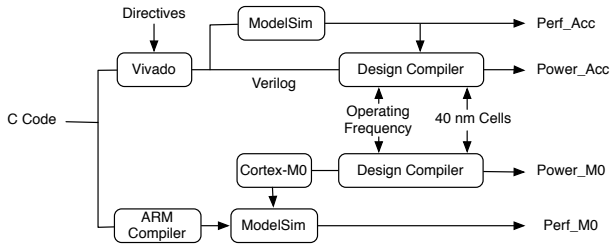

Fig. 1: Workflow used to collect power/performance metrics for each design point.

power/performance penalty. This architecture lowers the area price of loosely-coupled accelerators.

Cong et al. present a design space trade-off study for reconfigurable FPGA-based high performance systems using a high-level synthesis tool [3]. The design space they focus on is relatively modest, as designer's knowledge was used to manually prune the space.

As more hardware accelerators are incorporated in SoC designs, memory subsystem and communication mechanisms will become increasingly important. However, the design space and trade-offs of the accelerator design are not well understood. With this work, and the others mentioned above, we have taken a bottom up approach to accelerator rich architectures. This understanding of the core design and power/performance trade-offs allows inclusive, systems level integration questions to be studied.

In contrast to prior work, we consider a much larger design space with low level circuit estimates of power, area, and timing results for custom, fixed function accelerators. Focusing on a large set of designs of loosely-coupled accelerators gives us the design freedom to study unconstrained accelerator designs. Memory subsystems and communication techniques are interesting, related problems but orthogonal to this work. Here we set out to answer core design questions and hope this work paves the way for future memory subsystem studies.

III. Methodology

This section explains the synthesis flow used to generate accelerators, benchmarks, and a general purpose core.

A. Workflow

Figure 1 shows how we generated the design space for each accelerator. Xilinx's Vivado, a tool for High Level Synthesis (HLS), transforms C code to RTL; applying various directives to the source C code allows us to generate hundreds of unique accelerator implementations. Mentor Graphics's ModelSim captures accurate cycle counts and activity factors of these accelerators. Activity factors and RTL are then fed to Synopsys's Design Compiler to obtain power and area estimates of the generated accelerators based on circuit-level power analysis and resource utilization. Both the general purpose core and accelerators assume an ideal memory model. This was done to make sure we fairly compare performance and to completely understand the computational limits before further restricting designs with different memory subsystems.

1) C-to-RTL Synthesis - Vivado: Vivado is Xilinx's HLS solution capable of generating RTL from C, C++, and SystemC. Vivado provides directives that perform well known architectural level optimizations, allowing designers to trade-off performance, power, and area at a high level. We swept a large subset of all available directives to understand their effects on accelerator design. Loop unrolling, memory partitioning, loop pipelining, and using multi-staged, pipelined multipliers had the most pronounced effects on power/performance.

Directive	Start	Stop	Step Size
Loop Unrolling	0	N	next = prev * 2
Array Partitioning	0	N/2	next = prev * 2
Pipelining	0	7	next = prev + 1
Multiplier Stages	0	6	next = prev + 3

TABLE I: Directives applied when sweeping architectural parameters, plus parameter ranges and steps.

Table I, shows how the directives were swept for each benchmark. Loop unrolling and functional unit directives shape the core of the accelerator. Higher unrolling factors and more aggressive functional units typically result in better performance and higher power. Array partitioning increases the bandwidth available to each core. Loop pipelining is mostly a scheduling technique. The pipelining directive coordinates the core's resources and memory bandwidth with natural partitions of work that the benchmark can be parsed into.

2) RTL Simulation - ModelSim: ModelSim was used to simulate the Verilog generated by Vivado. The simulations yield accurate cycle counts and detailed activity factors on a per gate basis. We found the activity factors have a large effect on power estimates in accelerators and as such were supplied as input to Design Compiler to generate a more accurate power estimate. Our problem sets, the benchmark inputs, are generated uniformly at random. We find that this distribution yields relatively high activity factors (a lot of switching).

3) RTL Synthesis - Design Compiler: Design Compiler synthesizes the RTL implementations of accelerators to a gate-level netlist representation using a commercial 40 nm standard cell library. During synthesis, activity factors generated during simulation are used to refine the power estimate for each accelerator. We used 2 ns, 5 ns, and 8 ns clock periods to capture cell selection differences at the circuit level. This frequency sweep ensures that a large design space is explored, as the period can greatly influence the power/performance of a given RTL design. We also used different multiplier implementations from Synopsys's Design Ware libraries to build a library of functional units.

B. Accelerator Workloads

We designed accelerators for workloads in the SHOC benchmark suite [4]. SHOC covers a large application space that includes both memory- and compute-bound algorithms. We study and present accelerator designs for five benchmarks from SHOC: Triad, Reduction, Scan, Stencil, and GEMM. Each of the benchmarks selected, though relatively simple, includes a unique, intrinsic algorithmic property that illustrates how different workloads require different design space analysis. The source C code is written in such a way that it compiles on a general purpose core and is synthesizable to Verilog. Each benchmark in the suite is scalable and as such our findings are independent of the problem size. To ensure a fair comparison, we use the sequential versions of each SHOC benchmark. This

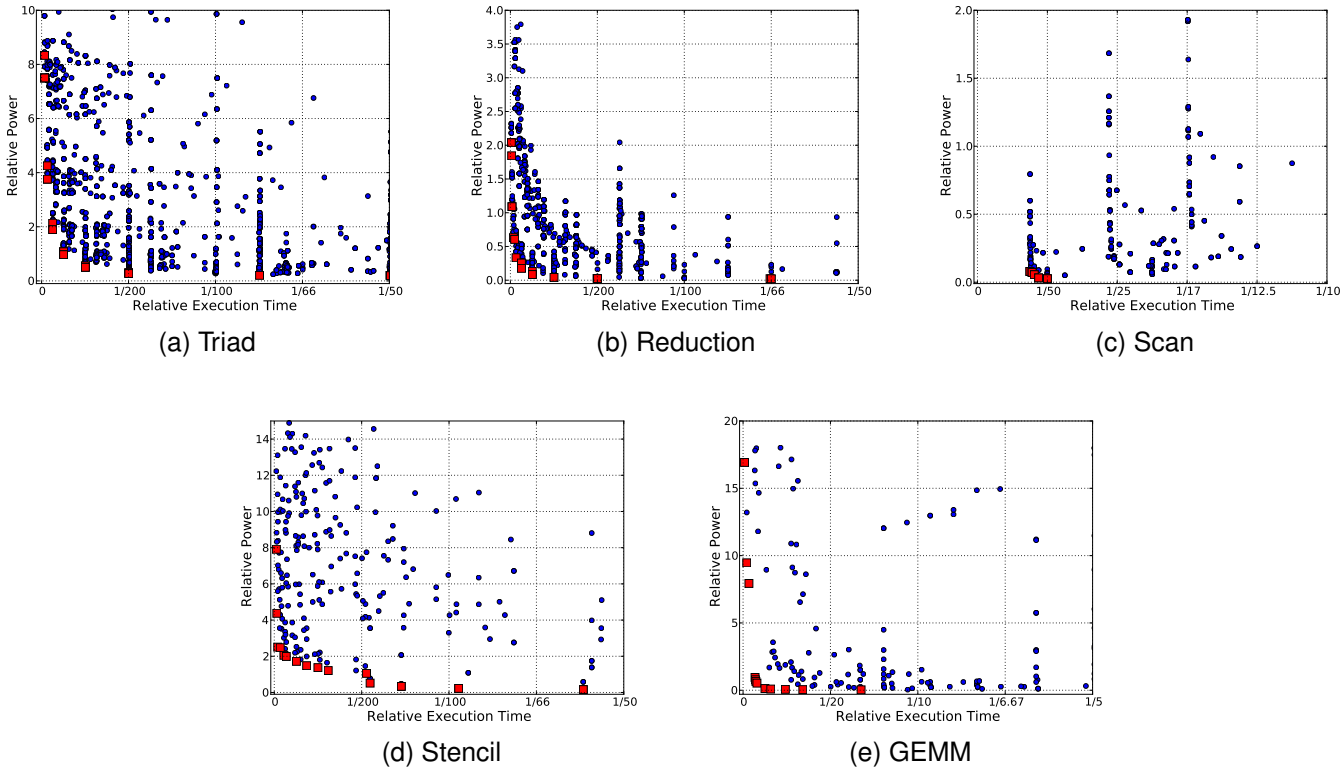

Fig. 2: Set of plots showing the highest performing and lowest power accelerator designs for each benchmark as a results of the design space search. All results are normalized to Cortex-M0's power and execution times. The Pareto points are highlighted as squares.

implementation is well suited for the Cortex-M0 and provides Vivado sufficient detail to extract necessary parallelism for the accelerators.

1) Triad: Triad takes as input a scalar s and two arrays A, B. It adds the i-th element of A to the product of the i-th element of $B \times s$ saving and returning results in an output array, C.

2) Reduction: Reduction computes and returns the sum of the elements of an input array.

3) Scan: Scan takes an input array of size N and outputs an array of size N whose j-th element is the sum of the first j elements of the input array.

4) Stencil: Stencil applies a 3×3 filter to an image stored in an array. The dot product of the filter and each possible 3×3 subset of the source image are computed, and the result is saved in an output array indexed by the middle pixel of the 3×3 source image's subset.

5) GEMM: GEMM here refers to an $O(n^3)$ implementation of dense matrix multiply. We use the GEMM notation as the benchmark uses integer, as opposed to single-precision, inputs.

C. General Purpose Comparison- ARM Cortex-M0

Cortex-M0 is the smallest ARM processor available. It has a 3-stage pipeline design and targets low-power, embedded applications. We chose a Cortex-M0 to investigate how much workloads benefit from acceleration when compared to a very low power general purpose core. We have a RTL implementation of the Cortex-M0[1] that can run the same C source code that we synthesize. To make a fair comparison, the RTL implementation of the core was run through the same synthesis flow as that used for the accelerators.

[1]ARM's RTL implementation available to universities through DesignStart

IV. Results

In this section, results of the design space exploration are presented, broken down, and analyzed. We begin by explaining the design space and high-level insights. Next, each workload's design space is decomposed. The decomposition includes quantifying the advantages of acceleration compared to running the benchmarks on a Cortex-M0 and the differences in Pareto optimal designs across each benchmark. We conclude by comparing benchmarks to see which accelerators offer the best performance relative to the Cortex-M0 and the differences in energy efficiency between workloads with different characteristics.

A. Design Space Exploration

The design space exploration can be broken into two distinct parts: the HLS sweeps done to generate different accelerator microarchitectures and the netlist synthesis sweeps.

1) HLS - Microarchitecture Level Sweeps: The four directives swept are loop unrolling, array partition, pipeline stages/issue latencies, and multiplier stages. Many more directives are available, however these four are the most notable as they have the most pronounced effects on each design's power/performance. As described in Section III, loop unrolling and functional unit selection constrain core resources, array partitioning governs available memory bandwidth, and loop pipelining schedules and partitions benchmarks into distinct parts. We found that increasing loop unrolling factors and array partitioning combine to exploit parallelism and increase

performance while simultaneously increasing power. Specifying more aggressive functional units (i.e, pipelined multipliers) also yields better performance with increased power costs. Loop pipelining provides performance benefits and, since it is mostly a scheduling construct, only uses the the functional units already instantiated so that no notable additional leakage or internal power costs are incurred.

2) RTL Synthesis - Operating Frequency Sweeps: The power and performance numbers presented here were generated by sweeping periods of 2, 5, and 8 nanoseconds. This sweep captures the different trade-offs of the same designs running at different frequencies. The trends of the frequency sweeps are as expected: high frequency results in better performance and higher power while low frequency typically provides lower power, slower solutions.

B. Quantitative Design Space Analysis

With an understanding of the effects of each directive on the accelerators design we now dissect the design spaces of each of the benchmarks.

Figure 2 shows the design space of each benchmark; the results are normalized to those of the Cortex-M0 to show the power/performance gains. Each benchmark originally consisted of thousands of points. To capture the differences in Pareto optimal designs, we pruned points on the extreme ends of the curves - points at which the Pareto curve's slope are nearly infinite or zero. Figure 2 suggests that even for a fixed number of basic optimizations applied to relatively simple benchmarks the design space is too large to comprehend with traditional methods.

1) Triad: Figure 2a shows Triad, a completely parallel, DOALL, non-nested benchmark performing a simple computation. Triad's intrinsic properties, its parallelism, represent accelerator designs that, with more resources, always result in greater performance at a linear power cost. This intuition is validated as the plot shows the performance can increase until execution takes a single step (load, execute, store) given unlimited resources. Triad is straightforward and the only real subtlety is confirming a design is Pareto optimal. The vertical points in Figure 2 show that a single speedup maps to many different power costs. Hand-coding an accelerator could easily result in one of these design points.

2) Reduction: Each iteration in Reduction produces some data used in the calculation of the final sum. The most efficient implementation of this algorithm is a tree adder. As shown in Figure 2b, the Reduction benchmark yields many optimal designs. We also find that, through expression balancing, Vivado is able to identify and synthesize a tree adder. Most of Reduction is control overhead as the only computation is add. This results in very fine-tuned circuitry and substantial power savings. The two curve patterns in the plot are for the frequencies swept at netlist synthesis time. Since the power/performance benefits of accelerating Reduction are so large, running the circuit with a 2 ns period yields the best results. Generally, intuition leads one to believe that higher frequency results in higher power consumption. The performance gains seen by Reduction are from eliminating most of the control overhead. The actual computation, the additions, dominate the power, which for Reduction means speeding up the control sections by running at a higher frequency decreases execution time for less increase in power than with other, more compute intensive, benchmarks.

3) Scan: Since each stage of Scan requires the sum to be computed, saved, and used as input to the next computation, the HLS tool cannot break this loop-carried dependence as it must wait for the previous iteration to finish. Even with an ideal memory model and unlimited hardware resources, the cycles necessary to complete the computation quickly reach the lower limit. Figure 2c shows that there exists a limit on the performance Vivado is able to extract from a given algorithm. The three sharp curves each correspond to different frequencies reaching their performance limit. Scan would benefit more from algorithmic changes that mask this dependence than from circuit or microarchitectural optimizations.

4) Stencil: Stencil is as parallel as Triad but at a coarser granularity. If we consider each 3×3 dot product of Stencil to be a single iteration then the iterations can be executed in parallel. The added complexity comes from the need to sum the outputs of the nine multiplications, a Reduction, and the amount of reused data between iterations. Unlike Triad, the input data is used more than once in many instances. Reusing loaded data increases computational density giving Stencil a much different characterization than Triad. In Figure 2d when considering a power budget less than or equal to that of the Cortex-M0, enough overhead is eliminated to support instantiating power hungry multipliers which increases performance by orders of magnitudes. These savings are possible as even modest unrolling factors allow heavy data reuse in Stencil while simultaneously removing control costs. Such optimization save considerable time by eliminating repetitive loads, stores, and simple bookkeeping.

5) GEMM: Matrix Multiply has a similar structure to Stencil but at an even coarser granularity. Since each calculation must use data from an entire row and column, it requires a much greater unrolling factor to exploit data reuse and execute individual calculations (independent iterations) completely in parallel. GEMM is shown in Figure 2e. As the reuse and parallelism are more coarse, it takes more effort to eliminate as much overhead as for Stencil. As such, GEMM must consume more power to achieve the same performance as other benchmarks. Moreover, GEMM's computational density is so high that the required multipliers cause the average power to far exceed that of the low-power Cortex-M0. Understanding the power requirements of GEMM is necessary in designing an accelerator as goals of high performance may be completely impractical given the power requirements.

C. Understanding Acceleration Gains

In this section we first, in part 1), we compare the benchmarks run on the Cortex-M0 to baseline accelerator designs (designs with no optimizations) to show where the benefits of acceleration come from compared to a Cortex-M0. In part 2) we compare different Pareto optimal designs and show which workloads are best suited for acceleration. Finally, in part 3) we go a step further and decompose points on the Pareto frontier of two benchmarks to show what increasing accelerator performance means to different workloads along different regions of the curve.

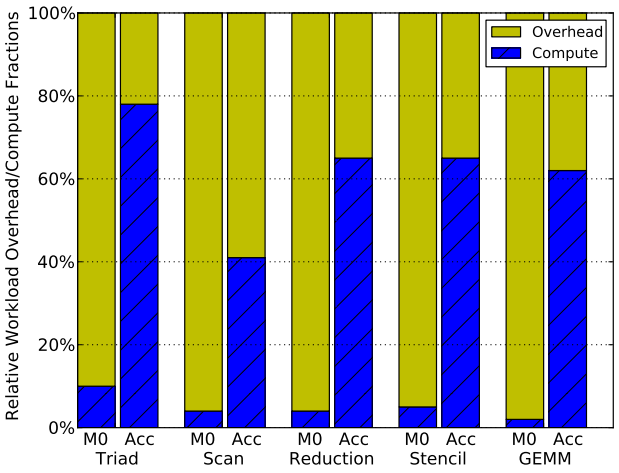

Fig. 3: A comparison of efficiencies of workloads run on a Cortex-M0 to a hardware implementation.

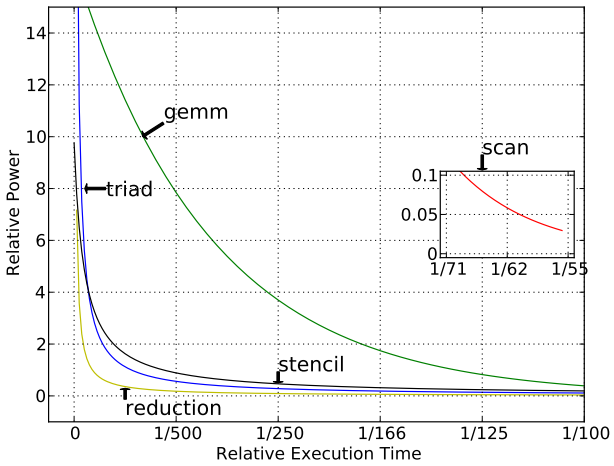

Fig. 4: Comparison of power/performance gains of accelerators against each other relative to Cortex-M0.

1) Comparing Cortex-M0 and Baseline Accelerator: There is no standard metric to compare the overheads of accelerators and general purpose cores. We chose instruction counts for general purpose cores and area for accelerators. For the Cortex-M0, only instructions executing essential computation count as compute. As there are no instructions for the accelerators we study, area not used for meaningful computation is classified as overhead. In doing so we measure the same property, although by different units, so that the two yield comparable data points when kept relative.

Figure 3 shows that despite the variation in workload structure, almost all of the Cortex-M0 execution is overhead. Each benchmark simulated on the Cortex-M0 was disassembled to machine instructions and each instruction was then tagged as either compute or overhead.

Area breakdowns presented in Figure 3 show accelerator implementations with no directives applied (baseline designs). An accelerator's area is tagged as compute or overhead by manually parsing its Verilog file and applying models of functional units included in Design Ware libraries along with post synthesis area reports. The breakdowns in Figure 3 show the M0 implementations to be much more control intensive than those using accelerators. By specializing on just one task, an accelerator incurs lower control costs and dedicates more of its resources to meaningful computation. This suggests that, as will be seen in Figure 4, the highest gains for a single accelerator are the ones with the highest control overhead and lowest computational density.

2) A Relative Understanding of Optimal Designs: With an understanding of what possible hardware implementations exist for different workloads, we compare each benchmark's Pareto frontier to see which benefit the most from acceleration. Each Pareto frontier was extracted from data presented in Figure 2 and fitted to a polynomial curve, which is plotted in Figure 4.

Surprisingly, we find Reduction to be the most effective workload implemented as an accelerator. When executed on a Cortex-M0, Reduction's cost is almost entirely control with almost negligible computational needs. An accelerator eliminates such overheads by building a custom control circuit; custom control and simple computational requirements make Reduction the best performing.

Triad has more demanding computational needs than Reduction but is completely parallel making performance gains unbounded. Triad always requires more power than Reduction to achieve a similar speedup as it requires a multiplier, which alone consumes enough power to shift Triad's Pareto above Reduction.

Scan's performance is limited at the algorithmic level and we see its performance gains are much less than the other benchmarks. However, since Scan's compute needs are similar to Reduction, only additions, we still see orders of magnitude in power savings.

Stencil's parallel structure lends well to large performance increases. However, unlike Triad, each point computation still consists of a considerable amount of control and demands more multipliers. Figure 4 shows that this increase in complexity and compute density means Stencil provides less performance as an accelerator given a fixed power budget.

GEMM is the most complex and costly of the benchmarks but also yields the highest absolute performance gains as it has high computational density. Figure 4 shows GEMM's Pareto optimal curve well above others, meaning substantial performance gains come at a large power cost to the system.

3) Efficiency Differences in Pareto Optimal Designs: To better understand the Pareto optimal designs of different workloads, Triad and Stencil are characterized further. Figure 5 highlights the optimal designs for Triad (top) and Stencil (bottom) in terms of energy and area, which present a different, more subtle finding from our design space explorations.

Triad's parallel nature implies nearly constant energy as shown in Figure 5b because such a workload can always be sped up with more resources at a fixed increase in power. Moreover, in Figure 5c it can be observed that increasing the performance of the benchmark (going from bar 6 to 1) results in a linear decrease in the percentage of overhead. This is both a positive and intuitive conclusion: optimizing for performance eliminates overhead.

Shown in Figure 5e, Stencil's energy characterization is less regular. Stencil has an energy optimal point and the energy costs

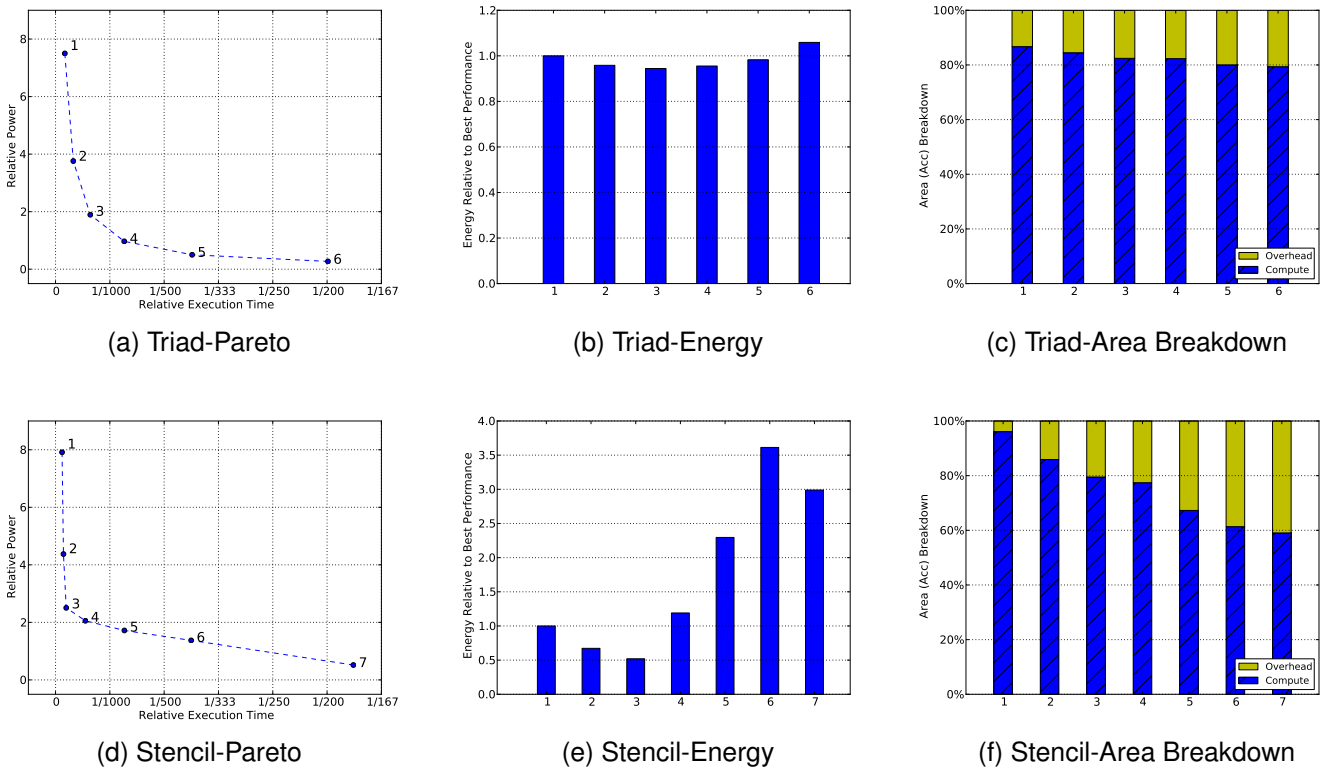

Fig. 5: Plots characterizing the Pareto optimal points in terms of energy and area breakdown.

of its Pareto optimal designs range from 0.5x to more than 3.5x as high as that of the best performing Pareto design. The fluctuation in energy is due to data reuse and resource scheduling lining up in an unintuitive way. The energy optimal point corresponds to a large unrolling factor, relatively modest array partitioning, pipelined multipliers, and non-pipelined loops. The large unrolling factor allows for the greatest utilization of loaded data for the reasonable bandwidth requirements. Since the multiplier is pipelined and the loop is not, bandwidth needs are amortized over multiple cycles maximizing the reuse (efficiency) of the given resources. This results in complex control flow and a non-intuitive energy optimal design solution.

The linear relationship between power/performance seen in Triad is not observed here. For more complex workloads, such as Stencil, the additional costs of a higher performing design may not always be justified.

V. Conclusion

In this paper we show which properties of workloads affect the shape of the large design space of hardware accelerators. Comparing the Pareto optimal curves produced by our analysis of several workloads reveals which benefit most from acceleration. By characterizing Pareto optimal implementations of benchmarks in terms of energy we have shown that non-intuitive combinations of architectural parameters can yield energy optimal designs. This work is a first step in shifting accelerator design to a quantitative, formal process.

VI. Acknowledgment

We would like to thank Glenn Holloway for his help revising this work and the anonymous reviewers for their feedback. This work was partially supported by STARnet, a Semiconductor Research Corporation program sponsored by MARCO and DARPA. This work was also partially supported by the National Science Foundation (NSF) Expeditions in Computing Award #: CCF-0926148. Any opinions, findings, and conclusions or recommendations expressed in this material are those of the authors and do not necessarily reflect the views of the National Science Foundation.

References

[1] E. S. Chung, P. A. Milder, J. C. Hoe, and K. Mai. Single-Chip Heterogeneous Computing: Does the Future Include Custom Logic, FPGAs, and GPGPUs. In *MICRO*, 2010.

[2] J. Cong, M. A. Ghodrat, M. Gill, B. Grigorian, and G. Reinman. Charm: a composable heterogeneous accelerator-rich microprocessor. In *Proceedings of the 2012 ACM/IEEE international symposium on Low power electronics and design*, ISLPED '12, pages 379–384, New York, NY, USA, 2012. ACM.

[3] J. Cong, K. Gururaj, and G. Han. Synthesis of reconfigurable high-performance multicore systems. In *FPGA*, 2009.

[4] A. Danalis, G. Marin, C. McCurdy, J. Meredith, P. Roth, K. Spafford, V. Tipparaju, and J. Vetter. The Scalable HeterOgeneous Computing (SHOC) Benchmark Suite . In *GPGPU*, 2010.

[5] V. Govindaraju, C.-H. Ho, and K. Sankaralingam. Dynamically specialized datapaths for energy efficient computing. In *HPCA*, 2011.

[6] R. Hameed, W. Qadeer, M. Wachs, O. Azizi, A. Solomatnikov, B. C. Lee, S. Richardson, C. Kozyrakis, and M. Horowitz. Understanding sources of inefficiency in general-purpose chips. In *ISCA*, 2010.

[7] H.-Y. Liu and L. P. Carloni. On learning-based methods for design-space exploration with high-level synthesis. In *Proceedings of the 50th Annual Design Automation Conference*, DAC '13, pages 50:1–50:7, New York, NY, USA, 2013. ACM.

[8] M. J. Lyons, M. Hempstead, G.-Y. Wei, and D. Brooks. The accelerator store: A shared memory framework for accelerator-based systems. volume 8, pages 48:1–48:22, New York, NY, USA, 2012. ACM.

[9] G. Venkatesh, J. Sampson, N. Goulding, S. Garcia, V. Bryksin, J. Lugo-Martinez, S. Swanson, and M. B. Taylor. Conservation cores: reducing the energy of mature computations. In *ASPLOS*, 2010.