FPGA High-level Synthesis versus Overlay: Comparisons on Computation Kernels

Colin Yu Lin[‡,*], Zhenghong Jiang[‡,†], Cheng Fu[‡], Hayden Kwok-Hay So[§], and Haigang Yang[‡,¶]
‡ System on Programmable Chip Research Department, Institute of Electronics, Chinese Academy of Sciences
§ University of Hong Kong
¶ Corresponding Author: yanghg@mail.ie.ac.cn

ABSTRACT

To promote FPGA to a wider user community and to increase design productivity, two new design methodologies, namely FPGA high-level synthesis (HLS) and FPGA overlay, are presented to use a high-level design abstraction. To make clear distinguish features of each design methodology, we make an comparison of a state-of-the-art FPGA HLS tool, Vivado HLS, and an FPGA overlay tool, ArchSyn, on two computation intensive kernels, matrix-matrix multiplication and fast Fourier transform.

In the comparison, FPGA overlay shows an overwhelming superiority in computation performance, which is 8X to 39X faster than FPGA HLS. However, FPGA HLS exhibits its advantages in dynamic power consumption metric. It achieves up to 17X lower power consumption than FPGA overlay. Power- and energy-efficiency are another two essential metrics evaluating trade-offs between performance and power consumption. As demonstrated with evaluation results, FPGA overlay is averagely 3.5X better in power-efficiency for FFT kernel, and achieves up to 2 orders of magnitude better energy-efficiency than FPGA HLS.

1. INTRODUCTION

FPGA has been widely used as computation accelerator in many applications for high performance and high power/energy-efficiency, such as large-scale datacenter services [1], and convolutional neural networks [2]. To achieve extreme performance and efficiency, experienced FPGA design engineer analyzes data dependencies and exploits potential computation parallelism from application, and designs cycle-accurate hardware using hardware description languages. Such FPGA design methodology guarantees near-optimal performance [3] and extreme hardware efficiency, but it requires hardware knowledge and skills and limits FPGA users to only hardware design engineers.

To promote FPGA to a wider user community and to increase design productivity, new design methodologies using a high-level design abstraction, such as FPGA high-level synthesis (HLS) and FPGA overlay, are presented recently. FPGA HLS [4, 5] accepts C-like high-level design and performs code transformations and synthesis optimizations to generate synthesizable cycle-accurate RTL. FPGA overlay [6–9] is a coarse-grained design abstraction layer over fine-grained FPGA resources. User design in high-level language will be complied, scheduled and mapped to the pre-implemented overlay architecture. Both FPGA HLS and overlay intend to turn FPGA hardware design and optimization into an automatic EDA process, as a result to improve FPGA usability and design productivity.

The performance and power/energy efficiency are among the most important metrics evaluating FPGA HLS and overlay tools. In this paper, we compare a state-of-the-art commercial FPGA HLS tool, *Vivado High-Level Synthesis* [10], with an academic FPGA overlay tool, *ArchSyn* [11], on two commonly used computation kernels, matrix-matrix multiplication and fast Fourier transform, to discuss the advantages and disadvantages of both design methodologies. Some conclusions are drawn in Table 1.

In the following section, the FPGA HLS and overlay tools under evaluation are briefly introduced. Section 3 introduces our evaluation methodology, and the comparison results are presented in Section 4. We have a discussion of both design methodologies and compare them with traditional HDL-based design methodology in Section 5. And we conclude the paper in the final section.

2. VIVADO HLS & ARCHSYN

2.1 Vivado HLS

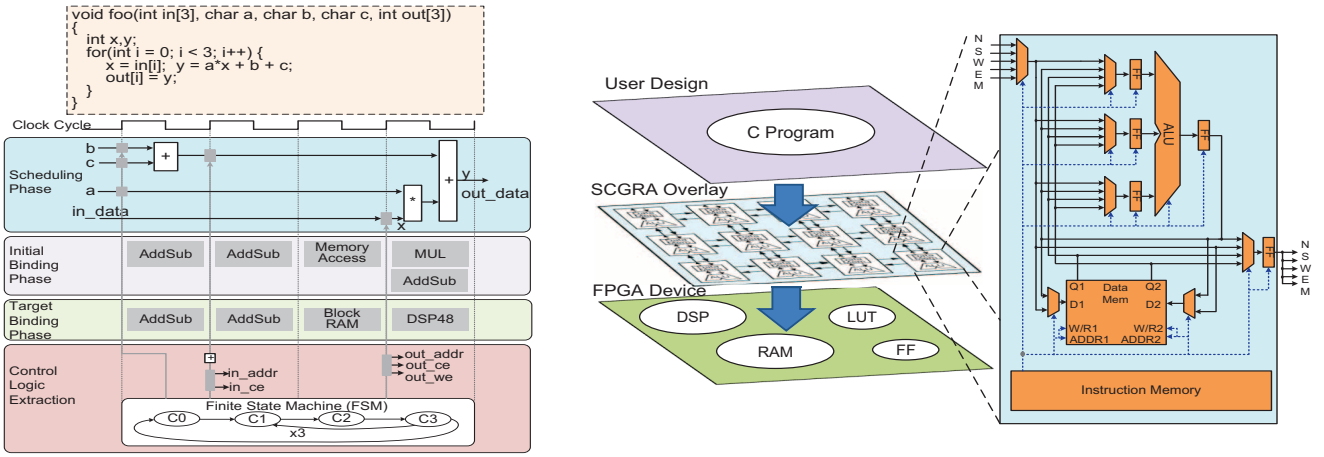
The Xilinx Vivado HLS tool transforms a C specification into a RTL implementation, and flow mainly includes *scheduling*, *binding* and *control logic extraction*. *Scheduling* determines which operation occur during each clock cycle; *binding* determines which hardware resource implements each scheduled operation; and a finite state machine (FSM) is created through *control logic extraction* to sequence the operations in the RTL design.

Figure 1(a) depicts an example in Vivado HLS compilation flow. During the scheduling phase, each operation is dispatched to one specific clock cycle to be executed, and registers are insert to isolate the operations at different cycles. In the initial binding phase, HLS implements the multiplier operation using a combinational multiplier (MUL), and implements both add operations using a combinational adder/subtractor (AddSub). The data fetch operations from arrays are binding as memory access, as arrays would be synthesized into block RAMs by default in Vivado HLS.

*This research is funded by the National Natural Science Foundation of China (61404140, 61271149).

†The first two authors contributed equally to this paper, and should be considered co-first authors.

This work was presented in part at the international symposium on Highly-Efficient Accelerators and Reconfigurable Technologies (HEART2016) Hong Kong, HK, July 25-27, 2016.



(a) Scheduling, Binding, and Control Logic Extraction phases in Vivado HLS compilation flow

(b) ArchSyn using an SCGRA overlay between user design and FPGA device

Figure 1: Overview of Vivado HLS & ArchSyn.

And then in target binding phase, the operations are binding to physical components inside the target FPGA device. Finally a FSM is created to control when the registers store data and the state of any I/O control signals.

2.2 ArchSyn

As illustrated in Figure 1(b), an layer called *Soft Coarse-Grained Reconfigurable Array (SCGRA) Overlay* is proposed by ArchSyn [11]. *SCGRA* is an overlay between user high-level design and target FPGA hardware. Instead of mapping user high-level design to FPGA resources directly as most HLS tools, the process of ArchSyn compilation is divided into two steps.

First, user high-level design is mapped to the *SCGRA* overlay. The *SCGRA* overlay consists of an array of directly connected simple *configurable processing elements (CPEs)*. Each *CPE* performs primitive compute operations according to a small local sequencer at each clock cycle. Data are communicated via multi-hop routing within the direct interconnect network. The *scheduler* schedules each compute operation to execute on a particular *CPE* at a particular cycle. It also determines the communication schedule of the intermediate data among the producing and consuming *CPEs*, optionally buffering them with distributed individual memory along the path.

After mapping user high-level design to the *SCGRA* overlay, ArchSyn implements the overlay on target FPGA. The *SCGRA* overlay takes advantage of reconfigurability of low-level FPGA. Both *CPEs* and interconnect network can be reconfigured at hardware implementation level. The coarse-grained *CPEs* can be pre-implemented or can be tailored according to scheduling results. The *CPE* implementation is optimized towards low-level FPGA architecture.

3. EVALUATION METHODOLOGY

3.1 Evaluation Flow

In the evaluation flow, computational kernel designs in C/C++ are first compiled into RTL designs using Vivado HLS. Then, Vivado Design Suite transforms the RTL implementations into device configuration through *logic synthesis* and *implementation*, and then reports the performance, hardware resource utilization and power consumption metrics. The latest version of Vivado suite, 2015.4, is used.

With the same kernels as inputs, ArchSyn outputs RTL design of the *SCGRA* overlay, which consists of an array of *CPEs*. The scheduler of ArchSyn generates a sequence of instructions, controlling execution of *CPEs* in the *SCGRA* overlay. Except for *the instruction ROM* of each *CPE*, the logic design of the *CPE* array remains unchanged, thus it can be pre-implemented on target FPGA device. Once the scheduler completes the scheduling, the size and content of the *the instruction ROMs* are determined, and *incremental implementation* flow is used to implement the ROM instances into each *CPE*. The Virtex-7 FPGA chip with most memory resource, XC7VX1140T, is chosen as the target device to implement the SCGRA overlay design.

3.2 Computation Kernels

In our evaluation, two computation intensive application kernels are selected, namely matrix-matrix multiplication and fast Fourier transform.

Matrix-matrix multiplication (MxM) calculates the product matrix C of two matrices A and B, $C = A \times B$. If A is an m-by-p matrix and B is a p-by-n matrix, the (i, j)-th entry of C is defined by $c_{ij} = \sum_{k=1}^{p} a_{ik} \cdot b_{kj}$, and the product C is an m-by-n matrix. In this paper, only the multiplication with square matrices are evaluated.

Fast Fourier transform (FFT) is an efficient algorithm to compute the discrete Fourier transform and its inverse, and it is widely used in many applications, including radar, sonar, MPEG audio compression and spectral analysis. The most common FFT algorithm, radix-2 Cooley-Tukey, is used as an evaluation benchmark kernel in this paper.

3.3 Computation Parallelism

To fully utilize the massively parallel compute resources of FPGA, both Vivado HLS and ArchSyn automatically exploit computation parallelism from high-level user design to improve computation performance while maintaining efficient resource utilization and low power consumption. With a given parallelism factor, p, ArchSyn generates an array

```
#define M 50
#define N 50
#define P 50
void matrix( char x[2*M][N],char z[M][P])
{
   row: for(int i=0; i<M; i++) {
   col:    for(int j=0; j<P; j++) {
   #pragma HLS UNROLL factor=F1
              z[i][j]=0;
   product: for(int k=0; k<N; k++) {
   #pragma HLS UNROLL factor=F2
              z[i][j] += x[i][k] * x[k+M][j];
           }
        }
     }
    return;
}
```

(a) MxM

```
#define L 13
#define N  8192
void fft( char in[3*N/2][2] )
{
  for(int k=0; k<L; k++) {
    int M = 1 << k;  int step = 1 << k+1;
    int t   = 0;          int k1    = N/(2*M);
external: for( int j = 0; j < M; j++) {
    #pragma HLS UNROLL factor=F1
internal:   for(int i = j; i < N; i += step) {
    #pragma HLS UNROLL factor=F2
              char wx = in[t+N][0];
              char wy = in[t+N][1];
              char x   = in[i+step/2][0];
              char y   = in[i+step/2][1];
              char tx  = wx * x - wy * y;
              char ty  = wx * y + wy * x;
              in[i+step/2][0] = in[i][0] - tx;
              in[i+step/2][1] = in[i][1] - ty;
              in[i+step/2][0] = in[i][0] + tx;
              in[i+step/2][1] = in[i][1] + ty;
    }
    t += k1;
  }
 }
 return;
}
```

(b) FFT

Figure 2: Loop Unrolling Directives used in Vivado HLS to Exploit Computation Parallelism.

consisting of p CPEs. And the scheduler will schedule compute operations into these CPEs for parallel computation.

In Vivado HLS design, user directives, including *loop unrolling*, *loop pipelining* and *array partitioning*, are required to instruct the compiler to make different trade-offs between performance and FPGA resource utilization. Some other research work has focused on the design space exploration with such directives [13]. In our comparisons, we only use *loop unrolling* to exploit computation parallelism. When a loop is unrolled, more hardware resources are allocated to execute compute operations. Figure 2 illustrates the loop unrolling directives used in the two computation kernels.

In MxM kernel, the inner-most loop with label "*product*" is unrolled in first priority. Only when the inner-most loop is already fully unrolled, the second inner-most loop with label "*col*" will be unrolled. This is because MxM kernel has pretty simple and regular data dependencies, and compiler are easily to exploit computation parallelism from the inner-most loops. Experiment results demonstrate that this unrolling strategy provides the best results in performance.

Figure 2(b) illustrates the loop unrolling directives used in FFT kernel. A higher priority is given to the loop with "*external*" label, which is opposite to MxM. The reason is that FFT kernel has more complex and irregular data dependencies between iterations of the inner-most loop, which makes the unrolling of inner-most loop ineffective. Data dependencies between loop "*external*" are comparably less complex, and this results in better performance with loop "*external*" unrolled.

4. COMPARISON RESULTS

In Figure 3, comparison results of Vivado HLS and ArchSyn on computation latency, dynamic power consumption, power-delay product and energy-delay product are presented. As off-chip memory access bandwidth is an important parameter in evaluation, the same off-chip memory access capability is guaranteed in implementations of Vivado HLS and ArchSyn. Two overlay systems with different input data memory access methods are referenced in the comparison. *Overlay Off-chip* indicates an overlay system that all input data are stored off-chip, and computation elements access the memory to fetch the data during computation. *Overlay On-chip* indicates an overlay system that all input data are stored in on-chip memory resources through FPGA configuration. No extra memory access for input data are needed.

In the comparisons, four different kernels are implemented. *MxM-50* and *MxM-25* indicate that matrices evolving in multiplication are all 50×50 and 25×25 in size, respectively; and *FFT-8K* and *FFT-1K* indicate FFT kernels with input length of 8,192 and 1,024, respectively. For MxM kernels, the unroll factors/CPE numbers are 1, 10, 25, 50 and 100; and for FFT kernels, they are 1, 8, 16, 32 and 64[1]. To clearly show all results in different computation parallelism and the improvement trends, both axes are in logarithmic scale.

4.1 Computation Latency

Computation latency is the time intervals between when input data starts and when all computed results are output. As computation accelerator, computation latency is the essential metric to valuate FPGA computation performance. Figure 3(a) reveals the computation latency comparison between Vivado HLS and ArchSyn.

When no computation parallelism is exploited, ArchSyn gives a 1.8X better computation performance result than Vivado HLS for MxM and FFT kernels averagely. With a larger unroll factor or more computation elements, computation latency of Vivado HLS and ArchSyn reduces for both MxM and FFT kernels. An obvious observation is that the decrease rate is slower for Vivado HLS than ArchSyn. When unroll factor/CPE number increases from 1 to 100 for MxM-50, Overlay On-chip gives a 100X performance, while the improvement of Vivado HLS is only 4X. And the performance difference between Overlay On-chip and Vivado HLS increases from 1.5X to 39X.

It is worthy to mention that computation latency of Vivado HLS for FFT kernels almost keeps unchanged, which is less than 6% improvement with a unroll factor of 32. And this indicates that Vivado HLS works inefficient in application kernels with complex and irregular data dependencies. ArchSyn is able to develop computation parallelism with complex and irregular data dependencies, and it gives a 15X and 25X performance improvement for FFT-8K with 32 and 64 CPEs.

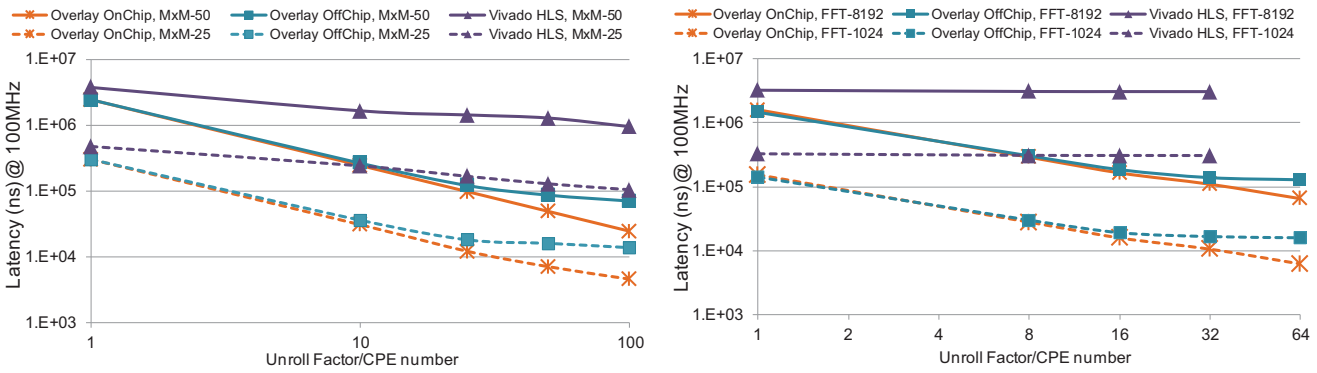
Overlay On-chip and Off-chip implementations exhibit similar improvement in computation performance when CPE number is small. However, when high computation parallelism is required, the Overlay Off-chip implementation begins to slow down in latency optimization, while Overlay On-chip implementation continues to show a near linear improvement. The reason is that Overlay On-chip gets rid of IO limitation with input data storing on-chip through FPGA configuration. And it is able to exploit computation parallelism only under constraint of computation resource and gives better computation performance.

4.2 Dynamic Power Consumption

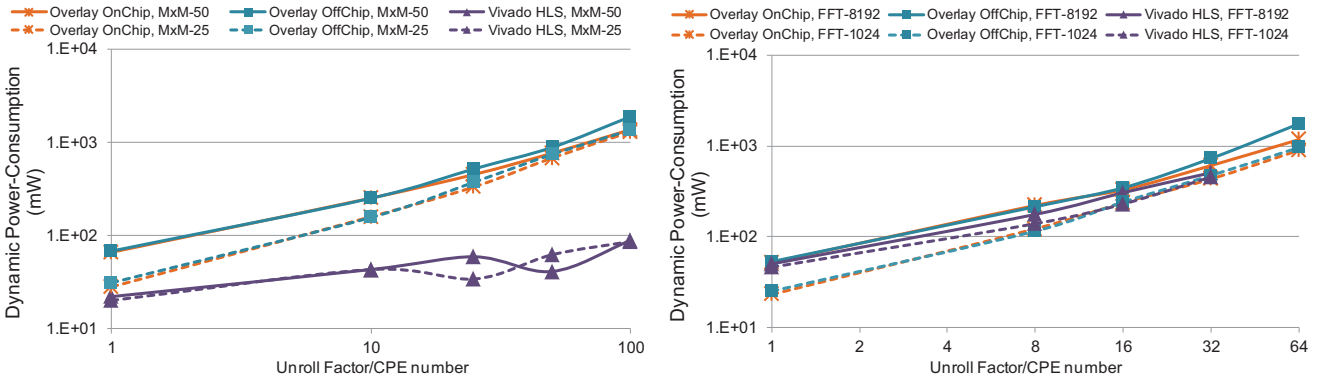
Dynamic power consumption is an import metric in evaluating computation accelerator. Figure 3(b) demonstrates the dynamic power consumption of both Vivado HLS and ArchSyn implementations.

Clearly opposite to latency results, Vivado HLS exhibits a 3X and 1.5X lower power consumption for MxM-50 and MxM-25 kernels respectively with no computation parallelis-

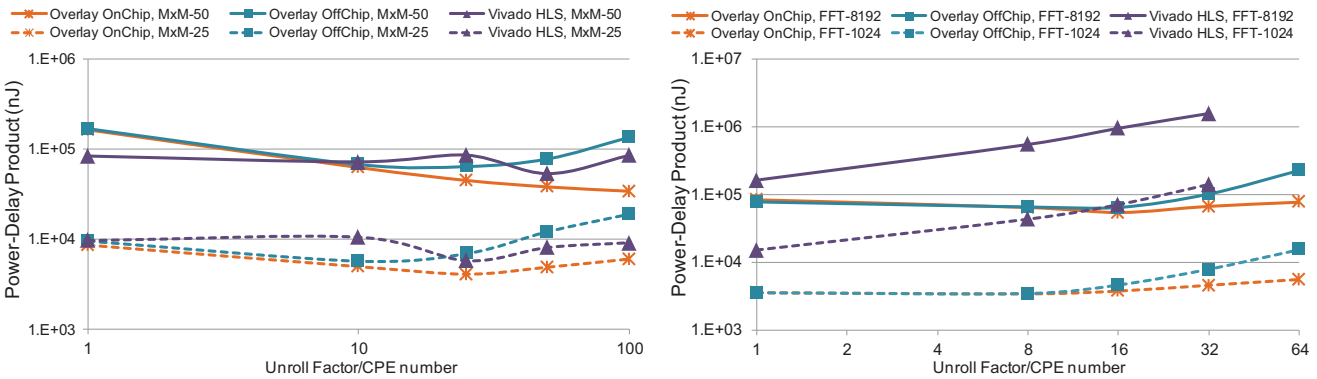
[1]Vivado HLS results with a unroll factor of 64 for FFT kernel are not presented as memory usage exceeds our 4GB-memory PC platform.



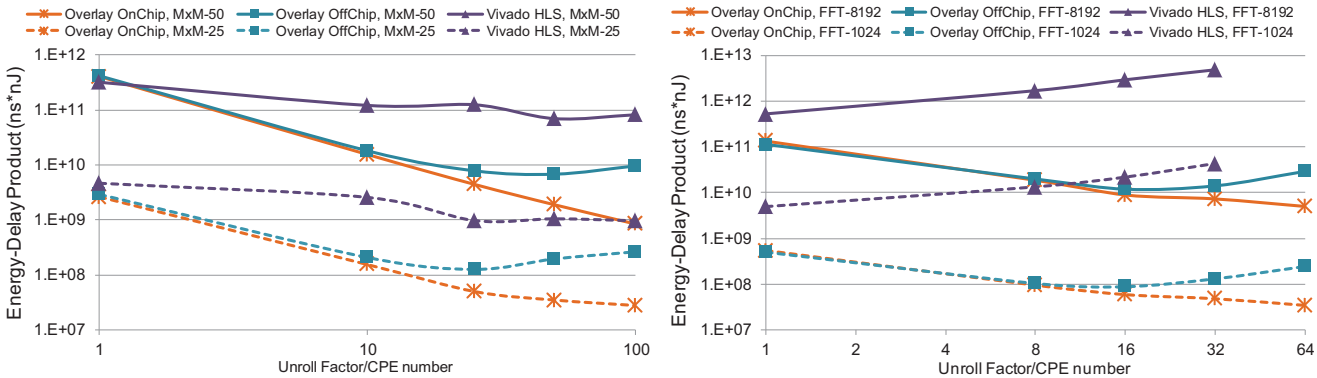
(a) Computation Latency



(b) Dynamic Power Consumption



(c) Power-Delay Product



(d) Energy-Delay Product

Figure 3: Vivado HLS vs ArchSyn on Computation Latency, Dynamic Power Consumption, Power-Delay Product and Energy-Delay Product.

m. The power consumption difference enhances as unroll factor/CPE number increases. When unroll factor/CPE number is 100, Vivado HLS gives a 17X lower power consumption.

An obvious observation is that, Vivado HLS power consumption results has a significant drop at a unroll factor of 50 for MxM-50. And the same improvement happens at a unroll factor of 25 for MxM-25. The reason is that, when the inner-most loop is fully unrolled, the loop can be completely remove. Logic resources for loop/FSM control will be completely released, and as a result, saving a large amount of resource and power consumption.

For FFT kernels, Vivado HLS gains no benefit in power consumption. Due to complex and irregular data and operation dependencies, Vivado HLS is not able to reuse same hardware resources for different iterations of the unrolled loop. As a result, the power consumption increases about linearly as unroll factor increases, same as ArchSyn.

4.3 Power-Efficiency

Chasing high performance through massive computation parallelism inevitably increases power consumption of computing system/accelerator. Improving performance with limited power consumption is an important design goal of computing hardware. We use the metric *Power-Efficiency*, which is equivalent to the value of *Performance per Power* or *Power-Delay Product*, to evaluate the balance between performance and power consumption. Figure 3(c) displays the power-delay product values for both Vivado HLS and ArchSyn.

Both Vivado HLS and ArchSyn exhibit dropping and raising trends in power-delay product curves for MxM kernels, since the improvement in latency by exploiting computation parallelism cannot catch up the increasement of power consumption overhead. For Vivado HLS, the best power-efficiency point is achieved with only the inner-most loop fully unrolled, which is a unroll factor of 50 for MxM-50 and 25 for MxM-25. With IO bandwidth constraint on input data, the best power-efficient point of ArchSyn is at a CPE number of 25 and 10. The best power-efficiency of Overlay Off-chip is 16% worse than the best of Vivado HLS for MxM-50; and for MxM-25, the best results for Overlay Off-chip and Vivado HLS are pretty close, only about 1% difference. While without the IO constraint, ArchSyn prefers a larger number of CPE for higher computation parallelism and performance. The best power-efficiency of Overlay On-chip is 57% and 41% better than Vivado HLS for MxM-50 and MxM-25 respectively.

But for FFT kernels, Vivado HLS implementations keep getting worse in power-efficiency with a higher unroll factor, and a non-unrolled implementation becomes the most power-efficient choice for Vivado HLS. This is because that computation latency has little improvement when unrolled, while much more hardware resources are utilized. For ArchSyn, power-efficiency improves with relatively small CPE number, while begins to degenerate with more CPEs. ArchSyn shows an average of 3.5X better power-efficiency than Vivado HLS.

4.4 Energy-Efficiency

Besides power-efficiency, energy-efficiency is another important metric in evaluating computing system/accelerator. We define *energy-efficiency* as a measurement of computation performance with a specified amount of energy, which is *Performance per Energy*. Similar to power-efficiency in previous section, *Energy-Delay Product* is used to evaluate the energy-efficiency of Vivado HLS and ArchSyn in Figure 3(d).

Compared to power-efficiency, latency plays a more significant role in energy-efficiency evaluation. Therefore, ArchSyn exhibits much better behavior in energy-efficiency due to its outperformance in reducing computation latency. In statistic, Overlay On-chip gives 35X to 144X better energy-efficiency than Vivado HLS; and Overlay Off-chip is 8X to 56X better.

5. DISCUSSIONS

In Table 1, we summarize advantages and disadvantages of Vivado HLS and ArchSyn. Also, they are compared to traditional FPGA design methodology based on hardware description language.

5.1 FPGA Design

Compared to traditional HDL-based FPGA design methodology, both Vivado HLS and ArchSyn take C-like user design and automatically generate RTL design for FPGA implementation. As a higher-level design abstraction is provided, FPGA design and debug process becomes more productive. Also, taking a popular and simple design language, C, as input significantly lowers the barrier to entry for FPGA users.

As Vivado HLS uses a compilation method of directly mapping operation and data to FPGA resources as shown in Figure 1(a), the final FPGA implementation is sensitive to user design style. So hardware design knowledge is strongly recommended to Vivado HLS users. While in compilation flow of ArchSyn, user high level design is finally translated into instructions of processing elements, which will only affect size and configuration content of instruction ROMs, and most part of the SCGRA overlay implementation remains the same. So hardware design knowledge is not necessary to ArchSyn users.

5.2 Design Optimization

Design optimization is a critical process affecting final implementation on FPGAs. It determines the computation latency and resource and power consumption of final FPGA implementation. In HDL-based design methodology, optimization of improving performance or reducing resource/power consumption of final FPGA implementation highly relys on hardware design knowledge and skills.

For Vivado HLS, it provides optional user directives to generate different final FPGA implementations. Some directives are used to guide the code transformation before or during scheduling, and some affects the hardware-related binding and control logic extraction. But their effects on computation latency and resource/power consumption of final FPGA implementation are dependent on application. To explore design space efficiently using Vivado HLS, design engineer needs to be very familiar with its user directives and their effects on code transformation and RTL design/implemenation [13].

While for ArchSyn, user only needs to input a parallelism factor, p. The scheduler will try to exploit computation parallelism from application by a factor of p. The optimal performance speedup will be p as in the example of kernel MxM in Figure 3(a). Of course, the performance speedup

Table 1: Comparisons between Vivado HLS, ArchSyn and Traditional FPGA Design Methodology

	Vivado HLS	ArchSyn	Traditional FPGA Design
Design & Optimization (Manual)	* High-level language input * Optional user directives * Code transformation and hardware design knowledge required	* High-level language input * Only require a parallelism factor	* Hardware description language input * Hardware design knowledge required
Implementation (Automatic)	* Sensitive to user directives * Limited speedup through automatically parallel computing * Relatively low resource and power consumption overhead to speedup computation	* High speedup through automatically parallel computing * Resource and power consumption linear to parallelism factor	* High speedup, resource sharing, low power consumption through hand-optimized hardware design

will be inevitably affected by data and operation dependencies, but ArchSyn guarantees a linear speedup in a certain range of p as in the example of kernel FFT in Figure 3(a). Such performance speedup comes with cost of resource and power consumption increased by the factor of p.

5.3 FPGA Implementation

For all three design methodologies, implementation process from RTL to FPGA hardware is automatic. Here we analyze and compare the implementation results of different methodologies. For hand-optimized hardware design, FPGA implementation can achieve very high performance, and even close to optimal in some well studied application kernels like matrix multiplicaiton [3]. At the same time, low resource and power consumption can be achieved by design skills like efficient resource sharing.

For Vivado HLS, FPGA implementation is sensitive to user design and directives as discussed in previous subsection. Our evaluation results show that it can improve performance by limited speedup through automatically parallel computing in some cases. And the overhead in resource and power consumption to speedup computation is relatively low in some examples.

While for ArchSyn, it uses a scheduling method instead of directly mapping operation and data into FPGA resources. It guarantees a higher speedup through automatically parallel computing even with complex and irregular data dependencies and limited IO bandwidth. But the penalty is a linear increase in resource and power consumption. Such penalty results in a problem that for a large computation kernel, resources, especially memory resources, may be short and it requires more than one configuration for one computation kernel.

6. LIMITATIONS & FUTURE WORK

Though the comparisons present different features of FPGA HLS and Overlay design tools, there are several limitations. First, this paper intends to compare the two design tools from the perspective of software engineers rather than hardware design experts. Only minimal design efforts are taken and no sophisticated techniques are explored with Vivado HLS to reach its ultimate performance. Second, limited benchmarks are used, which only reflect the characteristics in computation-intensive applications.

In our future work, more kernels will be applied and benchmarks of application-level will be added to have more comprehensive comparisons. On other hand, the comparisons indicate the inefficiency of overlay design in power consumption metric. Optimization strategies and design efforts will be applied for further improvement.

7. CONCLUSIONS

In this paper, we make a comparison between a state-of-the-art FPGA HLS tool, *Vivado HLS*, and an FPGA overlay design tool, *ArchSyn*, in order to have an insight into these two design methodologies. With careful evaluation on two computation intensive application kernels, ArchSyn exhibits an overwhelming superiority in computation performance, while Vivado HLS shows the dominance in dynamic power consumption of implementation. On the other hand, ArchSyn proves to be better in power- and energy-efficiency due to its outperformance in computation latency.

8. REFERENCES

[1] A Putnam, AM Caulfield, ES Chung, D Chiou, et al. A reconfigurable fabric for accelerating large-scale datacenter services. In *ISCA*, pages 13–24. IEEE, 2014.

[2] C Zhang, P Li, G Sun, Y Guan, B Xiao, et al. Optimizing FPGA-based accelerator design for deep convolutional neural networks. In *FPGA*, pages 161–170. ACM, 2015.

[3] CY Lin, HKH So, et al. A model for matrix multiplication performance on FPGAs. In *FPL*, pages 305–310. IEEE, 2011.

[4] J Cong et al. High-level synthesis for FPGAs: From prototyping to deployment. *Computer-Aided Design of Integrated Circuits and Systems, IEEE Transactions on*, 30(4):473–491, 2011.

[5] A Canis, J Choi, M Aldham, V Zhang, et al. LegUp: High-level synthesis for FPGA-based processor/accelerator systems. In *FPGA*, pages 33–36. ACM, 2011.

[6] HKH So and C Liu. FPGA overlays. In *FPGAs for Software Engineers*. Springer, 2016.

[7] CY Lin and HKH So. Energy-efficient dataflow computations on FPGAs using application-specific coarse-grain architecture synthesis. *ACM SIGARCH Computer Architecture News*, 40(5):58–63, 2012.

[8] D Capalija and TS Abdelrahman. A high-performance overlay architecture for pipelined execution of data flow graphs. In *FPL*, pages 1–8. IEEE, 2013.

[9] AK Jain, SA Fahmy, et al. Efficient overlay architecture based on DSP blocks. In *FCCM*, pages 25–28. IEEE, 2015.

[10] Xilinx Inc. *Vivado Design Suit User Guide, High-Level Synthesis*, UG902 (v2016.1), `http://www.xilinx.com/support/documentation/sw_manuals/xilinx2016_1/ug902-vivado-high-level-synthesis.pdf`, 2016.

[11] Y Lin. *ArchSyn: An Energy-efficient FPGA High-level Synthesizer*. PhD thesis, Univ. of Hong Kong, Hong Kong, `http://hub.hku.hk/bib/B49799599`, December 2012.

[12] Xilinx Inc. *Vivado Design Suit User Guide, Design Flows Overview*, UG892 (v2016.1), `http://www.xilinx.com/support/documentation/sw_manuals/xilinx2016_1/ug892-vivado-design-flows-overview.pdf`, 2016.

[13] G Zhong, A Prakash, Y Liang, T Mitra, and S Niar. Lin-analyzer: a high-level performance analysis tool for FPGA-based accelerators. In *DAC*, page 136. ACM, 2016.