Energy Efficiency of Full Pipelining: A Case Study for Matrix Multiplication

Peipei Zhou,* Hyunseok Park,* Zhenman Fang,* Jason Cong,* André DeHon†
* UCLA, Dept. of Computer Science, Los Angeles, CA 90095
† University of Pennsylvania, Dept. of ESE, 200 S. 33rd Street, Philadelphia, PA 19104
Email: {memoryzpp,parkhyu,zhenman,cong}@cs.ucla.edu, andre@ieee.org

***Abstract*—Customized pipeline designs that minimize the pipeline initiation interval (II) maximize the throughput of FPGA accelerators designed with high-level synthesis (HLS). What is the impact of minimizing II on energy efficiency? Using a matrix-multiply accelerator, we show that matrix multiplies with II>1 can sometimes reduce dynamic energy below II=1 due to interconnect savings, but II=1 always achieves energy close to the minimum. We also identify sources of inefficient mapping in the commercial tool flow.**

1. Introduction

To meet the ever-increasing demand for high computation performance and energy efficiency, numerous commodity acceleration platforms have been proposed and developed, including the well-known many integrated cores (MICs, or Intel Xeon Phi processors), graphical processing units (GPUs), field-programmable gate arrays (FPGAs), and application-specific integrated circuits (ASICs) [1, 2, 3]. The utilization wall [4] has stimulated interest in FPGAs, since FPGAs provide both low power and customization capability to accelerate different applications (more flexible than ASICs).

Compared to the general-purpose MICs and GPUs, FPGAs allow designers to look beyond parallelization and customize accelerators. The customized pipeline design has been one of the most successful and widely used optimizations to improve the performance of FPGA accelerators [5, 6, 7]. At the same time, the recent success of commercial HLS tools like Xilinx Vivado HLS [8] has made design space exploration for a customized pipeline easier compared to conventional register transfer level (RTL) designs.

Among various tunable parameters in such pipeline customization, the pipeline initiation interval (II)—which is defined as the number of cycles between two consecutive pipeline iterations [9]—is one of the most important customization parameters since it reflects the throughput of the pipeline design and has been widely studied (e.g. [5, 6, 7]). Most prior studies, except [5], have focused on minimizing the pipeline II so as to maximize the throughput of FPGA accelerators. Meanwhile, there are also examples [6] indicating that smaller pipeline II can reduce the energy consumption of FPGA logic gates. Motivated by these studies, this paper begins to explore a key question: *Does a customized pipeline with the minimum II always minimize energy? If not, how does the pipeline II affect the energy consumption?*

To get initial insight into this question, we focus on the classical matrix-multiplication algorithm specified in C for HLS. We build an analytical model of the energy consumption for the kernel as a function of matrix dimension, N, and pipeline II, including the effects of computational, interconnect, memory, and leakage energy. This allows us to identify how the energy should scale with problem size and II. We synthesize the HLS kernel with Vivado HLS and fit constants within the analytical model. Along the way, we identify sources of inefficiency in the commercial tool flow that can cause the HLS solution to diverge from the ideal scaling for the matrix-multiply kernel, which include:

1. Missing opportunities for register sharing.
2. Missing opportunities for address generator sharing.
3. Lack of power-gating for unused memory banks.

We find that the logic component of energy remains flat, while memory and leakage components increase with II, but interconnect can decrease with increasing II. Interconnect savings are large enough that we can identify cases where $II > 1$ minimizes energy. Nonetheless, we see that the increasing energy term is modest, such that the $II = 1$ case is always within a few percent for matrix-multiplies that can fit on a single FPGA today. The energy framework identified here should translate to other HLS kernels, but will have different compute and interconnect scaling that should be characterized and better understood in future work.

2. Related Work in Energy Modeling

FPGA energy models have been widely used to provide guidance in design space explorations. Recent studies in [10] developed analytical models to characterize energy consumption of designs ranging from a sequential design (processor) to a spatial design (FPGA), using Rent's rule [11] as a modeling tool. Earlier works [12, 13, 14] employed models to provide in-depth analysis of FPGA power decomposition and the impact of look-up table (LUT) size, cluster size, and segment lengths on power consumption. Recent work introduced FPGA memory models to analyze the effect of the memory architecture (including block size, banking, physical spacing) and parallelism on an application's energy efficiency [15]. While these works present detailed energy models, they do not directly address the microarchitectural structure that results from tuning the II in HLS designs. To the best of our knowledge, this work is the first to model the impact of the pipeline II on energy consumption of FPGA accelerators from a high-level perspective.

3. Matrix-Multiplication Kernel

```
void matrix_multiply(float a[N][N],
                     float b[N][N], float c[N][N]) {
  int i, j, k, p;
  k_loop:  for(k = 0; k < N; k++) {
    i_loop:  for(i = 0; i < N; i++) {
    //i_loop PIPELINE II = II_i
      p_loop:  for(p = 0; p < N; p += N/II_i) {
      #pragma HLS PIPELINE II = 1
        j_loop:  for(j = 0; j < N/II_i; j++) {
        #pragma HLS UNROLL
        c[i][p+j] += a[i][k] * b[k][p+j];
} } } } }
```

Listing 1: Pseudo code of square matrix multiplication

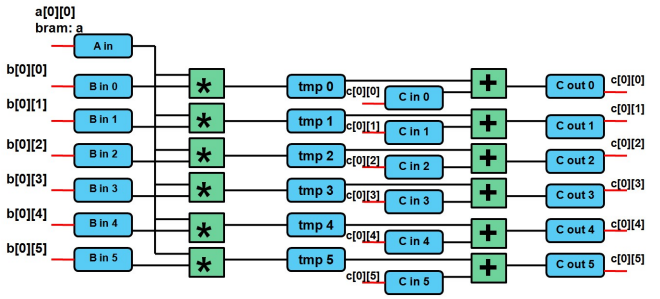

Figure 1: Architecture for N = 6, II = 1

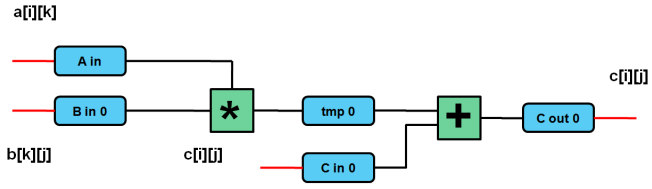

Figure 2: Architecture for N = 6, II = N = 6

To make it easier to understand, we use square $N \times N$ matrix-multiplication as an example to demonstrate the energy model for mapping applications with perfectly shareable processing elements (PEs) onto a commodity FPGA. As shown in Listing 1, to pipeline the i_loop with a specific II (e.g., II_i = 1), we change the increment value of the p index in the p_loop and apply the HLS PIPELINE pragma. Consequently, the inner-most j_loop will be unrolled automatically (HLS UNROLL pragma is shown to illustrate this unroll but not needed explicitly). In the j_loop, $\frac{N}{II\_i}$ elements within one row of the c matrix, $c[i][p]$, $c[i][p+1]$, ... , $c[i][p+\frac{N}{II\_i}-1]$ are updated using $a[i][k]$ and $\frac{N}{II\_i}$ elements within one row of b matrix, $b[k][p]$, $b[k][p+1]$, ... , $b[i][p+\frac{N}{II\_i}-1]$. The PIPELINE II of i_loop (i.e., II_i, for simplicity, we will directly use II for II_i in the rest of the paper) determines the throughput, resource utilization, and dynamic energy to execute this matrix multiplication kernel. Please note that the most straightforward matrix-multiply algorithm would not have had a separate p_loop and j_loop. However, when we initially used that description, Vivado HLS did not share the local registers optimally when we increase II. Fig. 1 and Fig. 2 show the architecture for N = 6, II = 1 and II = N = 6 respectively when registers are perfectly shared.

The resources and cycles to finish the matrix multiplication kernel can be generalized in terms of problem size N and PIPELINE II of the i_loop.

1. There are $\frac{N}{II}$ multiplier(s) and $\frac{N}{II}$ adder(s), and each computes II elements within one row. The number of B or C input/output registers and temporary registers between multipliers and adders is $\frac{N}{II}$.
2. There are $\frac{N}{II}$ independent memory bank(s) for the b matrix, each with II column(s). It is the same for the c matrix. Only one memory bank is needed for the a matrix since $a[i][k]$ is shared within one i_loop iteration.
3. The number of cycles to finish the kernel is $N^2 \times II$.

TABLE 1: HLS reported resource usage for multiplier and adder under different IIs, N=24

II	1	2	3	4	6	8	12	24
DSP	120	60	40	30	20	15	10	5
FF	8520	4260	2840	2130	1420	1065	710	355
LUT	8376	4188	2792	2094	1396	1047	698	349

4. Energy Model

4.1. Computation Energy

The computation energy includes arithmetic energy (multiply-add operations) and register energy for holding the inputs, outputs, and temporaries. We can perfectly share these PEs and registers by factor of II, making the total PE and register energy consumption:

$$E_{compute} \propto \frac{N}{II} \times \left(N^2 \times II\right) = N^3 \quad (1)$$

For Xilinx 7 Series FPGAs, each multiply-add needs three DSP48E [16] for the floating-point multiplier and two DSP48E for the adder along with a fixed number of LUTs and FFs. Table 1 shows the resource usage for multipliers and adders decreasing as $\frac{1}{II}$ since PEs are perfectly shared.

4.2. Memory Energy

In order to fully pipeline the matrix multiplication, each PE needs to access each column of the b and c matrix simultaneously. HLS provides comprehensive partition pragmas [8] to easily partition an array into individual memory banks. For example, we use the complete partition pragma to partition b along the column direction. Each column, $b[..][N]$, becomes an individual memory block.

As II increases, the number of simultaneous accesses to the b matrix decreases, which means more columns can be placed in the same memory bank with size of $N \times II$. In this design, cyclic partition pragma is applied to the b and c matricies to automatically split the memory along the column direction in $\frac{N}{II}$ equally sized blocks interleaving the original array.

In general, there are $\frac{N}{II}$ b or c banks; within each, II columns of data are stored. We need to consider the total memory energy when accessing these banks. In total, there are N^3 b memory reads, N^3 c memory reads and writes, and N^2 a memory reads, which we could safely ignore when N is large enough. Each memory access is from a logic memory bank with size of $N \times II$. On Xilinx 7 series FPGAs, all the logic memory banks are constructed using the embedded BRAM18K memory banks on the chip [17].

If we activate a single BRAM for each read within a bank, the total energy reading from BRAMs is constant at:

$$E_{mem} \propto \frac{N}{II} \times \left(N^2 \times II\right) = N^3 \quad (2)$$

When the logical memory bank size is larger than physical BRAM18K bank size, it needs to be constructed using multiple BRAM18K banks, and the area of each logical memory bank increases. This impacts the wiring as we see in the next section.

4.3. Interconnect Wire Energy

The wire energy can be decomposed into wires within PEs and wires connecting PEs and memory. Wiring within the PE is fixed and will scale with the compute energy.

$$E_{wire.in.pe} \propto \frac{N}{II} \times \left(N^2 \times II\right) = N^3 \quad (3)$$

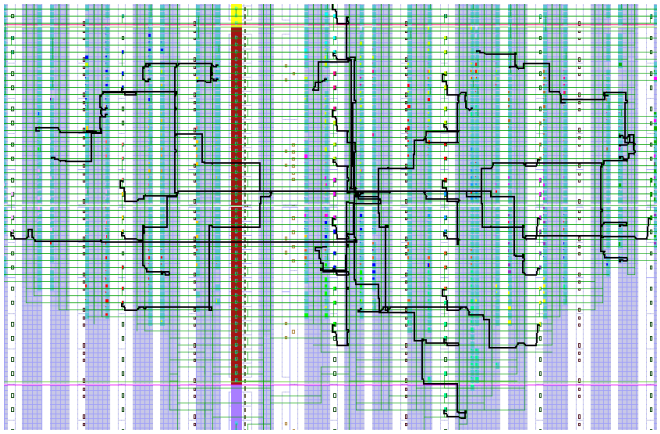

Figure 3: Routing of broadcasting a[i][k] to all 24 multipliers, N = 24, II = 1

Wire transferring broadcast data. In this matrix-multiply algorithm, broadcast wires must transfer $a[i][k]$ from the memory bank storing the a matrix to the multipliers as shown in Fig. 3. The BRAM blocks storing the a matrix are close to the input register, *Ain*, and close to one multiplier. This broadcast should take energy proportional to the total area of all the PEs it is feeding.[1] As II increases, the total PE area scales as $\frac{N}{II}$, until we can no longer fit $N \times II$ elements of the b matrix into a single BRAM. Thus the total energy for broadcasting $a[i][k]$ is:

$$E_{wire.share.A} \propto \frac{N}{II} \times N^2 = \frac{N^3}{II} \tag{4}$$

After $N \times II > BRAM18K$, this scaling changes in interesting ways. At this point, the total layout area for all the PEs becomes dictated by BRAMs not DSPs, and the area does not change with II. If we must broadcast to all the BRAMs, this means the broadcast energy does not shrink with II.

$$E_{wire.share.A} \propto N^2 \times N^2 = N^4 \tag{5}$$

However, we really only need to broadcast to a few BRAMs within a PE, allowing the broadcast energy to continue to shrink with increasing II. Current synthesis tools do not exploit this opportunity.

Wire transferring private data. When the logical memory bank size $N \times II$ is smaller than size of the physical BRAM18K bank (18432 bits, 576 floating-point numbers), the wiring between the private c and b matrix memory banks and the PE logic is constant, so the total energy for wiring also scales proportionally, independent of II:

$$E_{wire.priv.B,C} \propto \frac{N}{II} \times \left(N^2 \times II\right) = N^3 \tag{6}$$

When the logical memory bank size is larger than physical BRAM18K bank size, the wiring between the private b and c memory banks and the PE logic also grows as the square root of the memory capacity or $\sqrt{N \times II}$. Thus, the total energy routing memory is

$$\begin{aligned} E_{wire.priv.B,C} &\propto \frac{N}{II} \times \left(N^2 \times II\right) \times \sqrt{N \times II} \\ &\propto N^{3.5} II^{0.5} \end{aligned} \tag{7}$$

1. [18] shows the H-tree layout has linear layout area, which implies linear wirelength in the area, which in turn implies linear energy.

4.4. Leakage

During the computation, the FPGA will also leak energy proportional to the time for the computation and the resources that leak during the computation. If we put nothing else on the FPGA and use a fixed size FPGA that does not offer any power gating for unused components, leakage increases with runtime and hence II:

$$E_{leak} \propto N^2 \times II \times E_{FPGA\_leak} \tag{8}$$

However, if we use a design with perfect power gating of unused components, the leakage should scale with the utilized logic. For the case where $N \times II < BRAM18K$:

$$E_{leak} \propto \frac{N}{II} \times N^2 \times II = N^3 \tag{9}$$

If we exploit the smaller resources of the $II > 1$ designs to use a smaller FPGA, we can get some of the effects of Eq. 9. Similarly, if we exploit the smaller resource utilization of the $II > 1$ to put additional logic onto the FPGA that fills the resources unused by the matrix-multiply, the leakage attributable to the multiply should scale closer to Eq. 9.

4.5. Total Energy

Putting all the energy components together and assuming perfect power gating (Eq. 9), we have total energy as the follows:

$$\begin{aligned} E_{total} &= E_{compute} + E_{memory} + E_{wire} + E_{leak} \\ &= \begin{cases} N^3 \left(c1 + \frac{c2}{II}\right), \\ \text{if } N \times II \leq \text{BRAM18K} \\ N^3 \left(c3 + c4 \times N + c5 \times II^{0.5}\right), \\ \text{if } N \times II > \text{BRAM18K} \end{cases} \end{aligned} \tag{10}$$

5. Results

For small N, when the design is not memory dominated, we can expect to see decreasing energy until $N \times II = BRAM18K$ driven by broadcast wiring energy. Beyond that, we expect to see energy increase with II due to wiring energy between BRAMs and computation within a PE.

We mapped the HLS designs to an Virtix-7 XC7VX485T using Vivado 2015.1.5. We simulated each mapped design in Vivado with random a and b matricies. We then used the Switching Activity Interchange format (SAIF) file generated from post-implementation simulation to estimate the energy required by the mapped designs. From the mapped designs, we used linear regression fit to determine the constants $c1$–$c5$ in Eq. 10.

Fig. 4 shows how the energy components scale with II from the Vivado mapped designs along with the total energy model from our fit model. We can see the mostly flat DSP and logic energies that match the analytic description. We also see that the interconnect energy and the overall energy drop with increasing to $II = 16$ where $N \times II = BRAM18K$. We see the interconnect energy grow after that as expected. However, we also see that BRAM energy, rather than remaining flat, increases with II after $II = 16$. Here, Vivado mapped designs are unnecessarily activating all of the BRAMs, not just the BRAM that holds the data needed on each cycle. This makes the total design energy unreasonably high for large II. It should be possible to avoid activating the unused BRAMs as illustrated in [15, 19]. Making the perfect power gating assumption, we see that energy is minimized at $II = 8$. However, the effect is small and the benefit over $II = 1$ is

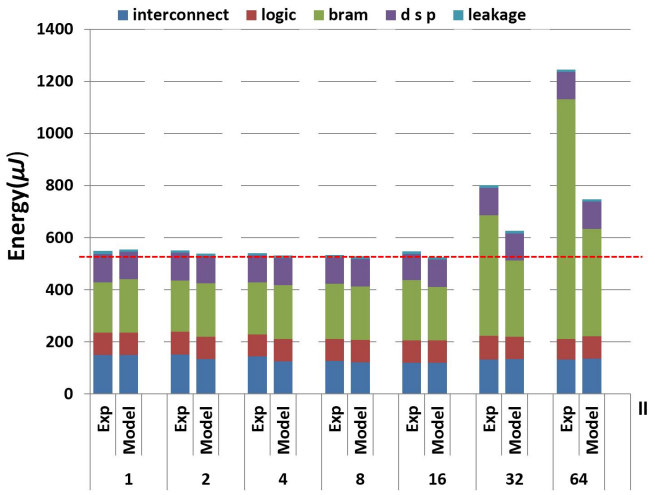

Figure 4: Energy Scaling with II for $N = 64$ Matrix Multiply

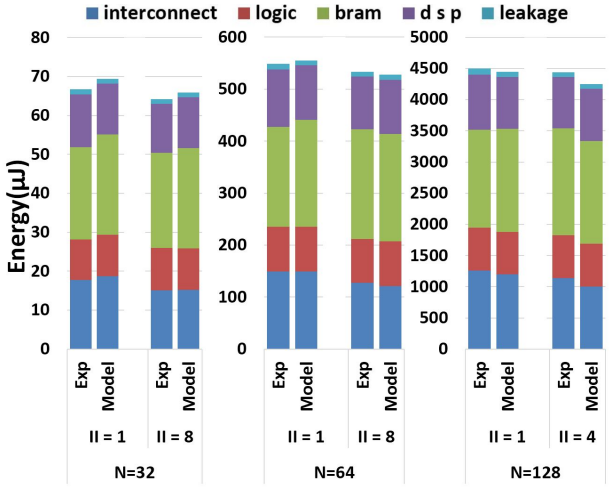

Figure 5: Scaling with N for Matrix Multiply

less than 3%. If we get less than perfect power gating, this effect will easily be dominated by an increase in leakage energy with II.

Fig. 5 shows how energy scales with N, including how this effects the optimal II and our model fit. The II for the minimum energy point decreases with N since larger N means the single BRAM capacity is reached at a lower II. Since all energy components scale as N^3 for the region where $N \times II \leq BRAM18K$, the energy proportions remain the same as N grows.

Note that all the PEs are generating the same addresses for their local b and c memories. Consequently, the design can use a single address generator for all the BRAMs. However, for some values of N and II, Vivado HLS will not share the address generators, resulting in much larger logic energy at those design points. Also, as we pointed out earlier, without adding the p loop in Listing 1, the HLS tool fails to share the registers properly.

6. Conclusion and Future Work

Interconnect energy within our matrix-multiply kernel is minimized for an II that is typically greater than one. With efficient power gating or alternate use of chip resources, this can lead to minimum total energy at a point other than the fully pipelined, II=1 point. Nonetheless, the effect is small and the fully pipelined design often uses the least energy in practice, both due to leakage and other discrete and non-ideal scaling effects.

The energy modeling framework illustrated here should be adaptable to other kernels. However, there is good reason to believe that kernels will differ in how they scale in key areas. We expect interconnect energy to scale differently for other tasks or even implementations of the same task. For example, using the systolic-array implementation of matrix multiply [20], one may see different scaling. Our matrix-multiply kernel had near perfect sharing of logic as II increased, which will not be the case for less regular tasks. Consequently, it will be useful to characterize how these components scale for other tasks and develop a suitably parameterized energy model that can be adapted to various tasks characteristics. Ultimately, we hope model generation can be automated and provide high-level guidance for designers. As illustrated here, these models may also help to identify inefficiencies in current mapping tools that should be addressed to achieve energy efficient designs.

Acknowledgments

This work is partially supported by the Center for Domain-Specific Computing under the Intel Award 20134321 and NSF Award CCF-1436827. It is also supported in part by C-FAR, one of the six centers of STARnet, a Semiconductor Research Corporation program sponsored by MARCO and DARPA.

References

[1] A. Duran and M. Klemm, "The intel many integrated core architecture," in *HPCS*, July 2012, pp. 365–366.
[2] S. Che *et al.*, "Accelerating compute-intensive applications with GPUs and FPGAs," in *SASP*, June 2008, pp. 101–107.
[3] H. K. Phoon *et al.*, "A highly compatible architecture design for optimum FPGA to structured-ASIC migration," in *ICSE*, Oct 2006, pp. 506–510.
[4] M. Taylor, "Is dark silicon useful? harnessing the four horsemen of the coming dark silicon apocalypse," in *DAC*, June 2012, pp. 1131–1136.
[5] P. Li *et al.*, "Resource-aware throughput optimization for high-level synthesis," in *FPGA*, 2015, pp. 200–209.
[6] J. Cong *et al.*, "A fully pipelined and dynamically composable architecture of cgra," in *FCCM*, May 2014, pp. 9–16.
[7] J. Cong et al., "Automatic memory partitioning and scheduling for throughput and power optimization," *TODAES*, vol. 16, no. 2, pp. 15:1–15:25, Apr 2011.
[8] Xilinx, "Vivado High-Level Synthesis." [Online]. Available: http://www.xilinx.com/products/design-tools/vivado/integration/esl-design/index.htm
[9] M. Lam, "Software pipelining: An effective scheduling technique for vliw machines," in *PLDI*, 1988, pp. 318–328.
[10] A. DeHon, "Fundamental underpinnings of reconfigurable computing architectures," *Proc. IEEE*, vol. 103, no. 3, pp. 355–378, March 2015.
[11] B. Landman and R. L. Russo, "On a pin versus block relationship for partitions of logic graphs," *TC*, vol. C-20, no. 12, pp. 1469–1479, Dec 1971.
[12] K. Poon *et al.*, "A detailed power model for field-programmable gate arrays," *TODAES*, vol. 10, no. 2, pp. 279–302, Apr. 2005.
[13] F. Li *et al.*, "Power modeling and characteristics of field programmable gate arrays," *TCAD*, vol. 24, no. 11, pp. 1712–1724, Nov 2005.
[14] S. Rajavel and A. Akoglu, "An analytical energy model to accelerate FPGA logic architecture investigation," in *ICFPT*, Dec 2011, pp. 1–8.
[15] E. Kadric *et al.*, "Impact of memory architecture on FPGA energy consumption," in *FPGA*, 2015, pp. 146–155.
[16] Xilinx 7 Series DSP48E1 Slice User Guide.
[17] Xilinx 7 Series FPGAs Memory Resources.
[18] C. E. Leiserson, "Area efficient graph layouts (for VLSI)," in *FOCS*, 1980, pp. 270–281.
[19] R. Tessier *et al.*, "Power-efficient RAM mapping algorithms for FPGA embedded memory blocks," *TCAD*, vol. 26, no. 2, pp. 278–290, Feb 2007.
[20] J. wook Jang, S. Choi, and V. K. Prasanna, "Energy-efficient matrix multiplication on FPGAs," *TVLSI*, vol. 13, no. 11, pp. 1305–1319, November 2005.