Characterizing the Costs and Benefits of Hardware Parallelism in Accelerator Cores

Steven J. Battle and Mark Hempstead
Drexel University
ECE Department
Philadelphia, PA USA
Email: sjb328@drexel.edu, mark.hempstead@coe.drexel.edu

***Abstract*—Power and utilization constraints are limiting the performance gains of traditional architectures. Designers are increasingly embracing specialization to improve performance in the era of dark-silicon. General purpose processors are beginning to resemble SOC's from the embedded domain, and now include many specialized accelerator cores to improve computation-throughput while reducing the energy-cost of computation.**

The design-space of accelerator cores is wide and varied. Designers are able to specify how much parallelism to expose in hardware by varying input width, pipeline depth, number of compute-lanes, etc. In this paper we study three accelerator cores: DES, FFT, and Jacobi Transform, exhibiting three different types of computation: streaming cryptographic, butterfly DSP, and stencil. We investigate methods to increase parallelism within the accelerator while remaining on the pareto-frontier, and examine the trade-offs faced by designers with respect to area, power, and throughput. We present models of these trade-offs and provide insight into the design of cores under real-world constraints.

Index Terms*—*Accelerator architectures, Analytical models, Computer architecture, System-on-chip

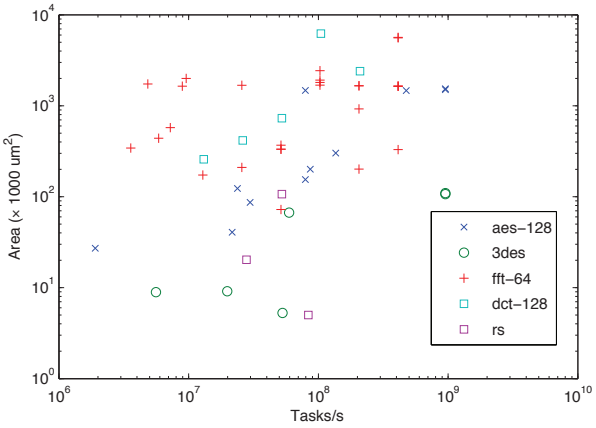

Fig. 1. Area vs. Performance (tasks/s) for fixed-function accelerator cores synthesized from OpenCores[8] and Spiral DSP Framework [9]. Optimal designs are low area (y-axis) and high throughput (x-axis).

I. Introduction

Contemporary microprocessor architectures have hit a power wall that is constraining performance. With the end of Dennard scaling[1], power density (i.e. W/mm^2) is increasing with each transistor process technology generation. To deal with power budgets constrained by cooling costs and battery life requirements, designers must shrink the size of the microprocessor or leave sections of the die intermittently unpowered, a condition recently named *Dark Silicon*[2], [3].

Hardware specialization increases energy efficiency and performance over general purpose cores through the use of specialized circuits that complete more computations for every transistor switch. This technique is used widely in system-on-chip (SoC) designs and in the hardware and software co-design communities. Many industry leaders expect the amount of specialization on chip to increase over the next 30 years.

Hardware specialization and acceleration has not been confined to embedded processors, as specialized instructions for SIMD and multimedia have been a part of general purpose chips for decades. As transistor power-density has increased, the granularity of hardware specialized accelerators has increased to encompass entire applications, similar to accelerator cores in the embedded domain. For example, mobile processors, such as processors from TI, Marvel and Apple, include accelerators for video, audio, and image processing [4], [5]; high performance systems like the IBM PowerEN processor for network processing include accelerators for packet processing and encryption [6]. Industry sponsored ITRS reports predict that a single SoC will contain over a thousand processing cores by 2021 [7]. Designers must have a detailed understanding of the performance benefits and costs of these specialized processing engines in order to make the predictions of large core counts a reality.

System designers must not only select the tasks to accelerate with hardware, they must also choose a specific implementation from a wide design space. As processors implement more functions in hardware, it is beneficial to study accelerator dataflow designs to understand the trade-offs encountered when extracting parallelism. We illustrate this wide design space in Figure 1, which shows the performance in tasks/s (x-axis) vs area (y-axis) for twelve 128-bit AES and seven DES-3 encryptor cores, 28 FFT_{64} and 6 DCT_{128} DSP cores, and four Reed Solomon cores, where each core is synthesized in an industry standard ASIC flow and performance measured at the maximum operating frequency. Each point represents a design built referencing RTL from OpenCores[8] or generated using the Spiral DSP generator[9].

Figure 1 shows a wide range of accelerator performance and area-cost; for example, the AES-128 encryptor designs from opencores (marked × in the figure) have a performance

978-1-4799-2987-0/13/$31.00 ©2013 IEEE

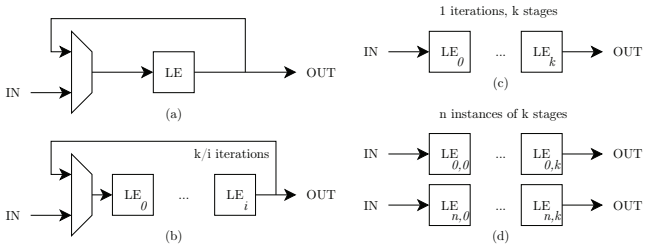

Fig. 2. Dataflow Types: (a) iterative (b) partially unrolled (c) fully unrolled (d) replicated for $n\times$ streaming throughput

range of almost three orders of magnitude, from 10^6 to 10^9 encrypts/s, and an area cost from 10^4 to 10^6 μm^2. There is a large selection of IP to choose from and for each IP many different hardware implementations of the desired function. Designers of accelerator cores can vary input streaming-width, i.e. the amount of data processed by the core logic elements; pipeline depth, the number of independent stages in the design; and initiation interval, the time between successive computations on the core. Designers need a methodology and models to better understand this wide design space and remove *poor designs* that are not area and energy efficient.

In this paper we examine several representative accelerator cores to investigate how we can extract parallelism from each kernel. We study three cores implementing different computation with different dataflow types: a DES cryptographic core, an FFT DSP core, and a Jacobi stencil core. We present models for each kernel capturing effect on area, energy, and throughput while varying the dataflow to increase parallelism in the design. These models, that capture the costs of hardware parallelism, will be of use over the next 30 years as designers continue to embrace specialization.

II. Hardware Parallelism

The amount of parallelism in an accelerator core is a function of the dataflow architecture and number of computing elements. By classifying different types of dataflow architectures, we can model and bound the amount of parallelism in any given accelerator core. Figure 2 shows several canonical dataflow classes. A serial model (a), iterates over a computation block k times until the computation is done. This model results in a blocking accelerator with the lowest throughput because new data cannot be processed until k tasks are completed. However, an iterative design requires the fewest resources: a single logic element (LE) and input mux [10].

Parallelism can be extracted by unrolling the loop in hardware at the cost of additional resources (LE's). Figure 2 (b) shows the partially unrolled function. The loop of k tasks is unrolled i times, yielding an $i\times$ increase in throughput, requiring only $\frac{k}{i}$ cycles between successive tasks. Such an architecture requires i logic elements, and still requires the input mux to select between new or iterated data. In Figure 2 (c), the loop is fully unrolled, removing the input mux overhead at a cost of k logic elements. This architecture provides the maximum throughput, pipelining successive tasks over k stages. Furthermore, parallelism can be extracted by replicating the hardware to stream wider data vectors, shown in Figure 2 (d). This is necessary when system bandwidth requirements exceed the capability of a single pipeline.

Algorithm 1 Data Encryption Algorithm

1: $IP \leftarrow permute(plainText[0:63])$ {Crypto Loop}
2: **for** $i = 0$ to 15 **do**
3: **if** $i = 0$ **then**
4: $L_i \leftarrow IP[32:63]$
5: $R_i \leftarrow IP[0:31]$
6: **else**
7: $L_i \leftarrow FINAL[0:31]$ {Criss-cross R_{i-1} XOR OP_{i-1}}
8: $R_i \leftarrow L_{i-1}$ {Criss-cross prior round}
9: **end if**
10: $KeyMixed_i \leftarrow expandPermute(L_i)$ **XOR** $subkey_i$
11: $CRP_i \leftarrow subs(KeyMixed_i)$ {Substitution look-up table: 8×6-bit → 8×4-bit}
12: $OP_i \leftarrow permute(CRP_i)$
13: $FINAL \leftarrow [R_i$ **XOR** $OP_i, L_i]$ {Final output}
14: **end for**

We explore hardware optimizations that can be applied to dataflows (a), (b), and (c) such that each case does not simply cost $k \times \{Area(LE), Power(LE)\}$. For example. a designer can increase the pipeline depth of the accelerator to improve task-throughput if each stage operates on new data, or logic may be simplified if it can be made unique per-stage. This can yield area-efficiencies if smaller logic gates can be used for increased throughput if cycle-time is reduced.

III. Methodology

To explore the accelerator core design space, we synthesize RTL descriptions of hardware in an industry standard 32nm flow using Synopsys Design Compiler. We perform functional simulations on the gate-level netlist, recording activity factors in Synopsys VCS for accurate power measurement using Primetime-PX. We choose three different accelerator types representative of the computation patterns found in many heterogeneous systems: an encryption accelerator (DES), a signal processing core (FFT), and a stencil computing engine (Jacobi Transform). We select the pareto-optimal design for each core-type from our design space as the starting point for our design studies.

IV. DES Encryptor Core

We begin our study of accelerator designs with a DES crypto-core. The Data Encryption Standard [11] codifies the Data Encryption Algorithm (DEA) shown in algorithm 1. DEA is iterative, with 16 identical processing stages in the outer loop (line 2). In each processing *round*, 64-bits of input are permuted, transformed via a lookup table substitution and *XOR*ed with a 48-bit subkey derived from the input encryption key. The subkey is computed according to an iterative key-schedule algorithm, permuting the input key to generate a unique subkey for each of the 16 encryption rounds.

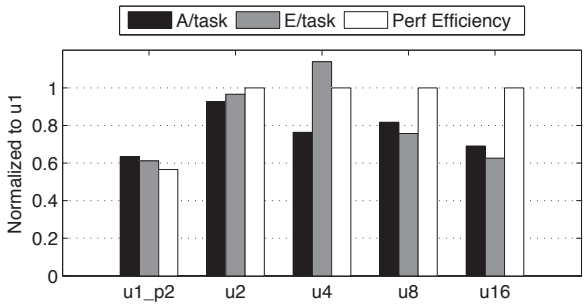

Fig. 3. **DES characteristics.** $\frac{Area}{Task}$ (Area cost of unrolling, i.e. area per number of logic elements), $\frac{Energy}{Task}$ (Energy cost of unrolling), and Performance-efficiency normalized to the baseline iterative *u1* DES core.

Design	Area (μm^2)	f (MHz)	P_{dyn} (mW)	P_{leak} (mW)	BW (Gbps)	Perf (M-tasks/s)	Para
u1	5226	625	0.93	0.09	2.58	40.3	1
u1_p2	6331	730	1.29	0.10	2.92	45.6	2
u2	9690	625	1.74	0.22	5.16	80.6	2
u4	22498	625	4.34	0.28	10.32	161.3	4
u8	34154	625	5.55	0.63	20.64	322.6	8
u16	57768	625	9.05	1.11	41.29	645.2	16

TABLE I
DES ACCELERATOR CORE CHARACTERISTICS

The basic encryption core (or logic element) consists of subkey generator, permutation, substitution transform, and XOR blocks. The baseline design selecting from opencores[12] implements a single encryption round of algorithm 1. This "blocking" core iterates over the full 16 rounds before encrypting or decrypting new data. This design is synthesized in 32nm at a 625 MHz target frequency.

A. Exposing Parallelism

We compare the costs of extracting parallelism from the iterative baseline (denoted *u1* to indicate 1× unrolling) with designs that unroll the loop 2× (*u2*) through 16× (*u16*). Unrolling the loop allows the core to support *n*-independent operations in the pipeline. Table I shows the area and power costs for each design, along with the performance of the design in tasks/s and amount of parallelism (*Para*) in the design.

A naïve approach to unrolling replicates all the hardware identically across each stage, yielding a theoretical minimum cost of $n \times A(u1)$ and $n \times P(u1)$ for n stages of unrolling. This is only a minimum, as "input-select" logic complexity can vary depending on the algorithm and the amount of unrolling, yielding designs that are superlinear in cost.

In the DES encryptor core, the cryptography function remains the same in each stage. However, the subkey generator must select the appropriate input key for the each round. There is opportunity to optimize this subkey block according to the the amount of parallelism (unrolling) in the design. The subkey generator in the iterative design generates all 16 keys, selecting the appropriate key for the current round. Unrolling the design $n\times$ means each stage's key-block is responsible for only $\frac{16}{n}$ keys. With $n = 2$ rounds, subkey logic is divided into odd- and even-rounds, requiring a thinner key-mux and less logic.At 8- and 16- rounds, a pipelined key-block is more optimal, generating all subkeys in stage 1, rather than including multiple key-select logic and master-key registers in each stage. In the *u8* design, this optimization requires only a 1-bit mux to select subkeys 0-7 vs. subkeys 8-15. In the *u16* case, there is no mux required as each key has a 1-to-1 correspondence with each round.

As mentioned before, increasing the pipeline depth is another means of increasing parallelism. We measure this effect by partitioning the basic encryption core into two stages: (1) permutation and subkey generation, and (2) crypto-substitution. This design is denoted *u1_p2* (unrolled 1×, partitioned 2×) in Table I and Figure 3. This design supports two encrypts per LE with only a single crypto and keygen block, however it requires additional registers to store the original keys and more complex mux-select logic to toggle back and forth between keys each cycle.

B. Design Analysis

We synthesized each RTL implementation, measuring the resource costs (power, energy, and area) and the performance benefits of each implementation in Table I. Figure 3 shows several characteristics normalized to the baseline *u1* design. We introduce the metric of *performance-efficiency* in this figure and shown in the following equation to compare how well each new logic element is used to improve performance (task/s).

$$PE_n = \frac{Perf_n}{Perf_1 \times n} \quad (1)$$

An ideal design will have a performance-efficiency of 1 or more, indicating that improving parallelism (number of computations in flight at once) improves performance. Efficiency less than 1 indicates that the system would be better off with *n* independent cores. We also show Area- and Energy-per-task supported by the design, relative to the *u1* baseline, measuring the *cost* of each level of loop-unrolling.

Partitioning. Design *u1_p2* achieves higher task throughput than the baseline, supporting 2 independent encrypts in flight. However, task-latency has increased from 16- to 32-cycles, while the cycle time is only 13% lower, rather than an ideal 50% reduction. The moderate frequency increase does not improve performance (task/s) beyond the *u2* design which also supports 2 tasks, as indicated by performance-efficiency ¡ 1. The *u1_p2* design has the lowest energy-per-task cost, as a single core is amortized across 2 encrypts. Deeper pipelines and more 'equitable' partitioning were explored, but required additional state elements and a superlinear increase in power and area.

Unrolling. Figure 3 shows that designs generally become cheaper as the loop is unrolled. The feedback mux is amortized over more logic elements or eliminated completely in the fully-unrolled case. Design *u4* has the most complicated subkey select mux, selecting among 4 keys in each stage and has the highest energy cost and lowest area-efficiency of the unrolled designs. Designs *u8* and *u16* with simpler muxes and pipelined key-generator require less energy per task than less unrolled designs with 'heavier' key-gen blocks. We exclude designs

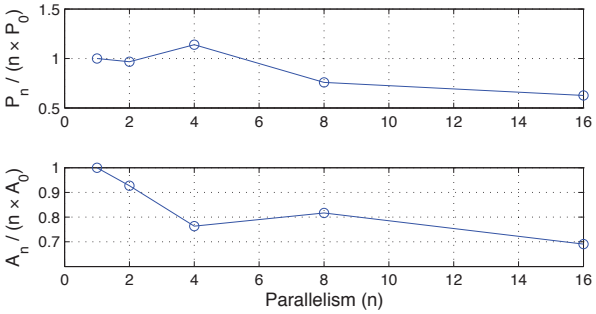

Fig. 4. **DES Parallelism**. Effective power and area cost (y-axis) per core vs. parallelism (x-axis). Naïvely instantiating n cores will cost $1\times$. Designs with lower cost for high-parallelism are optimal as core-cost is amortized.

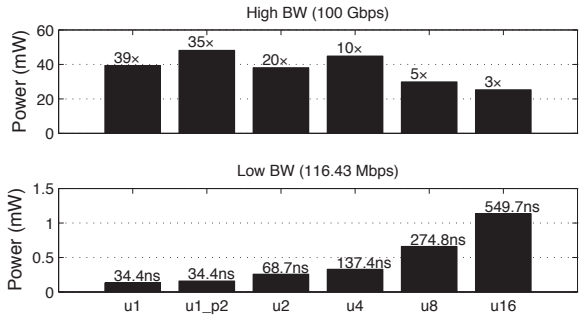

Fig. 5. **Use Case**: Power for system composed of each design in high-BW and frequency-scaled low-BW use-cases. High BW systems require multiple instances (# shown above each bar). Low BW systems are leakage-dominant with single instance (cycle-time above each bar).

without a pipelined key-generator beyond $8\times$ as these designs were not pareto-optimal. Design *u16* is most power- and area-efficient due to its less complex key-muxes.

We model area and power as a function of unrolling the loop by bounding it within the best- and worst-case designs. We see that unrolling the loop is beneficial when it reduces complexity. Figure 4 shows the effective scaling of the baseline design as the loop is unrolled. While the general trend is a reduction in power and area, point 4 is interesting. $4\times$ is the worst point in terms of power, consuming more than the baseline, yet it is more area efficient than $8\times$. Figure 4 shows that partially unrolling the loop is not power-efficient below $8\times$. Complexity shifts from mux overheads in iterative designs to sub-key generation in partially unrolled designs. These overheads are amortized when unrolling $< 4\times$ suggesting that if the power budget allows for it, unrolling is optimal when more throughput is required.

C. Use Case Analysis

It is informative to study real use-cases to understand which design is optimal given a set of design constraints.

High bandwidth. A back-end network router could be expected to process 100 Gbps of encrypted data. Each version of the DES core requires multiple instances to meet the system constraints, ranging from 39 instances of design *u1* to 3 instances of *u16*. In this case, designs *u8* and *u16* consume the least power, with the $16\times$ case able to be under-clocked to 1.9ns and still meet the system constraints.

Low bandwidth. A typical low bandwidth use-case consists of a set-top cable box recording two HD programs while watching a third, requiring access to three 10 MBps programs within three separate 38.81 Mbps MPEG transport streams[13]. Assuming the whole stream is encrypted, the decryptor chip will need to processes 116.43 Mbps of data performing 1.8192 M decrypts/s. Figure 5 shows the frequency-scaled power for each design to meet the low-bandwidth constraint with the effective cycle-time above each bar. In this domain, leakage power is the dominant factor. In this case, design *u1* is the best choice, running at 29 MHz, as it has the lowest area and lowest leakage power.

D. Model

For DES encryptors, the optimal design is unrolled $16\times$ for best throughput, power, and area efficiency. Designers should check for optimizations when partially unrolling the design. In the DES case, the fully unrolled and pipelined key-generator costs less energy than the conventional key-generator. As shown in Figure 5, the iterative designs are best for low bandwidth architectures.

Figure 4 shows that area per task decreases as the loop is unrolled n times, The $4\times$ design is an outlier, where unrolling overhead exceeds the gains from amortizing hardware. We can model the general behavior with a linear equation of the form: $A_n = A_1 \times n + b$, showing that hardware for each new round is amortized and costs less area to implement. Power also grows linearly with these pareto optimal designs, modeled by equation 3. This is due to the previously mentioned subkey generator optimizations required to stay on the pareto-curve when fully unrolled.

$$Area_{DES}(n) = 3500 \times n + 4600 \ \ (\mu m^2) \tag{2}$$

$$Power_{DES}(n) = 0.64 \times n + 0.09 \ \ (W) \tag{3}$$

$$Throughput_{DES}(n) = n \ \ (\frac{ops}{cycle}) \tag{4}$$

V. Discrete Fourier Transform

DFT cores are an example of another common hardware kernel, the butterfly operation. Figure 6(a) shows shows radix-2 butterfly kernel. This performs an addition and subtraction on each branch, where the output of the subtractive branch is scaled by a constant multiplier. Figure 6(b) shows a radix-4 butterfly, which operates on four inputs. This higher-order operation can be composed of radix-2 kernels, shown in (c).

Typical implementations of the Discrete Fourier Transform use Fast Fourier Transforms to factor high order DFTs into a series of multiplications [9], [14]. This is due to the cost of wiring and computation resources for high order radix butterfly kernels. In particular, the Spiral framework uses the Pease FFT algorithm to represent a DFT as k iterations of sparse matrix multiplication. The stages are identical except for the scaling factors applied to each kernel output. The DFT kernel is composed of an adder, subtractor, and multiplier, where

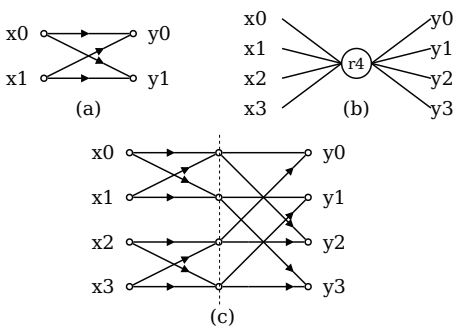

Fig. 6. (a) DFT_2 butterfly operation dataflow. (b) DFT_4 butterfly (c) DFT_4 dataflow composed of two stages of DFT_2 blocks.

Design (radix)	**Area** ($\times 10^6 \mu m^2$)	f_{max} (MHz)	P_{dyn} (mW)	P_{leak} (mW)	**BW** (Gbps)	**Perf** (M-task/s)	**Stages**
r2	1.13	140	53.9	19.3	143.8	35.1	8
r4	0.75	151	39.6	13.5	155.4	37.9	5
r8	0.82	119	18.9	16.2	122.3	29.5	4
r16	0.79	145	40.9	14.1	148.4	36.2	6
r32	0.95	103	21.1	18.0	105.5	25.7	6
r64	0.72	100	17.1	14.2	102.1	24.9	2

TABLE II
64-POINT FFT DSP CORES, VARYING BUTTERFLY RADIX

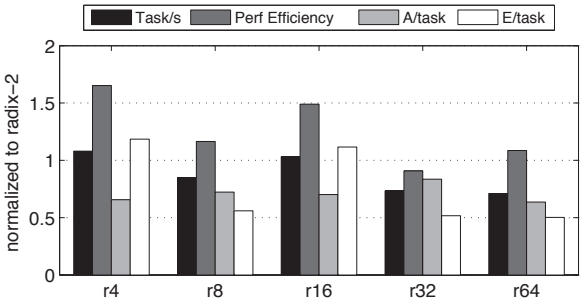

Fig. 7. **DFT_{64} Core Characteristics.** Performance ($\frac{task}{s}$), Performance-efficiency, $\frac{Area}{Task}$ (area per parallelism in the design), and $\frac{Energy}{task}$, varying butterfly-radix from radix-4 to radix-64 normalized to radix-2.

higher order DFT modules also includes permutation blocks to connect the output of $stage_n$ with the input of $stage_{n+1}$.

To limit the design space and afford maximum configurability, we use the Spiral DFT framework [9] to generate 64-point DFT designs, sweeping the butterfly kernel radix in powers of 2 from radix-2 (*r2*) to radix-64 (*r64*). The kernel function is isolated by completely unrolling the loop so each design accepts 64 input words per cycle. The designs are synthesized in 32nm with target frequency of 100 MHz and are shown in Table II. By varying the configured radix in the compute kernel, we alter the number of compute and permute stages in the design, along with the pipeline depth and the complexity of each stage.

A. Varying the Kernel

Sweeping the kernel complexity of the DFT_{64} has noticeable effects on the performance of the DSP core. Figure 7 shows the area cost ($\frac{area}{task}$), and energy cost ($\frac{energy}{task}$) for each design,

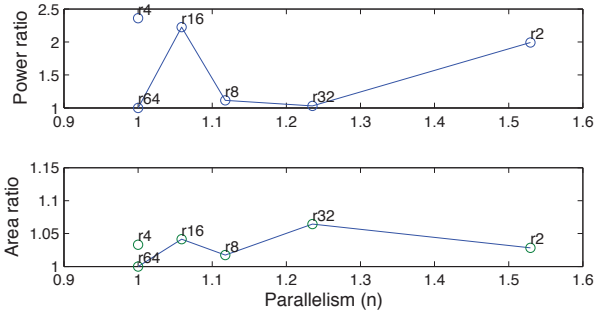

Fig. 8. **DFT Parallelism**. Power and Area cost vs. parallelism (pipeline-depth) for each DFT core, normalized to R64 which has the shallowest pipeline and lowest number of tasks in flight per-cycle.

normalized to the radix-2 design. Even though the problem size (64-point DFT) and streaming width (64-point) are the same among all designs, the number of tasks in the pipeline varies because pipeline- and stage-depth are a function of the kernel size. A larger butterfly kernel (e.g. radix-64) can be built with fewer stages and a shorter pipeline depth than a smaller kernel (radix-2).

In order to perform a 64-point DFT, the radix-2 kernel DSP requires 8 stages with a latency of 26 cycles. With a radix-2 kernel, the core performs a sequence of permute→compute operations. As shown in Figure 7 and Table II, the radix-2 design consumes the most area and the most power, as it has the most stages and logic elements. However, this design has a deeper pipeline and higher frequency, yielding higher performance and throughput. For DFT's with less complex butterfly kernels, synthesis tools generate designs exceeding the 100 MHz frequency, improving throughput further.

At the other extreme, the radix-64 design only has two stages for computation and permutation, with a latency of 17 cycles. However, the design is not able to exceed the target frequency, and suffers from lower throughput due to its shorter pipeline. Running at a lower frequency means the kernel uses the least power out of all the designs. However, it is not energy optimal. Radix-8 uses the least energy per task, and is the most efficient design in terms of energy. Radix-64, and kernels 8-32 have the same order of magnitude in terms of computation complexity, i.e. the number of addition and multiplication operations. The difference comes about from how the computations are organized, the number of permute stages, and the number of state-elements in the design. R64 has fewer stages and dedicated permute stages than other designs.

B. Model

We use Figure 8 and Table II to build our model of the DFT kernel space. The pipeline depth varies from 17 cycles for radix-64 to 26 cycles for radix-2, or from 1 to 1.53× the baseline radix-64 design. We normalize parallelism to 1 for the radix-64 designs. Iterative designs will have $P < 1$, while designs with more than 17 pipeline stages will exhibit $P > 1$. In this case, the designer should avoid kernels that are sub-optimal, such as radix-32 and radix 4. Area has a shallow

slope as a function of parallelism (P), and can be modeled by the following equations:

$$Area_{FFT}(P) = 0.77 \times P - 0.04 \ \ (mm^2) \tag{5}$$

$$Power_{FFT}(P) = 4.4 \times e^{1.9P} \ \ (mm^2) \tag{6}$$

$$Throughput_{FFT}(P) = P \ \ (\frac{ops}{cycle}) \tag{7}$$

An iterative design will have parallelism (n) of less than 1 when compared to the unrolled DFT-64 designs, corresponding to a smaller area and power as expected. Task throughput of the design is linear with parallelism.

C. Exploring the Pareto Frontiers

We consider the affect of the kernel on a use case that couples a DES encryptor front-end with a DFT back-end. We sweep the input bandwidth from 1 Gbps to 240 Gbps and build a system instantiating multiple DES and DFT cores. These cores are frequency scaled and instantiated multiple times in order to meet, but not exceed the required bandwidth.

Figure 9 shows the design space, highlighting the pareto-optimal area and pareto-optimal power designs. Note that only in a few cases do the pareto-optimal area and power designs overlap. These design spaces can vary up to 2.5× in terms of area and power. At the 1 Gbps point, the radix-64 and *u1* encryptor consume the least area and power and is the pareto optimal design. Increasing the bandwidth constraints requires multiple instances of any DES design to meet the throughput needs of the system. Now there is a trade-off between area and power, as the pareto-power design prefers the *u8* encryptor while the pareto-area prefers the *u16* design.

At even higher bandwidth constraints, i.e. $> 100\ Gbps$, the preferred FFT radix changes from radix-64 for the pareto-power case to radix-4 in the pareto-area case. This is due to radix-4's superior area-efficiency. However, this throughput-efficiency comes at a large cost in terms of power.

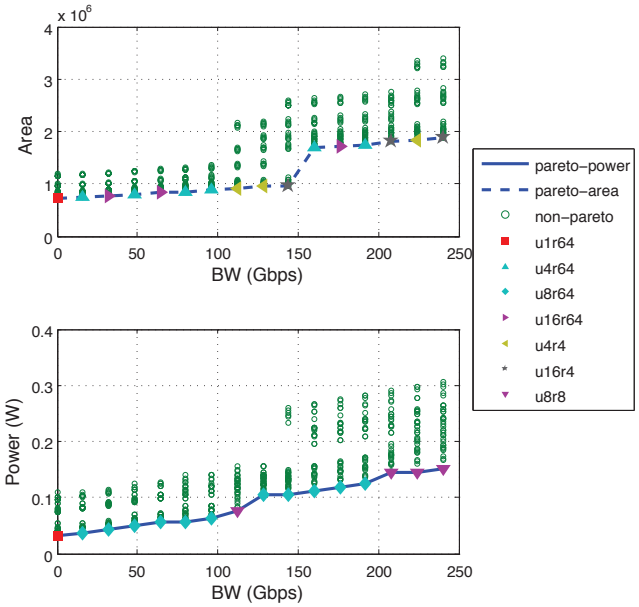

Fig. 9. **FFT-DES Design Space**: Area(top) and Power (bottom) of combined DES-FFT core while sweeping BW constraints. When necessary, each design is frequency-scaled and instantiated multiple times to meet, but not exceed the constraints.

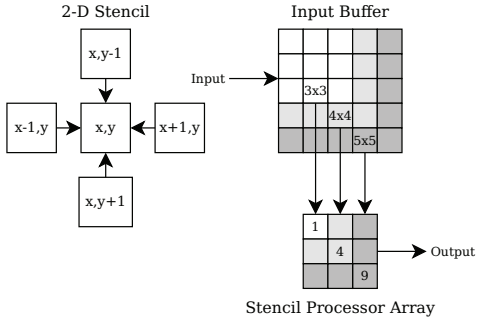

Fig. 10. **Jacobi Stencil** computation performs an average of the cell's Gaussian neighborhood. A single stencil-compute core requires a 3x3 array of elements. n^2 stencil-compute cores require $(n+2)^2$ elements, i.e. 4 cores requires a 4x4 array of elements, 9 cores a 5x5 array, and so on.

VI. Jacobi Stencil Computation

Stencil computations are another representative kernel often implemented in SOC architectures [15], [16]. These kernels are iterative, operating on an array where a computation for an element, *i*, depends on the neighboring elements. For example, the Jacobi kernel used in linear equation system solvers has a 2-dimensional stencil. The 'next' value for the center element, *i*, is computed as the average value of the Gaussian neighborhood (adjacent cells), as shown in Figure 10.

A. Extracting Parallelism

We designed a Jacobi accelerator incorporating the memory system described above, similar to the architecture described in Nilikantan et al.'s SRAD accelerator [16]. The architecture tightly couples stencil computing cores to an array of shift registers. A single stencil core requires a 3×3 shift register array to store and shift the input data from memory. Once all 9 elements are loaded, the stencil-core can begin computing the "next" value for the center element. After each cycle, the data in the register array shifts by one column so computation can begin on the next center element.

As can be seen in Figure 10, there is a dependency between the number of stencil-kernels and the size of the register array required to provide data to them. The stencil compute kernels operate on the center points of the register array. A system with n^2 stencil-kernels requires an $(n+2)^2$ array of registers.

This architecture is highly scalable due to the tight coupling of state and computation elements. We study the overhead of extracting parallelism by varying the number of compute cores (n^2) from $n = 1$ to $n = 4$, with input array sizes from 3×3 to 6×6 elements. We synthesize the designs with a 500 MHz frequency target, and simulate operation at peak throughput. Design characteristics are shown in Table III.

B. Design Analysis

Figures 11 and 12 show the effects of scaling the baseline stencil accelerator core. Unlike unrolling the DES computation

Design n	Area (μm^2)	f (MHz)	P_{dyn} (mW)	P_{leak} (μW)	Perf (G-tasks/s)	Para
1	1425	500	0.45	28.9	0.5	1
2	3679	500	1.30	75.3	2.0	4
3	7010	500	2.58	145	4.5	9
4	10999	500	3.51	235	8.0	16

TABLE III
JACOBI STENCIL ACCELERATOR CORES

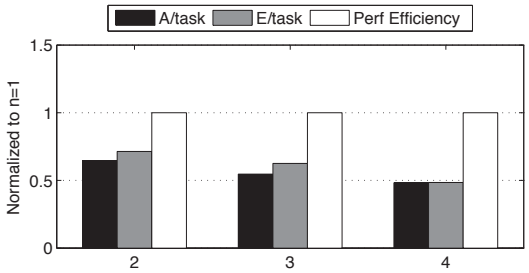

Fig. 11. **Jacobi Core Characteristics.** Area/task, Energy/task and Performance efficiency for $n = 2$ to $n = 4$, normalized to baseline $n = 1$.

loop or varying the DFT compute kernel, this is a highly regular transformation with immediate benefits. Increasing parallelism from 1- to 16-operations per core reduces both energy and area costs. Power density remains consistent, varying from $1\times$ to $1.14\times$ the baseline design. Energy per task decreases by $2\times$ as the larger register array is amortized among $16\times$ more computations. Similarly, the area-cost per task is reduced by $0.5\times$. In terms of real performance, the throughput in tasks/s increases by $16\times$ but the area only increases by $7.7\times$. As parallelism increases, the cost to move from n to $n+1$ decreases. Increasing from $n = 1$ to $n = 2$ costs $2.58\times$ as much area, while from $n = 3$ to $n = 4$ costs only $1.56\times$.

C. Model

Unlike the DES or FFT models, extracting parallelism in the Jacobi stencil computation for small values of n is essentially unbounded; limited only by the input bandwidth available to the core and the power and area envelopes available to the designer. For reasonable values of n we can model performance, power, and area as a function of the number of stencil compute kernels (n^2). As n gets very large, new models will need to be developed to account for the additional wire delay and potential changes to the register-array architecture.

$$Area_{Jacobi}(n) = 1018 + 635 \times n^2 \ (\mu m^2) \quad (8)$$

$$Power_{Jacobi}(n) = 1.12 \times n^2 - 0.73 \ (W) \quad (9)$$

$$Throughput_{Jacobi}(n) = n^2 \ (\frac{ops}{cycle}) \quad (10)$$

VII. CONCLUSION

In this paper we studied three accelerator cores: DES, FFT, and Jacobi Transform, exhibiting three different types of computation: streaming cryptographic, butterfly DSP, and stencil. Where others have focused on modeling resource consumption in the FPGA space [17], we examine several representative kernels focusing on specialization in ASICs.

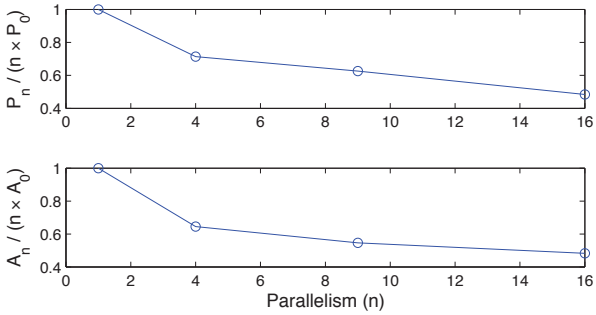

Fig. 12. **Jacobi Parallelism**. Power and Area vs. parallelism, varying the number of stencil cores compared to baseline single-stencil design. Naïvely instantiating n cores will cost $1\times$. Designs with lower cost are optimal.

We examined the underlying algorithms for each kernel, and investigated the physical effects of extracting parallelism from the design. We measured the costs and benefits associated with extracting parallelism from within these kernels, and model the pareto frontier for each kernel type. We observed a wide design space, in both power, area, and performance, even when using optimized designs. Our models of the pareto-frontier provide guidance to the designer, and will be able to help avoid costly design mistakes at an early stage.

REFERENCES

[1] R. H. D. et al., "Design of Ion-Implanted MOSFET's with Very Small Physical Dimensions," *IEEE Journal of Solid-State Circuits*, vol. SC, no. 9, pp. 256–268, October 1974.

[2] G. Venkatesh *et al.*, "Conservation cores: reducing the energy of mature computations," in *Proceedings of ASPLOS 15*, 2010.

[3] H. Esmaeilzadeh *et al.*, "Dark Silicon and the End of Multicore Scaling," in *ISCA 38, 2011*, June 2011.

[4] Texas Instruments, "TI OMAP4460 Platform," http://www.ti.com.

[5] Marvell, "Marvell ARMADA 618 Application Processor," http://www.marvell.com.

[6] J. Brown, S. Woodward, B. Bass, and C. Johnson, "Ibm power edge of network processor: A wire-speed system on a chip," *Micro, IEEE*, vol. 31, no. 2, pp. 76 –85, march-april 2011.

[7] Semiconductor Industry Association, "International Technology Roadmap for Semiconductors (ITRS), 2010, update 2011," http://www.itrs.net.

[8] OpenCores, "OpenCores," http://www.opencores.org.

[9] G. Nordin, P. A. Milder, J. C. Hoe, and M. Püschel, "Automatic Generation of Customized Discrete Fourier Transform IPs," in *Proc. of the 42nd Design Automation Conference*, ser. DAC '05, 2005.

[10] P. A. Milder, F. Franchetti, J. C. Hoe, and M. Püschel, "Formal datapath representation and manipulation for implementing dsp transforms," in *Proc. of the 45th Design Automation Conference*, 2008.

[11] DES, "Data Encryption Standard," in *In FIPS PUB 46, Federal Information Processing Standards Publication*, Washington, DC, 1977, pp. 46–2.

[12] R. Usselmann and S. Yasuhiro, "DES core," http://opencores.org/project, des.

[13] "Advanced Television Systems Committee Digital Television Standard Part 1:2013," *ATSC Std. A/53*, 2013.

[14] P. Kumhom, J. Johnson, and P. Nagvajara, "Design, optimization, and implementation of a universal FFT processor," in *IEEE ASIC/SOC Conference*. IEEE, 2000, pp. 182–186.

[15] A. R. Brodtkorb, C. Dyken, T. R. Hagen, J. M. Hjelmervik, and O. O. Storaasli, "State-of-the-art in Heterogeneous Computing," *Sci. Program.*, pp. 1–33, Jan. 2010.

[16] S. Nilakantan, S. Annangi, N. Gulati, K. Sangaiah, and M. Hempstead, "Evaluation of an accelerator architecture for speckle reducing anisotropic diffusion," in *CASES 14*, 2011.

[17] P. A. Milder, M. Ahmad, J. C. Hoe, and M. Püschel, "Fast and accurate resource estimation of automatically generated custom DFT IP cores," in *FPGA*, 2006, pp. 211–220.