Automatic Overlay Customization For High-Productivity Nested Loop Acceleration On FPGA

ABSTRACT

Compiling high level compute intensive kernels to FPGAs via an abstract overlay architecture has been demonstrated as an effective way to improve designers' productivity. Furthermore, by customizing the overlay architecture specifically to the user application, it offers a unique opportunity to significantly improve the power-performance of the resulting accelerators. However, achieving the desired performance and overhead constraints requires exploration in a complex design space involving multiple architectural parameters. It is a daunting task for even the most experienced designer, counteracting the benefit of utilizing an overlay architecture as a productivity enhancer.

In this work, an automatic overlay customization framework targeting nested loop acceleration on a hybrid CPU-FPGA system is presented. Using a soft coarse-grained reconfigurable array as an overlay, the customization framework provides not only architectural customization, but also high level compilation options and communication customization between the accelerator and host CPU. Taking advantage of the monotonic influence on performance from major design parameters, the design space was effectively pruned using simple marginal performance revenue metric into a feasible region. Within the feasible design space, optimal customizations for various design goals including performance, energy and energy delay product can then be searched rapidly. Experiments show that our marginal performance revenue aware search can achieve similar Pareto-optimal curve to that acquired by using an exhaustive search while its run time is around two orders of magnitude shorter on average.

1. INTRODUCTION

Overlay architectures are intermediate virtual architectures overlaid on top of the physical FPGA fabric. As demonstrated by many researchers, the use of overlay architectures is a useful technique to improve designers' productivity, security, and portability [13, 11, 9, 6, 7, 1, 16, 18, 4]. As a virtual architecture, utilizing overlays provide a unique opportunity to significantly improve power-performance of the synthesized designs through architectural customization. Unfortunately, to take advantage of this potential, users must navigate through a labyrinth of architectural design parameters, making non-trivial design choices along the way.

To allow users to fully exploit this opportunity, an automatic overlay customization framework is presented here. Our customization framework targets the soft coarse-grained reconfigurable array (SCGRA) overlay as presented in [13]. Nested loops expressed in high-level languages are accelerated on a hybrid CPU-FPGA platform, with the accelerated loops executed on the FPGA while the rest of the user application runs on the CPU. Given the user application, the proposed customization framework automatically explores the design space in producing the optimal overlay architecture. It achieves that by first pruning the overall design space into a feasible space, followed by searching the optimal configuration within this feasible space.

As part of a high-productivity compilation framework, it is desirable that this customization should not become a new run-time bottleneck of its own. With the observation that the major overlay design parameters such as loop unrolling factor and SCGRA overlay size typically have monotonic influence on performance, a marginal performance revenue aware algorithm is proposed for the design space exploration (DSE). This algorithm is straightforward yet effective to prune the design space and reduce the number of the invocation of the SCGRA compilation, which is the most time-consuming step of the DSE.

With the feasible design space is determined, we can further perform customization for various design goals such as minimum energy and minimum energy delay product (EDP). As the structure of the SCGRA overlay is quite regular, it is relatively straightforward to predict its hardware overhead, implementation frequency and power. This helps to estimate the final design goals rapidly through simple models.

Experiments using a real-world benchmark show that the proposed DSE achieves similar results as an exhaustive search while reducing its run time by 2 orders of magnitude on average. It delivers similar Pareto optimal in performance-overhead tradeoff, and provides optimal EDP customization as those results from an exhaustive search.

In Section 2, the SCGRA overlay compilation process that this work bases on is briefly introduced and the observation that leads to the proposed DSE algorithm is presented. The automatic overlay customization framework is elaborated in Section 3. Then, experimental results are presented in Section 4 and related work is introduced in Section 5. Finally, we conclude this paper in Section 6.

2. BACKGROUND AND OBSERVATION

This work mainly relies on top of an SCGRA overlay based FPGA compilation. Thus both the overlay and the compilation framework are introduced in this section. In addition, we performed a number of experiments to show the predictable overlay characteristics such as performance hardware overhead, power and implementation frequency, and these characteristics contribute to the proposed design space exploration and customization framework.

2.1 SCGRA Overlay Compilation

The SCGRA overlay based FPGA compilation framework is developed particularly for the sake of design productivity [13]. With a pre-built overlay, it can produce an FPGA implementation in the order of seconds.

Figure 1 summarizes the SCGRA overlay compilation framework. It targets a CPU-FPGA system and includes two paths corresponding to software and hardware compilation flows. The path marked with green arrows represents the conventional software compilation flow and the final product is the binary code targeting the host CPU. The path marked with orange arrows shows the SCGRA overlay based accelerator compilation flow. It compiles a nested loop compute kernel to a data flow graph (DFG) and then schedules the DFG to the specified SCGRA through an operation scheduler. The scheduling result can then be embedded directly into the pre-implemented SCGRA overlay to produce the final FPGA configuration bitstream.

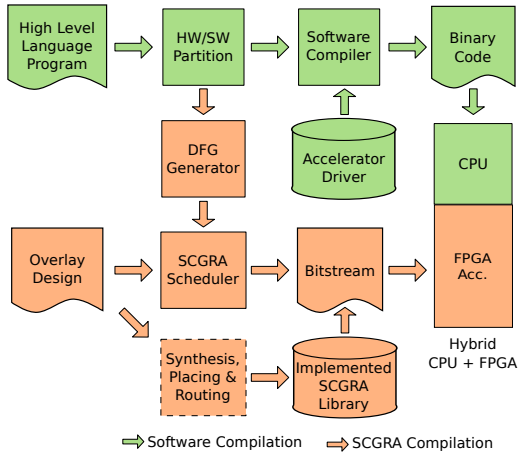


Figure 1: SCGRA Overlay Compilation Framework

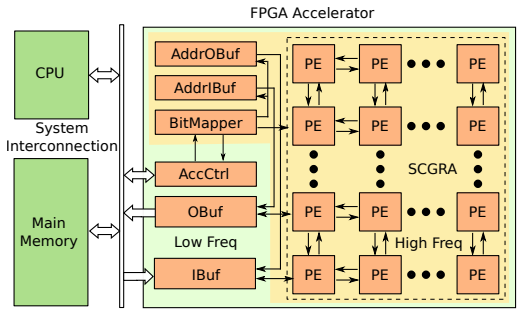


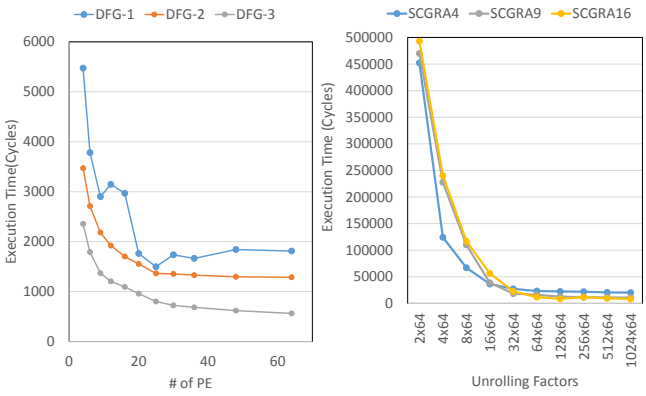
Figure 2: SCGRA Overlay Based FPGA Accelerator

Figure 2 shows the design of a typical SCGRA overlay based FPGA accelerator. In the accelerator, on-chip memory i.e. IBuf and OBuf are used to buffer data between the host CPU and the accelerator. A controller is also presented in hardware to control the operations of the accelerator as well as memory transfers. The SCGRA, which is the kernel computation fabric, consists of an array of processing elements (PEs) and it achieves the computation task through the distributed control words stored in each PE. The AddrBuf stores all the valid IO buffer accessing addresses of the DFG while the valid signals come from the BitMapper. In order to amortize the IO data transmission cost, we may group a number of DFG and have the IO transmitted in the granularity of a group. Accordingly, AddrBuf should store all the addresses of the group while the control words in each PE can still be reused.

2.2 Predictivity of SCGRA Overlay

In order to simplify the customization, we have performed a series of experiments on Zedboard [2] to analyze how different design parameters affect the performance, power and overhead.

Figure 3 shows the influence of SCGRA size and unrolling factor on the execution time of a loop kernel. Although there are slightly variation when the extracted DFG or the unrolling factor is small, both parameters basically present monotonic impact on the execution time.



(a) (b)

Figure 3: Design Parameter Influence on Execution Time, (a) SCGRA Size, (b) Unrolling Factor

Figure 4 shows the overhead, implementation frequency and power of a group of SCGRA overlay implementation. The basic configuration is listed in Table 1. According to this figure, it is found that the results typically present good linearity over the SCGRA overlay size and therefore simple models can be used to achieve an adequate estimation.

Table 1: Experiment Configuration

DataWidth	InstRom	DataMem	IBuf/OBuf	AddrBuf
32	1kx72bit	256	2048	4kx16bit

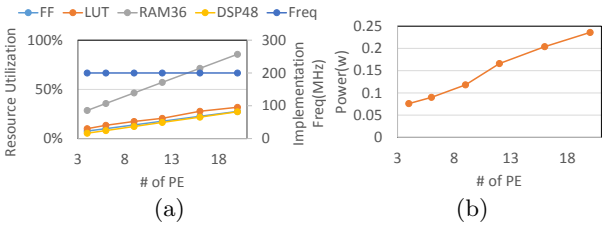


Figure 4: SCGRA Overlay Overhead and Power, (a) Overhead, (b) Power

3. AUTOMATIC OVERLAY CUSTOMIZATION

A SCGRA overlay customization framework is proposed to automatically adjust the overlay architecture to achieve better design trade-off for a specific compute intensive nested loop. A marginal performance revenue aware customization DSE is used to acquire a smaller feasible design space. Within the feasible design space, customization for various design goals such as minimum energy consumption and minimum energy delay production can be obtained rapidly.

3.1 Customization Framework

Figure 5 shows the proposed customization framework. It can be roughly divided into three steps including marginal performance revenue aware DSE, SCGRA overlay customization and SCGRA overlay compilation. The SCGRA overlay compilation with specified overlay configuration and loop transformation can be found in Section 2, and we will mainly focus on the remaining two steps in this section.

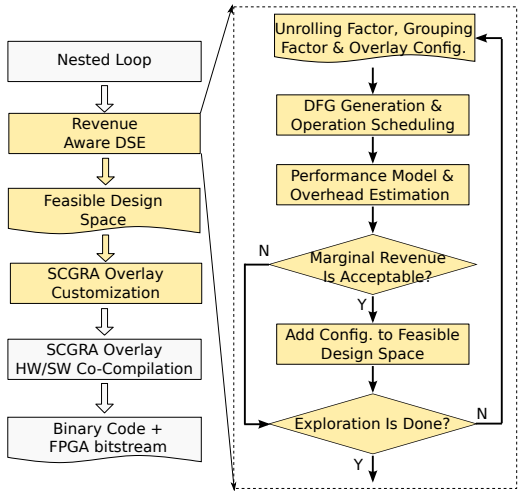


Figure 5: FPGA Accelerator Customization Framework

The DSE is essentially used to determine the feasible design space. As presented in Figure 5, the DSE needs to go through the operation scheduling first to acquire the DFG performance. Then the nested loop performance can be calculated with corresponding performance model. The detail of this model will be illustrated in next section. By comparing this configuration to previous acceptable configurations in the feasible design space, performance revenue can be obtained. If the revenue of a particular design parameter is lower than a pre-defined threshold, this configuration will be regarded as unacceptable. In addition, according to the observation in previous section, the configurations that are larger on this particular parameter will be set to be unacceptable as well because the marginal performance revenue decreases. Finally, these unacceptable configurations can be safely pruned and thus saves the scheduling and estimation efforts. Similarly, overhead constraints can also be used to prune the design space. The configurations that are accepted will be added to the feasible design space. Overhead as well as performance of each feasible configuration will be stored and they can be used for further customization in the next step.

After the DSE, customization for various design goals such as energy consumption and energy delay production will start. It basically acquires these design goals through corresponding models and then search for the optimal design and configuration.

Finally, customized overlay configuration and loop transformation options will be passed to the overly compilation step. The nested loop can be then compiled to FPGA bitstream and CPU binary code targeting the hybrid CPU-FPGA computation system.

3.2 Performance Revenue Aware DSE

The proposed performance revenue aware DSE algorithm is formalized and detailed in this section. Suppose $\boldsymbol{\Psi}$ represents the accelerator design space of a nested loop. $\boldsymbol{C} = (C_0, C_1, ..., C_N) \in \boldsymbol{\Psi}$ represents a possible configuration of the nested loop acceleration. C_i stands for the ith design parameter of the configuration $\boldsymbol{C}$ and $C_i \in (C_i^0, C_i^1, ..., C_i^{R_i})$ where R_i stands for the maximum No. of possible C_i. $\boldsymbol{C}(i,j) = (C_0, C_1, ..., C_i^j, ...)$ denotes a configuration in which the ith design parameter adopts the jth possible option. ϵ is the performance revenue metric and Φ denotes the feasible design space.

The algorithm is illustrated in Algorithm 1. It is essentially a brute-force search with online design space pruning using marginal performance revenue. As different design parameters may influence with each other, the marginal performance revenue is merely analyzed between two configuration vectors that differ on a single dimension of the vectors and it can be calculated by (1).

$$Revenue(i,k) = 1 - \frac{LoopExeTime(\boldsymbol{C}(i,k+1))}{LoopExeTime(\boldsymbol{C}(i,k))} \quad (1)$$

In addition, there may be minor fluctuation due to the complex heuristics used in the operation scheduling. For example, larger SCGRA size may result in worse performance as displayed in Figure 3. This may mislead the design space pruning. To cope with this problem, multiple consecutive performance revenue should be evaluated as presented in this algorithm.

Algorithm 1 Revenue Aware Design Space Exploration.

```
procedure
  Init State[C] as False
  for all C ∈ Ψ ∧ (¬State[C]) do
    i = 1
    while i < N do
      k is integer such that C_i == C_i^k
      Perform simulation with configuration C(i, k)
      Estimate performance LoopExeTime(C(i, k))
      Estimate overhead Overhead(C(i, k))
      if Revenue(i, k − 1) > ε ∨ Revenue(i, k) > ε then
        Add C(i, k) to Φ
        Set State[C] as True
      else
        j = k
        while j <= R_i do
          Set State[C] as True
        end while
      end if
      i = i + 1
    end while
  end for
end procedure
```

3.3 Performance and Overhead Models

In this section, we will mainly explain the performance and hardware overhead models which will be used for both the design space exploration and customization.

3.3.1 Performance Model

In order to map the nested loop to an SCGRA overlay, the loop will be partially unrolled and the unrolled part will be transformed to a DFG. Then the DFG will be scheduled to the overlay. In order to improve the transmission efficiency, a few identical DFGs will be further grouped and each group needs only a single IO transmission.

Suppose each group consists of $DFGPerGroup$ DFGs and the nested loop contains $GroupPerLoop$ groups. $DFGExeTime$ denotes the execution time of a DFG which can be acquired from the operation scheduler. The execution time of the loop which includes both the computation time and the communication time can be calculated by (2). We use DMA for the communication between the SCGRA overlay and host CPU and the DMA can be estimated using a piece wise linear modeled function denoted by $DMALat(x)$ where x stands for the amount of data to be transmitted. Then the communication can be calculated by (3).

$$\begin{aligned} LoopExeTime =& GroupPerLoop \times DFGPerGroup \\ & \times DFGExeTime + CommTime \end{aligned} \tag{2}$$

$$\begin{aligned} CommTime =& (DMA(GroupIn) + DMA(GroupOut)) \\ & \times GroupPerLoop \end{aligned} \tag{3}$$

3.3.2 Overhead, Power and Energy Models

FPGA hardware overhead mainly involves DSP, LUT, FF and RAM block. DSP and RAM block consumption can be calculated directly with specified overlay configuration while LUT and FF are estimated through a linear model of SCGRA size.

Power consumption is consisted of RAM block power, clock power, signal power, DSP power and so on. We notice that the BRAM power consumption has better linearity with the amount of BRAM consumption while the rest part of the power is linear to the SCGRA size. Thus the power consumption can be calculated by (4) where α and β are linear model coefficients.

$$\begin{aligned} Power =& RAMOverhead \times RAMUnitPower \\ & + \alpha \times SCGRASize + \beta \end{aligned} \tag{4}$$

With the performance and power models, we can further acquire the energy consumption model which is the production of power and loop execution time and energy power delay product model which is the production of energy consumption and loop execution time.

4. EXPERIMENTS

We take four applications including Matrix Multiplication (MM), FIR, Kmean(KM) and Sobel Edge Detector (SE) as our benchmark. The benchmark configurations are presented in table Table 2.

Table 2: Benchmark Configurations

Benchmark	Parameters	Loop Structure
MM	Matrix Size(128)	$128 \times 128 \times 128$
FIR	# of Input (1024) # of Taps+1 (64)	1024×64
SE	# of Vertical Pixels (128) # of Horizontal Pixels (8)	$128 \times 8 \times 3 \times 3$
KM	# of Nodes(1024) # of Centroids(4) # of Dimensions(2)	$1024 \times 4 \times 2$

The benchmark is customized to a CPU-FPGA system using both the proposed revenue aware (RA) customization framework and an exhaustive search(ES) based customization. Then the customization speed and quality of the two are compared.

4.1 Experiment Setup

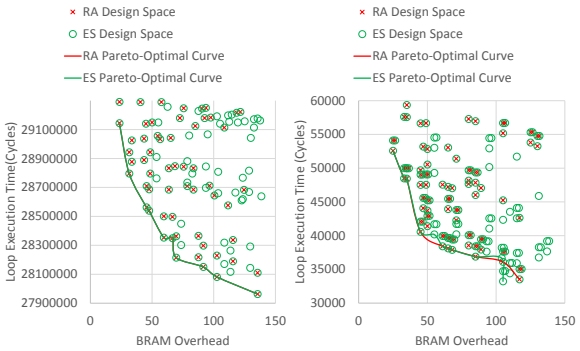
All the run time was obtained from a computer with Intel(R) Core(TM) i5-3230M CPU and 8GB RAM. Zedboard which has an ARM processor and an FPGA was used as the hybrid computation system. Vivado 2013.3 was used for the hardware design.

The overlay typically runs at 200MHz on Zedboard and we assume the implementation frequency can be scalable to all the different overlay configurations. The power used in this work is acquired from XPower which is part of the Vivado design suite.

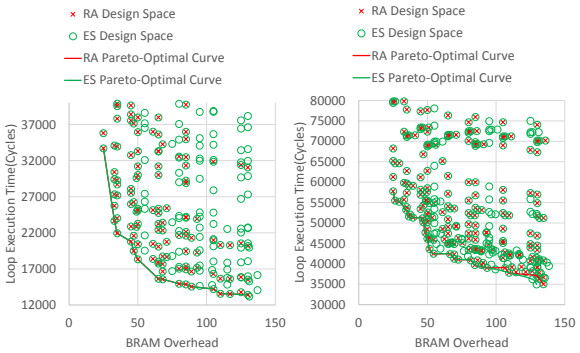
There are many different design parameters for compiling a nested loop to the CPU-FPGA system through an SCGRA

Table 3: Design Parameters of Overlay Customization For Nested Loop Acceleration

Design Parameters		Is Supported?
Nested Loop Compilation	Loop Unrolling Factor	Yes
	Grouping Factor	Yes
Overlay Configuration	SCGRA Topology	No, 2D Torus
	SCGRA Size	Yes
	Instruction Mem	Yes
	Data Mem	Yes
	IO Buffer	Yes
	Address Buffer	Yes
	Operation Set	No
	Pipeline Depth	No



(a) MM (b) FIR

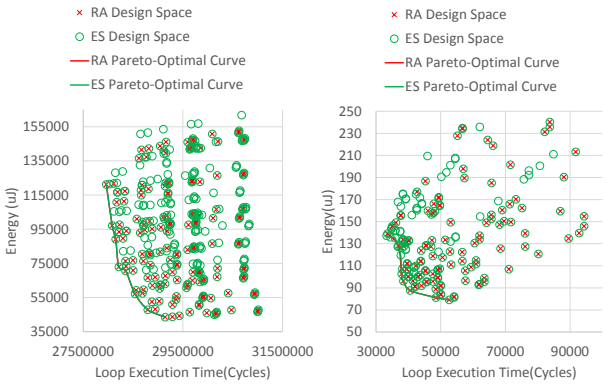


(c) SE (d) KM

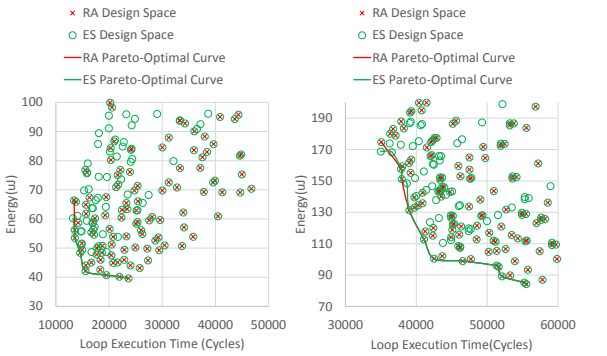
Figure 6: Performance-Overhead Pareto-optimal Curve and Possible Design Options Around The Curve

overlay. Table 3 is a list of the major design parameters of the overlay customization for nested loop acceleration. As identified in the table, most of the parameters are covered in this work while pipeline depth, SCGRA topology and operation set of the overlay are not configurable at the moment. We adopt a deterministic 200MHz pipeline design and 2D torus topology. The operation set supported by the overlay are mainly simple operations with three source operands and a single destination operands. Also note that IO buffer partition is not fully supported and we can only change the IO buffer depth and data width at the moment.

4.2 Experiment Results



(a) MM (b) FIR



(c) SE (d) KM

Figure 7: Performance-Energy Pareto-optimal Curve and Possible Design Options Around The Curve

Table 4: Time Cost for RA DSE and ES DSE

Benchmark	MM	FIR	KM	SE
RA DSE (min)	20.2	10.6	13.4	11.4
ES DSE (min)	2880.6	835.2	1140.5	946.2
Speedup	142.6	78.8	85.1	86.2

Table 4 shows the exploration time comparison of the two DSE methods. The proposed RA DSE is around 100x faster than the ES DSE on average. In particular, it can be found that ES DSE can be extremely slow on MM which has three levels of loop with relatively large loop count and thus larger design space. Though RA DSE also needs longer time to complete the DSE of MM, it can skip most of the unfeasible configurations and the run time is less sensitive to the size of the design space.

In order to demonstrate the quality of RA DSE, we presented the comparison of the exploration results acquired from both DSE methods. The exploration results includes performance-overhead, performance-energy and performance-EDP trade-offs as shown in Figure 6, Figure 7 and Figure 8 respectively. In the figures, both Pareto-optimal curves and the design options around the curves are presented.

Basically, the Pareto-optimal curves acquired using RA DSE is quite close to that obtained from an ES DSE. In particular, all the three Pareto-optimal curves of MM are highly over-

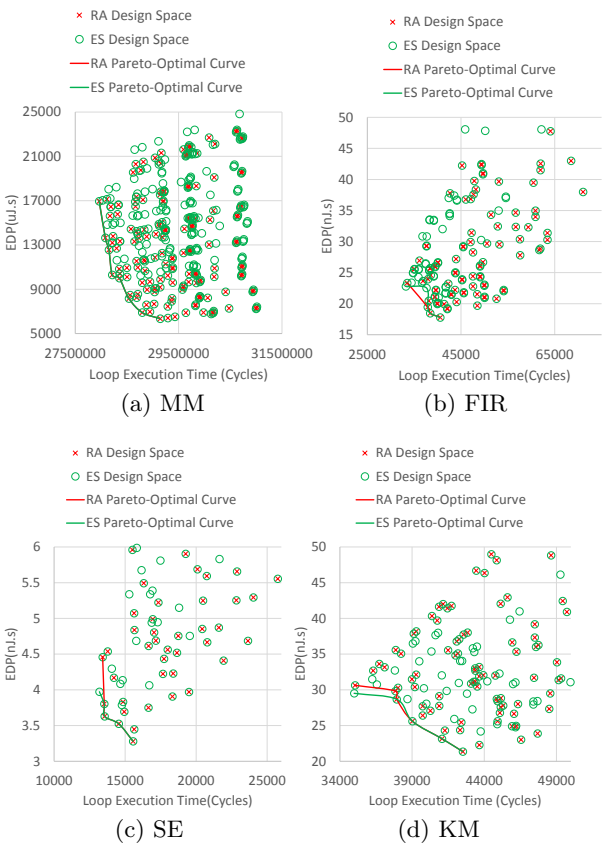


Figure 8: Performance-EDP Pareto-optimal Curve and Possible Design Options Around The Curve

lapped and RA DSE successfully skips the design options that are further from the Pareto-optimal curves. However, RA DSE may prune the design options that involve a larger overlay size but better performance according to the revenue metric. As a result, the Pareto-optimal curves of the rest three nested loops mostly deviate at the higher performance area. Fortunately, this can be improved by lowering the performance revenue threshold and affording longer DSE time.

These Pareto-optimal curves can also be used for customization. When minimum energy consumption or minimum EDP is taken as the design goal, RA aware DSE can achieve the optimal design in all the four benchmarks.

5. RELATED WORK

FPGA design productivity has become a major obstacle that hinders the widespread adoption of FPGA as commodity computing devices. To address the problem, the community has developed a number of design tools that can compile high level applications all the way to FPGA [20, 3, 17, 14] and design frameworks that can provide automatic customization to achieve better design trade-off [21, 5, 8, 15, 12, 10, 19].

High level synthesis (HLS) is one of the most widely adopted methods to compile high level applications to FPGA circuits. Thus previous automatic customization frameworks were mostly developed on top of the HLS tools [21, 5, 8, 15, 12]. They typically considered an HLS tool as a black box. Generic algorithms were adopted to perform the DSE and customization in [5, 8, 12]. Divide and conquer method was used in [15] and A calibration tree algorithm was used in [10].

Overlay which is an intermediate layer built on top of the off-the-shelf FPGA devices has been demonstrated to be an effective way to improve the FPGA design productivity[13, 6]. A number of different types of overlays [11, 9, 6, 7, 1, 16, 18, 4] have been developed for different trade-offs between flexibility and overhead. SCGRA overlay is one of the most promising overlays for compute intensive application acceleration on FPGA [13, 19]. The authors in [19] focused on topology customization and showed the potential benefits of the SCGRA overlay customization. This work will provide a more intensive customization for nested loop acceleration on a hybrid CPU-FPGA system. It will not only include the overlay fabric customization but also the compilation customization such as loop unrolling and communication customization between the FPGA and host CPU.

6. CONCLUSION

In this work, we have presented an automatic SCGRA overlay customization framework for nested loop acceleration on a hybrid CPU-FPGA system. By utilizing a marginal performance revenue metric, it effectively prunes the design space to a small feasible region. In the feasible design space, customization for different design goals can be obtained rapidly. When compared to an exhaustive search, the proposed customization framework reduces run time by 2 orders of magnitude on average while achieving comparable Pareto-optimal in performance-overhead tradeoff and the same customized architecture that results in minimum energy and minimum energy-delay product.

7. REFERENCES

[1] A. Al-Dujaili, F. Deragisch, A. Hagiescu, and Weng-Fai Wong. Guppy: A gpu-like soft-core processor. In *Field-Programmable Technology (FPT), 2012 International Conference on*, pages 57–60, Dec 2012.

[2] Avnet. Zedboard. `http://www.zedboard.org/`, 2014. [Online; accessed 25-June-2014].

[3] Andrew Canis, Jongsok Choi, Mark Aldham, Victor Zhang, Ahmed Kammoona, Jason H. Anderson, Stephen Brown, and Tomasz Czajkowski. LegUp: High-level Synthesis for FPGA-based Processor/Accelerator Systems. In *Proceedings of the 19th ACM/SIGDA International Symposium on Field Programmable Gate Arrays*, FPGA '11, pages 33–36, New York, NY, USA, 2011. ACM.

[4] D. Capalija and T.S. Abdelrahman. An architecture for exploiting coarse-grain parallelism on fpgas. In *Field-Programmable Technology, 2009. FPT 2009. International Conference on*, pages 285–291, Dec 2009.

[5] B. Carrion Schafer and K. Wakabayashi. Machine learning predictive modelling high-level synthesis design space exploration. *Computers Digital Techniques, IET*, 6(3):153–159, May 2012.

[6] R. Ferreira, J.G. Vendramini, L. Mucida, M.M.

Pereira, and L. Carro. An fpga-based heterogeneous coarse-grained dynamically reconfigurable architecture. In *Proceedings of the 14th international conference on Compilers, architectures and synthesis for embedded systems*, pages 195–204. ACM, 2011.

[7] David Grant, Chris Wang, and Guy G.F. Lemieux. A cad framework for malibu: An fpga with time-multiplexed coarse-grained elements. In *Proceedings of the 19th ACM/SIGDA International Symposium on Field Programmable Gate Arrays*, FPGA '11, pages 123–132, New York, NY, USA, 2011. ACM.

[8] M. Holzer, B. Knerr, and M. Rupp. Design space exploration with evolutionary multi-objective optimisation. In *Industrial Embedded Systems, 2007. SIES '07. International Symposium on*, pages 126–133, July 2007.

[9] Dmitrij Kissler, Frank Hannig, Alexey Kupriyanov, and Jürgen Teich. A dynamically reconfigurable weakly programmable processor array architecture template. In *ReCoSoC*, pages 31–37, 2006.

[10] Maciej Kurek, Tobias Becker, Thomas C.P. Chau, and Wayne Luk. Automating optimization of reconfigurable designs. In *Field-Programmable Custom Computing Machines (FCCM), 2014 IEEE 22nd Annual International Symposium on*, pages 210–213, May 2014.

[11] I. Lebedev, Shaoyi Cheng, A. Doupnik, J. Martin, C. Fletcher, D. Burke, Mingjie Lin, and J. Wawrzynek. MARC: A many-core approach to reconfigurable computing. In *Reconfigurable Computing and FPGAs (ReConFig), 2010 International Conference on*, pages 7 –12, dec. 2010.

[12] Hung-Yi Liu and L.P. Carloni. On learning-based methods for design-space exploration with high-level synthesis. In *Design Automation Conference (DAC), 2013 50th ACM / EDAC / IEEE*, pages 1–7, May 2013.

[13] removed. removed for blind review.

[14] ROCCC. Roccc2.0. `http://www.jacquardcomputing.com/roccc/`, 2014. [Online; accessed 19-January-2014].

[15] Benjamin Carrion Schafer and Kazutoshi Wakabayashi. Divide and conquer high-level synthesis design space exploration. *ACM Trans. Des. Autom. Electron. Syst.*, 17(3):29:1–29:19, July 2012.

[16] D. Unnikrishnan, Jia Zhao, and R. Tessier. Application specific customization and scalability of soft multiprocessors. In *Field Programmable Custom Computing Machines, 2009. FCCM '09. 17th IEEE Symposium on*, pages 123–130, April 2009.

[17] Xilinx. Vivado hls. `http://www.xilinx.com/products/design-tools/vivado/integration/esl-design/`, 2014. [Online; accessed 18-October-2014].

[18] Peter Yiannacouras, J. Gregory Steffan, and Jonathan Rose. Fine-grain performance scaling of soft vector processors. In *Proceedings of the 2009 International Conference on Compilers, Architecture, and Synthesis for Embedded Systems*, CASES '09, pages 97–106, New York, NY, USA, 2009. ACM.

[19] Hayden Kwok-Hay. Yu, Colin Lin. So. Energy-efficient dataflow computations on FPGAs using application-specific coarse-grain architecture synthesis. In *Highly Efficient Accelerators and Reconfigurable Technologies, The 4th International Workshop on*. IEEE, 2012.

[20] Zhiru Zhang, Yiping Fan, Wei Jiang, Guoling Han, Changqi Yang, and Jason Cong. Autopilot: A platform-based esl synthesis system. In *High-Level Synthesis*, pages 99–112. Springer, 2008.

[21] Guanwen Zhong, V. Venkataramani, Yun Liang, T. Mitra, and S. Niar. Design space exploration of multiple loops on fpgas using high level synthesis. In *Computer Design (ICCD), 2014 32nd IEEE International Conference on*, pages 456–463, Oct 2014.