2014 IEEE 22nd Annual International Symposium on Field-Programmable Custom Computing Machines

Kung Fu Data Energy—Minimizing Communication Energy in FPGA Computations

Edin Kadric, Kunal Mahajan, and André DeHon
Dept. of Electrical and Systems Engineering
University of Pennsylvania, Philadelphia, PA, USA
Email: ekadric@seas.upenn.edu

***Abstract*—The energy in FPGA computations can be dominated by data communication energy, either in the form of memory references or data movement on interconnect (*e.g.*, over 75% of energy for single processor Gaussian Mixture Modeling, Window Filtering, and FFT). In this paper, we explore how to use data placement and parallelism to reduce communication energy. We further introduce a new architecture for embedded memories, the Continuous Hierarchy Memory (CHM), and show that it increases the opportunities to reduce energy by strategic data placement. For three common FPGA tasks in signal and image processing (Gaussian Mixture Modeling, Window Filters, and FFTs), we show that data movement energy can vary over a factor of 9. The best solutions exploit parallelism and hierarchy and are 1.8–6.0× more energy-efficient than designs that place all data in a large memory bank. With the CHM, we can get an additional 10% improvement for full voltage logic and 30–80% when operating the computation at reduced voltage.**

I. Introduction

Energy is emerging as a dominant constraint for modern computations. Mobile devices are limited by battery life; server rooms are limited by power density.

Within FPGAs, for many applications, data movement energy—the energy for moving data from one physical location on the FPGA to another—can dominate computational energy. Data movement includes both energy for accessing memory and energy for moving bits over interconnect segments between processing elements. *Just as Kung Fu aims for maximum effect with minimum movement, energy-conscious FPGA designs should aim to minimize data movement.*

We look specifically at two opportunities: parallelism and hierarchy. You might expect more parallel designs to consume greater energy. However, we note that, to first order, energy efficiency, or energy per work done, should remain unchanged. Then, we observe that by making a design more parallel, we can often reduce the size of memories used and hence memory energy, so that increasing parallelism can reduce total energy. Nevertheless, increasing parallelism also makes the design less compact, potentially increasing the length of the wires and hence the interconnect energy. We characterize this trade-off and find the level of parallelism that minimizes total energy efficiency. We further reduce data movement energy by using hierarchy to place data that will be used in the near future close to the consumer, allowing items that will not be needed as soon to be placed farther away.

In addition to examining how to minimize energy on current FPGAs, we explore how we might change FPGA architectures to enhance our opportunity to reduce memory energy. We introduce the Continuous Hierarchy Memory (CHM) that (1) largely eliminates the cost of using an embedded memory that is larger than necessary for a computation and (2) facilitates aggressive use of hierarchy to reduce the average cost of a memory access when the data access pattern is localized.

Our contributions include:

- quantifying the energy contribution of compute, memory, and interconnect
- characterizing how parallelism impacts total dynamic energy consumption and each of the above components
- providing guidance for energy reduction on current FPGAs with concrete application to Gaussian Mixture Modeling (Sec. IV), Window Filters (Sec. V), and Fast Fourier Transforms (Sec. VI), chosen for their different levels of locality and communication requirements
- introducing the Continuous Hierarchy Memory and characterizing its benefits (Sec. VII)

II. Data Movement Energy

A. Memory Energy

The energy required to access a memory depends on its capacity and the number of output bits. Roughly, the energy (and delay) minimizing organization for an N-bit memory is a $\sqrt{N} \times \sqrt{N}$ array. This means that all the main wires in the memory are of length $\sqrt{N}$, so that their capacitance scales as $\sqrt{N}$ and, consequently, their switching energy scales as $\sqrt{N}$. Everything else being equal, a memory of four times the capacity will cost twice the energy. Furthermore, bringing twice as many bits out of an array roughly doubles the energy. *From an energy standpoint, we should use as small a memory block as possible, with as small an output width as possible*

B. Interconnect Energy

a) For Large Memories: Memory energy is driven by wire lengths. This means we cannot "cheat" the $\sqrt{N}$ energy growth effect by decomposing the memory into several smaller memories and wiring to them. We would still end up with address wires of length $\sqrt{N}$ and input/output wires of length $\sqrt{N}$. If we were dominated by memory cell capacitance, breaking the large memory into many smaller *memory banks* and activating only one memory bank at a time could reduce the memory cell energy, but we are still left with wiring energy that also has a $\sqrt{N}$ dependence.

The wiring effect between banks is even larger if they are not densely packed (*e.g.* M9K columns are 5–16 columns apart on a Stratix IV). Building a single, large capacity memory from many, distributed, small capacity embedded memory banks, as exist in most FPGAs [1], [2], consumes more energy than a custom, large capacity memory as we might get in an ASIC, contributing to the energy gap between FPGAs and custom ASICs [3]. *From an energy standpoint, we would like to use a single memory block of the right size for a problem rather than building it from small memory blocks that are sparsely distributed in a computing array.*

b) Between Computations: For many computations, we can reduce the size of the memory by performing the computation in parallel. Rather than having a single processing element (PE) or computational datapath with a large memory to hold all data values, we can have multiple PEs, each with its own, smaller memory. In the extreme, we may be able to eliminate the memories altogether and simply connect together datapath elements. For example, we could build a completely spatial FFT Butterfly network that had no internal memories. As we make the design more parallel, we reduce the size of memories, reducing memory energy. However, we also increase the physical size of the computation, potentially increasing the length of the wires in the system and hence increasing energy. In tasks like the FFT, data must now be moved from the PE where it is produced to the PE where it is consumed, and this data movement costs energy.

c) Question: These two effects raise our main question: *How do we minimize total communication energy?* Is total energy minimized at either the parallel or sequential extreme? Or, more generally, what is the optimal parallelism to minimize energy in a computation? This question is closely related to the ones explored in [4], [5], and we similarly show it is often better to distribute the data and computation than to centralize it in a single memory. Here, we look at concrete applications and explore the use of embedded memories to perform sharing of operators on FPGAs.

III. Background

A. Stratix IV Architecture

We use the Stratix IV architecture [2] as a concrete starting point for exploring data movement energy because it has embedded memory blocks of two different sizes. It includes both M9K blocks that are organized as 256×36 memory banks and M144K blocks that are organized as 2048×72 memory banks. Both memories are dual-ported, capable of performing one memory read and one memory write per cycle. Since the M144K memories have 16 times the capacity ($\frac{144}{9}$=16) and twice the datapath width (72/36), we expect each M144K memory operation to be about $\sqrt{16}\times2=8$ times the energy of each M9K memory operation. A more careful energy model of delay-optimized memory blocks in CACTI 6.5 [6] suggests the ratio should be 4.7. Using Altera's PowerPlay [7], we estimated the energy of each of these memories and determined the M9K spends 7.2 pJ per operation while the M144K spends 32.6 pJ per operation, a ratio of 4.5. Per bit read, the energy premium for reading from the 72b-wide M144K instead of a pair of 36b-wide M9K memories is a factor of 2.3. Further experiments confirm that reading fewer bits than the native output width (36b for M9K, 72b for M144K) does not reduce the energy consumed per memory read. This suggests two rules of thumb for energy minimization. When possible:

1) keep data in small memories
2) read at the maximum native memory width

The newer Stratix V architecture has only a single embedded memory block size, M20K memories organized as 512×40 memory banks [8]. The analysis in [8] justified the move to wider basic memories on the basis of soft-error reliability and to a single block size based on area. There was no discussion of the impact on energy efficiency.

B. Quartus Power Optimizations

In our work, we set Quartus to perform power-driven synthesis [9], which includes power-aware memory balancing: When several discrete memories are combined to form a single, larger memory, higher speed can be traded for lower power by either accessing all memories or only activating some of them based on the current address. For example, consider a $1K\times36$ memory implemented as 4, 256×36 M9K blocks. With speed optimization, each M9K would be configured as a $1K\times9$ block, and each would be read on every cycle to form the 36b output. With power optimization, each M9K could be configured as 256×36, and only one of them would be activated, thus reducing power, but introducing extra logic to select the proper output and memory.

C. Energy Modeling

Prior work on FPGA energy accounting has shown that interconnect consumes the majority ($> 60\%$) of the switching energy [10]. The most comprehensive works to date on FPGA power modeling have focused on logic blocks and interconnect [11], [12] and not addressed memory energy. These prior works focus on RTL optimization where the logic content, and hence level of parallelism, is fixed. In contrast, this paper explores parallelism transforms that would be performed at the behavioral level, changing the logic content and, more importantly, the shape of the memories used in the computation.

D. Related

Chen [13] notes that FFT can be dominated by data movement. As a result, he explores a streaming FFT [14] that avoids the physical interconnect of a large, spatial FFT and shows that it keeps the data movement overhead down to a fraction of the compute cost. [13] addresses exactly the issues we are concerned with here. It emphasizes the need to read at the maximum native width and shows how to do this for the FFT. Chen used the most recent Virtex 7 architecture that has only a single size of embedded memory so was not able to explore the broader options of memory bank size, nor did he look at how memory energy concerns might drive memory block architecture. He compares only two levels of parallelism on a single kernel while we perform a more general sweep of PE count across 3 kernels with diverse interconnect requirements.

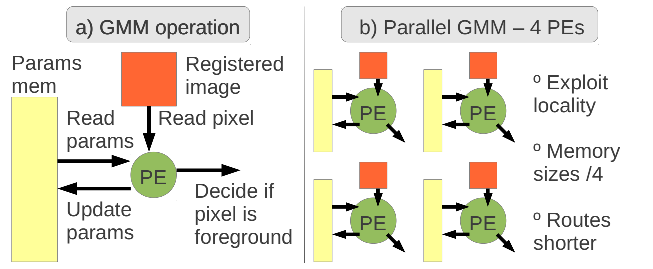

Fig. 1. GMM structure and parallelization

IV. Application 1: High locality, GMM

A. GMM Kernel Description

As a starting example, we consider Gaussian Mixture Models (GMM) [15] to separate foreground objects from background (Fig. 1a). Since it has no interaction between the data pixels, it has a Rent exponent [16] of $p_{rent} = 0$.

The task is to identify moving foreground objects from a background in a registered image. For concreteness, imagine a fixed surveillance or web camera. The goal is to separate out the fixed background from the objects that move in front of it. As a first attempt, we might note that an object moving in front of a pixel will cause the pixel value to change, whereas the background will have a fixed and steady pixel value. So, the algorithm should "learn" the fixed values and use that to differentiate the background from the object. In practice, lighting varies, so it is useful to model the background pixel not as a single value but as a Gaussian distribution. Furthermore, when we have complex backgrounds, like trees or water, a pixel position can actually take on multiple different values but still be part of the background. Consequently, effective GMM algorithms adapt multiple Gaussians to model the distributions for background pixels (we use 3 for our example). Values that do not fit the Gaussian prediction models are then classified as foreground. Each model stores a mean and variance, indicating the likelihood that the given pixel has the given intensity, as well as a weight, indicating the current confidence in the model. Over time, these model parameters converge to the values of the background image being monitored. Then, when an object moves across the background, a deviation in typical pixel values is observed, and the object can be tracked. Each PE reads a pixel value, 9 parameters (3 models $\times$ 3 parameters per model), decides whether the pixel is background or foreground, and updates the parameter values. If each pixel is stored using $w = 16$ bit, for an image of size 64$\times$64, we need to store $w \cdot N^2 = 64$ Kbit for the pixel memory, and $3^2 \cdot w \cdot N^2 = 576$ Kbit for the Gaussian models. We restrict ourselves to this small image so that it fits on-chip on a Stratix IV. Going to external memory off-chip would be an even larger cost per bit.

B. More PEs, smaller memories, smaller routes

When we have only one PE ($P = 1$), we use one 576 Kbit memory and one 64 Kbit memory. In contrast, with P PEs, we need P memories of size $(576/P)$ Kbit, and P memories of size $(64/P)$ Kbit. Using multiple PEs allows us to make the memories smaller, reducing the cost of each memory access, and place them closer to their associated PEs, reducing the routing cost between memory and logic (Fig. 1b).

In the Stratix IV, 576 Kbits can be stored using either 4 M144Ks or 64 M9Ks. The 64 Kbits can be stored using either 1 M144K or 8 M9Ks. We leave this choice to Quartus, as well as the choice of how to compose the memories, using power-driven synthesis (Sec. III-B).

C. GMM Results

We implement the GMM kernel in Bluespec System Verilog [17], translate it to Verilog and compile it in Quartus 13.0.1 for a Stratix IV FPGA (EP4SGX230KF40C2). We use virtual pins instead of routing the outputs of the kernel to the edges of the FPGA. After compiling, for a 50 MHz clock, we simulate the design using random test cases in ModelSim to get signal activities and static probabilities that we use with PowerPlay to evaluate power consumption for the design.

The resources and energy results for different P values are shown in Tab. I. The memory component of total energy is separated from the compute component ("Comp" column), which includes registers, DSPs, clock enable and combinational logic. We also separate routing due to memory (Mroute) and routing due to compute (Croute).

We notice a 1.7$\times$ energy reduction between the 1 PE and 64 PE cases. Even though the total number of memory bits remains unchanged, they are split into smaller memory chunks, each read by a different PE, thus reducing the cost of each memory access. The table also shows that routing cost per PE is reduced, thanks to the increase in locality. In contrast, the work performed by the logic remains unchanged. Memory energy alone is reduced by a factor of 4.6.

The first row reports results when using 1 PE with M144Ks. The second row is still for 1 PE, but uses combinations of M9Ks to form larger memories. As expected, here the memory cost is lower, but there are also increased routing costs to combine the memories, ultimately making this design less efficient. However, for the next cases, combining M9Ks gives an advantage over using M144Ks, so we only show results with M9Ks (this is the choice made by power-optimized synthesis). We notice two discrete jumps in memory energy as the number of PEs is increased. With $P = 1$, the 64 M9Ks for the parameter memory are organized into 4 banks of 16 M9Ks each, so that only 16 of the 64 M9Ks are on at a time. The pixel memory is organized as two banks of 4 M9Ks each. For P = 2, 4, and 8, the banks contain 8 M9Ks for the parameter memory, and 1 M9K for the pixel memory, hence the lower memory energy. Even though memory energy is the same for those cases, the cost of selecting the right bank is higher for lower PE count, since there are more of them (4 parameter banks for $P = 2$, 2 banks for $P = 4$), which contributes to the decrease in Croute. P = 16, 32, and 64 each use 1 M9K for the pixel memory and 4 M9Ks for the parameter memory, hence the next drop in memory energy (all 4 M9Ks need to be on all the time because of the data's 144b width).

TABLE I
GMM Energy Consumption (64×64 image)

PEs	Resources			Mem blocks/pixel		Percentage of total energy				Energy components (pJ/pixel)				Total
P	LUTs	Regs	DSPs	Image	Params	Mem	Mroute	Comp	Croute	Mem	Mroute	Comp	Croute	(pJ/pixel)
1	1343	13	6	1 M144K	4 M144K	35%	12%	23%	31%	300	105	197	264	866
1	1536	21	6	8 M9K	64 M9K	25%	3%	22%	50%	230	29	206	458	923
2	3066	20	12	4 M9K	32 M9K	16%	3%	28%	53%	115	25	206	392	738
4	5968	13	24	2 M9K	16 M9K	18%	4%	31%	48%	115	24	201	311	651
8	10704	10	48	1 M9K	8 M9K	20%	13%	34%	33%	115	72	196	190	572
16	21312	9	96	1 M9K	4 M9K	12%	13%	36%	40%	65	68	195	211	539
32	42592	8	192	1 M9K	4 M9K	12%	13%	37%	38%	65	71	199	209	545
64	83136	7	384	1 M9K	4 M9K	13%	13%	38%	37%	65	69	195	190	520

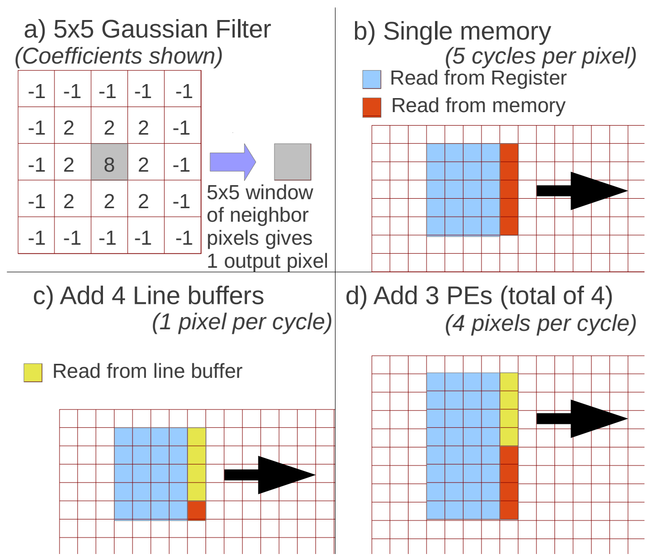

Fig. 2. Window Filter Configurations

V. Application 2: Medium locality, WINF

The second application we explore is Window Filtering (WINF), which has medium locality, $p_{rent} = 0.5$. We apply the same communication-minimizing techniques as in GMM but also optimize for data that is reused after an intermediate amount of time.

A. WINF Kernel Description

Window filters compute each output pixel of an image as the weighted sum of neighboring pixels. As an example, we implement a Gaussian Filter for edge detection with a 5×5 window and power of 2 coefficients (Fig. 2a).

We choose an image of size 128×128, and 16 bits per pixel intensity. This size is, again, selected to keep all memory on chip. The memory requirement for WINF with a single memory and a single PE is 0.25 Mbit, or 2 M144Ks.

The PE mostly consists of shifts, adders, and registers. In this simplest setting, one pixel is read from the main memory on each cycle and stored in one of the registers. It takes 5 cycles to fill one column of registers and produce one pixel (Fig. 2b). Without registers, all reads would be from the main memory, and it would take 25 cycles to produce each pixel. Since most reads go back to a single, large memory, there is significant communication overhead.

B. Line buffers: more locality

Since each pixel needs information from its neighbors, which belong to different rows, there is non-locality that forces us to read from memory multiple times to process each pixel. The problem with the previous configuration is that those reads all go to the main, large memory. Instead, we can exploit memory hierarchy and use line buffers to store the pixels that will be needed in the near future in smaller memories (2 Kbit, or a quarter of an M9K for each line).

As shown in Fig. 2c, using 4 line buffers, we can produce one pixel every cycle, where 20 pixels are available from the PE's registers, 4 are read from the 4 line buffers, and one pixel is read from the main memory (and stored in the appropriate line buffer to be used in the near future). This line buffering scheme is typically used in window filters for delay optimization [18], and our experiments show that these also benefit energy, with a $2.5\times$ improvement in total energy efficiency, as shown in Tab. II, and a $3.1\times$ improvement in memory energy alone

C. Multiple pixels per cycle

Next, we use multiple PEs (Fig. 2d). If we read 2 or 4 pixels from the main memory every cycle instead of 1, we can update 2 or 4 line buffers at the same time, and produce 2 or 4 neighboring pixels at the same time (that are on the same column). This both reduces the number of memory accesses and the registers' overhead, since their content is shared among multiple pixels in the same cycle. This gives an additional $2\times$ improvement in energy efficiency from 1 PE to 4 PEs (Tab. II), and a $3.9\times$ improvement in memory energy alone.

D. Sub-images, smaller memories, smaller routes

In the spirit of the strategy for GMM, one simple improvement is to divide the image into B sub-images, each storing (0.25/B) Mbit and each having their own PEs, thus reducing routing and lowering energy per memory access. The only communication between the sub-images happens at their borders, thus the operations are still localized within a sub-image. With $B = 4$ sub-images, this gives an additional $1.2\times$ improvement over the previous case with 4 PEs (Tab. II), a $1.4\times$ improvement in memory energy alone.

Overall, for WINF, there is a consistent decrease in both memory and routing energy as we increase parallelism and locality. Line buffers and extra PEs also decrease compute energy (whereas using sub-images leaves it unchanged).

TABLE II
WINF ENERGY CONSUMPTION (128×128 IMAGE)

Parameters			Resources		Mem blocks/B		Percentage of total energy				Energy components (pJ/pixel)				Total
P	LBuff?	B	LUTs	Regs	Image	LBuff	Mem	Mroute	Comp	Croute	Mem	Mroute	Comp	Croute	(pJ/pixel)
1	N	1	435	418	2 M144K	0	46%	4%	17%	33%	395	30	146	281	852
1	Y	1	474	417	32 M9K	4 M9K	33%	3%	25%	39%	126	11	81	124	342
2	Y	1	753	498	32 M9K	4 M9K	22%	3%	33%	42%	63	6	67	84	220
4	Y	1	1449	661	32 M9K	4 M9K	15%	2%	37%	46%	32	3	61	75	170
4	Y	2	2847	1308	16 M9K	4 M9K	11%	2%	42%	45%	23	3	59	63	148
4	Y	4	5647	2587	8 M9K	4 M9K	11%	2%	42%	45%	23	3	57	60	143

TABLE III
FFT ENERGY CONSUMPTION ($N = 16K$)

Parameters			Resources			Memory Blocks		Percentage of total energy				Energy components (pJ/point)				Total
R	P	H	LUTs	Regs	DSPs	Data	Twiddle	Mem	Mroute	Comp	Croute	Mem	Mroute	Comp	Croute	(pJ/point)
2	1	1	730	170	10	4 M144K	2 M144K	50%	3%	19%	28%	2324	136	902	1291	4652
2	1	16	821	176	12	4 M9K/M144K	2 M144K	34%	6%	24%	36%	1308	210	925	1366	3810
2	4	4	1909	381	35	88 M9K	63 M9K	25%	2%	29%	43%	878	75	1019	1493	3465
2	16	1	8159	1234	129	64 M9K	62 M9K	16%	3%	35%	45%	421	85	896	1164	2566
2	32	1	16473	2379	257	64 M9K	63 M9K	9%	4%	38%	49%	206	85	913	1173	2377
4	1	1	1813	260	26	4 M144K	3 M144K	22%	2%	31%	45%	643	53	887	1314	2897
4	1	16	1897	266	27	4 M9K/M144K	3 M144K	15%	5%	33%	46%	389	133	855	1190	2567
4	4	4	5689	689	99	80 M9K	60 M9K	9%	2%	34%	55%	241	59	892	1462	2654
4	16	1	21433	2394	385	64 M9K	63 M9K	4%	2%	34%	61%	109	39	906	1615	2669

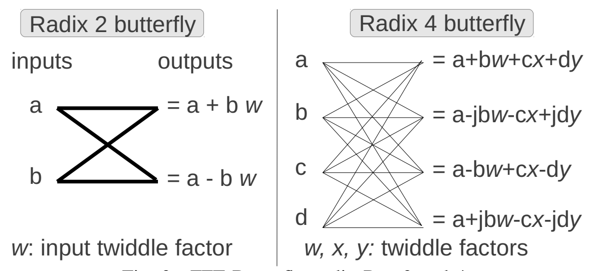

Fig. 3. FFT Butterfly, radix R = 2 and 4

VI. APPLICATION 3: LOW LOCALITY, FFT

The third application we explore is FFT (Fast Fourier Transform), which has little locality, $p_{rent} = 1$.

A. FFT Kernel Description

We compute an N-point FFT, where each output point depends on every input point, thus the non-locality and $p_{rent} = 1$. The CooleyTukey FFT algorithm [19] is performed by recursively subdividing it into R sub-FFTs of size N/R each, where R is the radix of the algorithm. The data is recursively divided into R subsets until the last R points, which are transformed using a radix R butterfly unit (Fig. 3). Furthermore, each sub-FFT can be performed in parallel with the other ones and can thus be assigned its own PE. We use a total of P PEs, each implementing its own butterfly unit. We use 18-bit fixed-point arithmetic so that each input to the FFT is a complex number requiring 18 × 2 = 36 bits.

Fig. 4 shows the FFT network. Fig. 5 shows the hardware datapath for the network. On each cycle, one value is read from each memory. The data is permuted and fed into the PEs, then de-permuted and written back to the memory location it came from. The first column labeled "memories" in Fig. 4 shows

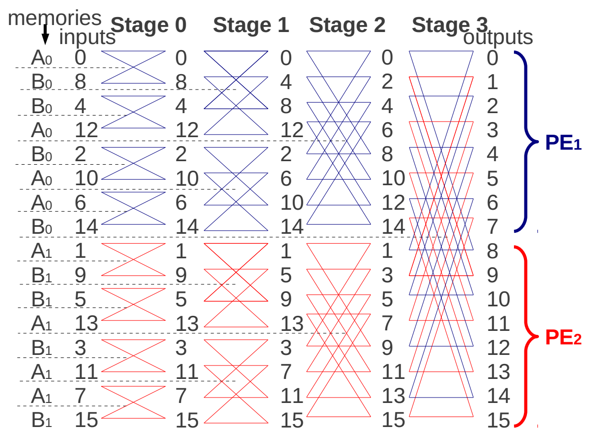

Fig. 4. Basic FFT network for $N = 16$, $R = 2$, $P = 2$

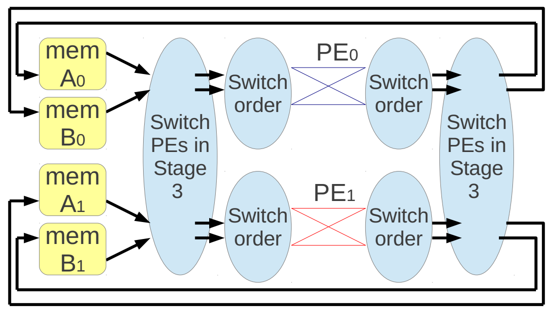

Fig. 5. Basic FFT implementation for $N = 16$, $R = 2$, $P = 2$

how to distribute the data in the memories to ensure that no two datapoints from the same memory are needed at the same time at a particular stage. With this scheme, we need $P \times R$ memories of depth $N/(P \times R)$ with 1 read port and 1 write port. Not shown in the figures, we also need $(P \times R - 1)$ Read-Only-Memories (ROMs) of depth $N/(P \times R)$ with 1 read port for the coefficients (twiddle factors). The latency of this scheme is $[(N\log(N))/(R\log(R))]/P$.

Similarly to GMM and WINF, using multiple PEs decreases memory sizes and makes each memory access less expensive. Furthermore, each PE is closer to its own memory, reducing routing overhead. However, FFT has low locality: data eventually moves across the whole network between PEs, increasing routing significantly with PE count. Note that with multiple PEs, we do exploit locality, since each PE can work independently of the other ones, until the last $(\log(P)/\log(R))$ stages. For example, in Fig. 4, all communication is local to each PE, until stage 3 where data is shared across PEs.

B. Hierarchical FFT

With the scheme described so far, each PE needs to perform a sub-FFT of size N/P before the recombining stages. Each sub-FFT is, itself, split recursively into R sub-FFTs at each stage. Within one PE's task, the most straightforward way to sequence operations is to compute all the data from one stage (column) before moving to the next. For example, in Fig. 4, we would compute all of stage 0, then all of stage 1, etc.

Alternately, we can perform all the stages of one sub-FFT before moving to the next sub-FFT. For example, in Fig. 4, we would compute the first two butterflies of stage 0, then the first two of stage 1, and only then move to the 3rd and 4th butterflies of stage 0, then the 3rd and 4th of stage 1. With this depth-first or recursive approach [20], we could use a smaller memory for stages 0 and 1, and only use the N/P size memory at stage 2. This further reduces the memory sizes and the cost of each memory access: we only use larger memories when they are needed. To characterize this optimization, we introduce a parameter H for the ratio of the larger PE memory that is needed in both cases to the smaller memory used for leaf FFTs. In the previous example, H=2.

C. FFT Results

Tab. III shows experimental results for an $N = 16K$-point FFT. As we scale the number of PEs and exploit hierarchy, we observe a $2.0\times$ improvement in total energy for the radix 2 case, an $11.3\times$ improvement in memory energy alone. This is achieved with 32 PEs.

For low PE count, we find that the FFT performs better with radix 4 than radix 2. This is not only because it uses memories that are exactly the right size, but also because the radix 4 algorithm requires fewer compute operations per FFT input. However, as the number of PEs is increased, the radix 4 version's routing increases faster than that of radix 2 due to non-locality, its total energy increases, and it eventually becomes less efficient than using radix 2. The optimum point for radix 4 is obtained with 1 PE and $H = 16$, with a $1.13\times$ improvement in total energy over the simple 1 PE case, a $1.65\times$ improvement in memory energy. Beyond that, although increasing parallelism reduces memory energy, it also increases routing energy (Croute), resulting in an increase in total energy. Radix 2 continues energy reduction at least down to 32 PEs.

Addrs	0–15	16–63	64–255
Organization	16×36	48×36	192×36
Shape (μm×μm)	50×34	90×65	93×130
E_{mem} (pJ)	2.0	2.9	6.5
E_{awires} (pJ)	0.0	0.041	0.11
E_{dwires} (pJ)	0.0	0.15	0.39

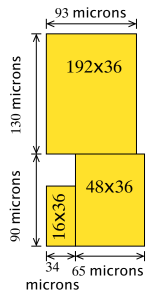

Fig. 6. M9K Three-Level Memory

VII. Memory Architecture

We saw that the Stratix IV is able to construct memories with sizes in between M9K and M144K, or over M144K, by composing the memory blocks it has available. However, this comes with decoder, multiplexer and routing overheads that make the memory energy higher than it would have been had there been blocks of exactly the right size.

With proper decomposition, we can eliminate most of this cost. By internally banking the embedded memories and preferentially using the small banks close to the memory I/O ports, we can reduce the costs of using larger memories and enable hierarchical memory optimizations. Furthermore, by asymmetrically organizing the addresses within the memory, we can place some memory content closer to the output and hence access it with lower energy.

To concretely illustrate this idea in isolation, we consider modifying the M9K and M144K memories in the Altera Stratix IV architecture while leaving the rest of the architecture unchanged. This allows us to use the same designs and design tools (Quartus) for the architectural comparison, but prevents us from exploring more comprehensive architectural alternatives, which we leave for future work.

A. Three-Level Hierarchical Memory for M9K

For the M9K, we consider a 3-level hierarchy that allows the block to be used as a 16×36, 64×36, or 256×36 memory. Furthermore, addressing maps them into a single address space, with the lower addresses mapped to the smaller memory banks (Fig. 6). This revised memory can be used efficiently as the smaller banks and can be used to reduce energy by placing more frequently accessed data in the low addresses.

Using the Stratix III M9K area from [21] and the aspect ratio of LABs from [2], we estimate the Stratix IV M9K block as $4700F \times 5600F$, where F is the technology half pitch. For the 40 nm technology, this is 190μm$\times 220\mu$m. We use 100μm$\times 220\mu$m for the memory banks (Fig. 6), leaving the rest of the area for the interconnect and interfacing logic. [22] suggests half of the LAB area, or 2.6MF^2, is needed for interconnect at each LAB or M9K site. We use CACTI 6.5 [6] to estimate memory block area and energy.

CACTI does not model the exact technology used by Altera for the Stratix IV, so we simply use CACTI to obtain relative energies for various block sizes and scale the energy estimate

TABLE IV
M144K Hierarchical Memory with 8 Banks

Addrs	Org.	Shape (μm$\times\mu$m)	E_{mem} (pJ)	E_{awires} (pJ)	E_{dwires} (pJ)
0–255	256×72	190×130	11.2	0.0	0.0
256–767	2×256×72	2×190×130	11.2	0.27	1.9
768–1279	2×256×72	2×190×130	11.2	0.53	3.8
1280–1791	2×256×72	2×190×130	11.2	0.80	5.8
1792–2047	256×72	190×130	11.2	1.1	7.7

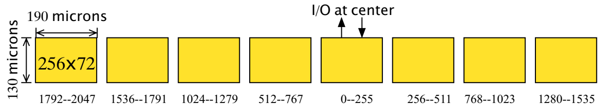

Fig. 7. Possible Layout of 8 Bank M144K (rotated 90°)

obtained from Quartus for the M9K (7.2 pJ) to get comparable energy estimates for the smaller blocks. The 192×36 is 90%, the 48×36 is 40%, and the 16×36 is 28% of the energy of the 256×36 memory block.

We add energy costs for the wiring to the various banks. We assume the smallest bank is immediately adjacent to the interconnect ports and requires no additional wiring. Then, we must route across that bank to reach each of the larger banks as shown in Fig. 6. Each bank gets at most 9b of address and returns 36b. We add a bit for an enable so we have 10b address and 36b return. In this technology, wires are roughly 2 pF/cm [23, Table INTC6]. Based on PowerPlay energy estimates for R4 segment energy, we assume wires are buffered sufficiently to cost 3 pF/cm. Connecting to the 48×36 memory requires wires that run 34μm across the 16×36 memory. The energy for the address wires (E_{awires}) is 0.041 pJ, calculated as:

$$E_{awires} = {}^{1}\!/_{2} \times 0.9^2 \times 10 \times 34\mu\text{m} \times 3\text{pF/cm} \qquad (1)$$

$V_{dd} = 0.9V$, hence the 0.9^2 term. The data wires (E_{dwires}) are 0.15 pJ. We make a similar calculation for connecting to the 192×36 memory, where wires must run 90μm across the height of the 48×36 memory, and summarize all wiring energies in Fig. 6. We calculate the energy used per access based on the activity of the respective wires (α terms):

$$E_{chm} = \alpha_{addr} E_{awires} + E_{mem} + \alpha_{data} E_{dwires} \qquad (2)$$

B. Banked, Hierarchical Memory for M144K

If the M144K memory was organized in a square, we would use a decomposition into exponentially sized memories similar to the M9K. However, since the M144K is spread across 8 rows, we consider each row a 256×72 memory bank. We assume the interconnect interfacing is in the middle, and assign higher address groups to banks further away from the center (Tab. IV, Fig. 7). We use the same methodology for calculating memory bank and wire energy and summarize in Tab. IV.

The worst-case memory is actually less energy than the PowerPlay M144K (11.2+1.1+7.7=20 pJ<32.6 pJ). One reason is that we split the length of the longest wire. Since we feed from the middle and only activate one bank, we do not need to send address and data across all 7 unused banks, which would have cost 26.6 pJ, only the four banks in a particular direction. An unbanked version would have address and bit lines that ran the entire length. Banking will likely be slower since it adds logic to determine the lines to drive.

TABLE V
GMM Energy with CHM

PEs	Mem blocks/pixel		Mem (pJ/pixel)			Total (pJ/pixel)		
P	Image	Params	Base	Bank	CHM	Base	Bank	CHM
32	1 M9K/4	4 M9K/2	65	54	42	545	533	522
64	1 M9K/8	4 M9K/4	65	54	25	520	508	479

TABLE VI
WINF Energy with CHM

		Mem blocks/B		Mem (pJ/pixel)			Total (pJ/pixel)		
P	B	Image	LBuff	Base	Bank	CHM	Base	Bank	CHM
1	1	32 M9K	4 M9K/4	126	105	79	342	321	295
2	1	32 M9K	4 M9K/4	63	52	40	220	209	196
4	1	32 M9K	4 M9K/4	32	26	20	170	164	158
4	2	16 M9K	4 M9K/4	23	19	13	148	144	138
4	4	8 M9K	4 M9K/4	23	19	13	143	139	133

C. CHM results

GMM is dominated by the parameter memory, for which at least 4 M9Ks are active at all times to supply the 144b wide data. However, only half and a quarter of those M9Ks are actually needed at 32 and 64 PEs respectively. With M9K-CHM memory we only pay for that fraction of the cost.

Tab. V shows improved GMM results using CHM. We get benefits just from replacing the M9K block with the banked version discussed here, even if we use all of it (all 9 Kbits). The gains from this effect alone are captured in the "Bank" column: this is the CHM with worst-case access patterns; quantifying the impact of the absolutely lower energy for the M9K. The CHM column captures the full benefits of using a smaller memory bank. Tab. V shows that we can further reduce GMM memory energy by 2.6×, making the GMM design 1.1× more energy efficient. The total gains are modest because memory is no longer the dominant contributor to energy.

In WINF, the line buffers are implemented with M9Ks, but only need a quarter of their capacity (2 Kbits). Once again, with an M9K-CHM memory, we only pay for that fraction of the cost. Tab. VI shows an additional 1.1–1.2× improvement over the optimized cases from Tab. II, including a 1.6–1.8× improvement in memory energy alone.

Performing the FFT computation recursively (hierarchically) further benefits from the CHM. With CHM memories, we do not need a separate small memory. Instead, since the access pattern to the memory is non-uniform—most of the addressing is to the lower addresses—we pay the cost of the smaller CHM configuration more often. This results in the largest CHM benefits (Tab. VII) with a 1.3–2.3× reduction in memory energy alone for a 1.04–1.14× total improvement. Including the effect of banking the memories, this is a 1.1–1.5× improvement in total energy.

D. Discussion of the CHM Results

The results show modest CHM gains, in part because by the time we optimize memories, the dominant energy becomes that of the interconnect and logic. Nonetheless, energy-conscious designs will scale voltage for interconnect and logic more aggressively, as memories are notoriously the first to fail

TABLE VII
FFT ENERGY WITH CHM

Param			Mem (pJ/point)			Total (pJ/point)		
R	P	H	Base	Bank	CHM	Base	Bank	CHM
2	1	1	2324	1090	822	4652	3418	3150
2	1	16	1308	705	487	3810	3207	2989
2	4	4	878	726	316	3465	3313	2902
2	16	1	421	348	151	2566	2493	2296
2	32	1	206	170	74	2377	2341	2245

at small feature sizes and low voltages. For example, [24] shows 84% energy reduction for WINF in compute and interconnect when scaling voltage. With this voltage scaling for computational energy, the total energy consumption for the best WINF case drops from 42 pJ/pixel to 32 pJ/pixel when using CHM, a $1.31\times$ benefit, compared to $1.08\times$ previously (143 pJ/pixel to 133 pJ/pixel, Tab. VI); overall, this is a $26\times$ reduction in energy over the base case. Similarly, CHM gives a $1.84\times$ benefit for the best GMM case ($1.08\times$ previously); $18\times$ over the baseline. Finally, CHM gives a $1.27\times$ benefit for the best FFT case ($1.06\times$ previously); $10\times$ over the baseline.

VIII. Discussion and Open Issues

This paper has focused on energy reduction. The more parallel versions will consume higher dynamic power. If the concern is battery life (mAh) or a utility bill (kWh), minimizing energy is the goal. Reducing the clock frequency commensurate with the parallelism will turn the energy saving into a dynamic power saving.

While the focal exploration here is on FPGAs, the results hint at larger lessons. The most sequential cases are memory-energy dominated, meaning the FPGA design cannot be substantially less efficient than ASIC or sequential processor cases. Furthermore, the general trend of increased parallelism using smaller, distributed memories reducing memory energy and hence total energy should hold as well.

It will be worthwhile to perform a more comprehensive architectural study of the size and distribution of CHM blocks to identify energy-minimizing FPGA architectures.

IX. Conclusions

The energy of FPGA computations can be dominated by data movement, which is minimized by using the *right size* memories. Parallelism often allows us to split larger memories into smaller blocks that use less energy than larger ones. For GMM, WINF, and FFT, we show that more parallel designs where PEs only use a few Stratix IV M9Ks for memory can use significantly less energy than single PE designs with large memories. For highly interconnected designs, like the FFT, data movement energy on interconnect wiring can dominate memory energy, suggesting a limited benefit to parallelism. We also introduced a Continuous Hierarchy Memory that can reduce the impact of size mismatches in FPGA embedded memories and offers further energy reduction when the frequency of data access varies in the application.

X. Acknowledgments

This research was funded in part by DARPA/CMO contract HR0011-13-C-0005. Any opinions, findings, and conclusions or recommendations expressed in this material are those of the authors and do not reflect the official policy or position of the Department of Defense or the U.S. Government.

References

[1] Xilinx Virtex 2.5V Field Programmable Gate Arrays, Xilinx, Inc., 2100 Logic Drive, San Jose, CA 95124, April 2001.
[2] D. Lewis, E. Ahmed, D. Cashman, T. Vanderhoek, C. Lane, A. Lee, and P. Pan, "Architectural enhancements in Stratix-III and Stratix-IV," in FPGA, 2009, pp. 33–42. [Online]. Available: http://doi.acm.org/10.1145/1508128.1508135
[3] I. Kuon and J. Rose, "Measuring the gap between FPGAs and ASICs," IEEE Trans. Computer-Aided Design, vol. 26, no. 2, pp. 203–215, February 2007.
[4] A. DeHon, "Location, location, location: The role of spatial locality in asymptotic energy minimization," in FPGA, 2013, pp. 137–146. [Online]. Available: http://doi.acm.org/10.1145/2435264.2435291
[5] ——, "Wordwidth, instructions, looping, and virtualization: The role of sharing in absolute energy minimization," in FPGA, 2014, pp. 189–198. [Online]. Available: http://doi.acm.org/10.1145/2554688.2554781
[6] N. Muralimanohar, R. Balasubramonian, and N. P. Jouppi, "CACTI 6.0: A tool to model large caches," HP Labs, Palo Alto, CA, HPL 2009-85, April 2009, latest code release for CACTI 6 is 6.5. [Online]. Available: http://www.hpl.hp.com/techreports/2009/HPL-2009-85.html
[7] PowerPlay Early Power Estimator, Altera Corporation, 2013. [Online]. Available: http://www.altera.com/support/devices/estimator/pow-powerplay.jsp
[8] D. Lewis, D. Cashman, M. Chan, J. Chromczak, G. Lai, A. Lee, T. Vanderhoek, and H. Yu, "Architectural enhancements in Stratix V," in FPGA, 2013, pp. 147–156. [Online]. Available: http://doi.acm.org/10.1145/2435264.2435292
[9] PowerPlay Optimization, Altera Corporation, 2013. [Online]. Available: http://www.altera.com/literature/hb/qts/qts_qii52016.pdf
[10] T. Tuan, A. Rahman, S. Das, S. Trimberger, and S. Kao, "A 90-nm Low-Power FPGA for Battery-Powered applications," IEEE Trans. Computer-Aided Design, vol. 26, no. 2, pp. 296–300, 2007.
[11] K. Poon, S. Wilton, and A. Yan, "A detailed power model for field-programmable gate arrays," ACM Tr. Des. Auto. of Elec. Sys., vol. 10, pp. 279–302, 2005.
[12] F. Li, Y. Lin, L. He, D. Chen, and J. Cong, "Power modeling and characteristics of field programmable gate arrays," IEEE Trans. Computer-Aided Design, vol. 24, no. 11, pp. 1712–1724, Nov. 2005.
[13] R. Chen, N. Park, and V. K. Prasanna, "High throughput energy efficient parallel FFT architecture on FPGAs," in HPEC, 2013, pp. 1–6.
[14] P. A. Jackson, C. P. Chan, J. E. Scalera, C. M. Rader, and M. M. Vai, "A systolic FFT architecture for real time FPGA systems," in HPEC, 2004.
[15] M. Genovese and E. Napoli, "ASIC and FPGA implementation of the gaussian mixture model algorithm for real-time segmentation of high definition video," IEEE Trans. VLSI Syst., vol. 22, no. 3, pp. 537–547, March 2014.
[16] B. S. Landman and R. L. Russo, "On pin versus block relationship for partitions of logic circuits," IEEE Trans. Comput., vol. 20, pp. 1469–1479, 1971.
[17] Bluespec, Inc., "Bluespec SystemVerilog 2012.01.A." [Online]. Available: http://www.bluespec.com
[18] H. Yu and M. Leeser, "Automatic sliding window operation optimization for FPGA-based," in FCCM, April 2006, pp. 76–88.
[19] J. W. Cooley and J. W. Tukey, "An algorithm for the machine calculation of complex Fourier series," Math. Comput., vol. 19, pp. 297–301, 1965.
[20] M. Frigo and S. G. Johnson, "The design and implementation of FFTW3," Proc. IEEE, vol. 93, no. 2, pp. 216–231, 2005.
[21] H. Wong, V. Betz, and J. Rose, "Comparing FPGA vs. custom CMOS and the impact on processor microarchitecture," in FPGA, 2011, pp. 5–14.
[22] M. S. Abdelfattah and V. Betz, "Design tradeoffs for hard and soft FPGA-based networks-on-a-chip," in ICFPT, 2012, pp. 95–103.
[23] "International technology roadmap for semiconductors," <http://www.itrs.net/Links/2010ITRS/Home2010.htm>, 2010.
[24] E. Kadric, K. Mahajan, and A. DeHon, "Energy reduction through differential reliability and lightweight checking," in FCCM, 2014.