Impact of Memory Architecture on FPGA Energy Consumption

Edin Kadric
ekadric@seas.upenn.edu

David Lakata
dlakata@seas.upenn.edu

André DeHon
andre@acm.org

Department of Electrical and Systems Engineering
University of Pennsylvania
200 S. 33rd St., Philadelphia, PA 19104

ABSTRACT

FPGAs have the advantage that a single component can be configured post-fabrication to implement almost any computation. However, designing a one-size-fits-all memory architecture causes an inherent mismatch between the needs of the application and the memory sizes and placement on the architecture. Nonetheless, we show that an energy-balanced design for FPGA memory architecture (memory block size(s), memory banking, and spacing between memory banks) can guarantee that the energy is always within a factor of 2 of the optimally-matched architecture. On a combination of the VTR 7 benchmarks and a set of tunable benchmarks, we show that an architecture with internally-banked 8Kb and 256Kb memory blocks has a 31% worst-case energy overhead (8% geomean). In contrast, monolithic 16Kb memories (comparable to 18Kb and 20Kb memories used in commercial FPGAs) have a 147% worst-case energy overhead (24% geomean). Furthermore, on benchmarks where we can tune the parallelism in the implementation to improve energy (FFT, Matrix-Multiply, GMM, Sort, Window Filter), we show that we can reduce the energy overhead by another 13% (25% for the geomean).

Categories and Subject Descriptors

B.7.1 [**Integrated Circuits**]: Types and Design Styles

Keywords

FPGA, Energy, Power, Memory, Architecture, Banking

1. INTRODUCTION

Energy consumption is a key design limiter in many of today's systems. Mobile devices must make the most of limited energy storage in batteries. Limits on voltage scaling mean that even wired systems are often limited by power density. Reducing energy per operation can increase the performance delivered within a limited power envelope.

FPGA'15, February 22–24, 2015, Monterey, California, USA.

ACM 978-1-4503-3315-3/15/02 ...$15.00.
http://dx.doi.org/10.1145/2684746.2689062.

Memory energy can be a significant energy component in computing systems, including FPGAs. This is particularly true when we assess the total cost of memories, including interconnect energy on wire segments that carry data to distant and distributed memory blocks or to off-chip memory.

This leads to an important architectural question: *How do we organize memories in FPGAs to minimize the energy required for a computation?* We have several choices: What are the sizes of memory blocks? Where (how frequently) are memory blocks placed in the FPGA? How are they activated? How are they decomposed into sub-block banks? Then, there are choices available to the RTL mapping flow: When mapping a logical memory to multiple blocks, should they each get a subset of the data width and be activated simultaneously? or should they each get a subset of the address range and be activated exclusively? In this paper, we first develop simple analytic relations to reason about these choices (Section 2). After reviewing some background in Section 3, we describe our methodology in Section 4. In Section 5, we perform an empirical, benchmark-based exploration to identify the most energy-efficient organization for memories in FPGAs and quantify the trade-offs between area- and energy-optimized mappings.

Section 6 takes the study from Section 5 one step further. For high-level tasks, we are not stuck with a single memory organization—the choice of parallelism in the design impacts the memory organization needed and, consequently, the total energy for the computation. A highly serial design might build a single processing element (PE) and store data in a single large memory; whereas a more parallel version would use multiple PEs and multiple, smaller memory blocks. The parallel version would then have lower memory energy, but may spend more energy on routing. Consequently, there is additional leverage to improve energy by selecting the appropriate level of parallelism. In Section 6, we explore how this selection allows energy savings and how it further drives the selection of energy-efficient FPGA memory architectures.

Contributions:

- Analytic characterization of the energy overhead that results from mismatches between the logical memory organization needed by a task and the physical memory organization provided by an FPGA (Section 2)
- First empirical exploration of memory architecture space for energy minimization (Section 5)
- VTR-compatible memory mapping for energy minimization (Section 4.2)
- Joint exploration of memory architecture space and high-level parallelism tuning (Section 6)

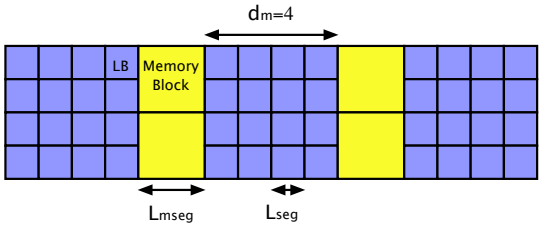


Figure 1: Column-Oriented Embedded Memories

2. ARCHITECTURE MISMATCH ENERGY

FPGA embedded memories generally improve area- and energy-efficiency [10]. When it perfectly matches the size and organization needed by the application, an FPGA embedded memory can be as energy-efficient as the same memory in a custom ASIC. Nonetheless, the FPGA has a fixed-size memory that is often mismatched with the task, and this mismatch can be a source of energy overhead.

First, let us consider just the memory itself. Memory energy arises almost directly from the energy to charge wire capacitance, which grows as the side length of the memory; that is, in an energy-minimizing layout, a memory block will roughly be square and the length of address lines, bit lines, and output wires grow as the square root of the memory capacity. A memory that is four times as large will require twice the energy. Therefore, when the FPGA memory block (M_{arch}) is larger than the application memory (M_{app}), there is an energy overhead that arises directly from reading from a memory bank that is too large ($E(M_{arch})/E(M_{app})$).

There is also a mismatch overhead when the memory block is smaller than the application memory. To understand this, we must also consider the routing segments needed to link up the smaller memory banks into a larger memory bank. To build a larger bank, we take a number of memory banks ($\lceil M_{app}/M_{arch} \rceil$) and wire them together, with some additional logic, to behave as the desired application memory block. In modern FPGAs it is common to arrange the memory blocks into periodic columns within the FPGA logic fabric (See Fig. 1). Assuming square memory and logic blocks, the set of smaller memory blocks used to realize the large memory block would roughly be organized into a square of side length $\left\lceil \sqrt{M_{app}/M_{arch}} \right\rceil$, demanding that each address bit and data line connected to the memory cross roughly $d_m \times \left\lceil \sqrt{M_{app}/M_{arch}} \right\rceil$ horizontal interconnect segments to address the memory, where d_m is the distance between memory columns in the FPGA architecture. If E_{seg} is the energy to cross a length-1 segment over a logic island in the FPGA and $E_{mseg}(M_{arch})$ is the energy to cross a length-1 segment over a memory block of capacity M_{arch}, the horizontal and vertical routing energy to reach across the memory is:

$$E_h = (d_m E_{seg} + E_{mseg}(M_{arch})) \times \left\lceil \sqrt{M_{app}/M_{arch}} \right\rceil \quad (1)$$

$$E_v = E_{mseg}(M_{arch}) \left\lceil \sqrt{M_{app}/M_{arch}} \right\rceil \quad (2)$$

Since the routing energy of wires comprises most of the energy in a memory read, and since each bit must travel the height of the memory block (bit lines) and the width (output select), per bit, the energy of a memory read is roughly the energy of the wires crossing it:

$$E_{bit}(M) \approx 2E_{mseg}(M) \quad (3)$$

Therefore:

$$E_{mseg}(M_{arch}) \left\lceil \sqrt{\frac{M_{app}}{M_{arch}}} \right\rceil \approx E_{mseg}(M_{app}) \approx 0.5E_{bit}(M_{app})$$

This gives us the following mismatch ratio, driven by the ratio of the energy for routing between memory banks to the energy for routing over memory banks:

$$\frac{E_h + E_v}{E_{bit}(M_{app})} \approx 1 + \frac{d_m E_{seg}}{2E_{mseg}(M_{arch})} \quad (4)$$

To illustrate the mismatch effects, Fig. 3a shows the result of an experiment where we quantify how the energy compares between various matched and mismatched designs. Each of the curves represents a single-processing-element matrix-multiply design that uses a single memory size; the size of the memory varies with the size of the matrices being multiplied. Each curve shows the energy mismatch ratio (Y-axis) between the energy required on a particular memory block size (X-axis) and the energy required at the energy-minimizing block size (typically the matched size); hence all curves go to 1.0 at one memory block size and increase away from that point. In contrast to the previous paragraph where we used deliberately simplified approximations to provide intuition, Fig. 3a is based on energy from placed-and-routed designs using tools and models detailed in the following sections; Fig. 3 also makes no *a priori* assumption about large memory mapping, allowing VTR [15] to place memories to minimize wiring. The figure shows how the energy mismatch ratio grows when the memory block size is larger or smaller than the matched memory block size. In practice, designs typically demand a mix of memory sizes, making it even harder to pick a single size that is good for all the memory needs of an application. Nonetheless, this single-memory size experiment is useful in understanding how each of the mismatched memories will contribute to the total memory energy overhead in a heterogeneous memory application.

There is also a potential energy overhead due to a mismatch in memory placement. Assuming we accept a column-oriented memory model, this can be stated as a mismatch between the appropriate spacing of memories for the application ($d_{m_{app}}$) and the spacing provided by the architecture ($d_{m_{arch}}$). If the memories are too frequent, non-memory routes may become longer due to the need to route over unused memories. If the memories are not placed frequently enough, the logic may need to be spread out, effectively forcing routes to be longer as they run over unused logic clusters. This gives rise to a mismatch ratio:

$$\frac{\left\lceil \frac{d_{m_{app}}}{d_{m_{arch}}} \right\rceil (d_{m_{arch}} E_{seg} + E_{mseg}(M_{arch}))}{d_{m_{app}} E_{seg} + E_{mseg}(M_{arch})} \quad (5)$$

Note that if we make $d_{m_{arch}} E_{seg} = E_{mseg}(M_{arch})$, the mismatch ratio due to route mismatch (Eq. 5) is never greater than 2×. Similarly, the mismatch ratio due to memories being too small (Eq. 4) is never greater than 1.5×. We can observe this phenomenon in Fig. 3a by looking at the 32Kb memory size that never has an overhead greater than 1.2×. In Fig. 3, we also identify the $d_{m_{arch}}$ that minimizes max-overhead (shown between square brackets for each memory size in Fig. 3). This approximately corresponds to the intuitive explanation above, where the energy for routing across memories is balanced with the energy for routing across

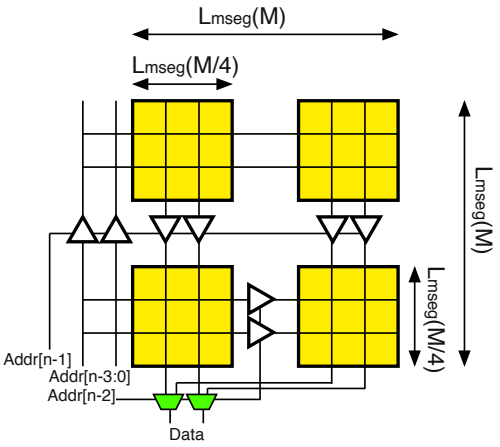


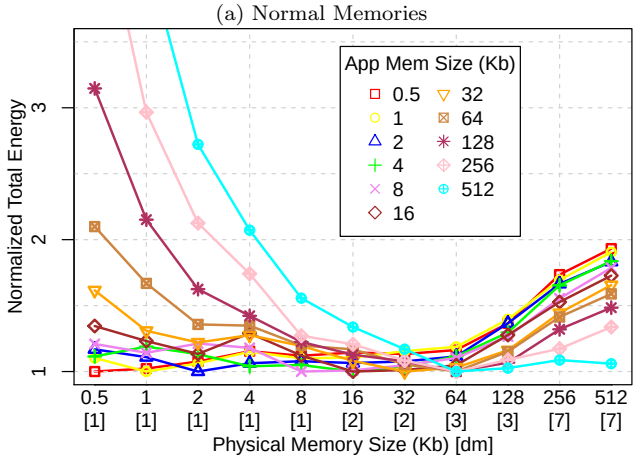
Figure 2: Internal Banking of Memory Block

logic. The 32Kb case has $E_{mseg}(M_{arch})/E_{seg} = 2.53$, suggesting a $d_{m_{arch}}$ of 2 or 3. For this 32Kb case, we found $d_{m_{arch}} = 2$ experimentally. Since segment energy is driven by wire length, $d_{m_{arch}} E_{seg} = E_{mseg}(M_{arch})$ roughly means $d_{m_{arch}} L_{seg} = L_{mseg}(M_{arch})$; when we populate memories this way, half the FPGA area is in memory blocks. This design point gives us an *energy-balanced* FPGA that makes no *a priori* assumptions about the mix of logic and memory in the design. In contrast, today's typical commercial FPGAs could be considered *logic-rich*, making sure the energy (and area) impact of added memories is small on designs that do not use memories heavily.

While the $d_{m_{arch}} E_{seg} = E_{mseg}(M_{arch})$ balance can limit the overhead when the memories are too small, we can still have large overhead when the memory blocks are too large ($E(M_{arch})/E(M_{app})$). One way to combat this problem is to use *internal banking*, or Continuous Hierarchy Memories (CHM) [8]: We can bank the memory blocks internally so that we do not pay for the full cost of a large memory block when we only need a small one. For example, if we cut the memory block into four, quarter-sized memory banks, and only use the memory bank closest to the routing fabric when the application only uses one fourth (or less) of the memory capacity, we only pay the memory energy of the smaller memory bank (See Fig. 2). In the extreme, we might recursively decompose the memory by powers-of-two so that we are never required to use a memory more than twice the size of the memory demanded by the application. There are some overheads for this banking which may suggest stopping short of this extreme. Fig. 3b performs the same experiment as Fig. 3a, except with memory blocks that can be decomposed into one-quarter and one-sixteenth capacity sub-banks. With this optimization, the curves flatten out for larger memory sizes. The physical size with smallest max-overhead is now shifted to 128Kb, still at 1.2×.

Another way to reduce the impact of memory block size mismatch is to include memory blocks of multiple sizes in the architecture. This way, the design can use the smallest memory block that will support the application memory. For example, if we had **both** 1Kb and 64Kb memories, we could map the 2Kb and smaller application memories to the 1Kb memory block and the 4Kb and larger application memories to the 64Kb block and reduce the worst-case overhead to 1.1× (Fig. 3a). However, this raises an even bigger question

(a) Normal Memories



(b) Memories with Internal Banking (one quarter, one sixteenth)

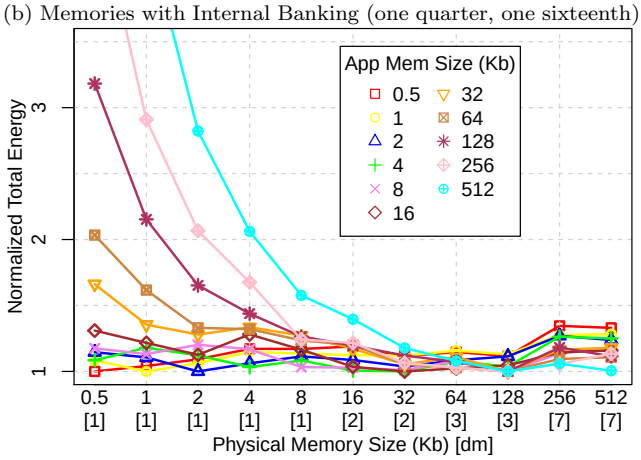


Figure 3: Energy Overhead due to Architectural Mismatch for Matrix-Multiply

about balance among logic and the multiple memory sizes. In particular, routes may now need to pass over the memory blocks of the unused size. We can generalize the previous observation about balancing routing over memories and logic to: $d_{m_1} E_{seg} = E_{mseg}(M_1)$, $d_{m_2} E_{seg} = E_{mseg}(M_2)$. However, since there are now three different resource types, in the worst-case, a route could need 3× the energy of the optimally-matched architecture instead of 2× when there were only two resource types (logic and one memory size).

Another point of mismatch between architecture and application is the width of the data read or written from the memory block. Memory energy also scales with the data-width. In particular, energizing twice as many bit lines costs roughly twice the energy. While FPGA memory blocks can be configured to supply less data than the maximum width, this is typically implemented by multiplexing the wider data down to smaller data **after** reading the full width—the same number of bit lines are energized as the maximum width case, so these smaller data reads are just as expensive as the maximum width read, and hence more expensive than they could have been with an optimally-configured memory. While width mismatch is another, important source of mismatch, it is beyond the scope of this paper. We stick to a single raw data-width of 32 bits throughout our experiments.

Another potential point of mismatch is the simultaneous ports provided by the memories. We assume all memories are dual-ported (2 read/write ports) throughout this paper.

3. BACKGROUND

3.1 FPGA Memory Architecture

We build on the standard Island-Style FPGA model [3]. The basic logic tile is a cluster of K-LUTs with a local crossbar providing connectivity within the cluster (*c.f.* Xilinx CLB, Altera LAB). These clusters are arranged in a regular mesh and connected by segmented routing channels.

To incorporate memories into this mesh, we follow the model used by VTR [15], Xilinx, and Altera, where select columns are designated as memory columns rather than logic columns (Fig. 1). Organizing the memory tiles into a homogeneous column rather than placing them more freely in the mesh allows them the freedom to have a different size than the logic tiles. For example, if the memory block requires more area than the logic cluster, we can make the memory column wider without creating irregularity within rows or columns. Altera uses this column memory model in their Cyclone and Stratix architectures, and the M9K blocks in the Stratix III [12] are roughly 3× the area of the logic clusters (LABs) [20], while being logically organized in the mesh as a single tile. Large memories can span multiple rows, such as the M144K blocks in the Stratix III, which are 8-rows tall while remaining one logical row wide, accommodated by making the column wider as detailed above.

Within this architectural framework, we can vary the proportion of memory tiles to logic tiles by selecting the fraction of columns that are assigned to memory tiles rather than logic tiles. One way to characterize this is to set the number of logic columns between memory columns, d_m. VTR identifies this as a `repeat` parameter (`repeat`=$d_m + 1$). For two memory sizes, d_m still gives the spacing between memory columns, but we first use h_1/h_0 memory columns with small memories of height h_0, followed by one column of large memories of height h_1, so that the area occupied by the small memories is equal to the area occupied by the large ones.

3.2 Energy Modeling and Optimization

Poon [17] developed energy modeling for FPGAs and identified how to size LUTs (4-LUTs), clusters (8–10), and segments (length 1) to minimize energy. However, Poon did not identify an energy-minimizing memory organization. FPGA energy modeling has since been expanded to modern direct-drive architectures and integrated into VTR [6].

Recent work on memory architecture has focused on area optimization rather than energy. Luu examined the area-efficiency of memory packing and concluded that it was valuable to support two different memory block sizes in FPGAs [14]. Lewis showed how to size memories for area optimization in the Stratix V and concluded that a single 20Kb memory was superior to the combination of 9Kb and 144Kb memories in previous Stratix architectures [13], but did not address energy consumption, leaving open the question of whether energy-optimized memory architectures would be different from area-optimized ones.

3.3 Memory Energy Modeling

We use CACTI 6.5 [16] to model the physical parameters (area, energy, delay) of memories as a function of capacity, organization, and technology. In addition to modeling capacity and datapath width, CACTI explores internal implementation parameters to perform trade-offs among area, delay, throughput, and power. We use it to supply the memory block characteristics for VTR architecture files at 22 nm. We set it to optimize for the energy-delay-squared product.

Addresses	0–63	64–255	256–1023
Shape	16×32	48×32	192×32
Size (μm^2)	38×15	52×31	94×58
E_{mem} (pJ)	0.31	0.54	1.1
E_{wires} (pJ)	0.0	0.46	0.84

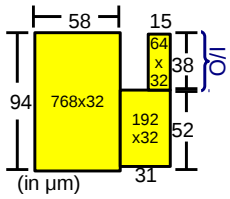


Figure 4: Internal Banking for 32K memory

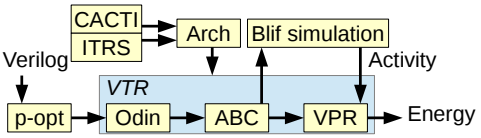


Figure 5: Energy Estimation Tool Flow

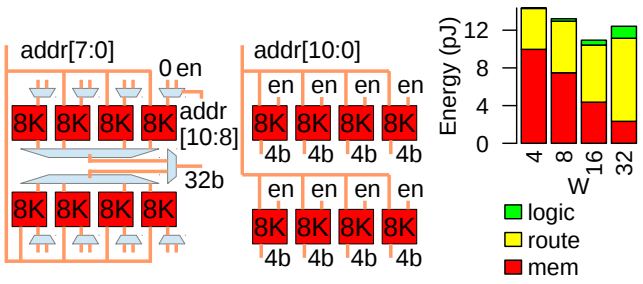
For internal banking (Fig. 2), CACTI gives us the area and energy (E_{mem}) of the memory banks, and we compute wire signaling energy (E_{wires}) to communicate data and addresses between the referenced memory bank and the memory block I/O. For example, consider the data in Fig. 4 for a 1024×32b (32Kb) internally-banked memory. A monolithic 32Kb memory block is 67μm×113μm, which is high enough to contain the 94μm required for the height of the 768 × 32 memory of size 58μm×94μm (plus room for extra logic), as shown in Fig. 4. The total width in Fig. 4 is 58 + 31 = 89μm, or 89/67 = 1.33× that of the monolithic 32Kb memory block. We therefore adjust E_{mseg}(32K-banked)= 1.33 × E_{mseg}(32K). CACTI directly provides E_{mem}. For the largest bank, E_{wires} is (31μm)(1 + 10 + 64)C_{wire} $(V_{dd})^2$. C_{wire} = 180pF/m, V_{dd} = 0.95V. (1 + 10 + 64) corresponds to one signal for the enable, 10 for the address bits, 64 for the 32b input and 32b output. 31μm is the distance to reach the large bank. The medium-size bank has a similar E_{wires} equation but with 38μm instead of 31μm, and the small bank has E_{wires} = 0. Then, the energy of an internally-banked memory is given by $E_{banked} = E_{mem} + \alpha E_{wires}$, where α is the average activity factor over all signaling wires.

4. METHODOLOGY

Fig. 5 shows our tool flow. We developed and added several components on top of a stock VTR 7 release.

4.1 Activity Factor Simulation

Activity factors and static probabilities assigned to the nets of a design have a major impact on the estimated energy. Common ways to estimate activity include assigning a uniform activity to all nets (*e.g.*, 15%), or performing vectorless estimation with tools such as ACE [11], as done by VTR. For better accuracy, our flow obtains activity factors by simulating the designs. We run a logic simulation on the BLIF output of ABC (`pre-vpr.blif` file) on a uniformly random input dataset. For example, for the matched 32Kb-memory matrix-multiply design in Fig. 3a, the average simulated activity factor is 11%, whereas ACE estimates it to be 3%, resulting in an energy estimation that is off by ≈ 3.7×. The tunable benchmarks are designed in a streaming way that activates all memories all the time (independent of the random data), except Matrix Multiply (`MMul`), for which the clock-enable signal is on ≈ 1/3 of the time. The VTR benchmarks do not come with clock-enable for the memories, so we set them to be always on.



a: 2K×32 (W=32) **b:** 2K×32 (W=4) **c:** Sweep W

Figure 6: Effect of Memory Block Activation and Output Width Selection on Energy Consumption

4.2 Power-Optimized Memory Mapping

When mapping logical memories onto physical memories, FPGA tools can often choose to optimize for either delay or energy using power-aware memory balancing [19]. For example, when implementing a 2K×32b logical memory using eight 256×32b physical memories, the tool could choose to read $W = 4$b from each memory (delay-optimized, Fig. 6b). Since each memory internally reads at the full, native width, the cost of the memory operation is multiplied by the number of memory blocks used. Alternatively, it could read $W = 32$b from only one of the memories (Fig. 6a), in which case only one memory is activated at a time (reducing memory energy), but extra logic and routing overhead is added to select the appropriate memory and data. The power-optimized case often lies between these extremes. For example, as our experiment shows in Fig. 6c, the optimum is to activate 2 memories at once and read $W = 16$b from each.

Unfortunately, the VTR flow does not perform this kind of trade-off: it always optimizes for delay. Odin decomposes the memories into individual output bits [18], and the packer packs together these 1-bit slices as much as possible within the memory blocks to achieve the intended width [14]. In fact, VTR memories do not have a clock-enable so they must be activated all the time. Instead, we use VTR architectures with special memory block instantiations that contain a clock-enable, modify VTR's architecture-generation script (`arch_gen.py`) to support these blocks and to support two memory sizes, and add a `p-opt` stage before Odin to perform power-optimized memory mapping based on the memories available in the architecture. This includes performing memory sweeps as illustrated in Fig. 6c to select the appropriate mapping for each application memory. The impact of this optimization is shown for the best architectures in Tab. 2. We find that mapping without p-opt adds 4-19% geomean energy overhead for the optimum architectures, comparable to the 6% benefit reported in [19]. Not using p-opt adds 40–108% worst-case energy overhead, suggesting that this optimization is more important for the designs with high memory overhead. Our p-opt code and associated VTR architecture generation script can be found online [7].

4.3 Logic Architecture

The logic architecture uses k4n10 logic blocks (clusters of 10 4-LUTs) and 36×36 embedded multipliers (which can be decomposed into two 18×18 multiplies, or four 9×9 multiplies) with d_{mpy}=10 and the same shape and energy as in VTR's default 22 nm architectures (a height of 4 logic tiles, plus we use $L_{mpyseg} = 4L_{seg}$). The routing architecture uses direct-drive segments of length 1 with Wilton switch-boxes.

4.4 Technology

We use Low Power (LP) 22 nm technology [1] for logic evaluation and Low Stand-by Power (LSTP) for memories. We use ITRS parameters for constants such as the unit capacitance of a wire at 22 nm ($C_{wire} = 180\,\text{pF/m}$). Then:

$$C_{metal} = C_{wire} \times \text{tile-length} \tag{6}$$

We evaluate interconnect energy based on this C_{metal}, instead of the constant one that is provided in the architecture file. This way, the actual size of the low-level components of the given architecture and technology, as well as the computed channel width, are taken into account when evaluating energy. It is important to model this accurately since routing energy dominates total FPGA energy (See Fig. 6c).

4.5 Energy and Area of Memory Blocks

VTR assigns one type of block to each column on the FPGA (logic cluster, multiplier, or memory), and can give them different heights, but assumes the same horizontal segment length crossing each column. However, some memories can occupy a much larger area than a logic tile, and laying them out vertically to fit in one logic tile width would be inefficient. For energy efficiency, the memories should be closer to a square shape, and to that end, we allow the horizontal segment length crossing memories to be longer (which costs more routing energy, hence $E_{mseg}(M) \neq E_{seg}$ in Section 2). We fix the height of the memory (h) ahead of time, but keep the horizontal memory segment length (L_{mseg}) floating:

$$h = \left\lceil \frac{\sqrt{A_{sw}(W_0) + A_{mem}}}{\sqrt{A_{logic}(W_0)}} \right\rceil \tag{7}$$

Here W_0 is a typical channel width for the architecture and benchmark set. We use $W_0 = 80$. Then, when VPR finds the exact channel width, W_{act}, and hence the tile-length and area (A_{logic}), we can adjust L_{mseg} accordingly:

$$L_{mseg} = \frac{A_{sw}(W_{act}) + A_{mem}}{h\sqrt{A_{logic}(W_{act})}} \tag{8}$$

A_{mem} is the area for the memory obtained from CACTI, and A_{sw} is the switch area required to connect the memory to the FPGA interconnect. We obtain A_{sw} from VPR's low-level models, similar to the way it computes $A_{logic} = A_{luts} + A_{sw}$.

4.6 Benchmarks

To explore the impact of memory architecture, we use the VTR 7 Verilog benchmarks[1] [15] and a set of tunable benchmarks that allow us to change the parallelism level, P, in Section 6. Tab. 1 summarizes the benchmarks that have memories. We expect future FPGA applications to use more memory than the VTR 7 benchmarks. Some of them, such as stereovision, only model the compute part of the application and assume off-chip memory. We expect this memory to move on chip in future FPGAs. The tunable benchmarks provide better coverage of the large memory applications we think will be more typical of future FPGA applications. For this reason, we do not expect a simple average of the benchmarks, such as the geometric mean, to be the most meaningful metric for the design of future FPGAs—it is weighted too heavily by memory-free and memory-poor applications.

We implemented the tunable benchmarks in Bluespec SystemVerilog [4]; they are the following:

[1]Except `LU32PEEng` and `LU64PEEng` (similar to `LU8PEEng`) on which VPR routing did not complete after 10 days.

Table 1: Memory Requirements for the Benchmarks

Benchmark	Mem Bits	# Memories	Largest Mem
VTR			
boundtop	32K	1	1K×32
ch_intrinsics	256	1	32×8
LU8PEEng	45.5K	9	256×32
mcml	5088K	10	64K×36
mkDelayWorker32B	520K	9	1K×256
mkPktMerge	7.2K	3	16×153
mkSMAdapter4B	4.35K	3	64×60
or1200	2K	2	32×32
raygentop	5.25K	1	256×21
spree	1538K	4	32K×32
Tunable			
MMul128	516K	2P	(16K/P)×32
GMM128	7680K	P	(16K/P)×480
Sort8K	1440K	(12-logP)+2P	(8K/P)×45
FFT8K (-twiddle)	1023K	4P	(4K/P)×32
WFilter128 (-line buffer)	256K	P	(16K/P)×16

GMM: Gaussian Mixture Modeling [5] for an $N \times N$ pixel image, with 16b per pixel and $M = 5$ models. P pixels are computed every cycle. This operation is embarrassingly parallel, since each PE is independent of the other ones.

WFilter: 5×5 Gaussian Window Filter for an $N \times N$ pixel image, with 16b per pixel and power-of-2 coefficients. $P = 1/5$ corresponds to a single PE that needs 5 main memory reads per pixel (storing the last 24 values read in registers). $P = 1$ adds line buffers so that only 1 main memory read and 4 line buffer reads translate into 1 pixel per cycle. $P = 2$ and $P = 4$ extend the filter's window, share line buffers, and compute 2 and 4 pixels per cycle, respectively. For $P > 4$, every time P is doubled, the image is divided into two sub-images, similar to the GMM benchmark.

MMul: $N \times N$ matrix-multiply ($A \times B = C$), with 32b integer values and datapaths (See Fig. 11).

FFT: N-point 16b fixed-point complex streaming Radix-2 Fast Fourier Transform, with $P \times \log(N/P)$-stage FFTs followed by $\log(P)$ recombining stages.

Sort: N-point 32b streaming mergesort [9], where each datapoint also has a $\log(N)$-bit index. One value is processed per cycle, and the parallelism comes from implementing the last $\log(P)$ stages spatially.

In Section 5, we use $P = 1$ for each of these benchmarks.

4.7 Limit Study and Mismatch Lower Bound

Section 5 shows the energy consumption for different applications and memory architectures (*e.g.*, Figs. 7, 8, 9). In order to identify bounds on the mismatch ratio, we also set up *limit study* experiments. Our limit study assumes that each benchmark gets exactly the physical memory depth it needs (the width stays at 32), as if the FPGA were an ASIC. Therefore, there is no overhead for using memories that are too small (no need for internal banking as in Section 2) or too large (no need to combine multiple memory blocks as in Section 4.2). We further assume that the limit study memories have the same height as that of a logic tile, making them widely available and keeping the interconnect energy low for vertical memory crossings. Finally, we place memory blocks every 2 columns ($d_m = 1$), so that place-and-route tools can always find a memory right where they need one. To avoid overcharging for unnecessary memory columns, we modify routing energy calculations, and ignore horizontal memory-column crossings for the limit study ($E_{mseg} = 0$). Some large-memory benchmarks drop slightly below the limit study. When the large memories are decomposed, they can see benefits similar to internal banking, where memory references to component memory blocks close to the output require less energy than references to the full, application-sized memory bank.

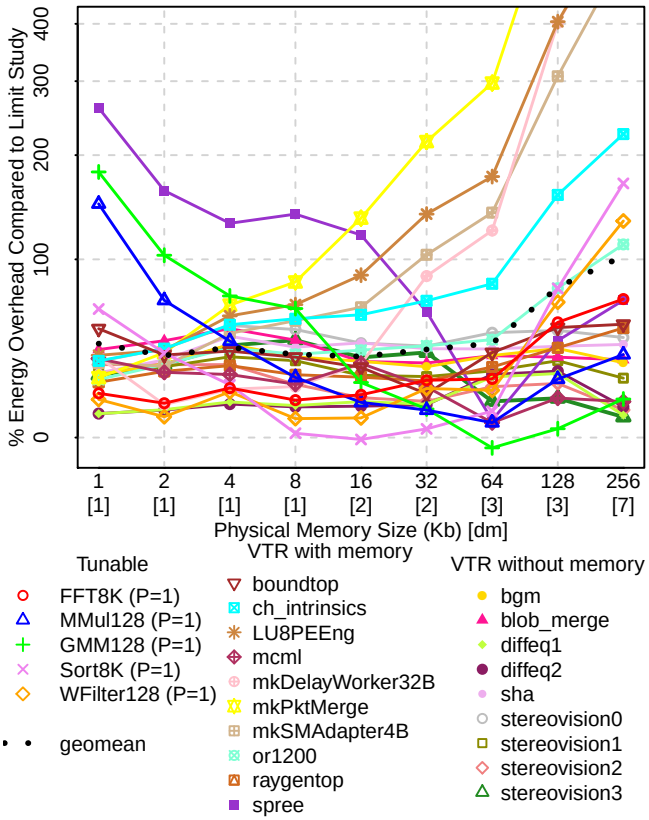


Figure 7: Single Memory Block Size Sweep

5. EXPERIMENTS

5.1 Memory Block Size Sweep

We start with the simplest memory organization that uses a single memory block size and no internal banking (Fig. 7), with the d_m values from Fig. 3a. For comparison, energy is normalized to the lower-bound obtained using the limit study. Most of the curves have an energy-minimizing memory size between the two extreme ends (1Kb and 256Kb). Benchmarks with little memory have an energy-minimizing point at the smallest memory size (1Kb). Benchmarks with no memory have a close-to-flat curve, paying only to route over memories, but not for reads from large memories. The 16Kb memory architecture minimizes the geometric mean energy overhead of all the benchmarks at 37%. As noted (Section 4.6), the geometric mean is weighted heavily by the many benchmarks with little or no memory, so may not be the ideal optimization target for future FPGA applications. `spree` and `mkPktMerge` define the maximum energy overhead curve and suggest that a 4Kb memory minimizes worst-energy overhead at 130% the lower bound.

Fig. 8 shows the detailed breakdown of energy components for three benchmarks as a single memory block size is varied, both for the normal case (top row) and for the internally-banked memories (bottom row) described in Section 3.3.

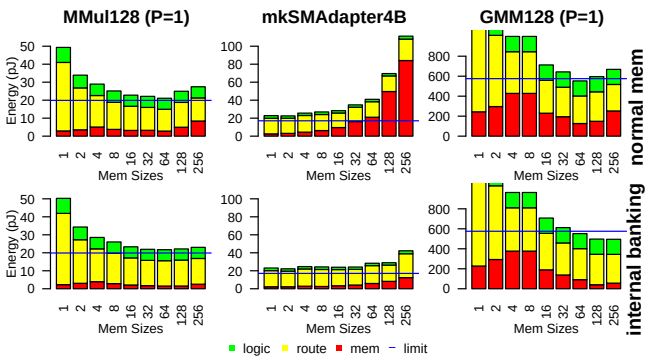


Figure 8: Detailed Breakdown of Energy Components When Sweeping Memory Block Size

We also show a blue line highlighting the optimistic lower bound obtained from the limit study Most benchmarks have the shape of MMul128, with an energy-minimizing memory size between 1Kb and 256Kb. Small benchmarks with small memories have the shape of mkSMAdapter4B, with large increases in memory energy with increasing physical memory size. Fig. 8 shows how internal banking reduces this effect. Sometimes this allows the minimum energy point to shift. For example, in GMM128 the minimum shifts from 64Kb to 256Kb, reducing total energy at the energy-minimizing block size from 553 pJ to 495 pJ, a reduction of 10%.

5.2 Full Parameter Sweep

In Section 2 we showed analytically why the spacing between memory columns, d_m, should be chosen to balance logic and memory in order to minimize worst-case energy consumption. For simplicity, we limited Fig. 7 to only use the d_m values from our mismatch experiment (Fig. 3a). Since the optimal values of d_m may vary among benchmarks, Fig. 9 shows geomean (a) and worst-case (b) energy overheads when varying both memory block size and d_m. In Tab. 2, we identify the energy-minimizing architectures for each of the four architectural approaches (1 or 2 memory sizes, internal-banking or not). We also compare to the lower-bound energy ratios for our closest approximation to the Cyclone (C with $d_m = 9$ and 8Kb blocks) and Stratix (S with $d_m = 9$ and 16Kb blocks) architectures.[2] The heatmap (Fig. 9) gives a broader picture than the specific energy minimizing points, showing how energy increases as we move away from the identified points. We see broad ranges of values that achieve near the lowest geometric mean point, with narrower regions, often single points, that minimize the worst-case overhead. The heatmap shows that overhead has a stronger dependence on memory block size than memory spacing. The commercial designs are appropriately on the broad energy-minimizing valley for geometric mean, but the large spacing, d_m, leaves them away from the worst-case energy-minimizing valley. Multiple memories and internal banking both reduce energy, and their combination achieves the lowest energies. Compared to the commercial architectures, we identify points that reduce the worst-case by 47% ((2.47-1.31)/2.47) while reducing the geomean by 13%. In all the energy minimizing cases, the memories are placed more frequently than the commercial architectures, closer to the balanced point identified analytically in Section 2.

[2] Modeled points have square logic clusters and memories, whereas real Stratix and Cyclone devices are rectangular.

Table 2: Energy Minimizing Architectural Parameters

	Architecture				Energy % Overhead		+% w/o p-opt		Area % Overhead	
ID Fig. 10	mem1 (Kb)	mem2 (Kb)	int bank	dm	geo	worst	geo	worst	geo	worst
[best worst-case]										
4f	16	0	no	6	24	114	13	108	43	143
06b	1	64	no	2	17	46	7.7	64	70	199
5b	32	0	yes	2	25	52	8.0	70	151	384
38d	8	256	yes	4	8.0	31	3.7	40	77	162
[best geomean]										
3g	8	0	no	7	21	204	17	93	37	303
06c	1	64	no	3	14	48	7.0	58	52	180
5g	32	0	yes	7	11	63	8.1	71	53	225
37e	8	128	yes	5	7.0	48	3.7	41	63	490
[commercial-like]										
C(3i)	8	0	no	9	22	216	19	74	46	361
S(4i)	16	0	no	9	24	147	11	52	41	172

5.3 Area-Energy Trade-off

Since there are valleys with many energy points at or close to the minimum energy, parameter selection merits some attention to area. Furthermore, it is useful to understand how much area we trade off for various energy gains. Fig. 10 shows the energy-area trade-off points when varying memory size and organization (1 vs. 2 memories, internal banking vs. not). To simplify the figure, we only show the pareto-optimal points of each set. Energy is normalized to the limit study, while area is normalized to the smallest area achieved.

The architectures with two memory sizes are particularly effective at keeping both worst-case area and worst-case energy overhead low. The architecture that minimizes worst-case area-overhead is a single 16Kb memory design with $d_m = 5$, requiring 80% more energy than the design that minimizes worst-case energy (which requires 33% more area). The designs that minimize geomean form a tight cluster spanning 28% area and 16% energy. Overall, the Stratix and Cyclone architectures fit into this geomean area- and energy-minimizing cluster, but are far from the pareto optimum values in the worst-case energy-area graph. This suggests the commercial architectures are well optimized for the logic-rich design mix captured in the VTR benchmark set. However, as FPGAs see more computing tasks with greater memory use and a larger range of logic-memory balance, our results suggest there are architectural options that provide tighter guarantees of low energy and area overhead.

5.4 Sensitivity

The best memory sizes and the magnitude of benefits achievable are sensitive to the relative cost of memory energy compared to interconnect energy. Since PowerPlay [2] estimates that the Altera memories are more expensive (about 3× the energy—perhaps because the Altera memories are optimized for delay and robustness rather than energy) than the energy-delay-squared-optimized memories CACTI predicts are possible, it is useful to understand how this effect might change the selection of architecture. Therefore, we perform a sensitivity analysis where we multiply the energy numbers reported by CACTI by factors of 2×, 3×, and 4×. The results for a single memory block size are shown in Fig. 9c. Without internal banking, the relative overhead cost of using an oversized memory is increased, shifting the energy-minimizing bank size down to 4Kb or 2Kb. At 2× the CACTI energy, the benefit of internal-banking is roughly the same at 30%, but drops to 19% by 4×.

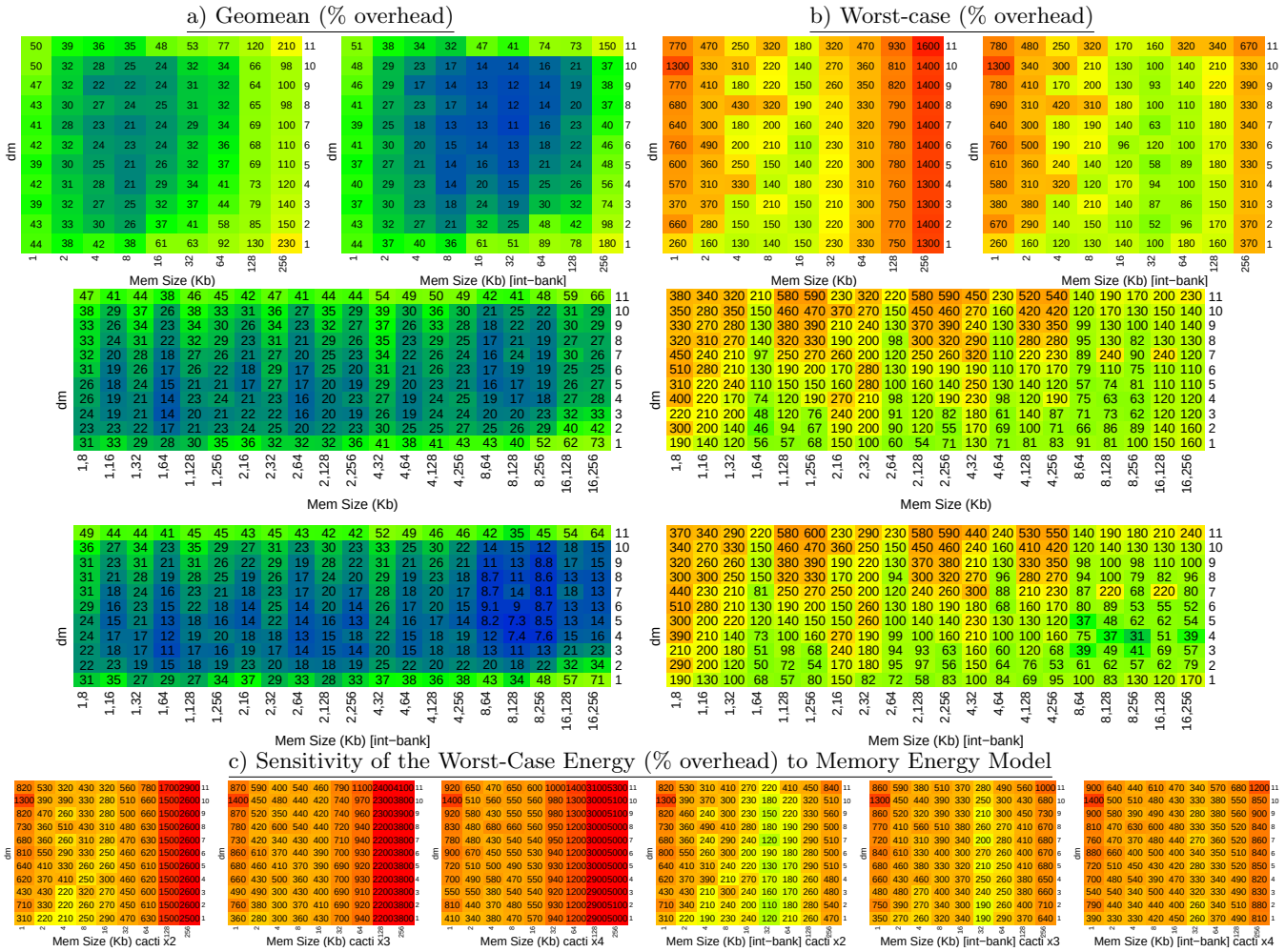


Figure 9: Energy overhead versus memory block size(s) and d_m

6. PARALLELISM TUNING

For many designs, we can choose either to serialize the computation on a single processing element (PE), requiring a large memory for the PE, or to parallelize the computation with many PEs, each with smaller memories. For the Stratix IV with two memory levels (9Kb and 144Kb), we previously showed [8] that parallel designs with many PEs improved the energy-efficiency over sequential designs. In this section, we ask: what is the optimal memory architecture, and minimum energy achievable, when we can vary the parallelism in the design to find the energy-minimizing configuration for each memory architecture?

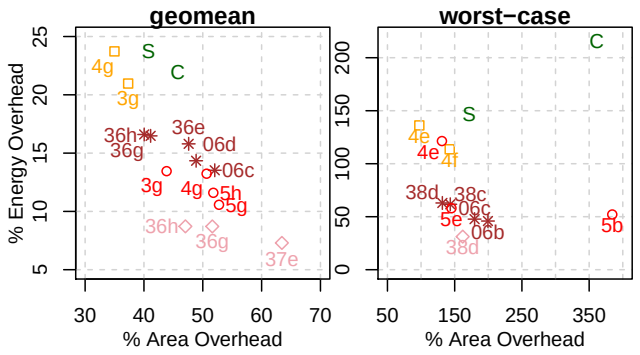
6.1 Issues

When we increase the number of PEs, we can reduce the size of the dataset that must be processed by each PE, decomposing the memory needed by the application into multiple, smaller memories (smaller M_{app}) and thus lowering the energy per memory access. Specifically, doubling the number of PEs often halves the M_{app} and hence reduces memory energy by $\sqrt{2}$. For most designs, increasing the number of PEs also increases the data that must be routed among PEs and hence increases inter-PE routing energy. As long as the fraction of energy in memory remains larger than the inter-PE routing energy, increasing the number of PEs results in a net reduction in total energy.

For example, Fig. 11 shows the shape of an $N \times N$ by $N \times N$ matrix-multiply $A \times B = C$ for different parallelism levels P ($N = 4$ is shown). The computation is decomposed by columns, with each PE performing the computation for N/P columns of the matrix. The B data is streamed in first and stored in P memories of size N^2/P, then A is streamed in row major order. Each A datapoint ($A[i,k]$) is stored in a register, data for each column (j) is read from each B memory, a multiply-accumulate is computed ($C[i,j] = C[i,j] + A[i,k] \cdot B[k,j]$), and the result is stored in a C memory of size N/P.[3] Once all the A datapoints of a row have been processed, the results of the multiply-accumulates can be streamed out, and the C memories can be used for the next row. When $P = N$, C does not need memories. Either way, increasing P keeps the total number of multiply-accumulates and memory operations constant. However, since the memories are organized in smaller banks, each memory access now costs less, and energy is reduced, as long as the interconnect-per-PE does not increase too much.

Fig. 12 shows how energy-efficiency changes with PE count for three tunable benchmarks. It shows how energy is reduced with additional parallelism up to an energy-minimizing

[3]This is different from the matrix-multiply in Section 2, where C was stored in an output memory of size N^2 ($P = 1$), keeping only one size of memory for the application.



label	0	1	2	3	4	5	6	7	8
mem size (Kb)	1	2	4	8	16	32	64	128	256

label	a	b	c	d	e	f	g	h	i	j	k
dm=	1	2	3	4	5	6	7	8	9	10	11

Figure 10: Energy-Area Trade-off

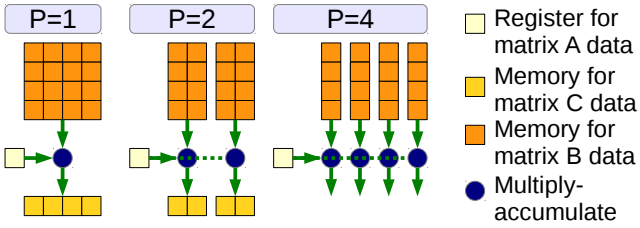


Figure 11: Parallelism Impact on Memory and Interconnect Requirements for a $(4 \times 4)^2$ matrix-multiply

number of PEs, with benefits of 86% (WFilter128), 27% (Sort8K), and 34% (MMul128).

6.2 Experiments

Once the memory architecture is fixed (with a given memory block size), being able to tune the parallelism level of the benchmarks as in Section 6.1 allows us to reduce potential mismatches between the application's memory requirements and the memory architecture. For example, consider Fig. 13, showing a sweep of both memory block size and parallelism for MMul128. The curves are normalized to the same limit study point as in Section 5 ($P = 1$), hence the curves can go below 0% overhead: the more parallel versions are different designs, and they can be more energy-efficient. For example, within the space of internally-banked, 1-memory-size architectures, Section 5 concluded that a memory size

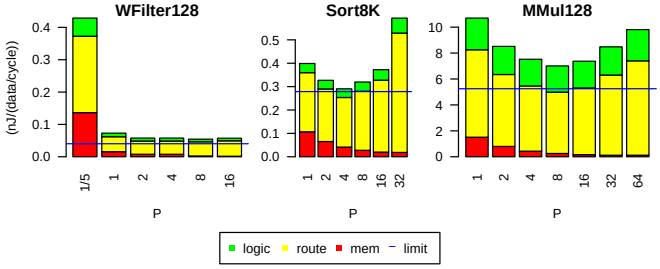


Figure 12: Energy breakdown versus P for different tunable benchmarks (4Kb int-bank with $d_m = 2$)

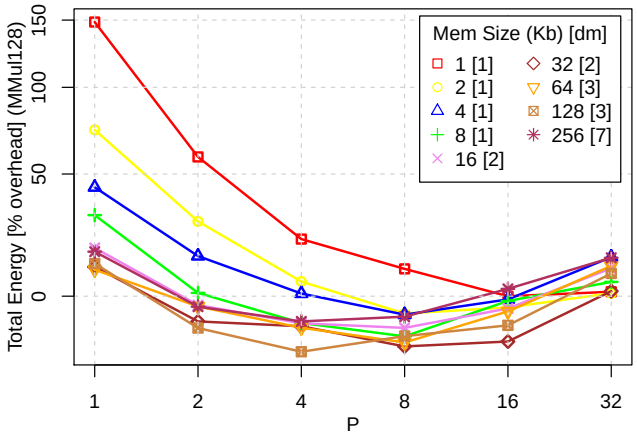


Figure 13: Energy versus P for MMul128 and varying mem sizes (int-bank) (normalized to the limit study for $P = 1$)

of 32Kb minimizes energy. In the case of MMul128, this gave an overhead of 10%. However, tuning the benchmark to $P = 8$ brings it down to -15%, a 23% reduction. This shows that parallelism can be a powerful optimization to reduce energy even when we do not have control over the memory architecture. In fact, MMul128 has a point at -17% overhead ($P = 4$ and a 128Kb memory), suggesting that the ability to tune P may shift the optimum architecture.

Fig. 14 shows the effect of tuning to optimum P for different memory sizes and d_m values. Due to space constraints, we show only the 2-memory, internally banked designs, which contain the lowest energy points. We can observe three effects from parallelism tuning that the previous sections have set up: (1) reduce the absolute energies, and hence overheads, achievable; (2) shift the energy-minimizing parameter selections to smaller memories (*e.g.*, 1Kb+64Kb vs. 8Kb+256Kb for worst-case); and (3) create broader near-minimum-energy valleys, making energy overhead less sensitive to the selection of memory block size.

7. CONCLUSIONS

We have shown how to size and place embedded memory blocks to guarantee that energy is within a factor of two of the optimal organization for the application. We focused on energy-balanced FPGA design points, which may be different from the logic-rich design points for current commercial architectures. On the benchmark set, we have seen that a two-memory design with 8Kb and 256Kb banks with internal banking and $d_m = 4$ keeps the worst-case mismatch energy overhead below 31% compared to an optimistic limit-study lower bound. Internal banking provided 19% of the energy savings. The optimal energy-balanced memory architecture for energy minimization differs from the logic-rich, area-minimizing points: we are driven to multiple memory sizes (8Kb and 256Kb vs. single 16Kb) and more frequent (d_m=4 vs. d_m=5–9) memories, spending 33% more worst-case area (28% more geomean area on logic-rich benchmarks) for 80% lower worst-case energy (16% lower geomean energy). Finally, tuning parallelism in the application can reshape the memory use, reducing the energy overhead by avoiding memory size mismatches. Joint optimization further reduces the worst-case energy overhead by 13%, and the geomean by 25%.

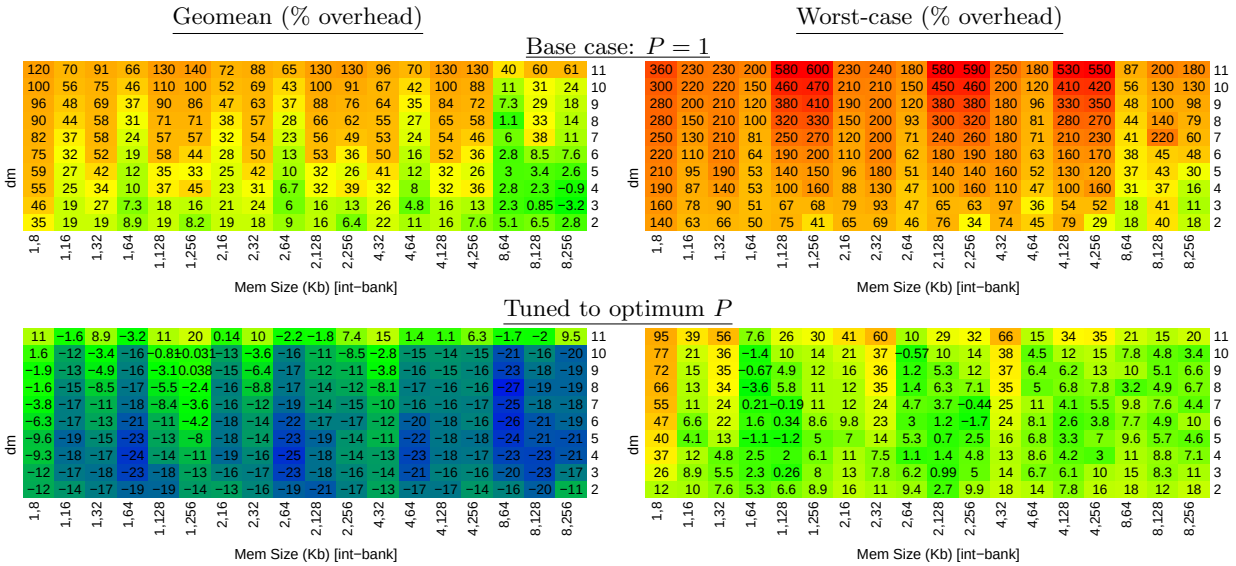


Figure 14: Energy overhead for the tunable benchmarks

8. ACKNOWLEDGMENTS

This research was funded in part by DARPA/CMO contract HR0011-13-C-0005. David Lakata was supported by the VIPER program at the University of Pennsylvania. 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.

9. REFERENCES

[1] International technology roadmap for semiconductors. <www.itrs.net/Links/2012ITRS/Home2012.htm>, 2012. 4.4

[2] Altera Corporation. *PowerPlay Early Power Estimator*, 2013. 5.4

[3] V. Betz, J. Rose, and A. Marquardt. *Architecture and CAD for Deep-Submicron FPGAs*. Kluwer Academic Publishers, Norwell, MA, 02061 USA, 1999. 3.1

[4] Bluespec, Inc. Bluespec SystemVerilog 2012.01.A. 4.6

[5] 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.*, 22(3):537–547, March 2014. 4.6

[6] J. Goeders and S. Wilton. VersaPower: Power estimation for diverse FPGA architectures. In *ICFPT*, pages 229–234, 2012. 3.2

[7] E. Kadric. Power optimization (p-opt) code and architecture files. `http://ic.ese.upenn.edu/distributions/meme_fpga2015/`, 2015. 4.2

[8] E. Kadric, K. Mahajan, and A. DeHon. Kung fu data energy—minimizing communication energy in FPGA computations. In *FCCM*, 2014. 2, 6

[9] D. Koch and J. Torresen. FPGASort: A high performance sorting architecture exploiting run-time reconfiguration on FPGAs for large problem sorting. In *FPGA*, pages 45–54, 2011. 4.6

[10] I. Kuon and J. Rose. Measuring the gap between FPGAs and ASICs. *IEEE Trans. Computer-Aided Design*, 26(2):203–215, February 2007. 2

[11] J. Lamoureux and S. J. E. Wilton. Activity estimation for field-programmable gate arrays. In *FPL*, pages 1–8, 2006. 4.1

[12] 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*, pages 33–42, 2009. 3.1

[13] D. Lewis, D. Cashman, M. Chan, J. Chromczak, G. Lai, A. Lee, T. Vanderhoek, and H. Yu. Architectural enhancements in Stratix V. In *FPGA*, pages 147–156, 2013. 3.2

[14] J. Luu, J. H. Anderson, and J. S. Rose. Architecture description and packing for logic blocks with hierarchy, modes and complex interconnect. In *FPGA*, pages 227–236, 2011. 3.2, 4.2

[15] J. Luu, J. Goeders, M. Wainberg, A. Somerville, T. Yu, K. Nasartschuk, M. Nasr, S. Wang, T. Liu, N. Ahmed, K. B. Kent, J. Anderson, J. Rose, and V. Betz. VTR 7.0: Next generation architecture and CAD system for FPGAs. *ACM Tr. Reconfig. Tech. and Sys.*, 7(2):6:1–6:30, July 2014. 2, 3.1, 4.6

[16] N. Muralimanohar, R. Balasubramonian, and N. P. Jouppi. CACTI 6.0: A tool to model large caches. HPL 2009-85, HP Labs, Palo Alto, CA, April 2009. Latest code release for CACTI 6 is 6.5. 3.3

[17] K. Poon, S. Wilton, and A. Yan. A detailed power model for field-programmable gate arrays. *ACM Tr. Des. Auto. of Elec. Sys.*, 10:279–302, 2005. 3.2

[18] J. Rose, J. Luu, C. W. Yu, O. Densmore, J. Goeders, A. Somerville, K. B. Kent, P. Jamieson, and J. Anderson. The VTR project: architecture and CAD for FPGAs from verilog to routing. In *FPGA*, pages 77–86, New York, NY, USA, 2012. ACM. 4.2

[19] R. Tessier, V. Betz, D. Neto, A. Egier, and T. Gopalsamy. Power-efficient RAM mapping algorithms for FPGA embedded memory blocks. *IEEE Trans. Computer-Aided Design*, 26(2):278–290, Feb 2007. 4.2, 4.2

[20] H. Wong, V. Betz, and J. Rose. Comparing FPGA vs. custom CMOS and the impact on processor microarchitecture. In *FPGA*, pages 5–14, 2011. 3.1