Automatic Nested Loop Acceleration on FPGAs Using Soft Coarse-Grained Reconfigurable Array Overlay

ABSTRACT

Offloading compute intensive nested loops to execute on FPGA accelerators have been demonstrated by numerous researchers as an effective performance enhancement technique across numerous application domains. However, the relatively low design productivity caused by compiling high level loops to FPGA accelerators and customizing the resulting FPGA accelerators to specific applications remains a major challenge to the wide adoption of FPGAs as computing devices.

To address the productivity and performance problems, an automatic nested loop acceleration framework on a hybrid CPU-FPGA system is presented. It adopts a regular soft coarse-grained reconfigurable array (SCGRA) overlay as the backbone of the FPGA accelerator and can be implemented much faster than a conventional high level synthesis (HLS) design flow. Most importantly, it automatically customizes not only the overlay architectural design parameters but also high level compilation options and communication interfaces between the accelerator and host CPU, which completely shields the hardware design details and makes it accessible to high level application designers. In addition, by taking advantage of the regularity of the SCGRA overlay, a straightforward yet effective customization method is developed. According to the experiments, the proposed customization method can achieve similar Pareto-optimal curve to that acquired by using an exhaustive search while its runtime is around two orders of magnitude shorter. Experiments also show that the customized accelerators achieve competitive performance compared to the implementations using off-the-shelf HLS design tools.

1. INTRODUCTION

Offloading compute intensive nested loops to FPGA accelerators has been demonstrated by many researchers to be an effective solution for performance enhancement across many application domains [10]. However, the relatively low productivity in developing FPGA-based compute applications remains a major obstacle that hinders their widespread employment [12].

To address this productivity challenge, researchers have recently turned to the use of virtual FPGA overlay architectures and high-level compilation tools that, when combined properly, are able to produce high-performance accelerators at near software compilation speed [17, 3, 6, 15, 18, 28]. Not only have these early works illustrated the promise of using overlays to improve productivity, they have also highlighted a unique potential of overlays — By customizing the architectures of these *virtual* yet often regular overlays for a target user design, in theory, it is possible to significantly improve the power-performance of the resulting accelerator. In practice, however, navigating through a labyrinth of architectural and compilation parameters to fine-tune an accelerator's power-performance 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 address both the design productivity and performance challenge, we have developed a soft CGRA (SCGRA) overlay based automatic nested loop acceleration framework targeting a hybrid CPU-FPGA system. In this framework, nested loops expressed in high-level languages are compiled and executed on the SCGRA overlay 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 as a performance enhancing technique. 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.

Particularly, by taking advantage of the regularity of the SCGRA overlay, a multitude of design metrics such as performance and energy consumption can be accurately estimated using analytical models once the overlay scheduling result is available. While the overlay scheduling depends on much less design parameters, the overall customization framework can be dramatically simplified. Moreover, the regular tiling SCGRA overlay allows prompt implementation by using macro-based compilation techniques [39] and efficient bitstream reuse [28] during design iterations of an application. As a result, the overall design automation framework with rapid back-end compilation and fast application specific customization ensures both 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 the SCGRA overlay and hardware-software development. The result is comparable to an exhaustive design space search while it runs at a fraction of time.
- We have developed a *rapid* hardware-software compilation framework for a hybrid CPU-FPGA platform based on a SCGRA overlay. The compilation speed is high while the resulting system's performance is competitive.

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

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 [27], parallel compilation [26, 16, 13, 14] and using hard-macro techniques [22, 19] 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 [35, 4, 40]. 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 [39, 7].

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 [29, 31], learning-based methods [24, 8], divide and conquer algorithm [30] and a calibration free algorithm [20] 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 [17, 3], array-of-FUs [6, 15], soft CGRA [18, 28], soft GPU [1], vector processors[37, 32] to configurable processors or multi-core processors [34, 23, 36, 5, 21, 9] 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 [33, 11]. Most importantly, it allows both rapid compilation by taking advantage of the overlays' tiling structure [39] and efficient bitstream reuse within the design iterations of an application [28], 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 [41, 25] 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 [38] 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

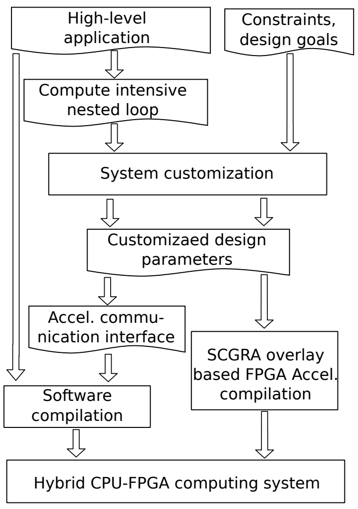

High-level application
Constraints, design goals
Compute intensive nested loop
System customization
Customizaed design parameters
Accel. commu-nication interface
SCGRA overlay based FPGA Accel. compilation
Software compilation
Hybrid CPU-FPGA computing system

Figure 1: Automatic nested loop acceleration framework

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 developed an automatic nested loop acceleration framework as shown in Figure 1. To make the framework accessible to high-level application developers, the application description and the design constraints as well as design goals must stay high-level. Design constraints may include various metrics such as FPGA overhead, power consumption and energy consumption. Design goals can be minimum energy consumption, minimum EDP or minimum execution time etc., and Pareto-optimal curves such as energy-performance trade-off are provided when there is no specified design goals.

To achieve the desired design goals and constraints, various design parameters including overlay architectural parameters such as row and column size, input buffer depth etc. and compilation options such as loop unrolling factor allow customization specifically to a user application. When the optimal design parameters are determined by the customization process, SCGRA overlay based FPGA accelerator as well as the communication interface between host CPU and the accelerator will be generated. Finally, the generated FPGA accelerator and software are compiled to the CPU-FPGA system rapidly.

3.1 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 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 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.

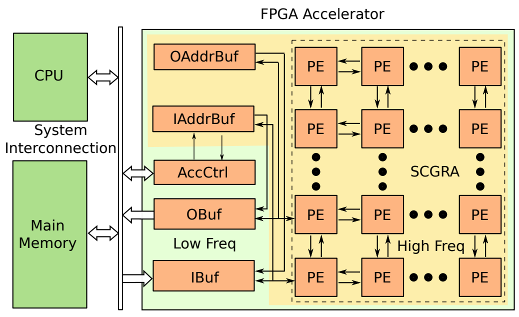
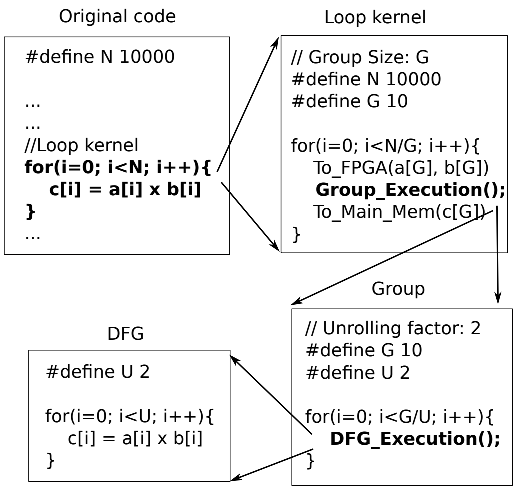


Figure 2: SCGRA overlay based FPGA accelerator



```
#define N 10000

...
...
//Loop kernel
for(i=0; i<N; i++){
    c[i] = a[i] x b[i]
}
...
```

```
// Group Size: G
#define N 10000
#define G 10

for(i=0; i<N/G; i++){
    To_FPGA(a[G], b[G])
    Group_Execution();
    To_Main_Mem(c[G])
}
```

```
// Unrolling factor: 2
#define G 10
#define U 2

for(i=0; i<G/U; i++){
    DFG_Execution();
}
```

```
#define U 2

for(i=0; i<U; i++){
    c[i] = a[i] x b[i]
}
```

Figure 3: 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.

3.2 Loop Execution On SCGRA Overlay Based FPGA Accelerator

Figure 3 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 3. 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.

3.3 SCGRA Overlay Compilation Framework

Combining the existing compilation techniques in [39, 28], we built a high productivity compilation framework targeting nest loop acceleration using SCGRA overlay. The framework is presented in Figure 4 and it basically consists of two compilation paths as shown in the figure.

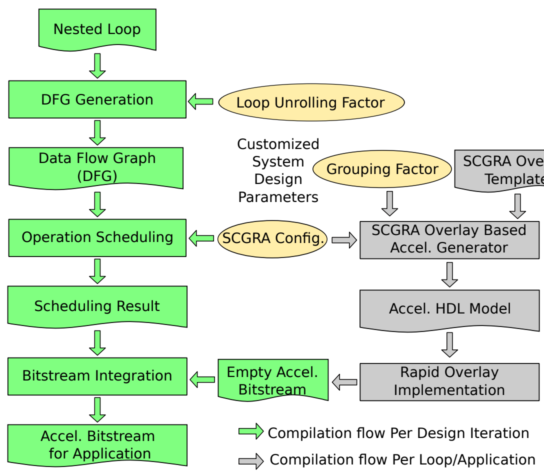


Figure 4: A high productivity SCGRA compilation framework

When a nested loop is initially compiled to the specified SCGRA overlay based accelerator, both design flows must be followed. The HDL model of the accelerator will be generated based on the specified parameters. Then the accelerator is rapidly implemented using the technique in [39] and an empty FPGA accelerator bitstream will be produced. Meanwhile, DFG is generated according to the specified unrolling factor and then scheduled to the SCGRA overlay. After the scheduling, control words are extracted and they can further be integrated into the empty FPGA accelerator bitstream. Finally, FPGA accelerator bitstream that can be used for the loop acceleration is generated. This process is typically 20 times faster than a standard HDL compilation flow[39].

When the nested loop is just slightly modified during the design iterations, the bitstream of the accelerator can be reused and we can simply follow the compilation flow marked with the green arrows. The compilation process is able to complete in a few seconds which is almost close to software compilation.

4. 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.

4.1 Customization Framework

Figure 5 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 (FDS) 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.

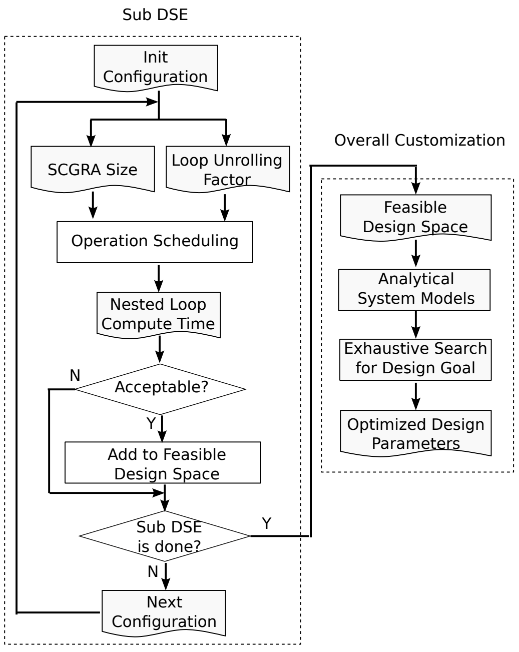


Figure 5: System customization framework.

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

4.2 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 $\boldsymbol{\Psi}$ represents the overall nested loop acceleration design space. $\boldsymbol{C} \in \boldsymbol{\Psi}$ represents a possible configuration in the design space and it includes a number of design parameters as listed in Table 1. 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

Table 1: Design Parameters of Nested Loop Acceleration

Design Parameters		Denotation
Nested Loop Compilation	Loop Unrolling Factor	$\boldsymbol{u} = (u_0, u_1, ...)$
	Grouping Factor	$\boldsymbol{g} = (g_0, g_1, ...)$
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

Null means there is no denotation for that parameter.

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

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

$RunTime(\boldsymbol{C})$ 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 Equation 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 Equation 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 Equation 5.

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

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 Equation 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 Equation 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 Equation 8.

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

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

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 Equation 9. BRAM overhead which is usually the overhead bottleneck for SCGRA overlay based FPGA accelerator design can be calculated by Equation 10.

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

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

4.3 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 shrink the large design space to a smaller FDS. Within the FDS, we further search the optimized design parameter for the entire system customization.

Suppose Φ denotes the FDS. 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 Equation 11 and Equation 12.

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

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

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.

In order to achieve the desired customization, we must make sure the FDS acquired from Equation 11 and Equation 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 Equation 11 or Equation 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 Equation 1 and Equation 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 Equation 11 and Equation 12 appears.

In addition, a series of experiments on Zedboard [2] as shown in Figure 6 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 Equation 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.

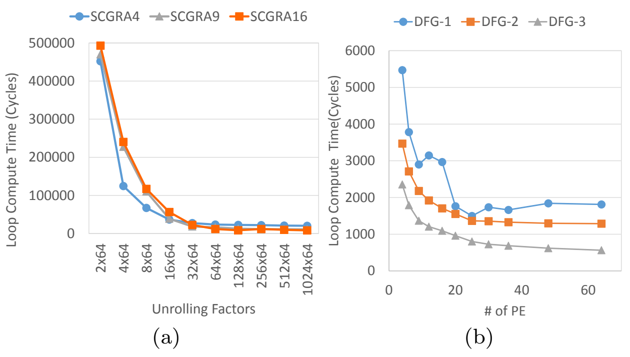


Figure 6: The design parameters typically have monotonic influence on the loop computation time and the computation time benefit degrades with the increase of the design parameter. (a) SCGRA Size, (b) Unrolling Factor

In 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.

4.4 Sub Design Space Exploration

To acquire the FDS, 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 6. 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.

5. 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.

5.1 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 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.

Algorithm 1 Sub Design Space Exploration.

```
procedure
  Initialize $r = 2, c = 2, \boldsymbol{u} = (1, 1, ...)$, FDS $\Phi = \emptyset$, $\boldsymbol{C} = (..., r, c, \boldsymbol{u}, ...)$, maximum SCGRA overlay $r_{Max} \times c_{Max}$.
  while $r < r_{Max}$ do
    while $c < c_{Max}$ do
      while $\boldsymbol{u}$ is not fully unrolled do
        Generate DFG with $\boldsymbol{u}$
        DFG Scheduling with configuration $\boldsymbol{C}$
        Estimate performance $CompuTime(\boldsymbol{C})$
        Get neighbor $\boldsymbol{C'} \in \Phi$ with smaller loop unrolling
        if $\boldsymbol{C'}$ exists and $Revenue(\boldsymbol{C}, \boldsymbol{C'}) \leq \epsilon$ then
          Break
        else
          Add $\boldsymbol{C}$ to $\Phi$
        end if
        update $\boldsymbol{u}$ with larger neighbor unrolling factor
      end while
      Get neighbor $\boldsymbol{C''} \in \Phi$ with smaller column size
      if $\boldsymbol{C''}$ exists and $Revenue(\boldsymbol{C}, \boldsymbol{C''}) \leq \epsilon$ then
        Break
      end if
      $c = c + 1$
    end while
    Get neighbor $\boldsymbol{C'''} \in \Phi$ with smaller row size
    if $\boldsymbol{C'''}$ exists and $Revenue(\boldsymbol{C}, \boldsymbol{C'''}) \leq \epsilon$ then
      Break
    end if
    $r = r + 1$
  end while
end procedure

procedure $Revenue(\boldsymbol{C}, \boldsymbol{C'})$
  return $\frac{CompuTime(\boldsymbol{C'}) - CompuTime(\boldsymbol{C})}{CompuTime(\boldsymbol{C'})}$
end procedure
```

Table 2: 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

5.2 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 2. 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.

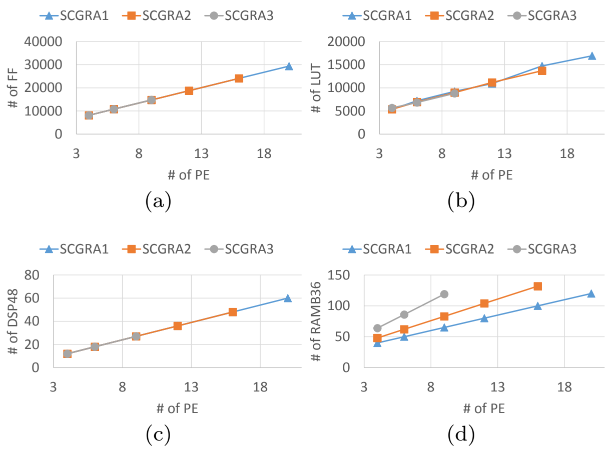


Figure 7: Relation between the accelerator overhead and overlay size, (a) FF overhead, (b) LUT overhead, (c)DSP overhead, (d)BRAM overhead

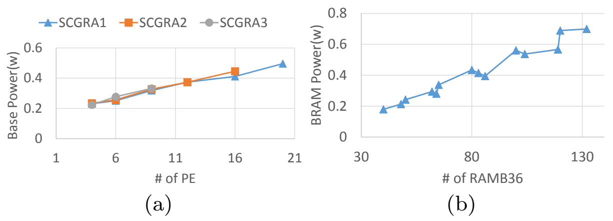


Figure 8: Power consumption of the SCGRA overlay based FPGA accelerator, (a) Base system power including DSP power, clock power, etc., (b) BRAM power

5.2.1 *Hardware Overhead*

Figure 7 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.

5.2.2 *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 8, 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.

5.3 Proposed Design Framework Evaluation

In this work, we take four applications including Matrix Multiplication (MM), FIR, Kmean(KM) and Sobel Edge De-

Table 3: Benchmark Configurations

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

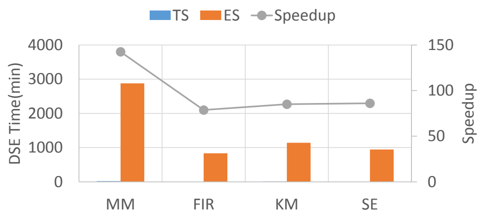


Figure 9: DSE time of the benchmark using both TS and ES

tector (SE) as our benchmark. The configurations of the benchmark are detailed in Table 3. 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 benchmark using Vivado HLS with moderate manual optimization and compared the implementations with the ones obtained using the proposed customization framework.

5.3.1 *Customization Methods Comparison*

Figure 9 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 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 10, 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.

5.3.2 *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

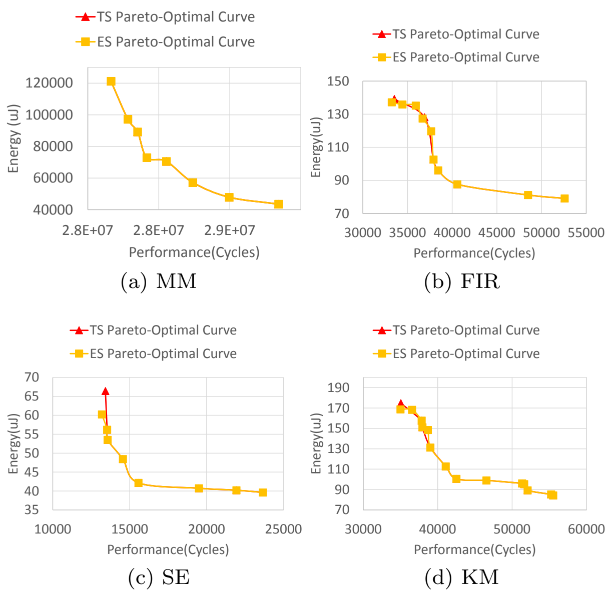


Figure 10: Performance-Energy Pareto-optimal curve

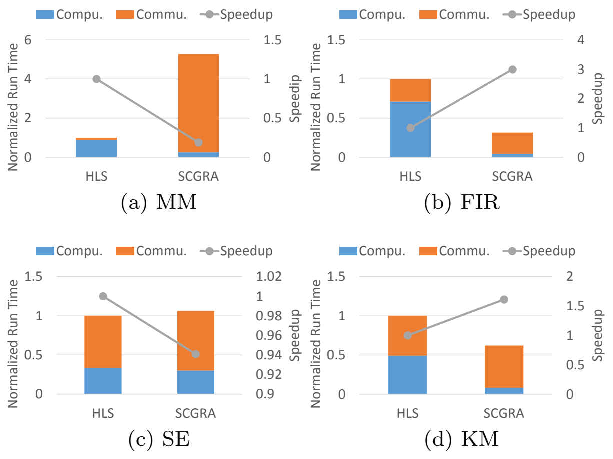


Figure 11: Performance comparison between the implementations using HLS and the proposed automatic customization framework. All the run time is normalized to that of the HLS based implementations.

were unrolled as much as possible and the input/output buffers were set as large as possible. The accelerators generated using 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 [28], we just brief the performance comparison for a quick reference.

Figure 11 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.

6. 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.

7. 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.

8. REFERENCES

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

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

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

[4] Canis, A., Choi, J., Aldham, M., Zhang, V., Kammoona, A., Anderson, J. H., Brown, S., and Czajkowski, T. LegUp: High-level Synthesis for FPGA-based Processor/Accelerator Systems. In *Proceedings of the 19th ACM/SIGDA International Symposium on Field Programmable Gate Arrays* (New York, NY, USA, 2011), FPGA '11, ACM, pp. 33–36.

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

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

[7] Capalija, D., and Abdelrahman, T. S. 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* (2014), IEEE, pp. 1–8.

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

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

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

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

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

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

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

[15] Ferreira, R., Vendramini, J., Mucida, L., Pereira, M., and Carro, L. An fpga-based heterogeneous coarse-grained dynamically reconfigurable architecture. In

Proceedings of the 14th international conference on Compilers, architectures and synthesis for embedded systems (2011), ACM, pp. 195–204.

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

[17] Grant, D., Wang, C., and Lemieux, G. G. 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* (New York, NY, USA, 2011), FPGA '11, ACM, pp. 123–132.

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

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

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

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

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

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

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

[25] Miniskar, N. R., Kohli, S., Park, H., and Yoo, D. 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* (2014), IEEE, pp. 1–9.

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

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

[28] removed. removed for blind review.

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

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

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

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

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

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

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

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

[37] Yiannacouras, P., Steffan, J. G., and Rose, J. Fine-grain performance scaling of soft vector processors. In *Proceedings of the 2009 International Conference on Compilers, Architecture, and Synthesis for Embedded Systems* (New York, NY, USA, 2009), CASES '09, ACM, pp. 97–106.

[38] Yu, Colin Lin. So, H. K.-H. 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* (2012), IEEE.

[39] Yue, X. Rapid overlay builder for xilinx fpgas.

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

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