Wordwidth, Instructions, Looping, and Virtualization The Role of Sharing in Absolute Energy Minimization

André DeHon
Department of Electrical and Systems Engineering, University of Pennsylvania
200 S. 33rd St., Philadelphia, PA 19104
andre@acm.org

ABSTRACT

When are FPGAs more energy efficient than processors? This question is complicated by technology factors and the wide range of application characteristics that can be exploited to minimize energy. Using a wire-dominated energy model to estimate the absolute energy required for programmable computations, we determine when spatially organized programmable computations (FPGAs) require less energy than temporally organized programmable computations (processors). The point of crossover will depend on the metal layers available, the locality, the SIMD wordwidth regularity, and the compactness of the instructions. When the Rent Exponent, p, is less than 0.7, the spatial design is always more energy efficient. When $p = 0.8$, the technology offers 8-metal layers for routing, and data can be organized into 16b words and processed in tight loops of no more than 128 instructions, the temporal design uses less energy when the number of LUTs is greater than 64K. We further show that heterogeneous multicontext architectures can use even less energy than the $p = 0.8$, 16b word temporal case.

Categories and Subject Descriptors

B.7.1 [**Integrated Circuits**]: Type and Design Styles—*VLSI*; C.0 [**General**]: Modeling of Computer Architecture; C.2.1 [**Computer Communication Newtorks**]: Network Architecture and Design

Keywords

Energy; Energy Modeling; Low Power; FPGA; Rent's Rule; Locality; Instructions; Multicontext; SIMD

1. INTRODUCTION

As we enter the era of mobile devices and *dark silicon* [4], minimizing energy becomes the dominant concern when engineering computations. Battery life and power-density envelopes limit the performance we can extract, not critical path delay or area. Under these constraints, architectures that reduce the energy per operation are the most beneficial.

In prior work [2], we established that spatial (FPGA-like) computations can use asymptotically lower energy than sequential (processor-like) computations. However, this analysis did not estimate when these asymptotic effects would begin to dominate. The goal of this paper is to estimate when the asymptotic effects are large enough to have a practical impact and to refine the comparison to account for common optimizations that allow processors to save energy. Consequently, this paper builds absolute energy models including constant terms and scale factors. To make the model results general, we introduce parameters for key technology factors (*e.g.*, relative size of memory bits compared to wire pitch, metal layers available for routing). To balance model complexity with accuracy, we use a wire-dominated energy model suitable for modern VLSI processes (Sec. 3.2).

To better characterize absolute energy in processors, we observe that processors reduce the overhead of instruction energy by sharing instructions using single-instruction, multiple data (SIMD) control of multibit words and looping (Sec. 4). The energy reduction is sufficient to allow the processor to use less energy for a computation than spatial (FPGA-like) designs (Sec. 5) when the design has little interconnect locality such that its Rent Exponent, p (Sec. 2.1), is greater than 0.75. When the Rent Exponent is less than 0.75, even with these optimizations, the FPGA uses less energy than the processor, but the optimizations reduce the absolute advantage. For the cases where the Rent Exponent is large, we identify architectural points between the FPGA and processor extremes that achieve lower energy than either (Sec. 6). Our analysis provides parameterized models that allow relative area and energy comparisons across a wide range of design and technology parameters.

Our contributions include:

1. parameterized, wire-dominated energy models for a programmable structure that can be tuned to represent single and multiple context FPGAs and processors
2. identified energy crossover points as a function of design size and a wide range of technology (*e.g.*, metal layers) and application characteristics including locality, word width, and loop size
3. optimized heterogeneous, synchronous, multicontext architecture and identification of the energy-minimizing architectural parameters
4. comparison of relative energies for select organizations across this design space

Permission to make digital or hard copies of all or part of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. Copyrights for components of this work owned by others than the author(s) must be honored. Abstracting with credit is permitted. To copy otherwise, or republish, to post on servers or to redistribute to lists, requires prior specific permission and/or a fee. Request permissions from permissions@acm.org.
FPGA'14, February 26–28, 2014, Monterey, CA, USA.
Copyright is held by the owner/author(s). Publication rights licensed to ACM.
ACM 978-1-4503-2671-1/14/02 ...$15.00.
http://dx.doi.org/10.1145/2554688.2554781.

2. BACKGROUND

Before developing our detailed energy models, we first review Rent's Rule, the hierarchical network style we will use for interconnect, and the asymptotic results from [2] that leverage Rent's Rule and the hierarchical networks.

2.1 Rent's Rule

Typical circuits do not look like random graphs. Rather, they have local clusters that can be physically placed to minimize the number of nets that enter and exit a region. First identified by E. F. Rent and published by Landman and Russo [8], Rent's Rule says that the number of wires that must cross into or out of a region of N components is reasonably modeled by:

$$BW = cN^p \tag{1}$$

where p is the Rent Exponent, which is typically in the range 0.5–0.7. The Rent Exponent can be used as a measure of locality. Designs with a smaller fraction of wires exiting a region, smaller BW, are characterized by a smaller p. Rent's Rule has been used to estimate wiring requirements in FPGAs (*e.g.* [7] used $p = 0.78$). As Hutton notes, both the circuit netlists and the FPGA substrate can each be characterized with their own Rent Parameters. For the scope of this paper, we make the simplifying assumption that the design and substrate Rent Parameters are matched.

Rent locality directly implies the wire length distributions in a design [3]. If the circuit has a Rent Exponent less than 0.5, the average wirelength is a constant independent of the number of gates, N. When the circuit has a Rent Exponent greater than 0.5, the average wirelength scales as $N^{p-0.5}$.

2.2 Hierarchical Network Review

Lieserson observed that excessive wiring in VLSI designs could lead to area inefficient implementations. As a result, he developed the Fat Tree interconnection network that has limited wiring bandwidth that grows according to Rent's Rule and, like Rent's Rule, can be parameterized by p [10]. The Butterfly Fat Tree (BFT) variant of the Fat Tree uses simple, constant size switches at the tree stages [6] (See. Fig. 1a). In this paper, we use a version of the BFT based on directional wiring [12, 11].

In these networks, the bandwidth into each subtree grows toward the root in accordance with Rent's Rule. The growth can be programmed discretely by selecting the use of bandwidth preserving 2:1 stages (2 parent links for 1 child link in each of the two sibling subtree) and bandwidth reducing 1:1 stages. By alternating 2:1 and 1:1 stages as shown in Fig. 1a, the bandwidth increase by a factor of 2 across two stages where the number of gates in the subtree has increased by a factor of 4. This corresponds to a p of 0.5 ($4^{0.5} = 2$). By changing the selection of 2:1 and 1:1 stages, we can configure networks for different p's. A $p = \frac{2}{3}$ network is realized by repeating a sequence of 2:1, 2:1, and 1:1 switches ($\left(2^3\right)^{0.67} = 4$).

In the heterogeneous multicontext design, we allow the tree to have a different set of Rent parameters ($c' = \frac{c}{C_t}$, $p' = p_t$) from the design it supports and place configuration memories local to the switches. To route the design, we time multiplex the original wiring requirements over this reduced physical network. When $p_t < p$, the time multiplexing factor, and hence the size of the memories, grows toward the root of the tree giving rise to the heterogeneous multicontext interconnect. At the root of a subtree of size N, the instruction memories are of depth $C_t N^{p-p_t}$. Fig. 1b shows a $C_t = 2$, $p_t = 0.25$, network used to implement the $c = 2$, $p = 0.5$ network shown in Fig. 1a. It realizes the $p = 0.25$ network by repeating the stage sequence of 2:1, 1:1, 1:1, and 1:1 switches ($\left(2^4\right)^{0.25} = 2$). The switch at the top of the 16-gate tree shown has memories of depth $C_t N^{p-p_t} = 2 \times \left(2^4\right)^{0.25} = 4$.

2.3 Asymptotic Review

To implement a computation sequentially, we must store the intermediate state of the computation in a memory. A circuit with N gates, stores each of the N gate outputs in a large memory. The size N memory has a side length of $O(\sqrt{N})$, meaning every input read or value stored will cost $O(\sqrt{N})$ energy. In contrast, if we spatially layout the circuit for the graph, we can try to minimize the average wirelength between producers and consumers. If the design has a Rent Exponent less than 0.5, the average wirelength is constant, and the energy per operation is $O(1)$. If we have an unlimited number of wire layers, we immediately know that the wires between gates will be at most $O(\sqrt{N})$ and, consequently, energy is no greater than $O(\sqrt{N})$. The wirelength relation above suggests the wirelength will only grow as $O(N^{p-0.5})$, meaning the energy will always be less than $O(\sqrt{N})$—less than the sequential energy—until $p = 1.0$. This wirelength phenomena largely explains the spatial and sequential asymptotic results from [2].

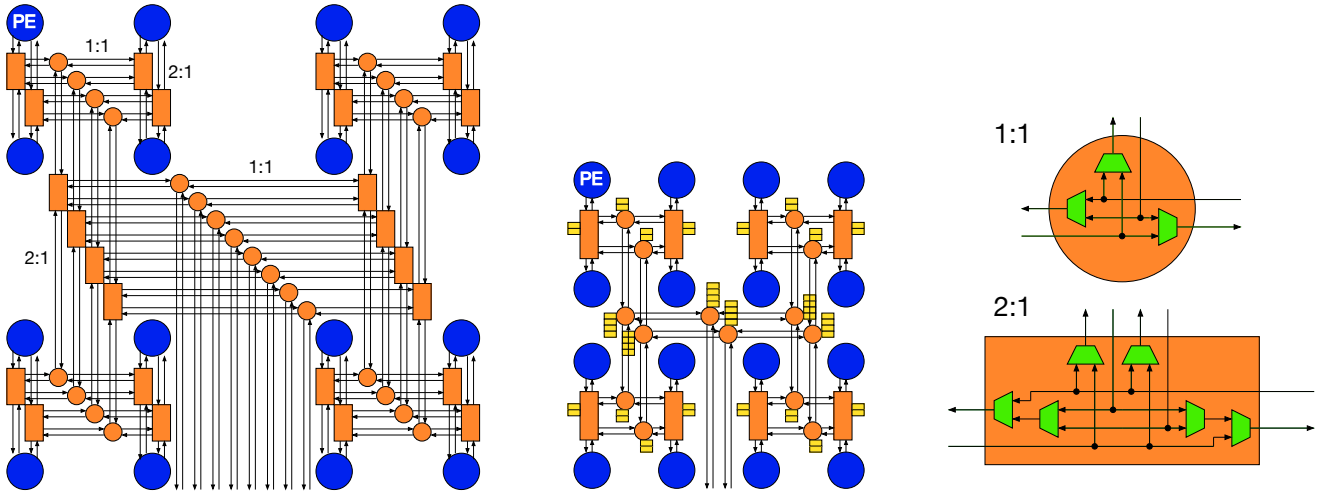
When we are limited to a constant number of metal layers, the side length grows as $O(N^p)$, and energy per gate can grow as $O(N^{2p-1})$, which is greater than the sequential energy $O(\sqrt{N})$ when $p > 0.75$. This larger area and energy is driven by the wiring required to support a large p design. However, [2] also shows that, by using a $p_t < 0.5$ heterogeneous multicontext BFT (previous section), we can bring the energy per operation below $O(\sqrt{N})$ for any $p < 1$.

While the spatial design has lower energy, asymptotically, it is not clear when the asymptotic effects begin to matter. Furthermore, while the asymptotic analysis points to the inefficiency of a sequential processor with a single memory, it does not rule out the possibility that sequential processors of some limited, fixed capacity could be less energy than fully spatial designs. The asymptotic results do not address how to organize the computation to minimize absolute energy.

3. MODELS

3.1 Computational Graph Model

Our computational model is a graph of interconnected operators that must all be evaluated on each cycle of operation. The graph could be a circuit netlist or a homogeneous synchronous dataflow (SDF) graph [9]. The graph may include state elements such as SDF delay elements or registers in a circuit netlist. The graph may contain cyclic paths as long as they are valid synchronous cycles, meaning there must be one or more registers or delay elements on each cyclic path. The operators are nodes in the graph. Nodes take in a fixed set of inputs, perform some logical operation on the inputs, and produce a fixed set of outputs. The node could be a logical gate or a flip flop in a circuit netlist or an arithmetic operation or delay element in an SDF graph. The edges in the graph describe how outputs from one graph node be-



(a) Spatial BFT (b) Heterogeneous Multicontext (c) Switch Composition

Figure 1: Directional Butterfly Fat Tree (BFT) with $c = 2$, $p = 0.5$

come inputs to others. Each edge has a single source and may have multiple destinations. Since the graph is a valid synchronous graph, a cycle of computation can be computed by ordering the nodes topologically and evaluating each node once based on its inputs. As a simplifying assumption, we assume every node switches on every evaluation cycle and leave the issue of non-homogeneous node switching or low activity to future work (Sec. 7).

3.2 Wire-Dominated Energy Model

To limit the complexity of the models and to improve their generality, we use a wire-dominated energy model. Assuming a fixed voltage for operation, dynamic switching energy is proportional to the capacitance switched. The capacitance switched will come from both the gates driven by each net and the capacitance of the wiring. In modern processes the capacitance in the wiring is the dominant component. At 32nm, the capacitance per unit wire of length one minimum feature width, F, is 6.4×10^{-18} Farads, while the capacitance of a transistor gate per minimum transistor width unit is 29×10^{-18} Farads (From [1], INTC2 and PIDS3-LOP). A dense SRAM cell is $140F^2$. Crossing the $12F$ width of the cell is $\frac{12\times6.4}{29} \approx 2.6\times$ more wire capacitance than the gate. Other gates are packed less densely than SRAM cells and have wiring that extends across multiple cells. As a consequence, we can reasonably focus on the wire capacitance to get an adequate approximation for dynamic capacitance switched. For this paper, we calculate the total wire lengths switched in order understand dynamic switching energy, E_{dyn}.

$$E_{dyn} \approx 0.5V^2 \sum_{\text{all wires i}} \alpha_i L_{wire_i} \times C_u \quad (2)$$

L_{wire_i} is the length of wire component i, and α_i is its switching activity. C_u is the capacitance per unit length of wire.

3.3 Memory Energy Models

Our memory energy model is based on and illustrates the wire-dominated model. For a random access memory with a W-bit wide interface containing M W-bit words, the total capacitance switched on a read or write operation is:

$$C_{rmem}(W, M) = (\log(M) + 2(2W + 2))\sqrt{WMA_{bit}}C_u \quad (3)$$

To minimize energy, the core array is organized into a square of length $\sqrt{WM}$ bit cells on a side. We must drive the $\log(M)$ address lines across the width of the array either for the word-line decoders or for the final mux selection. We must drive the bit-lines that run the height of the array for the W bits selected. The W selected bits will then run the width of the array in multiplexer selection. These two are captured in the $2W$ term. The bit-line and a word-select line run the width and height of the array, hence the "+2" term. We assume the bit-line and word-lines switch on and off for each memory operation and multiply those terms by two. The area for this memory is:

$$A_{rmem}(W, M) = \left(\sqrt{WMA_{bit}} + FP\left(\frac{\log(M)}{2}\right)\right)^2 \quad (4)$$

FP is the full wire pitch. CACTI [14] estimates at 32nm and below confirm the wire dominated energy assumption. Since CACTI is optimized more for performance than energy, the absolute CACTI results are uniformly 2.5× larger than the estimate above. At the risk of being generous to the sequential designs where the energy is dominated by memories, we use our smaller energy estimate as the main comparison point. In select places (Fig. 3, 4, and 11), we also report the larger memory energy case to illustrate the sensitivity of results to the ratio of memory energy to wire energy.

Sequentially accessed memories, as are appropriate for the instruction memories, can avoid the cost of addressing. A simple shift register can activate the appropriate rows and control the output multiplexer. The capacitance switched for a sequentially accessed memory is:

$$C_{smem}(W, M) = (2(2W + 1))\sqrt{WMA_{bit}}C_u \quad (5)$$

The area of the sequential memory is:

$$A_{smem}(W, M) = WMA_{bit} + \sqrt{WM}A_{shift} + \sqrt{\frac{M}{W}}A_{shift} + \left(\sqrt{WM} - W\right)A_{mux} \quad (6)$$

4. SEQUENTIAL PROCESSOR

We start with a simple sequential architecture composed of a 4-LUT to perform the computation, an instruction memory to specify the behavior of each logical 4-LUT in the netlist graph, and a data memory to hold the value on each edge, or net, in the netlist graph (Fig. 3 in [2]). We use a 4-LUT because previous work showed it to be the least energy for spatial designs [15, 13]; for the sake of this paper, we take the LUT size as a constant and do not explore it as an optimization parameter. For the wire-dominated energy model, we ignore the gate energy for the 4-LUT and control and focus on the memory energy. For each LUT evaluation, we must read the instruction, read 4 data values from the data memory, and write one value to data memory. This means a total of five memory operations on the random-access memory that holds data. The total capacitance switched when evaluating an N node graph is:

$$\begin{aligned} C_{seq} &= 5NC_{rmem}(1,N) \\ &\quad + I_{bits}(N,p)C_{smem}(1,I_{bits}(N,p)) \end{aligned} \tag{7}$$

The instruction bits, I_{bits}, depend on the size of the graph and the Rent Exponent, p. As [2] notes, a naive version would spend $\log(N)$ bits for each of the read addresses, but exploiting a Rent's-Rule style recursive bisection, we can use fewer bits to specify nodes that are "closer" in the recursive bisection tree. Only the nets that are cut at the root of the tree require $\log(N)$ bits; with $p < 1$, most require fewer. Examining the recursive bisection tree, we can account for the number of nets crossing at each tree level to understand the total number of bits to specify addresses, B_{src}:

$$B_{src} = \sum_{l=0}^{\log(N)} \left(2^{lp} \times \frac{N}{2^l}\right) = N \sum 2^{l(p-1)} = \frac{N}{1-2^{p-1}} \tag{8}$$

l is the height in the recursive bisection tree from the leaves ($l = 0$) to the root ($l = \log(N)$). Each instruction needs to specify 4 inputs and one output, so we multiply this value by 5. Each instruction must also specify the 4-LUT function, so we add 2^4 bits for the function, giving us:

$$I_{bits} = \left(\frac{5}{1-2^{p-1}} + 16\right) N \tag{9}$$

It is clear from examining Eq. 9 that the instruction memory is larger than the data memory by a significant constant factor. For $p = 0.7$, $I_{bits}/N \approx 43$. For regular graph operations, the instructions are the same for different data values and can potentially be reused. A common form of this is looping, where a set of instruction, the loop body, are reused across a large set of data. Low-level image processing or cellular automata are some of the most familiar examples of this kind of looping, applying the same set of operations to each neighborhood region of data. For a general formulation, we introduce a separate variable I for the total number of unique instructions required and allow that to be independent of N. To first order, I can also be viewed as modeling the impact of a first-level instruction cache, assuming the instruction trace is sufficiently localized that all references are effectively satisfied in this cache.

We can also reduce the number of instructions by sharing them across a wide word, W. One defining property of processors is that they do not operate on single-bit data elements, but rather operate on a set of bits (*e.g.*, 8, 16, 32, 64) grouped into words. This allows them to read many bits from memory, both reducing the number of addresses that must be specified and amortizing out the cost of specifying the address. Assuming the data bits stay fixed, this also reduces the number of instructions, since an instruction now applies to a group of W bits. The word operations are typically applying the same operation to each bit (*e.g.*, bitwise-AND, bitwise-XOR), and even arithmetic operations like ADD and SUBTRACT are effectively telling each bit of the datapath to act like an adder bitslice. This is known as a single-instruction, multiple-data (SIMD) operation since a single instruction is reused to control the W bits in the datapath. As a result, we can reformulate the energy for the processor in terms of the SIMD width, W and the total instructions, I.

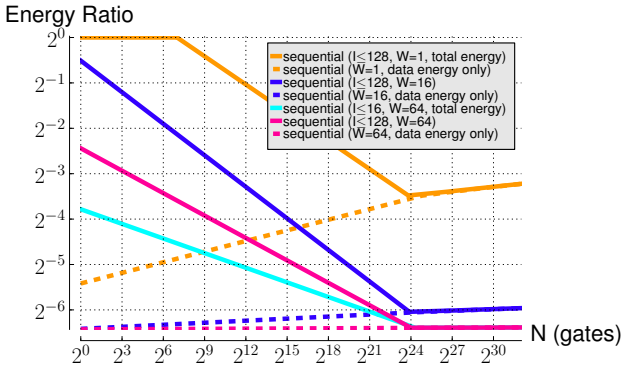


Figure 2: Sequential Energy Ratio to $W = 1$, $I = N$ for $p = 0.7$

$$\begin{aligned} C_{seqlw} &= 5\left(\frac{N}{W}\right) C_{rmem}\left(W, \frac{N}{W}\right) \\ &\quad + \left(\frac{I_{bits}(N,p)}{W}\right) C_{smem}(1, I_{bits}(I,p)) \end{aligned} \tag{10}$$

Fig. 2 compares the energy ratios at $p = 0.7$ to show the impact of limited instructions, I, and SIMD word width, W. All cases are normalized to the $I = N$, $W = 1$ sequential energy case. As the "data energy only" curves show, there is a clear crossover point where data energy begins to dominate instruction memory energy.

5. SPATIAL PROCESSING

The fully spatial processing architecture builds a 4-LUT for each node in the graph along with the BFT interconnection network. Instructions (configuration bits) are stored local to each LUT and interconnect multiplexer control bits and never change, consuming no dynamic energy. Dynamic energy is consumed switching the wire capacitance routing data between LUTs. To model this capacitance, we need to know the number and lengths of the wires, which, in turn, demands we know the area of the structure.

Area is driven either by the active area for LUTs, configuration bits, and switches or the wiring area for routing interconnect. Since these use different layers, an accurate calculation might take the maximum of the area required for wiring and interconnect. To simplify analytic calculation, we will add together area for wiring and active elements. The result is potentially a conservative overestimate, by as much as a factor of two when wiring area is the same as active

area. When one area component dominates, as it does for large designs, the sum becomes close to the maximum.

For simplicity, we use 2-input multiplexers as the basic building block for switching (Fig. 1c), including in the C-box connections between the network and the LUT. c is the number of base channels in the tree. Since this is a bidirectional network, there are both c inputs and c outputs from the network at each leaf (Fig. 1a). We will be using $c = 5$. The leaf area includes the LUT, its configuration bits, and the C-box multiplexers to select the LUT inputs from the tree network. C-box multiplexers can be depopulated to exploit the fact that they are feeding a 4-LUT [5]. The single output is driven into all of the network inputs at the leaf, leaving their selection to the network switches.

$$A_{leaf} = A_{4lut} + 2^4 A_{bit} + ((c-4)4))\,(A_{mux2} + A_{bit}) \quad (11)$$

From Figs. 1a and 1c, we see that each directional wire pair at the top of a subtree is associated with 3 two-input multiplexers. Counting based on wiring at each tree level gives the total switch area:

$$A_{sws} = \left(c \sum \left(2^{lp} \frac{N}{2^l}\right)\right)(3A_{bit} + 3A_{mux2}) \quad (12)$$

Putting these together:

$$A_{active} = N A_{leaf} + A_{sws} \quad (13)$$

For wiring we first count the number of wire channels needed across the width of the chip by looking across all overlapping wiring channels:

$$\begin{aligned} Wires &= 2\left(cN^p + 2cN^p\left(\frac{1}{2}\right)^{2p} + 4cN^p\left(\frac{1}{2}\right)^{4p} + \ldots\right) \\ &= 2cN^p \sum 2^{(1-2p)l} \end{aligned} \quad (14)$$

The constant two at the beginning of Eq. 14 accounts for the separate up and down links of the bidirectional channels. The exponent increases by a factor of two since we are counting every other stage to account the contribution of wires to a single dimension. The constant in front doubles since we're doubling the number of wire channels that parallel each other at every other tree stage.

The actual width required by the wires will depend on the number of metal layers available for programmable interconnect routing in the process. We will assume half the metal layers are available for horizontal wiring and half for vertical. We further assume we can perfectly use all the metal layers in each direction. This gives:

$$L_{wire} = \frac{Wires \times FP}{M_{layers}/2} = \frac{2FP \times Wires}{M_{layers}} \quad (15)$$

FP is the full pitch of the wires, which we take as 2 to normalize to the half-pitch feature size used in defining C_u. The length of the side of the entire design is thus:

$$L_{side} \leq \sqrt{A_{active}} + L_{wire} \quad (16)$$

We then determine the total wire capacitance switched by summing up the capacitance of all the wires.

$$C_{spatial} = \sum cN^p 2^i \left(\frac{N}{2^i}\right)\left(\frac{L_{side}}{2^{\lceil i/2 \rceil}}\right) \quad (17)$$

The ceiling in the sum accounts for the fact that every other stage extends in a different dimension, so the subtree width shrinks every other stage. Computing the subtree length as a fraction of L_{side} like this is a conservative overestimate of wire length since lower level wiring runs do not need to cross upper-layer wiring.

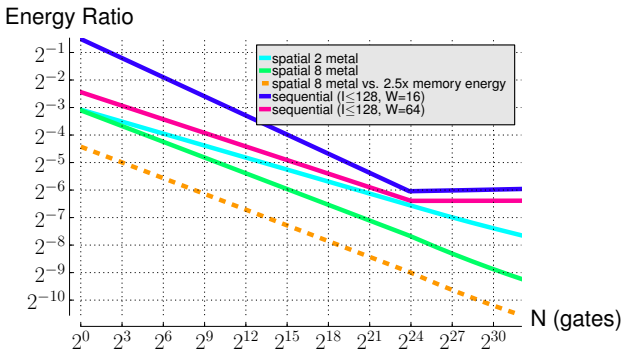


Figure 3: Energy Ratio to $W = 1$, $I = N$ Sequential for $p = 0.7$

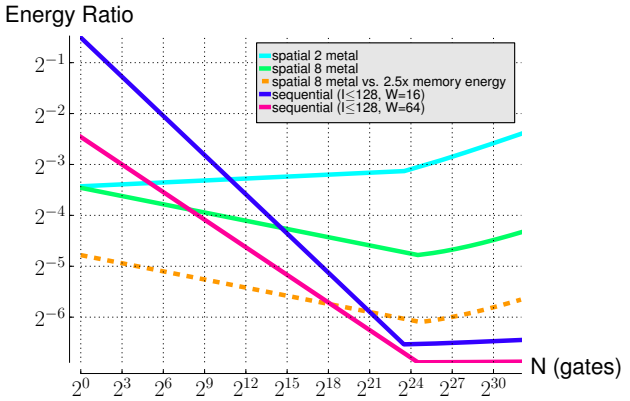


Figure 4: Energy Ratio to $W = 1$, $I = N$ Sequential for $p = 0.8$

Fig. 3 shows the energy ratios to the sequential architecture at $p = 0.7$. The spatial energy is always below the sequential energy and the benefit grows with design size, N. Large word width, W, and tight loops (small I) reduce the spatial benefit, but do not eliminate it. We include an 8-metal case as typical for current technology. At 32nm, the ITRS [1] suggests up to 13 metal layers are available, so even if some of those are dedicated to local wiring and power and clock distribution, eight metal layers may reasonable be available for network routing. If the technology or memory design is more expensive relative to wiring (*e.g.*, 2.5× as appropriate to match CACTI memories), the spatial advantage is even larger.

However, when $p > 0.75$, the benefits of the spatial design will eventually diminish or reverse, as illustrated in Fig. 4 for $p = 0.8$. Here, the spatial design is less energy for $W = 1$, $I = N$ up to billions of 4-LUTs, but the benefit is beginning to diminish. When the word width is larger and the instruction memory small, the sequential design can be more energy efficient. For 8 metal layers, $I \leq 128$, and $W = 16$, the sequential design becomes more energy efficient around 64K 4-LUTs. *Both the growing spatial benefit below $p = 0.75$ and the diminishing spatial benefits above $p = 0.75$ show that asymptotic effects do matter for this design size.* Asymp-

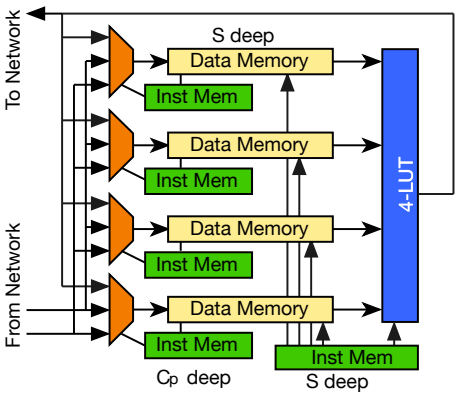


Figure 5: Multicontext Leaf Processing Element

totically, the sequential energy of $O(\sqrt{N})$ per operation is larger than the spatial energy of $O(N^{2p-1})$ when $p < 0.75$ and smaller when $p > 0.75$ (Sec. 2.3, [2]).

We do not evaluate the impact of looping and SIMD word width for the spatial design. The spatial design is already spending no switching energy on instructions, so there is no instruction energy to reduce with instruction sharing. Word grouping could reduce the number of configuration bits required. However, since the designs are wire rather than active area dominated, the savings is not significant.

6. MULTICONTEXT

The sequential and spatial designs are extremes in a larger design space. Since we see regions where each achieves lower energy, the natural question arises: is there an intermediate point in the design space that achieves even lower energy than the extremes? In this section, we extend our energy analysis to the heterogeneous multicontext design from [2].

We consider three main architectural variables S, p_t, and C_t. S is the leaf serialization. We assign S nodes to each physical leaf PE and evaluate them on a single 4-LUT. C_t is a constant serialization of the interconnect. p_t is the growth of the physical tree. As a result, the number of physical wires at the top of the hierarchical network (Fig. 1b) is $\frac{c}{C_t}N^{p_t}$ rather than cN^p when $p_t < p$, meaning there is a communication serialization of $C_t N^{p-p_t}$.

In this section, we first show the composition and resources for the leaf PE, then identify the interconnect resources, before addressing coordination. We identify the coordination challenge, examine the asynchronous proposal from [2], and introduce a synchronous version that is more efficient for typical designs. Finally, we determine the energy minimizing parameters and compare to the spatial and sequential extremes from the previous sections.

6.1 PE

Fig. 5 shows the multicontext PE. Each PE has a single 4-LUT fed by four independent data memories. Each data memory selects inputs from the incoming network wires to the PE. Independent instruction memories control each data memory write and the evaluation of the PEs. Each data memory holds S values so that the four together can provide the 4 inputs to each of the S nodes assigned to this PE. In the extreme case where there is only one PE ($S = N$), this is four times the data memory required since there are only N different data items. The input instruction memories need to be large enough to write S values into memory that may arrive over $C_t S^{p-p_t}$ cycles, so we define their depth:

$$C_p = \max\left(S, C_t S^{p-p_t}\right) \tag{18}$$

We compute the area of the PE as:

$$\begin{aligned} A_{pe} &= 4A_{mux}\left(\frac{c}{C_t}S^{p_t}\right) + 4A_{rmem}(1,S) + A_{4lut} \\ &\quad +4A_{smem}\left(\log\left(\frac{c}{C_t}S^{p_t}+1\right)+\log(S), C_p\right) \\ &\quad +A_{smem}(4\log(S)+16, S) \end{aligned} \tag{19}$$

We compute the capacitance contribution from the PE per node evaluated as:

$$\begin{aligned} C_{pe} &= 8C_{rmem}(1,S) \\ &\quad +4\frac{C_p}{S}C_{smem}\left(\log\left(\frac{c}{C_t}S^{p_t}+1\right)+\log(S), C_p\right) \\ &\quad +C_{smem}(4\log(S)+16, S) \\ &\quad +2\times 6C_{wire}\sqrt{A_{rmem}(1,S)}(\log(S)+1) \end{aligned} \tag{20}$$

The eight on the random access memory captures the fact that we read and write to each data memory once for each PE evaluation. The final term captures the fact that we must route address wires over data memories that cannot all be adjacent to the associated instruction memories.

6.2 Network

The network is designed to be parameterizable between the fully spatial network used in Sec. 5 (when $p_t = p$ and $C_t = 1$) and a binary tree (when $p_t = 0$ and $C_t = c$, Fig. 5(a) in [2]). When the network is not fully spatial, the data transfer will need to be sequentialized across some of the network links. In order to support this, the switches at those data links will have instruction memories to allow them to change behavior. With $p_t < p$, the amount of sequentialization and hence the size of the switching instruction memories grow toward the root of the fat tree. This means there will be points in the tree where an upper-level switch changes behavior while a lower-level switch remains unchanging.

To understand the area for the array, we must, again, account for both active and wiring area. The total area going into switches is:

$$\begin{aligned} A_{sw} &= \sum_{l=\log(S)}^{\log(N)}\left(\frac{N}{2^l}\right)\left(\frac{c}{C_t}\right)\left(2^l\right)^{p_t} \\ &\quad \times\left(3A_{mux2}+A_{smem}\left(3, C_t\left(2^l\right)^{p-p_t}\right)\right) \end{aligned} \tag{21}$$

This includes both the area of the switch and the area of the instruction memory for the switch. Three bits control each of the three multiplexers in the 1:1 switch.

$$A_{mcactive} = \left(\frac{N}{S}\right)A_{pe} + A_{sw} \tag{22}$$

We count the number of wire widths across each side of the array:

$$MCWires = 2\sum_{l=\log(S)/2}^{\log(N)/2}\left(\sqrt{\frac{N}{2^{2l}}}\right)\left(\frac{c}{C_t}\right)\left(2^{2l}\right)^{p_t} \tag{23}$$

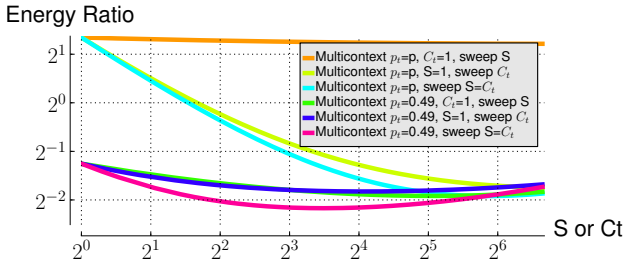


Figure 6: No Coordination Overhead Energy Ratio to Spatial for $p = 0.8$ at $N = 10^7$ and $M_{layers} = 8$

Side length due to wires is then:

$$L_{mcwire} = \frac{2 \times FP \times MCWires}{M_{layers}} \quad (24)$$

Including both wires and active area contributions:

$$L_{mcside} \leq \sqrt{A_{mcactive}} + L_{mcwire} \quad (25)$$

Interconnect energy will have contributions both from data transmission on wires and from switch instruction memories. For total wire data transmission capacitance, we compute a weighted sum based on wire length and usage.

$$C_{mcwire} = \sum_{l=\log(S)}^{\log(N)} \left(\frac{N}{2^l}\right) \left(c2^{lp}\right) \left(\frac{L_{mcside}}{2^{\lceil(\log(N)-l)/2\rceil}}\right) \quad (26)$$

For total switch instruction memory capacitance, we compute a weighted sum of the switch instruction memory uses.

$$C_{mcimem} = \sum_{l=\log(S)}^{\log(N)} \left(\frac{N}{2^l}\right) \left(\frac{c}{C_t}\right) \left(2^l\right)^{p_t} \left(C_t 2^{l(p-p_t)}\right) \times C_{smem}\left(3, C_t\left(2^l\right)^{p-p_t}\right) \quad (27)$$

Putting this together with the PE capacitance:

$$C_{mc} = NC_{pe} + C_{mcwire} + C_{mcimem} \quad (28)$$

Before dealing with coordination overhead, we can examine how multicontext energy compares to spatial energy assuming there is negligible overhead for coordination (Fig. 6). The multicontext designs shown here are normalized to the energy of the spatial design and achieve enough savings to drop below the sequential energy even for the $W = 64$ design (*c.f.* Fig. 4). At $S = C_t = 1$ and $p_t = p$ the multicontext design is larger than the spatial design (ratio above 2) due to memories and the PE structure in the multicontext case. We see that sequentializing the leaf alone (S) has little effect reducing energy since we still pay all the energy for the network above the sequential leaf. *Interconnect sequentialization, both through C_t and p_t, is effective at reducing energy.* These directly attack the larger area that goes into switches and wiring that is driving the wire lengths in the spatial design when $p > 0.5$.

6.3 Coordination

For multicontext evaluation, ordering of node evaluation in the graph is also an issue. For the fully sequential design, we can sequence the nodes topologically within the energy framework we've described. For the fully spatial design, all edges have their own physical wires, allowing LUTs to evaluate and send their results as dictated by precedence without coordination between the LUTs. However, for the multicontext design, we must control when graph nodes are evaluated on each physical PE and when an edge is routed on a shared wire. Because of precedence constraints, we cannot simply divide nodes by the PE sharing factor, S, and evaluate N/S nodes on each of S cycles.

To illustrate, consider the case of a depth 2 graph on an $S = C_t = 2$, $P_t = p$ design. For some graphs, we will be lucky such that (1) half the nodes are at depth 1 from the inputs and half at are depth 2, (2) each PE is assigned one node at depth 1 and one at depth 2, (3) we can route the outputs of the first half of the PEs through the network with exactly half the wire links. In this case, we can read the first configuration, evaluate the first half of the nodes, route their outputs through the network, read the second configuration from local memories, then evaluate the second half of the nodes and route their outputs. However, (1) the simple Rent's Rule spatial bisection we took to get p did not consider dividing wires and PEs by evaluation time, (2) we can get a pair of nodes at a PE that are the same depth from the input, and (3) there are designs where significantly more than half the nodes are at depth 2. That is, in the general case, the configuration memories may need to change at more than two different times. While each configuration memory only needs to provide one of two values, some PEs and interconnect links may require both to evaluate depth one nodes, while others may require both to evaluate depths two nodes. This means we can change configurations at any of four different times.

A depth D graph could need to switch at any of $D \times \max\left(S, C_t N^{p-p_t}\right)$ times. The simplest way to handle this would be to give every PE and switch a memory of this depth and globally clock the design. However, this would add substantial energy by (1) making the design larger and hence the wires longer, (2) making the memory energy more expensive, (3) adding substantial energy for clocking.

6.4 Asynchronous

A simple way to avoid making the memories larger is to allow the switches to operate asynchronously based on data presence [2]. The mapping from a spatial design to an asynchronous, heterogeneous multicontext design is a way to use the minimum instruction memory depths identified in Sec. 6.2. The switch instruction tells each switch which input it should wait upon and route next, so the switch only acts and uses energy as data becomes available.

The downside of asynchronous control is that we must route a handshake and acknowledgment with every data signal. This means all wiring channels have three times as many wires; for the asynchronous version of $MCWires$ (Eq. 23), we must multiply the channel widths by 3. Each asynchronous link makes four transitions to transfer data rather than one, meaning, in addition to the longer wires, we must account for more switching events in the asynchronous C_{mcwire} (Eq. 26). Furthermore, we expect the asynchronous switches to be larger than the synchronous switches; driven by the 3× wiring, we assume they are 3× the area.

6.5 Synchronous

As noted above (Sec. 6.3) it would be expensive to simply make all memories deeper and clock every switch and PE

at the worst-case sequentialization. Ideally, we would map the design directly for multicontext evaluation, carefully selecting which nodes can share a PE and scheduling PEs and wires to fit into a minimum number of cycles. However, for the reasons previously noted, without doing that mapping, we cannot guarantee how it will turn out. Here we describe synchronous evaluation strategies to minimize the memory depth expansion and estimate the likely range of values.

6.5.1 Heterogeneous Memory Clocking

We can clock the different tree levels proportional to their original multicontext depth ($C_t 2^{l(p-p_t)}$), meaning the tree switches closer to the root are clocked more often than the tree switches closer to the leaves. To do this, we evaluate the tree once for each leaf serialization C_t. That means, we evaluate the top of the tree N^{p-p_t} times, the next level $\left(\frac{N}{2}\right)^{p-p_t}$ times, and so forth until we clock the leaf once. At adjacent tree levels, the higher level will switch either once or twice for each time the lower level switch changes. At the level boundaries, the clock is divided where needed so that the lower level switches at half the rate of the upper level.

Fig. 7 shows the basic operation for heterogeneous clocking. Here, level $i+1$ is sequentialized at the same rate as i, which is sequentialized twice as heavily as level $i-1$. As a result, we must clock levels $i+1$ and i twice for every time we clock level $i-1$. To realize the connection set $A \rightarrow p0$, $C \rightarrow B$, $p0 \rightarrow C$, $D \rightarrow A$, $p1 \rightarrow D$, we pass $A \rightarrow p0$, $C \rightarrow B$, $p0 \rightarrow C$, on the first cycle and $D \rightarrow A$, $p1 \rightarrow D$ on the second. If $i-1$ is, itself, sequentialized, we allow it to switch on the second cycle.

To estimate clock energy, we must estimate the total capacitance switched across all the clocking required.

$$C_{clk} = CSF \cdot C_t \times (C_{tdist} + C_{sdist}) \tag{29}$$

CSF account for the fact that the clock switches multiple times per evaluation. We assume a 2-phase clock that switches up and down each evaluation, so use $CSF = 4$. C_{tdist} deals with the wiring to distribute clock to each tree level and is weighted by the number of times the tree level is switched.

$$\begin{aligned} C_{tdist} &= \sum \left(2^l\right)^{p-p_t} \times L_{mcside} \left(\frac{2^{l/2}}{2\sqrt{N}}\right) \left(\frac{N}{2^l}\right) \\ &= \frac{L_{mcside}}{2} \sum 2^{l(p-p_t)} \left(\frac{\sqrt{N}}{2^{l/2}}\right) \end{aligned}$$

C_{sdist} deals with the wiring to distribute the clock to the individual switch memories within a tree level.

$$C_{sdist} = \sum_{l=\log(S)}^{l=\log(N)} \left(2^l\right)^{p-p_t} \left(\frac{3}{2}\right) N_{sw}(l) \sqrt{A_{sw\_w\_mem}(l)} \tag{30}$$

The $\frac{3}{2}$ arises from H-Tree distribution over the switch area. $N_{sw}(l)$ is the total number of switches at level l in all subtrees:

$$N_{sw}(l) = \left(\frac{N}{2^l}\right) \left(\frac{c}{C_{txt}}\right) \left(2^l\right)^{p_t} \tag{31}$$

The area for a switch with its memory at a given level is:

$$A_{sw\_w\_mem(l)} = \sqrt{\left(A_{1:1} + A_{imem}\left(3, C_{txt}\left(2^l\right)^{p-p_t}\right)\right)} \tag{32}$$

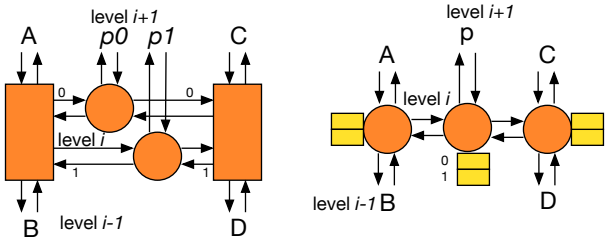


Figure 7: Heterogeneous Clocking Example

6.5.2 Context Switching

With the heterogeneous upper-level switching scheme, we route the entire interconnection pattern with C_t leaf switching cycles. Precedence constraints may prevent us from performing this route once; it may be necessary to route signals at different times. As a general formulation, we introduce a context factor, CF, as a multiplier on C_t, then define:

$$C'_t = CF \times C_t \tag{33}$$

Our evaluation strategy can then be to make the memories CF larger and clock the leaf levels CF as many times. In estimation, we replace C_t by C'_t when calculating memory depth and evaluation cycles (*e.g.*, Eq. 29). In the best case, $D = 1$, all results can be routed simultaneously and $CF = 1$. This may also be achievable with good packing of nodes into PEs and scheduling of wires. In the worst case, $CF = D$, and we must route the network at every graph depth. Modern designs are often pipelined to be shallow, resulting in a small D. As we will see the best values of C_t are around 8, so a typical operating regime will have $C_t \geq D$. In practice, when $C_t \geq D$, it should be possible to keep $CF \approx 2$. That is, we need at least C_t leaf routes to handle the leaf interconnect serialization, which is also enough to handle node serialization when $C_t \geq S$. Since $C_t \geq D$, we will have depths that need more than one context for routing, C_{t_i}. The routing required for nodes at a particular depth may not perfectly fill an integral number of contexts, meaning we may end up with one partially filled context per depth. So, we have:

$$C'_t = \sum_{i=1}^{D} C_{t_i} \leq C_t + D \leq 2C_t \tag{34}$$

Local routing hotspots could make this worse. In the following we use $CF = 4$ as a likely conservative bound, expecting realistic cases to have a CF somewhere between 1 and 4.

Note that the total switching energy on the wires and memories can remain proportional to the traffic that needs to be routed; it does not need to be multiplied by this CF factor. We must arrange for the switch configurations to change *only* when a different edge must actually be routed. To achieve this, we program the unused context memory slots to return the same value as the preceding cycle so they do not switch their output causing the switch to select a different input. Since we use sequential memories for instructions, selection of sequential memory locations is based on a shift register, and we avoid paying the cost of address line toggles for these extra memory contexts. We do have to pay for the longer wires associated with these memories, which we model by making the memories C'_t deep.

Fig. 8 compares the various coordination strategies, all normalized to the fully spatial case. For designs smaller than

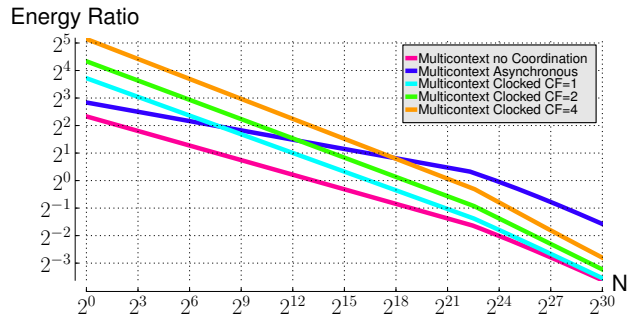


Figure 8: Energy Ratio to Spatial for $p = 0.8$ at $p_t = 0.49$, $C_t = S = 1$ and $M_{layers} = 8$

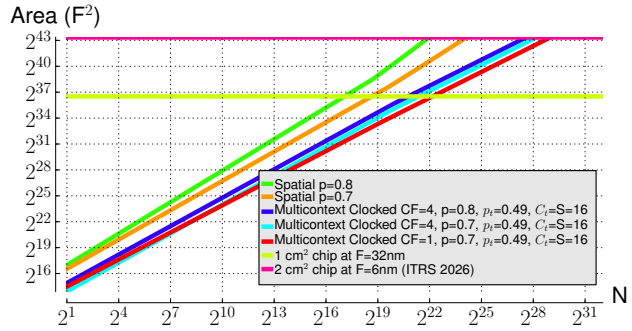


Figure 9: Area for $M_{layers} = 8$

32K 4-LUTs, the coordination energy makes all multicontext designs more expensive than spatial designs. For larger designs, the multicontext designs can be smaller. However, the asynchronous requires more energy than spatial up to 16 million 4-LUTs, by which point, even a context factor, CF, of 4 achieves less energy than the asynchronous design.

6.6 Comparisons

To understand the size of designs viable over the range of the ITRS 2012 roadmap [1], Fig. 9 plots the area of spatial and multicontext designs and marks the capacity of a moderate (1cm^2) chip at current 32nm technology and a large (2cm^2) chip at the 6nm technology predicted for 2026. This establishes that we could soon build multicontext designs with over 2 million 4-LUTs and establishes the potential viability of multicontext designs up to 512M LUTs.

Fig. 10 varies the physical tree Rent Exponent, p_t, for 100 million 4-LUT designs. This shows that p_t around 0.5 minimizes energy. *This is consistent with the asymptotic results that tell us that we need $p_t < 0.5$ to avoid wirelengths that grow faster than $\sqrt{N}$.* These absolute results shows that the binary tree ($p_t = 0$) is more expensive than a $p_t = 0.49$ tree and, more generally, the energy increases as the Rent Exponent decreases from around $p_t = 0.49$. Below $p_t = 0.5$, we can become switch and memory rather than wire dominated, such that additional interconnect sharing does not significantly reduce wire lengths. However, this sharing does increase instruction memory energy.

Fig. 11 shows how the various energy components contribute to total switching energy. Even at $p_t = 0.49$, wire energy dominates with no further sequentialization ($S = C_t = 1$). As the context and processor sharing increases,

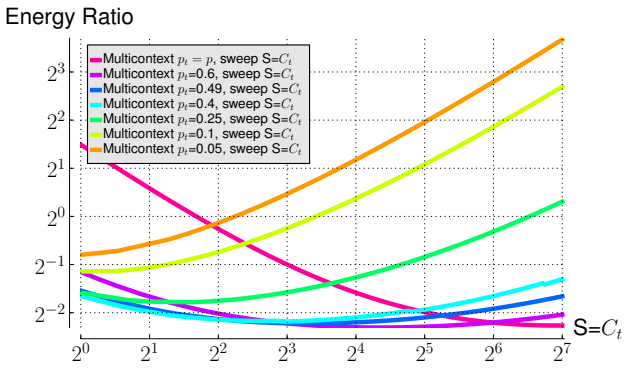


Figure 10: p_t Energy Comparison for $N = 10^8$, $p = 0.8$ and $M_{layers} = 8$ assuming $CF = 4$

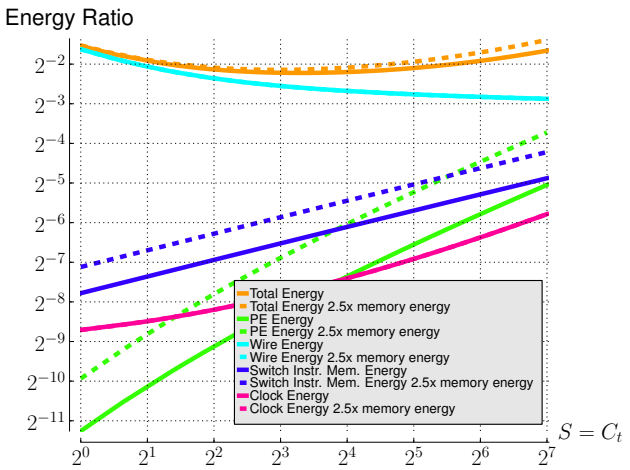


Figure 11: Energy Components for $N = 10^8$, $p = 0.8$, $p_t = 0.49$, and $M_{layers} = 8$ assuming $CF = 4$

the switch instruction memory and PE energy increase to the point where PE energy costs grow faster than the wiring energy savings, inducing an energy minimum around $C_t = S = 16$. Fig. 11 also shows the impact of memory being relatively more expensive. Since the minimum energy point is wire dominated, the impact of larger memories has a smaller effect at the energy-minimizing multicontext point than it does on a fully sequential design (*c.f.* Fig. 3).

The benefits of using a $p_t < 0.5$ tree increase with p for $p > 0.5$ as shown in Fig. 12. These savings is enough to match $W = 64$ at $p = 0.8$ for 512M 4-LUT designs. Nonetheless, at even higher p's, the multicontext design will require larger designs before it will achieve lower energy than the wide word sequential implementation—sizes that are not feasible in the next decade.

7. DISCUSSION AND OPEN ISSUES

Throughout this paper, we have assumed full activity for the communication links between nodes. This is an upper bound for absolute energy. However, activity will have a differential effect on the designs explored. The fully spatial design will directly benefit from a low activity edge—the output of a node drives a single wire, and if the node doesn't change values on a cycle, the wire does not switch.

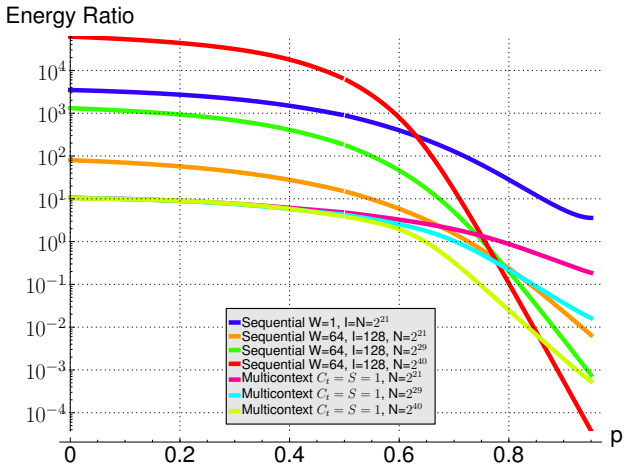


Figure 12: Energy Ratio to Spatial for $p_t = \min(0.49, p)$, and $M_{layers} = 8$ assuming $CF = 4$

A multicontext design that is sharing a wire between two different edges may switch twice on a cycle even if neither edge switches; if one node is non-active at 0 and the other at 1, when they are time multiplexed onto the same physical link, we pay two transitions on that cycle. As a result, we expect this effect to increase the advantage of spatial implementations over sequential implementations, and we expect this effect to reduce the benefits of multicontext evaluation over spatial compared to what we have shown here.

In some cases, it will be possible to exploit wordwidth optimizations for multicontext evaluation. For example, if $S = C_t = W$ and $p = p_t$ we may need no more than a single configuration like the spatial design. Even with $p_t < p$, the memories in the interconnect can be up to a factor of W smaller than in the $W = 1$ case assumed in Sec. 6. Additional development is needed to treat this properly.

8. CONCLUSIONS

With energy-limited technology, architectures that minimize energy will maximize the computational performance offered at a given power-envelope or energy budget. Communication locality, which we can quantify with Rent's Rule, is an important characteristics of computational tasks that determines how efficiently the task can be implemented. When the Rent Exponent, p, is less than 0.75, FPGAs use less energy than processors, even after considering SIMD word optimizations and tight loops that share instructions across operations. For the larger p designs, multicontext FPGAs can reduce energy compared to spatial designs (single-context FPGAs) by sharing interconnect wires to limit wire area. This allows them to reduce the energy requirements below sequential (processor) designs, at least up to $p = 0.8$, for the design sizes feasible in the next decade.

9. ACKNOWLEDGMENTS

Randy Huang and Andrew Sutherland provided valuable feedback on drafts of this work. This research was funded in part by National Science Foundation grant CCF-0904577 and 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 National Science Foundation, the Department of Defense, or the U.S. Government.

10. REFERENCES

[1] International technology roadmap for semiconductors. <http://www.itrs.net/Links/2012ITRS/Home2012.htm>, 2012. 3.2, 5, 6.6

[2] A. DeHon. Location, Location, Location—The Role of Spatial Locality in Asymptotic Energy Minimization. In *FPGA*, pages 137–142, 2013. 1, 2, 2.3, 4, 4, 5, 6, 6.2, 6.4

[3] W. E. Donath. Placement and average interconnection lengths of computer logic. *IEEE Trans. Circuits Syst.*, 26(4):272–277, April 1979. 2.1

[4] H. Esmaeilzadeh, E. Blem, R. S. Amant, K. Sankaralingam, and D. Burger. Dark silicon and the end of multicore scaling. In *ISCA*, pages 365–376, 2011. 1

[5] K. Fujiyoshi, Y. Kajitani, and H. Niitsu. Design of minimum and uniform bipartites for optimum connection blocks of FPGA. *IEEE Trans. Computer-Aided Design*, 16(11):1377–1383, Nov. 1997. 5

[6] R. I. Greenberg and C. E. Leiserson. *Randomness in Computation*, volume 5 of *Advances in Computing Research*, chapter Randomized Routing on Fat-Trees. JAI Press, 1988. Earlier MIT/LCS/TM-307. 2.2

[7] M. Hutton. Interconnect prediction for programmable logic devices. In *Proc. SLIP*, pages 125–131, 2001. 2.1

[8] B. S. Landman and R. L. Russo. On pin versus block relationship for partitions of logic circuits. *IEEE Trans. Comput.*, 20:1469–1479, 1971. 2.1

[9] E. A. Lee and D. G. Messerschmitt. Synchronous data flow. *Proc. IEEE*, 75(9):1235–1245, Sept. 1987. 3.1

[10] C. E. Leiserson. Fat-trees: Universal networks for hardware efficient supercomputing. *IEEE Trans. Comput.*, C-34(10):892–901, Oct. 1985. 2.2

[11] G. Lemieux, E. Lee, M. Tom, and A. Yu. Directional and single-driver wires in FPGA interconnect. In *ICFPT*, pages 41–48, 2004. 2.2

[12] D. Lewis, V. Betz, D. Jefferson, A. Lee, C. Lane, P. Leventis, S. Marquardt, C. McClintock, B. Pedersen, G. Powell, S. Reddy, C. Wysocki, R. Cliff, and J. Rose. The Stratix routing and logic architecture. In *FPGA*, pages 12–20, 2003. 2.2

[13] 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*, 24(11):1712–1724, Nov. 2005. 4

[14] 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

[15] 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. 4