An Automatic Nested Loop Acceleration Framework on FPGAs Using Soft CGRA Overlay

CHENG LIU, The University of Hong Kong
HAYDEN KWOK-HAY SO, The University of Hong Kong

Compiling high level compute intensive nested loops to FPGAs via an abstract overlay has been demonstrated as an effective way to improve design productivity. However, achieving the desired performance and energy constraints requires intensive application-specific customization over a complex design space while the customization can be extremely time-consuming counteracting the benefit of utilizing the overlay as a design productivity enhancer.

To address the above problem, we have developed a highly scalable and regular Soft CGRA (SCGRA) overlay as the backbone of the FPGA accelerators such that the design metrics such as implementation frequency, hardware overhead, and power consumption are highly predictable and analytical models can be easily built to evaluate the design metrics. With the analytical models, the complex application-specific customization targeting nested loop acceleration on FPGAs is reduced to a light-weight sub design space exploration centering the NP-complete SCGRA scheduling and a straightforward customization with all the potential configurations well estimated. According to the experiments, the proposed customization method achieves similar Pareto-optimal curves to that obtained using an exhaustive search with a fraction of runtime. When compiled to the implementations with best-effort optimization using off-the-shelf HLS design tools, the implementations using the proposed customization framework exhibit competitive performance and overhead.

General Terms: Nested Loop, FPGA, Overlay, Soft CGRA

Additional Key Words and Phrases: Overlay, FPGA Acceleration, Nested Loop, Soft CGRA

ACM Reference Format:
Cheng Liu and Hayden Kwok-Hay So, 2014. An Automatic Nested Loop Acceleration Framework on FPGAs Using Soft CGRA Overlay *ACM Trans. Reconfig. Technol. Syst.* 0, 0, Article 1 (2014), 20 pages.
DOI:http://dx.doi.org/10.1145/0000000.0000000

1. INTRODUCTION

Offloading compute intensive nested loops to FPGA accelerators has been demonstrated as an effective way of performance acceleration across various application domains[Chung et al. 2010]. However, the design productivity of developing such accelerators remains relatively low and it has become a major obstacle that hinders the wide adoption of FPGAs as compute engines. Although the use of high level synthesis (HLS) tools which allow the application designers to focus on high level functionality instead of low-level implementation details alleviates this shortcoming [Cong et al. 2011], the lengthy low-level FPGA implementation process greatly limits the number of compile-debug-edit cycles per day and dramatically affects the overall design productivity.

This work is supported in part by the Research Grant Concil of Hong Kong, under the General Research Fund project 716510 and in part by the Croucher Innovation Award of the Croucher Foundation.
Author's addresses: Cheng Liu and Hayden Kwok-Hay So Department of Electrical and Electronic Engineering, The University of Hong Kong, Pokfulam Road, Hong Kong

DOI:http://dx.doi.org/10.1145/0000000.0000000

To approach the above design productivity problem, researchers have recently turned to the use of virtual FPGA overlay architectures [Brant & Lemieux 2012; Capalija & Abdelrahman 2013; Ferreira et al. 2011; Grant et al. 2011; Kissler et al. 2006; Liu et al. 2013]. When combined properly to high level compilation tools, the overlay architecture based design methods are able to produce high-performance accelerators at near software compilation speed, but at the cost of hardware overhead, power and even performance. By customizing the architectures of these *virtual* overlays for a target user design, in theory, it is possible to significantly improve the performance-energy of the resulting accelerator. In practice, however, navigating through a labyrinth of architectural and compilation parameters to fine-tune an accelerator's performance-energy is a slow and non-trivial process. To require a user to manually explore such vast design space is going to counteract the productivity benefit of the utilizing overlay in the first place.

To obtain both high design productivity and advantages of application-specific customization, we have developed a soft CGRA (SCGRA) overlay based nested loop acceleration design framework. This framework targets a hybrid CPU-FPGA computing system where nested loop compute kernels expressed in high-level languages are compiled and executed on the SCGRA overlay built on top of FPGAs while the rest of the user application remains running on the host CPU. Given high-level design goals and design constraints, the framework automatically explores the design space and customizes architectural parameters specifically to the user application. In addition, the framework also exploits loop unrolling and hardware-software communication strategies in combination with buffer sizing and partition as performance enhancing techniques. Once the design goals and constraints are fulfilled, the corresponding hardware accelerator and communication interface are generated and both the hardware accelerator and software are compiled to the hybrid CPU-FPGA system.

As demonstrated in previous work, both the compilation from nested loops to the SCGRA overlay [Liu et al. 2013] and the SCGRA overlay implementation [Michael Yue & Lemieux 2015] are fast. Meanwhile, the SCGRA overlay is highly pipelined and has quite regular tiling structure, which makes the hardware overhead, power consumption and even implementation frequency highly predictable. Therefore, a multitude of design metrics such as performance and energy consumption can be accurately estimated using analytical models when the overlay scheduling result is available. And the nested loop specific acceleration problem can be reduced to a sub design space exploration centering an NP-complete SCGRA scheduling and a following customization with all the potential configurations well estimated. While the overlay scheduling depends on much less design parameters, the overall customization can be dramatically simplified. Accordingly, the overall design framework achieves both rapid compilation and fast application specific customization and ensures high design productivity and high performance of the resulting accelerators at the same time.

We performed a series of experiments to evaluate the efficiency and quality of the proposed design framework using a real-world benchmark. Compared to an exhaustive search, the proposed customization achieves similar results while reducing its runtime by 2 orders of magnitude on average. When compared to HLS implementations with moderate manual optimizations that can reasonably be expected from a novice user, the customized accelerators produced using the proposed framework has demonstrated competitive performance as well.

With that, we consider the main contribution of this work is in the following areas:

— We have developed a rapid customization framework that performs automatic design parameter tuning for SCGRA overlay based nested loop acceleration on a hybrid CPU-FPGA computing system. The result is comparable to an exhaustive design space search while it runs at a fraction of time.

— We have developed a parametric regular SCGRA overlay template. It can be used to generate FPGA accelerators with predictable implementation frequency, hardware overhead

and power consumption, which is essential to both the rapid compilation and customization.
— We have developed a hierarchy on-chip buffer. It allows flexible buffer partition and makes good use of the efficient lock-step computation of the SCGRA overlay.

In Section 2, related work is briefly introduced. The overall automatic nested loop acceleration framework is illustrated in Section 3. Then SCGRA overlay based FPGA accelerator is illustrated in Section 4 and the application-specific customization method is further detailed in Section 5. Experimental results are presented in Section 6 and limitations are discussed in Section 7. Finally, the paper is concluded in Section 8.

2. RELATED WORK

Despite the performance and power advantages, the design productivity of developing FPGA applications remains low due to the lengthy compilation and complex application-specific customization. And it has become the major obstacle that hinders the wide adoption of FPGAs as commodity computing devices. The community from both the industry and academia have developed many different methods from diverse angles to tackle the problem. These methods can be roughly classified into three categories. The first category mainly focuses on improving the low-level implementation tools. A number of approaches such as making quality/runtime trade-offs [Mulpuri & Hauck 2001], parallel compilation [Corporation 2015a,b; Goeders et al. 2011; Moctar & Brisk 2014] and using hard-macro techniques [Korf et al. 2011; Lavin et al. 2013] have been explored from this angle. The second category mainly centers the HLS design flow while the third one primarily relies on the overlay concept. They later two categories will be detailed in the following sections.

2.1. High-Level Synthesis

With many years of continuous endeavor, a number of tools have emerged as mature solutions for HLS [Canis et al. 2011; Xilinx 2014; Zhang et al. 2008]. They typically allow designers to express hardware designs using high-level description languages such as C, C++ etc. and also enable evaluation of different design choices using pragmas or directives. Indeed, they significantly improve the design productivity compared to the conventional hardware design flow using hardware description languages. However, when considering the overall design productivity of developing hybrid software-gateware applications, HLS is only addressing part of the problem, as the lengthy low-level compilation including synthesis, mapping, placing and routing remains a bottleneck for an application designer [Capalija & Abdelrahman 2014; Michael Yue & Lemieux 2015].

Customizing the generated hardware specifically to an user application is also time-consuming for designers and thus critical to the design productivity. A number of algorithms such as generic algorithms relied on local-search techniques [Schafer et al. 2009; Sengupta & Bhatia 1997], learning-based methods [Carrion Schafer & Wakabayashi 2012; Liu & Carloni 2013], divide and conquer algorithm [Schafer & Wakabayashi 2012] and a calibration free algorithm [Kurek et al. 2014] etc. have been developed to perform the DSE on top of HLS tools. The algorithms can efficiently help automate the customization or DSE process. However, the algorithms must rely on HLS tools to estimate the implementation information such as implementation frequency, overhead or power for the corresponding customization. While the hardware generated can be irregular and may vary dramatically, thus the accuracy of the estimation especially on implementation frequency and power can be rather limited, which may fail to optimize an HW/SW co-design problem.

2.2. Overlay Architectures

Overlay architecture which is a virtual intermediate architecture overlaid on top of off-the-shelf FPGA is increasingly applied as a way to address the productivity challenge.

Various overlays with diverse configuration granularities and flexibility ranging from virtual FPGAs [Brant & Lemieux 2012; Grant et al. 2011], array-of-FUs [Capalija & Abdelrahman 2013; Ferreira et al. 2011], soft CGRA [Kissler et al. 2006; Liu et al. 2013], soft GPU [Al-Dujaili et al. 2012], vector processors[Severance & Lemieux 2013; Yiannacouras et al. 2009] to configurable processors or multi-core processors [Capalija & Abdelrahman 2009; Cheah et al. 2012; LaForest & Steffan 2012; Lebedev et al. 2010; Unnikrishnan et al. 2009; Yiannacouras et al. 2007] have been developed over the years. SCGRA overlay provides unique advantages on compromising hardware implementation and performance for compute intensive nested loops as demonstrated by numerous ASIC CGRAs [Compton & Hauck 2002; Tessier & Burleson 2001]. Most importantly, it allows both rapid compilation by taking advantage of the overlays' tiling structure [Michael Yue & Lemieux 2015] and efficient bitstream reuse within the design iterations of an application [Liu et al. 2013], thus it is particularly promising for high productivity nested loop acceleration.

Indeed, SCGRA overlays have many similarities in terms of array structure and scheduling algorithm with ASIC CGRAs. ASIC CGRAs emphasize more on configuration capability and limited customization is allowed due to the overhead constraints [Miniskar et al. 2014; Zhou et al. 2014] while SCGRA overlays allow more intensive architectural customization because of the FPGA's inherent programmability. Moreover, hardware resources such as DSP blocks and RAM blocks available on FPGAs are discrete, which results in different design constraints for SCGRA overlay customization.

The authors in [Yu 2012] developed an SCGRA topology customization method using genetic algorithm and showed the potential benefits of the SCGRA overlay customization, but the rest of the system design parameters were not covered. In order to achieve both high design productivity and high performance with low overhead, a complete nested loop acceleration framework targeting CPU-FPGA system is developed in this work. It supports intensive application-specific customization including the overlay architectural customization, the compilation customization and communication interface customization for various design goals. When the customized design parameters are determined, corresponding hardware accelerator and software can be compiled to the target CPU-FPGA system rapidly eventually providing a push-button solution for a nested loop acceleration.

3. AUTOMATIC NESTED LOOP ACCELERATION FRAMEWORK

By using a regular SCGRA overlay built on top of the physical FPGA devices, we have developed an automatic nested loop acceleration framework targeting a hybrid CPU-FPGA system. The goal of the framework is to provide a high-productivity nested loop acceleration solution accessible to high-level application developers as well as a rapid application-specific customization process to achieve better performance-energy trade-off of the resulting accelerators.

Figure 1 shows the nested loop acceleration framework. Given a specified compute intensive loop kernel and high-level design goals as well as constraints, the framework automatically tunes the design parameters including the SCGRA overlay configuration, compilation options and on chip communication specifically to the loop kernel through the SCGRA customization process. After the customization, corresponding SCGRA overlay based FPGA accelerator is generated and implemented through the SCGRA compilation process. Meanwhile, the drivers that are needed to utilize the resulting FPGA accelerator are generated according to the SCGRA customization. Then the high level application is updated and compiled to general purposed processor (GPP) through a conventional software compilation process. The application binary code generated in combination with the FPGA accelerator bitstream forms the final application that will be executed on the hybrid CPU-FPGA system during run time.

SCGRA customization process is the focus of this work where optimal design choices are decided. Since it involves exploration in a vast design space, it is critical to the design

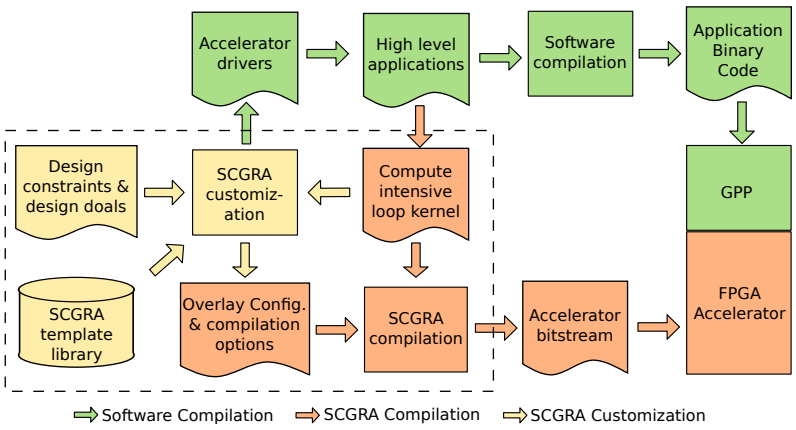


Fig. 1. Automatic nested loop acceleration framework

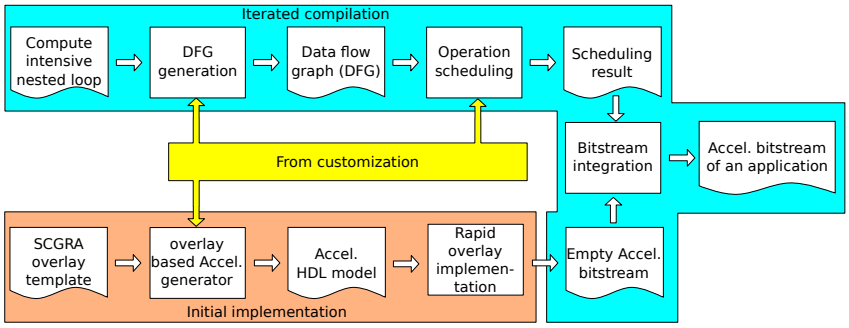


Fig. 2. High-productivity SCGRA overlay compilation, The whole diagram represents the initial compilation and it includes both the iterated compilation and the initial implementation.

productivity of the whole framework. In addition, it also determines the configurations of the resulting accelerators and thus affects the performance-energy trade-off of the final design. In this work, a dedicated customization method is proposed to meet both the requirements of the design productivity and performance-energy trade-off and it will be further illustrated in the following sections.

SCGRA compilation is responsible for mapping the high-level loop kernel to the physical FPGA bitstream through a specified SCGRA overlay. Figure 2 presents an overview of the compilation process and it includes two compilation flows for two typical development scenarios i.e. initial compilation and iterated compilation. In the first scenario when the specified SCGRA overlay is initially implemented on the target physical FPGA device, a standard implementation from HDL model may be time-consuming. Fortunately, a hardware-macro compilation technique [Michael Yue & Lemieux 2015] developed for the regular tiling architectures may significantly decrease the implementation time. In the second scenario when the specified SCGRA overlay has already been implemented, the bitstream can be reused as demonstrated in [Liu et al. 2013] and the compilation can be reduced to seconds.

4. SCGRA OVERLAY BASED FPGA ACCELERATION

In this work, parametric and scalable SCGRA overlay architectures, on-chip buffer structures and loop execution strategies have been developed for the nested loop acceleration

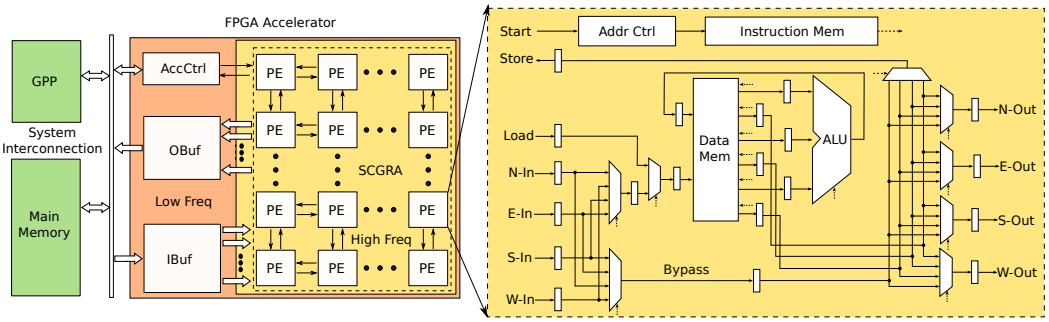


Fig. 3. SCGRA overlay based FPGA accelerator

framework. These design choices provide unique opportunity for application specific customization and help to achieve better performance-energy trade-off of the resulting accelerators.

4.1. SCGRA Overlay Based FPGA Accelerator

Figure 3 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 the communication 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.

PE is the key design element of the SCGRA overlay. It is simple yet highly pipelined which is beneficial to the hardware implementation. At the heart of PE is an ALU, which is supported by a multi-port data memory and an instruction memory. Three of the data memory's read ports are connected to the ALU as inputs, while the remaining ports are sent to the output multipliers for connection to neighboring PEs and the optional store paths external to the SCGRA overlay. At the same time, this data memory takes input from the ALU output, data arriving from neighboring PEs as well as from the optional IBuf loading path. The action of the PE is controlled by the AccCtrl unit that reads from the instruction memory. Finally a global signal from the AccCtrl block controls the start/stop of all PEs in the array.

ALU carries out the computations of the given applications. It can be easily customized to support operations of a specified application or a group of applications. The supported operations are fully pipelined in the ALU and may execute concurrently while they must complete in a deterministic number of cycles. Since ALU has only a single output port, the scheduler will ensure that there is never conflict at the output.

4.2. Loop execution on the accelerator

Figure 4 illustrates how the loop is executed on the FPGA accelerator. First of all, data flow graph (DFG) is extracted from the loop and then it is scheduled on to the SCGRA overlay based FPGA accelerator. Depending on how much the loop is unrolled and transformed to DFG, the DFG may be executed repeatedly until the end of the original loop. In addition, data transfers for multiple executions of the same DFG are batched into groups as shown in Figure 4. On the one hand, this technique is used to reduce the number of batching, which further helps to amortize the initial communication cost. On the other hand, it also results in larger on-chip memory overhead. The proposed customization framework can be used to make the right design choices to achieve an optimal design.

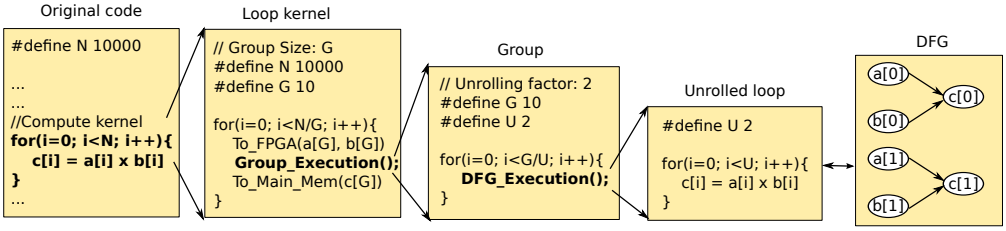


Fig. 4. Loop, group and DFG. The loop will be divided into groups. Each group will be partially unrolled and the unrolled part will be translated to DFG. IO transmission between FPGA and host CPU is performed in the granularity of a group.

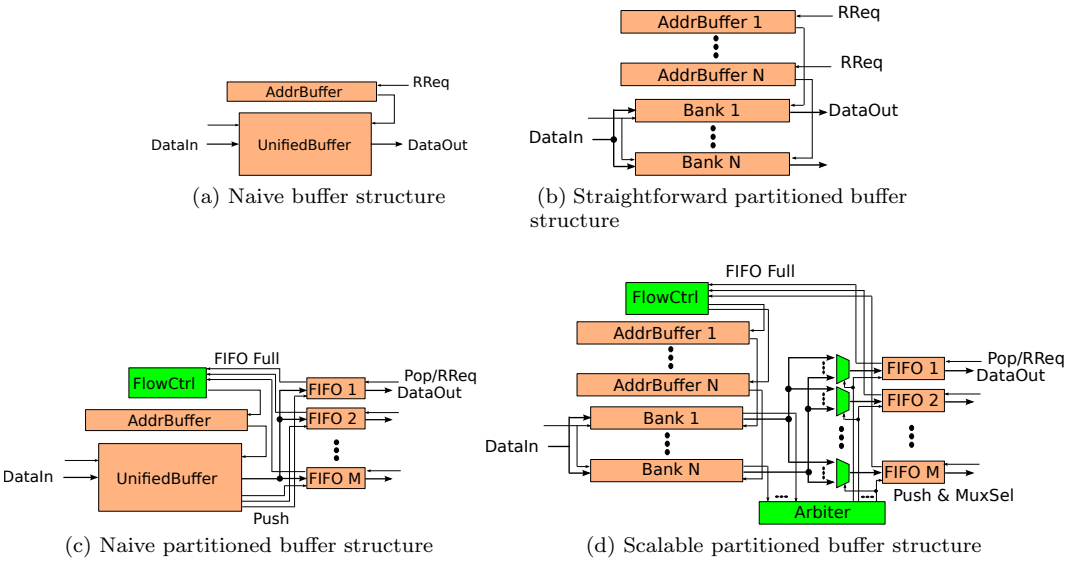
4.3. On-chip buffer structure

On-chip buffer is used to store data transmitted between the FPGA accelerator and main memory. (The input and output buffers are quite similar, so we just present the input buffer structure here for saving the space.) It is usually implemented with block RAM and a naive implementation as shown in Figure 5a provides limited bandwidth to the accelerator which may dramatically restrict the performance of the accelerator. A straightforward solution is to expose the primitive block RAM ports to the accelerator directly as shown in Figure 5b. However, the bandwidth is limited by the number of the primitive block RAM and it works only when the data is perfectly placed into different block RAM banks and each port of the accelerator accesses exactly the corresponding sub set of the data. In addition, we may want to have the input/output data of the same group stored in the input/output buffers as mentioned in previous section, while different input/output of the DFGs in the same group may have diverse layout pattern. For instance, one DFG may load its first input data from partitioned bank 0 while the following DFG may have to load its first input data from bank 1. As a result, the two DFGs will not be able to reuse the same lock-step computing on the SCGRA overlay. Although we may load/store input/output data for each DFG computation, the fine-grain data transmission between main memory and FPGA on-chip buffer is extremely expensive.

To solve this problem, we have introduced an additional buffering stage and developed a scalable on-chip buffer structure as shown in Figure 5c and Figure 5d. The first stage is the basic on-chip buffer as mentioned in Figure 5a and Figure 5b. It stores data of a single transmission between the accelerator and main memory. The additional buffer stage stores the input/output of the DFG computation on the CGRA overlay. It ensures the input/output data has exactly the same layout for all the DFG computation. Moreover, it can be implemented with arbitrary number of FIFOs providing sufficient bandwidth to the accelerator.

Although the additional buffer stage makes the computation data path even longer, data movement between the first buffer stage and the second buffer stage in input/output buffer and the DFG computing in the same group can be pipelined as illustrated in Figure 6. As long as the FIFOs are still available, FlowCtrl will keep the first stage buffer sending data to the second buffer stage with the pre-scheduled order which is obtained from the CGRA overlay scheduling and stored in the AddrBuffer. When the FIFOs have all the input for a single DFG execution, FlowCtrl will start the accelerator. When the accelerator completes a DFG computing, it will stop until it is activated next time. On the output side, the buffer and the accelerator work similarly. When the number of DFGs included in a group is big enough and DFG computing time is larger than the data movement cost, the cost of the additional buffer stages can be ignored.

To support the pipelining in Figure 6, the overall FIFO capacity is set to be twice the input/output of a single DFG. When the time consumed for data movement between the two buffer stages is shorter than the DFG computing time, Figure 5c can be used. Or else,



(a) Naive buffer structure

(b) Straightforward partitioned buffer structure

(c) Naive partitioned buffer structure

(d) Scalable partitioned buffer structure

Fig. 5. On-chip buffer structure

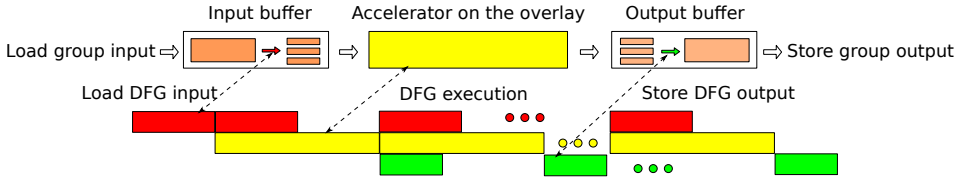


Fig. 6. Pipelined execution on the CGRA overlay

we may make use of all the potential bandwidth in the first buffer stage using the buffer in Figure 5d.

5. SCGRA OVERLAY BASED FPGA ACCELERATOR CUSTOMIZATION

Application-specific customization provides unique opportunity to improve the power-performance of the resulting accelerators. However, taking the system as a black box and exhaustively searching all the possible configurations may be inefficient and slow, which will result in low design productivity. In this work, by taking advantage of the regularity of the SCGRA overlay based FPGA accelerator, we can reduce the complex customization problem to a small DSE together with a simplified customization problem, and near optimal application-specific nested loop acceleration can be achieved rapidly.

5.1. Customization Framework

Figure 7 illustrates the overview of the customization framework. It can be roughly divided into two parts. In the first part, a sub DSE targeting loop execution time is performed and the feasible design space can be acquired. Since loop execution time is determined by the operation scheduling which merely depends on the loop unrolling factor and SCGRA size, the sub DSE is much simpler compared to the overall system DSE.

In the second part, each configuration in the feasible design space will be evaluated. Instead of using simulation based methods, analytical models are employed to estimate the accelerator metrics such as performance, overhead, energy consumption etc.. Because of the

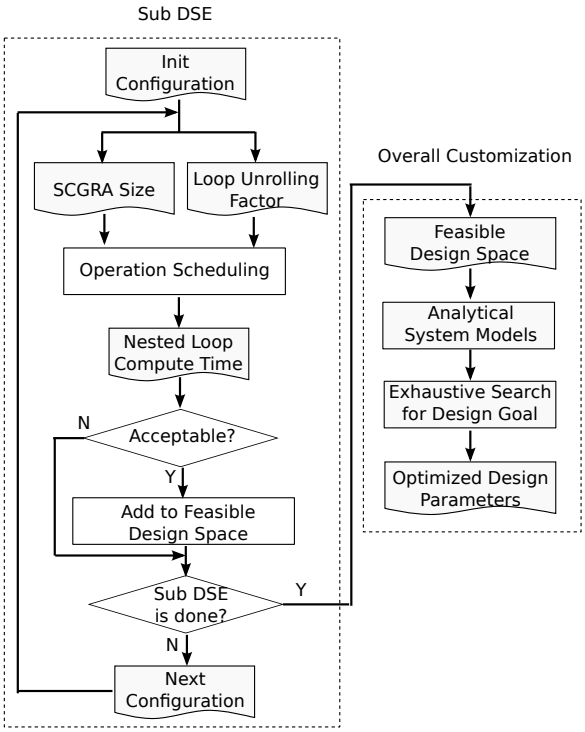


Fig. 7. System customization framework.

Table I. Design Parameters of Nested Loop Acceleration

Design Parameters		Denotation
Nested Loop Compilation	Loop Unrolling Factor	$\boldsymbol{u} = (u_0, u_1, \ldots)$
	Grouping Factor	$\boldsymbol{g} = (g_0, g_1, \ldots)$
Overlay Configuration	SCGRA Topology	Null, 2D Torus
	SCGRA Size	$r \times c$
	Instruction Mem	$imD \times imW$
	Data Mem	$dmD \times dmW$
	Input Buffer	$ibD \times ibW$
	Output Buffer	$obD \times obW$
	Input Address Buffer	$iabD \times iabW$
	Output Address Buffer	$oabD \times oabW$
	Operation Set	Null, fixed
	Implementation Frequency	f, fixed
	Pipeline Depth	Null, fixed

regularity of the SCGRA overlay, these analytical models are accurate. Even though the feasible design space is still large, it is fast to evaluate all the configurations in the feasible design space. After the evaluation process, customization for various design goals becomes trivial and the customized design parameters can be obtained immediately.

5.2. Customization Problem Formulation

In this section, we will formalize the customization problem of the nested loop acceleration on an SCGRA overlay based FPGA accelerator. Various design goals including performance, energy consumption and energy delay product (EDP) are used while energy consumption is taken as an example here.

Suppose $\mathbf{\Psi}$ represents the overall nested loop acceleration design space. $\boldsymbol{C} \in \mathbf{\Psi}$ represents a possible configuration in the design space and it includes a number of design parameters as listed in Table I. Assume that the loop to be accelerated has n nested levels and loop count can be denoted as $l = (l_1, l_2, ..., l_n)$. $R = (R_1, R_2, R_3, R_4)$ stands for the FPGA resource (i.e. BRAM, DSP, LUT and FF) that are available on a target FPGA and $Overhead(\boldsymbol{C}, i)$ denotes the four different types of FPGA resource overhead. $In(\boldsymbol{g})$ and $Out(\boldsymbol{g})$ stand for the amount of input and output of a group. Similarly, $In(\boldsymbol{u})$ and $Out(\boldsymbol{u})$ stand for the amount of input and output of a DFG. $DFGCompuTime(\boldsymbol{C})$ represents the number of cycles needed to complete the DFG computation. α_i and β_i are constant coefficients depending on target platform where $i = (1, 2, ...)$. With these denotations, the customization problem targeting minimum energy consumption can be formulated as follows:

Minimize

$$Energy(\boldsymbol{C}) = Power(\boldsymbol{C}) \times (\frac{RunTime(\boldsymbol{C})}{f}) \tag{1}$$

subject to

$$\begin{aligned} &Overhead(\boldsymbol{C}, i) \leq R_i, i = 1, 2, 3, 4 \\ &In(g) \leq ibD \\ &Out(g) \leq obD \\ &DFGCompuTime(\boldsymbol{C}) \leq imD \\ &\prod_{i=1}^{n} \frac{g_i}{u_i} \times In(u) \leq iabD \\ &\prod_{i=1}^{n} \frac{g_i}{u_i} \times Out(u) \leq oabD \end{aligned} \tag{2}$$

$RunTime(\boldsymbol{C})$ represents the number of cycles needed to compute the loop on the CPU-FPGA system. It consists of both the time consumed for computing on FPGA and communication between FPGA and host CPU, and it can be calculated using (3).

$$RunTime(\boldsymbol{C}) = CompuTime(\boldsymbol{C}) + CommuTime(\boldsymbol{C}) \tag{3}$$

Since the unrolled part of the loop will be translated to DFG and then scheduled to the SCGRA overlay. Thus the DFG computation time is essentially a function of $\mathbf{u}$, r and c, and it can also be denoted by $DFGCompuTime(\mathbf{u}, r, c)$. The nested loop is computed by repeating the same DFG execution, and the nested loop computation can be calculated using (4).

$$CompuTime(\boldsymbol{C}) = \prod_{i=1}^{n} \frac{l_i}{u_i} \times DFGCompuTime(\mathbf{u}, r, c) \tag{4}$$

DMA is typically used for the bulk data transmission. Communication cost per data can be modeled with a piecewise linear function and thus DMA latency can be calculated using (5).

$$DMA(x) = \begin{cases} (\alpha_1 \times x + \beta_1) \times x & : (x \leq T_0) \\ (\alpha_5 \times x + \beta_5) \times x & : (T_0 < x \leq T_1) \\ \alpha_6 \times x & : (x > T_1) \end{cases} \tag{5}$$

where x represents the amount of DMA transmission and T_0, T_1 stand for the turning points of the piecewise function. The communication time of the whole nested loop can be

calculated by (6).

$$CommuTime(\boldsymbol{C}) = \prod_{i=1}^{n} \frac{l_i}{g_i} \times (DMA(In(\mathbf{g})) + DMA(Out(\mathbf{g}))) \tag{6}$$

Power consumption of the FPGA accelerator consists of BRAM power, clock power, signal power and so on. Various experiments were done to measure the BRAM power and we can model BRAM power as (7). The rest part of the hardware resource mainly changes with the SCGRA size, so the power consumption can be roughly modeled with a linear equation of SCGRA size. Therefore, the power consumption of the accelerator can be modeled by (8).

$$Power\_BRAM(\boldsymbol{C}) = \alpha_7 \times r \times c \times Overhead(\boldsymbol{C}, 1) \tag{7}$$

$$Power(\boldsymbol{C}) = Power\_BRAM(\boldsymbol{C}) + \alpha_8 \times r \times c + \beta_6 \tag{8}$$

Hardware overhead on FPGA mainly includes DSP, LUT, FF and BRAM (block RAM). LUT, FF and DSP overhead can be roughly estimated with a linear function of SCGRA size and can be calculated using (9). BRAM overhead which is usually the overhead bottleneck for SCGRA overlay based FPGA accelerator design can be calculated by (10).

$$Overhead(\boldsymbol{C}, i) = \alpha_i \times r \times c + \beta_i, (i = 2, 3, 4) \tag{9}$$

$$\begin{aligned} Overhead(\boldsymbol{C}, 1) =& r \times c \times (imD \times imW + dmD \times dmW) + \\ & (ibD \times ibW + obD \times obW) + (iabD \times iabW + oabD \times oabW) \end{aligned} \tag{10}$$

5.3. Proposed Customization Method

Almost all the design metrics can be estimated by the analytical models when the operation scheduling result is available. Since the scheduling result merely depends on unrolling factor and the SCGRA overlay size, we can separate the scheduling from the customization problem and perform a sub DSE to reduce the large design space to a smaller feasible design space. Within the feasible design space, we further search the optimized design parameter for the entire system customization.

Suppose Φ denotes the feasible design space. ϵ indicates the percentage of the performance benefit obtained by the increase of loop unrolling or SCGRA size. It is a user defined threshold and must be small enough to prune the configurations that are inappropriate. The configurations in Φ must satisfy (11) and (12).

$$\begin{aligned} &\forall \boldsymbol{C} = (..., \boldsymbol{u}, r, c, ...) \in \Phi, \boldsymbol{C'} = (..., \boldsymbol{u'}, r', c', ...) \in \Phi, (r + 1 == r' \text{ and } c == c') \text{ or} \\ &(r == r' \text{ and } c + 1 == c') : \\ &\frac{CompuTime(\boldsymbol{C}) - CompuTime(\boldsymbol{C'})}{CompuTime(\boldsymbol{C})} > \epsilon \end{aligned} \tag{11}$$

$$\begin{aligned} &\forall \boldsymbol{C} = (..., \boldsymbol{u}, r, c, ...) \in \Phi, \boldsymbol{C'} = (..., \boldsymbol{u'}, r, c, ...) \in \Phi, \boldsymbol{u} \text{ and } \boldsymbol{u'} \text{ are consecutive unrolling factors :} \\ &\frac{CompuTime(\boldsymbol{C}) - CompuTime(\boldsymbol{C'})}{CompuTime(\boldsymbol{C})} > \epsilon \end{aligned} \tag{12}$$

Each configuration $\boldsymbol{C} \in \Phi$ must have the corresponding scheduling result known, and thus the computation time of the loop kernel, minimum instruction memory depth and data buffer depth are available as well. Then we can further evaluate energy consumption of each feasible configuration using the models built in previous section and acquire the optimized configuration.

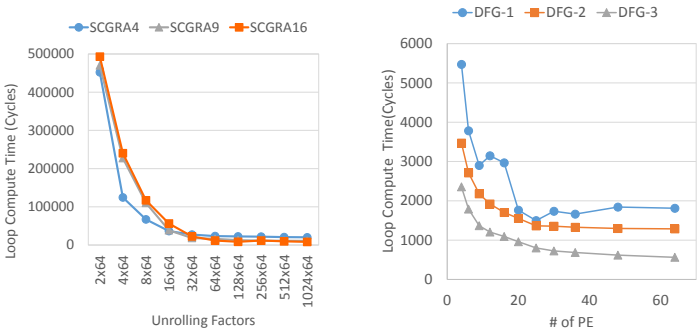


Fig. 8. The design parameters typically have monotonic influence on the loop computation time and the computation time benefit degrades with the increase of the design parameter. (a) SCGRA Size, (b) Unrolling Factor

In order to achieve the desired customization, we must make sure the feasible design space acquired from (11) and (12) can always cover the configuration that produces the optimal customization. A brief proof is presented as follows.

$\forall \boldsymbol{C'} \notin \Phi$, there must be a configuration $\boldsymbol{C}$ that fails (11) or (12). Suppose $\boldsymbol{C'} = (..., \boldsymbol{u}, r+1, c, ...)$ and $\boldsymbol{C} = (..., \boldsymbol{u}, r, c,)$. Thus we can conclude that

$$CompuTime(\boldsymbol{C'}) \geq (1-\epsilon) \times CompuTime(\boldsymbol{C}) \tag{13}$$

Since ϵ can be small and unrolling factor is not changed, $DFGCompuTime(\boldsymbol{C'})$ roughly equals to $DFGCompuTime(\boldsymbol{C})$ and therefore the instruction memory depth will not change. However, the increase of row size of the SCGRA overlay will result in significant overhead of BRAM and power consumption. Thus $Power(\boldsymbol{C'}) \geq Power(\boldsymbol{C})$. In addition, it will reduce the BRAM budget for on-chip buffer, which means $CommuTime(\boldsymbol{C'}) \geq CommuTime(\boldsymbol{C})$. According to (1) and (3), it is clear that $Energy(\boldsymbol{C'}) \geq Energy(\boldsymbol{C})$ and any configuration that is pruned during the sub DSE will not be an optimized configuration. Similarly, we can also draw the same conclusion when a different occasion in (11) and (12) appears.

In addition, a series of experiments on Zedboard [Avnet 2014] as shown in Figure 8 demonstrate that SCGRA size and unrolling factor present a clear monotonic influence on the loop compute time. The performance benefit of loop unrolling and increase of SCGRA size drops gradually and thus we can conclude that (14).

$$CompuTime(\boldsymbol{C_1}) - CompuTime(\boldsymbol{C_2}) > CompuTime(\boldsymbol{C_2}) - CompuTime(\boldsymbol{C_3}) \tag{14}$$

where $\boldsymbol{C_1} = (..., x1, ...)$, $\boldsymbol{C_2} = (..., x2, ...)$, $\boldsymbol{C_3} = (..., x3, ...)$ and $x1$, $x2$, $x3$ are three increasingly consecutive configurations of loop unrolling factor or row or column of the SCGRA overlay.

In other words, if $\boldsymbol{C_1}$ fails to be a feasible configuration, we can be sure that $\boldsymbol{C_2}$ and $\boldsymbol{C_3}$ will fail as well. With this observation, we can further simplify the sub design space exploration. The simplified sub DSE will be detailed in next section.

5.4. Sub Design Space Exploration

To acquire the feasible design space, we developed a dedicated sub DSE targeting nested loop computation time. Since the loop computation time merely depends on the SCGRA overlay size and the loop unrolling factor, the sub DSE is much simpler compared to the overall customization problem. In addition, we can further simplify the sub DSE with the observations shown in Figure 8. Whenever a configuration fails the sub DSE condition, all

the configurations which are larger on one design parameter and remain the same on the rest design parameters can be safely pruned. Thus a branch and bound algorithm as detailed in Algorithm 1 is used to efficiently explore the sub design space.

Algorithm 1 Sub Design Space Exploration.

```
procedure
  Initialize r = 2, c = 2, u = (1, 1, ...), feasible design space Φ = ∅, C = (..., r, c, u, ...),
  maximum SCGRA overlay r_Max × c_Max.
  while r < r_Max do
    while c < c_Max do
      while u is not fully unrolled do
        Generate DFG with u
        DFG Scheduling with configuration C
        Estimate performance CompuTime(C)
        Get neighbor C' ∈ Φ with smaller loop unrolling
        if C' exists and Revenue(C, C') ≤ ε then
          Break
        else
          Add C to Φ
        end if
        update u with larger neighbor unrolling factor
      end while
      Get neighbor C'' ∈ Φ with smaller column size
      if C'' exists and Revenue(C, C'') ≤ ε then
        Break
      end if
      c = c + 1
    end while
    Get neighbor C''' ∈ Φ with smaller row size
    if C''' exists and Revenue(C, C''') ≤ ε then
      Break
    end if
    r = r + 1
  end while
end procedure

procedure Revenue(C, C')
  return (CompuTime(C') − CompuTime(C)) / CompuTime(C')
end procedure
```

6. EXPERIMENTS AND RESULTS

The experiments mainly include two parts. In the first part, we analyze the implementations of the SCGRA overlay based FPGA accelerators with different configurations to demonstrate the regularity of the SCGRA overlay based FPGA accelerators. In the second part, we benchmark the efficiency and quality of the proposed customization framework.

6.1. Experiment Setup

All the run time was obtained using 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

Table II. SCGRA Based FPGA Accelerator Configuration

Group	Size	Inst. Rom	Data Mem	IBuf /OBuf	Addr Buf
SCGRA1	2x2, 3x2, 3x3, 4x3, 4x4, 5x4	1kx72	256x32	2kx32	4kx16
SCGRA2	2x2, 3x2, 3x3, 4x3, 4x4	2kx72	256x32	2kx32	4kx16
SCGRA3	2x2, 3x2, 3x3	4kx72	256x32	2kx32	4kx16

computation system. Vivado 2013.3 was used for the HLS based design and PlanAhead 14.7 was used for the SCGRA overlay based design.

The overlay implementations on Zedboard typically run at 200MHz. We conducted various experiments and found that the implementation frequency is almost the same for all the different overlay configurations. The power consumption used in this work was obtained from XPower which is part of the Xilinx design suite.

6.2. SCGRA Overlay Implementation Analysis

In order to analyze the overhead and power of the SCGRA overlay based FPGA accelerators, we had three groups of accelerators (SCGRA1, SCGRA2, SCGRA3) implemented on Zedboard. The configurations are detailed in Table II. Despite of the diverse configurations, all the implementations could meet 200MHz timing constrain. With this timing constraint, hardware overhead and power consumption are analyzed, which are described in the following sub sections.

6.2.1. Hardware Overhead. Figure 9 shows the relation between the four types of hardware resource overhead and SCGRA overlay size. It can be found that FF, LUT and DSP overhead do not change much with the memory configurations and they present good linearity to the overlay size, thus they can be estimated with linear models. BRAM overhead depends on both the overlay size and the memory configurations, and single variable linear model will not work for the estimation. Fortunately, it can be calculated precisely with the memory configurations.

6.2.2. Power Consumption. According to the power decomposition in Xpower, the power consumption of an FPGA design includes signal power, clock power, BRAM power and so on. To simplify the power model of the SCGRA overlay based FPGA accelerator, we divide the power consumption into BRAM power and base system power which essentially includes the power consumption of the rest part of the system. As shown in Figure 10, the base system power exhibits good linearity over the SCGRA overlay size while the BRAM power is near linear to the BRAM overhead. Therefore, both of them can be easily estimated with linear models.

6.3. Proposed Design Framework Evaluation

In this work, we take four applications including Matrix Multiplication (MM), FIR, Kmean(KM) and Sobel Edge Detector (SE) as our benchmark. The configurations of the benchmark are detailed in Table III. In order to evaluate the efficiency and quality of the proposed design framework, we implemented the benchmark using both the proposed two-step customization (TS) method and an exhaustive search (ES) method. Then we compared the Pareto-optimal curves acquired using both methods. In addition, we implemented the

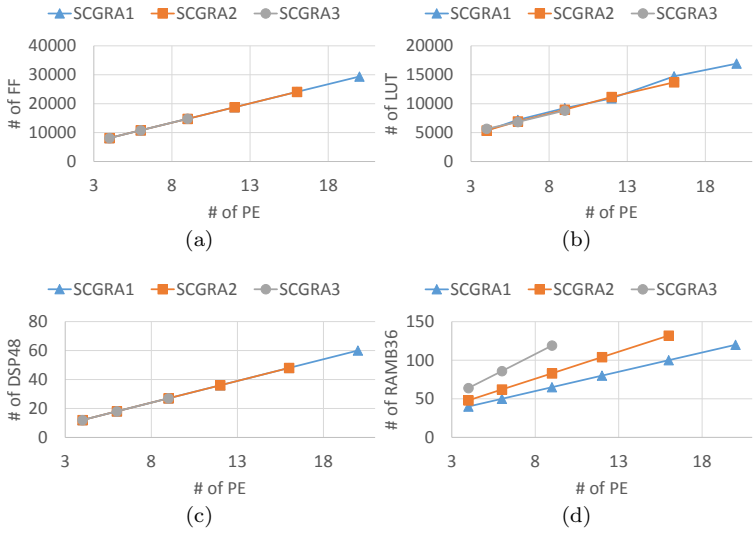


Fig. 9. Relation between the accelerator overhead and overlay size, (a) FF overhead, (b) LUT overhead, (c)DSP overhead, (d)BRAM overhead

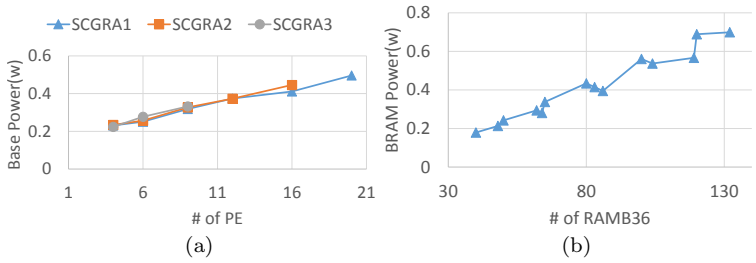


Fig. 10. Power consumption of the SCGRA overlay based FPGA accelerator, (a) Base system power including DSP power, clock power, etc., (b) BRAM power

Table III. 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)	$1024 \times 4 \times 2$

benchmark using Vivado HLS with moderate manual optimization and compared the implementations with the ones obtained using the proposed customization framework.

6.3.1. Customization Methods Comparison. Figure 11 shows the DSE time of both the TS DSE and ES DSE. The proposed TS DSE is around 100x faster than the ES DSE on average. In particular, ES DSE is extremely slow on MM which has three levels of loop with relatively large loop count and thus larger design space. Though TS DSE also needs longer time to

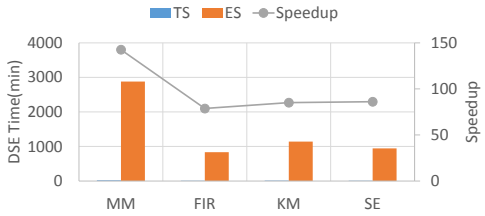


Fig. 11. DSE time of the benchmark using both TS and ES

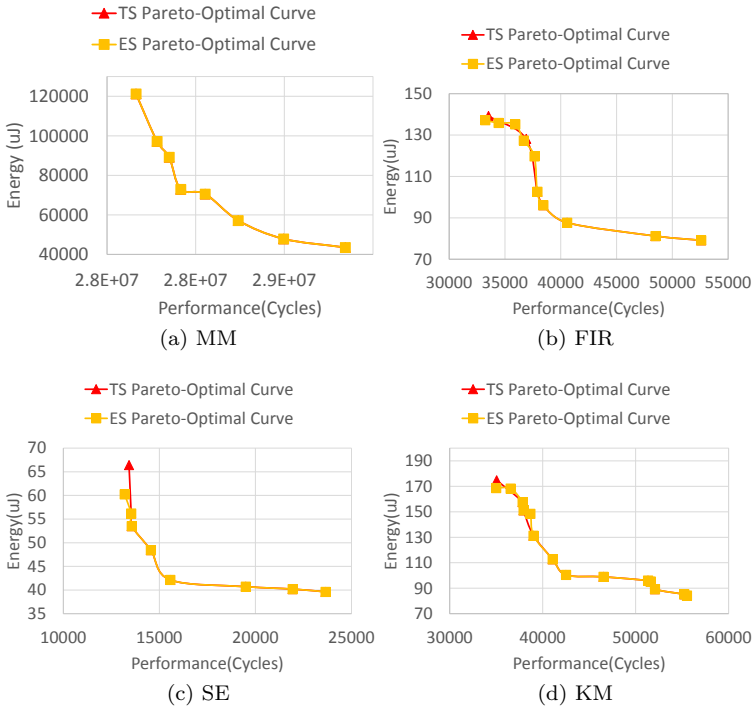


Fig. 12. Performance-Energy Pareto-optimal curve

complete the DSE of MM, it skips most of the unfeasible configurations and the runtime is less sensitive to the size of the design space.

In order to demonstrate the quality of proposed framework, we presented the Pareto-optimal curves acquired from both DSE methods. As shown in Figure 12, the Pareto-optimal curves obtained using the two DSE methods are quite close to each other. Since TS DSE may prune the design options that involve a larger overlay size and better performance according to the sub DSE metric, the Pareto-optimal curves may deviate slightly at the higher performance area. Fortunately, this can be improved by lowering the user defined metric ϵ while affording longer DSE time. When we customize the design for minimum energy consumption, TS DSE achieves the optimal design in all the benchmarks.

6.3.2. Design Tools Comparison. The proposed design framework will not be as useful if the performance of the generated system is not at least on par with similar systems created with conventional HLS tools. For that, we further implemented the benchmarks using Vivado HLS with moderate effort. The loops in the benchmark were unrolled as much as possible and the input/output buffers were set as large as possible. The accelerators generated using

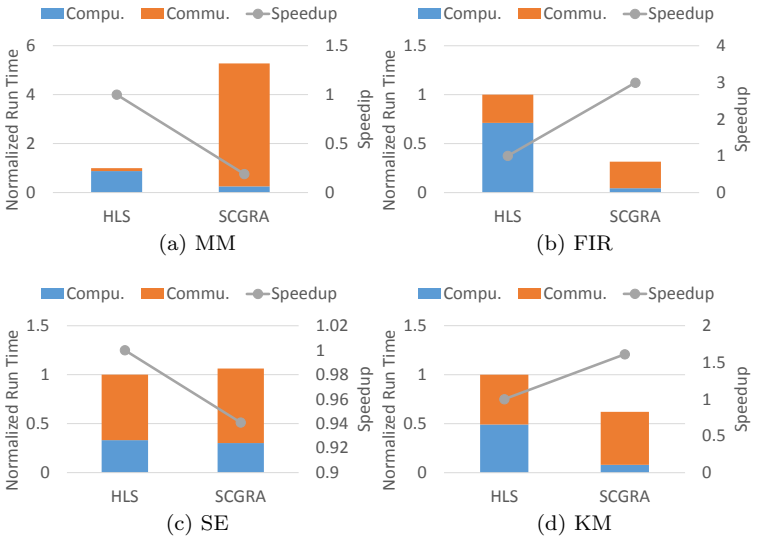


Fig. 13. Performance comparison between the implementations using HLS and the proposed automatic customization framework. All the run time is normalized to that of the HLS based implementations.

HLS run at 100MHz, the accelerators built on top of SCGRA overlay have the computation array running at 200MHz and the communication fabrics running at 100MHz. Then the accelerators generated using both tools were compared. Since similar comparison has already been done in previous work [Liu et al. 2013], we just brief the performance comparison for a quick reference.

Figure 13 shows the performance comparison of the implementations using both design tools. The proposed customization framework utilizes SCGRA overlay as the backbone and it can't afford large BRAM blocks for input/output buffers. As a result, the communication time is larger especially when there is a lot of data reuse between consecutive data transmissions. For instance, HLS based design for MM can afford a 32k-word input buffer storing all the input data. However, the SCGRA overlay based design can only provide a 4k-word input buffer and it needs to transmit the same data multiple times to complete the matrix multiplication (Note that it is possible to partly reuse data transmissions of different groups, but it is not supported in the experiments). Direct HLS can't support large loop unrolling due to the DSP resource constrain. SCGRA overlay based accelerator has a lot of distributed intermediate buffers and can accommodate larger loop unrolling. Therefore, the accelerators generated using the proposed framework provides competitive pure computation time and overall performance especially for compute intensive loops.

7. LIMITATIONS AND FUTURE WORK

Despite the promising advantages of the proposed customization framework, there are a few limitations that must be acknowledged and hopefully addressed in future work.

First of all, the proposed design framework mainly targets compute intensive nested loops. As such, it is not a generic method to perform HLS on random logic.

Secondly, the grouping strategy used in the framework makes the IO buffer partitioning difficult, because the input/output data of the DFGs in the same group may be located in diverse partitioned banks and the DFG scheduling may not be reused among these DFGs. This limitation will be fixed by introducing an additional buffering stage in future. Currently,

input/output buffer partitioning is not supported in this framework and the accelerators developed only have a single input buffer and a single output buffer.

Thirdly, data transmission and the computation are sequentially repeated to complete the nested loop on the accelerator. However, it is possible to pipeline the two processing stages for better performance while performance models need to be adjusted accordingly.

8. CONCLUSION

In this work, we have presented an automatic nested loop acceleration framework based on a homogeneous SCGRA overlay built on top of off-the-shelf FPGA devices. Given high level user constraints and design goals, the framework performs an intensive system customization specifically to a nested loop. When the optimized design configuration is acquired after the customization, the HDL model of the resulting accelerator as well as the communication interface can be generated. Finally, the accelerator and the software running on the host CPU can further be compiled rapidly to the hybrid CPU-FPGA system.

Particularly, by taking advantage of the regularity of the SCGRA overlay, various metrics of the accelerator such as overhead and energy consumption can be effectively evaluated via analytical models. This unique characteristic allows us to reduce the complex system customization to a simpler sub design space exploration and a straightforward search over the sub design space. Compared to an exhaustive search through the entire design space, the proposed two-step customization method reduces runtime by 2 orders of magnitude on average while achieving quite similar energy-performance Pareto-optimal curve and the same customized architecture that produces minimum energy consumption. When compared to the implementations using a state-of-the-art HLS tool, the customized design exhibits competitive performance.

REFERENCES

Al-Dujaili, A., Deragisch, F., Hagiescu, A., & Wong, W.-F. (2012). Guppy: A gpu-like soft-core processor. In *Field-Programmable Technology (FPT), 2012 International Conference on*, (pp. 57–60).

Avnet (2014). Zedboard. http://www.zedboard.org/. [Online; accessed 25-June-2014].

Brant, A. & Lemieux, G. (2012). Zuma: An open fpga overlay architecture. In *Field-Programmable Custom Computing Machines (FCCM), 2012 IEEE 20th Annual International Symposium on*, (pp. 93–96).

Canis, A., Choi, J., Aldham, M., Zhang, V., Kammoona, A., Anderson, J. H., Brown, S., & Czajkowski, T. (2011). 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, (pp. 33–36)., New York, NY, USA. ACM.

Capalija, D. & Abdelrahman, T. (2009). An architecture for exploiting coarse-grain parallelism on fpgas. In *Field-Programmable Technology, 2009. FPT 2009. International Conference on*, (pp. 285–291).

Capalija, D. & Abdelrahman, T. (2013). A high-performance overlay architecture for pipelined execution of data flow graphs. In *Field Programmable Logic and Applications (FPL), 2013 23rd International Conference on*, (pp. 1–8).

Capalija, D. & Abdelrahman, T. S. (2014). Tile-based bottom-up compilation of custom mesh-of-functional-units fpga overlays. In *Field Programmable Logic and Applications (FPL), 2014 24th International Conference on*, (pp. 1–8). IEEE.

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

Cheah, H. Y., Fahmy, S., & Maskell, D. (2012). idea: A dsp block based fpga soft processor. In *Field-Programmable Technology (FPT), 2012 International Conference on*, (pp. 151–158).

Chung, E., Milder, P., Hoe, J., & Mai, K. (2010). Single-chip heterogeneous computing: Does the future include custom logic, fpgas, and gpgpus? In *Microarchitecture (MICRO), 2010 43rd Annual IEEE/ACM International Symposium on*, (pp. 225–236).

Compton, K. & Hauck, S. (2002). Reconfigurable computing: a survey of systems and software. *ACM Computing Surveys (csuR)*, *34*(2), 171–210.

Cong, J., Liu, B., Neuendorffer, S., Noguera, J., Vissers, K., & Zhang, Z. (2011). High-level synthesis for FPGAs: From prototyping to deployment. *Computer-Aided Design of Integrated Circuits and Systems, IEEE Transactions on*, *30*(4), 473–491.

Corporation, A. (2015a). Quartus II 14.0 handbook. https://www.altera.com/en_US/pdfs/literature/hb/qts/quartusii_handbook.pdf. [Online; accessed 18-March-2015].

Corporation, X. (2015b). Command line tools user guide. http://www.xilinx.com/support/documentation/sw_manuals/xilinx14_7/devref.pdf. [Online; accessed 18-March-2015].

Ferreira, R., Vendramini, J., Mucida, L., Pereira, M., & Carro, L. (2011). An fpga-based heterogeneous coarse-grained dynamically reconfigurable architecture. In *Proceedings of the 14th international conference on Compilers, architectures and synthesis for embedded systems*, (pp. 195–204). ACM.

Goeders, J. B., Lemieux, G. G., & Wilton, S. J. (2011). Deterministic timing-driven parallel placement by simulated annealing using half-box window decomposition. In *Reconfigurable Computing and FPGAs (ReConFig), 2011 International Conference on*, (pp. 41–48). IEEE.

Grant, D., Wang, C., & Lemieux, G. G. (2011). 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, (pp. 123–132)., New York, NY, USA. ACM.

Kissler, D., Hannig, F., Kupriyanov, A., & Teich, J. (2006). A dynamically reconfigurable weakly programmable processor array architecture template. In *ReCoSoC*, (pp. 31–37).

Korf, S., Cozzi, D., Koester, M., Hagemeyer, J., Porrmann, M., Ruckert, U., & Santambrogio, M. D. (2011). Automatic hdl-based generation of homogeneous hard macros for fpgas. In *Field-Programmable Custom Computing Machines (FCCM), 2011 IEEE 19th Annual International Symposium on*, (pp. 125–132). IEEE.

Kurek, M., Becker, T., Chau, T. C., & Luk, W. (2014). Automating optimization of reconfigurable designs. In *Field-Programmable Custom Computing Machines (FCCM), 2014 IEEE 22nd Annual International Symposium on*, (pp. 210–213).

LaForest, C. E. & Steffan, J. G. (2012). Octavo: An fpga-centric processor family. In *Proceedings of the ACM/SIGDA International Symposium on Field Programmable Gate Arrays*, FPGA '12, (pp. 219–228)., New York, NY, USA. ACM.

Lavin, C., Nelson, B., & Hutchings, B. (2013). Improving clock-rate of hard-macro designs. In *Field-Programmable Technology (FPT), 2013 International Conference on*, (pp. 246–253). IEEE.

Lebedev, I., Cheng, S., Doupnik, A., Martin, J., Fletcher, C., Burke, D., Lin, M., & Wawrzynek, J. (2010). Marc: A many-core approach to reconfigurable computing. In *Reconfigurable Computing and FPGAs (ReConFig), 2010 International Conference on*, (pp. 7–12).

Liu, C., Lin, C. Y., & So, H. K.-H. (2013). A soft coarse-grained reconfigurable array based high-level synthesis methodology: Promoting design productivity and exploring extreme fpga frequency. In *Field-Programmable Custom Computing Machines (FCCM), 2013 IEEE 21st Annual International Symposium on*, (pp. 228–228).

Liu, H.-Y. & Carloni, L. (2013). On learning-based methods for design-space exploration with high-level synthesis. In *Design Automation Conference (DAC), 2013 50th ACM / EDAC / IEEE*, (pp. 1–7).

Michael Yue, D. K. & Lemieux, G. (2015). Rapid overlay builder for xilinx fpgas. In *Field-Programmable Custom Computing Machines (FCCM), 2015 IEEE 23rd Annual International Symposium on*, (pp. 1–8).

Miniskar, N. R., Kohli, S., Park, H., & Yoo, D. (2014). Retargetable automatic generation of compound instructions for cgra based reconfigurable processor applications. In *Compilers, Architecture and Synthesis for Embedded Systems (CASES), 2014 International Conference on*, (pp. 1–9). IEEE.

Moctar, Y. O. M. & Brisk, P. (2014). Parallel fpga routing based on the operator formulation. In *Proceedings of the The 51st Annual Design Automation Conference on Design Automation Conference*, (pp. 1–6). ACM.

Mulpuri, C. & Hauck, S. (2001). Runtime and quality tradeoffs in fpga placement and routing. In *Proceedings of the 2001 ACM/SIGDA ninth international symposium on Field programmable gate arrays*, (pp. 29–36). ACM.

Schafer, B. C., Takenaka, T., & Wakabayashi, K. (2009). Adaptive simulated annealer for high level synthesis design space exploration. In *VLSI Design, Automation and Test, 2009. VLSI-DAT'09. International Symposium on*, (pp. 106–109). IEEE.

Schafer, B. C. & Wakabayashi, K. (2012). Divide and conquer high-level synthesis design space exploration. *ACM Trans. Des. Autom. Electron. Syst.*, *17*(3), 29:1–29:19.

Sengupta, I. & Bhatia, N. (1997). A genetic algorithm approach to high-level synthesis of digital circuits. *International journal of systems science*, *28*(5), 517–522.

Severance, A. & Lemieux, G. (2013). Embedded supercomputing in fpgas with the vectorblox mxp matrix processor. In *Hardware/Software Codesign and System Synthesis (CODES+ISSS), 2013 International Conference on*, (pp. 1–10).

Tessier, R. & Burleson, W. (2001). Reconfigurable computing for digital signal processing: A survey. *Journal of VLSI signal processing systems for signal, image and video technology*, *28*(1-2), 7–27.

Unnikrishnan, D., Zhao, J., & Tessier, R. (2009). Application specific customization and scalability of soft multiprocessors. In *Field Programmable Custom Computing Machines, 2009. FCCM '09. 17th IEEE Symposium on*, (pp. 123–130).

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

Yiannacouras, P., Steffan, J., & Rose, J. (2007). Exploration and customization of fpga-based soft processors. *Computer-Aided Design of Integrated Circuits and Systems, IEEE Transactions on*, *26*(2), 266–277.

Yiannacouras, P., Steffan, J. G., & Rose, J. (2009). 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, (pp. 97–106)., New York, NY, USA. ACM.

Yu, Colin Lin. So, H. K.-H. (2012). 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.

Zhang, Z., Fan, Y., Jiang, W., Han, G., Yang, C., & Cong, J. (2008). Autopilot: A platform-based esl synthesis system. In *High-Level Synthesis* (pp. 99–112). Springer.

Zhou, L., Liu, D., Zhang, J., & Liu, H. (2014). Application-specific coarse-grained reconfigurable array: architecture and design methodology. *International Journal of Electronics*, (ahead-of-print), 1–14.