Automatic Nested Loop Acceleration on FPGAs Using Soft CGRA Overlay

Cheng Liu, Hayden Kwok-Hay So
Department of Electrical and Electronic Engineering
The University of Hong Kong
{liucheng,hso}@eee.hku.hk

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. Overlay architectures which are intermediate virtual layers built on top of off-the-shelf FPGAs are increasingly being used to build FPGA accelerators for the sake of design productivity. However, achieving the desired performance-overhead trade-off still requires application-specific customization over a large design space involving multiple architectural parameters and it is a challenging task for high-level application developers hindering the widespread adoption of FPGAs as computing devices.

In this work, an automatic nested loop acceleration framework utilizing a regular soft coarse-grained reconfigurable array (SCGRA) overlay is presented. Given high-level resource constraints, it automatically customizes the overlay architectural design parameters, high-level compilation options as well as communication between the accelerator and the host processor for optimized performance specifically to an application and thus is accessible to high-level application developers. In addition, by taking advantage of the regularity of the SCGRA overlay architecture, performance and overhead models are built and the customization problem is reduced to a much simpler sub design space exploration and a straightforward search problem. According to the experiments, it takes 10 minutes to 20 minutes to complete the customization achieving up to 10X speedup over a hard ARM processor on Zedboard.

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 [7]. However, the relatively low productivity in developing FPGA-based compute applications remains a major obstacle that hinders their widespread employment [9].

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 [11, 3, 5, 10, 12, 16]. 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 coarse-grained reconfigurable array (SCGRA) overlay based nested loop acceleration framework targeting a hybrid CPU-FPGA system. In this framework, given high-level design constraints, the framework automatically customizes the overlay architectural parameters, exploits loop unrolling and hardware-software communication in combination with buffer sizing specifically to an application. In addition, by taking advantage of the regularity of the SCGRA overlay, a multitude of design metrics such as performance and hardware 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. When the optimized design parameters are decided, the corresponding hardware accelerator and communication interface are generated and both the hardware accelerator and software are compiled to the hybrid CPU-FPGA system. Moreover, the SCGRA overlay allows rapid bitstream reuse [16] during design iterations of an application. With an initial overlay implementation, nested loops expressed in high-level languages can be compiled to the SCGRA overlay in seconds. With both the efficient application-specific customization and rapid bitstream reuse, the proposed design framework ensures both high design productivity and high performance of FPGA loop acceleration.

According to the experiments, it takes the proposed design framework around 10 minutes to 20 minutes to complete the loop accelerator customization. When compared to the performance of the benchmark executed on a hard ARM processor, the resulting FPGA accelerators achieve up to 10X speedup.

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

- We have developed an SCGRA overlay based FPGA loop accelerator design framework that automatically performs FPGA accelerator architectural design parameters, compilation options and communication tuning. The resulting accelerators have shown promising performance speedup over an ARM processor.
- By taking advantage of the regularity of the SCGRA overlay based FPGA accelerator, we have built a series of performance and overhead models and further simplified the customization problem using these models.

2. RELATED WORK

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 [11, 3], array-of-FUs [5, 10], soft CGRA [12, 16], soft GPU [1], vector processors[21, 17] to configurable processors or multi-core processors [19, 14, 20, 4, 13, 6] 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 [18, 8]. Most importantly, it allows both rapid compilation by taking advantage of the overlays' tiling structure [23] and efficient bitstream reuse within the design iterations of an application [16], 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 [24, 15] while SCGRA overlays allow more intensive architectural customization providing just enough hardware to the target application or application domains 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 [22] 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 optimized performance. 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 DESIGN FRAMEWORK

By using a regular SCGRA overlay built on top of the physical FPGA devices, we have developed an automatic nested

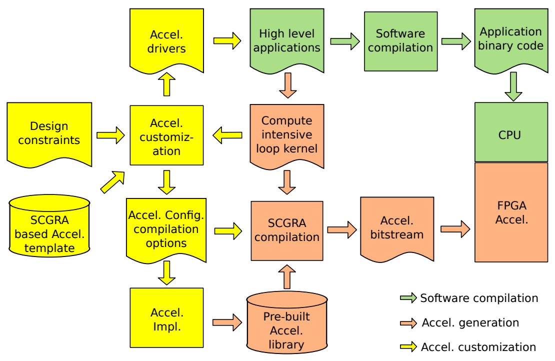


Figure 1: Automatic nested loop acceleration framework

loop acceleration framework as shown in Figure 1. It targets a hybrid CPU-FPGA computing system where the FPGA focuses on the compute intensive loop kernel and CPU handles the rest of the application. To that end, a standard software compilation routine is included to produce binary code for the CPU. The framework also has a fast and common SCGRA based accelerator generation path which is able to make use of pre-built bitstream to generate FPGA loop accelerator bitstream during the design iterations [16]. Meanwhile, it consists of a relatively slow yet per-loop based accelerator customization and implementation path. It automatically tunes the design parameters including overlay architectural parameters, compilation options as well as communication between the FPGA accelerator and host processor specifically to a user application under the user constraints such as hardware resource and energy consumption budgets. When the optimal design parameters are determined, SCGRA overlay based FPGA accelerator HDL models and drivers will be generated. Afterwards, the drivers will be used by the software compilation while the FPGA accelerator will be implemented and stored in the accelerator library which can be reused by the fast accelerator generation route.

3.1 SCGRA 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 computation.

Figure 3 shows the current implementation of a PE template that features an optional load/store path. At the heart of the PE is an ALU, which is supported by a multi-port data memory and an instruction memory. Data memory stores intermediate data during the computation while instruction memory stores all control words that determines the action of the PE. In addition, a global signal from the AccCtrl block controls the start/stop of all PEs in the array.

3.2 Loop execution on the FPGA accelerator

Figure 4 illustrates how the loop is executed on the FPGA accelerator. First of all, data flow graph (DFG) is extracted

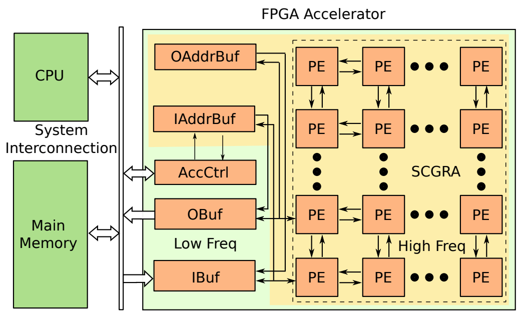


Figure 2: SCGRA overlay based FPGA accelerator

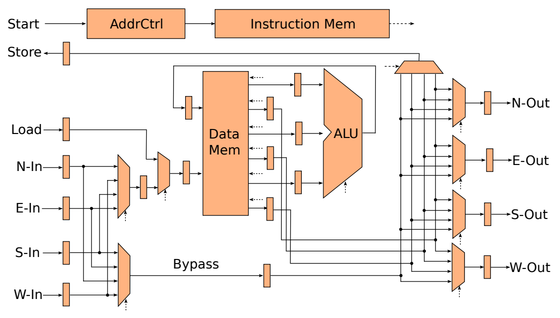


Figure 3: Fully pipelined PE structure. Each PE can be connected to at most 4 neighbours.

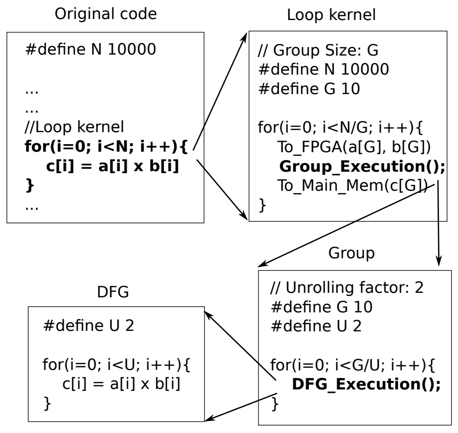
from the loop and then it is scheduled on to the SCGRA overlay based FPGA accelerator. Depending on how much the loop is unrolled and transformed to DFG, the DFG may be executed repeatedly until the end of the original loop. In addition, data transfers for multiple executions of the same DFG are batched 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.

3.3 SCGRA overlay compilation

With pre-built SCGRA overlay, the corresponding FPGA accelerator can be generated rapidly, which is also the basis of the high-productivity loop accelerator design framework. Figure 5 presents the detailed SCGRA overlay compilation. With the specified loop unrolling factor, DFG is generated and scheduled to the SCGRA overlay of the accelerator. After the scheduling, control words are extracted, and they can further be integrated into the pre-built FPGA accelerator bitstream creating the final FPGA loop accelerator bitstream. The compilation process typically completes in a few seconds as illustrated in [16]. This compilation process is particularly useful during the design iterations of an application when an initial implementation is done and minor modification is required.

4. SCGRA OVERLAY BASED FPGA ACCELERATOR CUSTOMIZATION

Application-specific customization provides unique opportunity to improve the energy and performance of the resulting



```
#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 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.

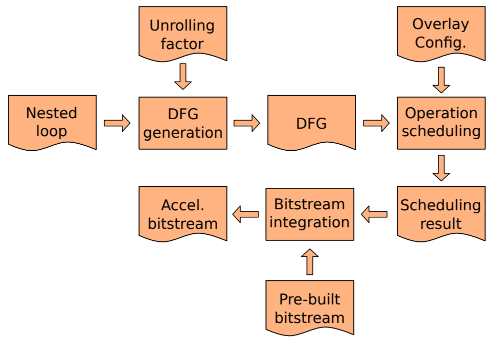


Figure 5: Rapid SCGRA overlay compilation

accelerators. However, taking the system as a black box and exhaustively searching all the possible configurations can be inefficient and slow. 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 much simpler sub design space exploration (DSE) together with a simplified search problem and optimized application-specific nested loop accelerator can be produced efficiently.

4.1 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 constraints including energy consumption and hardware resource overhead can be used while hardware overhead 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. $In(\boldsymbol{g})$

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.

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

$$RunTime(\boldsymbol{C}) = CompuTime(\boldsymbol{C}) + CommuTime(\boldsymbol{C}) \tag{1}$$

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 Equation 1.

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 3.

$$CompuTime(\boldsymbol{C}) = \prod_{i=1}^{n} \frac{l_i}{u_i} \times DFGCompuTime(\mathbf{u}, r, c) \tag{3}$$

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 $DMA(x)$ where x represents the amount of DMA transmission. The communication time of the whole nested loop can be calculated by Equation 4.

$$CommuTime(\boldsymbol{C}) = \prod_{i=1}^{n} \frac{l_i}{g_i} \times (DMA(In(\mathbf{g})) + DMA(Out(\mathbf{g}))) \tag{4}$$

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 5. BRAM overhead which is usually the overhead bottleneck for SCGRA overlay based FPGA accelerator design can be calculated by Equation 6.

$$Overhead(\boldsymbol{C}, i) = \alpha_i \times r \times c + \beta_i, (i = 2, 3, 4) \tag{5}$$

$$\begin{aligned}
Overhead(\boldsymbol{C}, 1) = &r \times c \times (imD \times imW + dmD \times dmW) +\\
&(ibD \times ibW + obD \times obW) +\\
&(iabD \times iabW + oabD \times oabW)
\end{aligned} \tag{6}$$

4.2 Customization framework

Figure 6 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 obtained. Since loop execution time is determined by the operation scheduling which simply depends on the loop unrolling factor and SCGRA size, the sub DSE is much simpler compared to the overall system DSE which includes more than 10 design parameters. 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 and overhead. These analytical models are accurate because of the regularity of the SCGRA overlay. Even though the feasible design space is still large, it is fast to evaluate all the configurations in it. After the evaluation process, customization for best performance becomes trivial and the customized design parameters can be obtained immediately.

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 Equation 7 and Equation 8.

$$\begin{aligned}
&\forall \boldsymbol{C} = (..., \boldsymbol{u}, r, c, ...) \in \Phi, \boldsymbol{C'} = (..., \boldsymbol{u'}, r', c', ...) \in \Phi,\\
&(r + 1 == r' \text{ and } c == c') \text{ or } (r == r' \text{ and } c + 1 == c') :\\
&\frac{CompuTime(\boldsymbol{C}) - CompuTime(\boldsymbol{C'})}{CompuTime(\boldsymbol{C})} > \epsilon
\end{aligned} \tag{7}$$

$$\begin{aligned}
&\forall \boldsymbol{C} = (..., \boldsymbol{u}, r, c, ...) \in \Phi, \boldsymbol{C'} = (..., \boldsymbol{u'}, r, c, ...) \in \Phi,\\
&\boldsymbol{u} \text{ and } \boldsymbol{u'} \text{ are consecutive unrolling factors :}\\
&\frac{CompuTime(\boldsymbol{C}) - CompuTime(\boldsymbol{C'})}{CompuTime(\boldsymbol{C})} > \epsilon
\end{aligned} \tag{8}$$

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

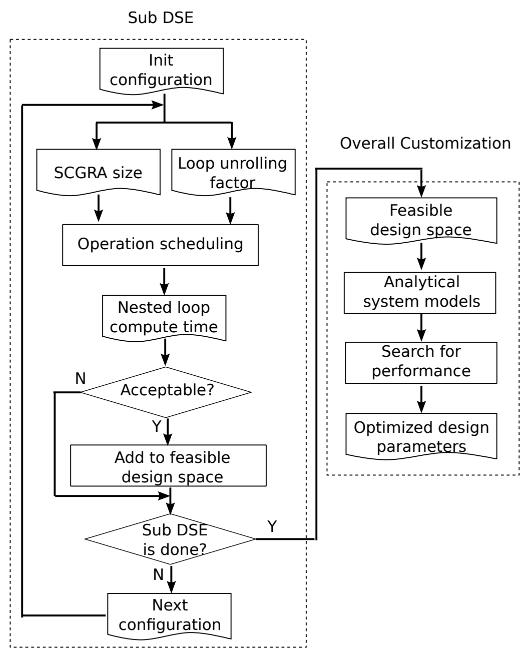
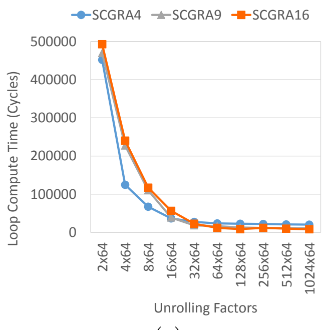
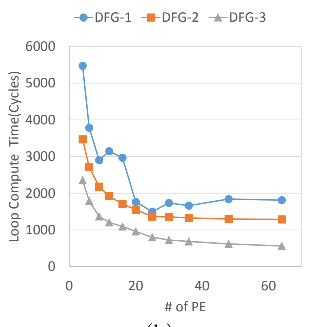


Figure 6: System customization framework.



(a)



(b)

Figure 7: 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 previous section and obtain the optimized configuration through a simple search.

In addition, a series of experiments on Zedboard [2] as shown in Figure 7 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. This observation further helps to simplify the sub DSE with a simple branch and bound algorithm.

5. EXPERIMENTS AND RESULTS

In the experiments, we measured the time needed to customize the loop accelerators and compared the performance of the resulting accelerators to that of an hard ARM processor.

Table 2: Benchmark Configurations

Benchmark	Parameters	Loop Structure
MM	Matrix Size(100)	$100 \times 100 \times 100$
FIR	# of Input (10000) # of Taps+1 (50)	10000×50
SE	# of Vertical Pixels (128) # of Horizontal Pixels (128)	$128 \times 128 \times 3 \times 3$
KM	# of Nodes(5000) # of Centroids(4) # of Dimensions(2)	$5000 \times 4 \times 2$

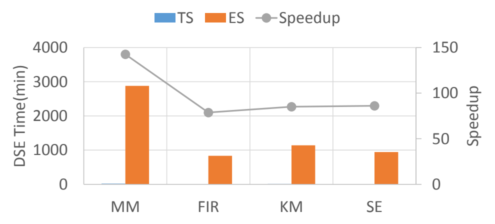


Figure 8: Benchmark customization time using both TS and ES

5.1 Experiment setup

The customization runtime 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 computation system. PlanAhead 14.7 was used for the SCGRA overlay based design. The customized overlay implementations on Zedboard run at 250MHz. To perform the customization, ϵ is set to be 0.05 and all the resource on Zedboard is set to be the resource constraint. Software runtime is obtained from ARM processor of Zedboard.

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 2.

5.2 Customization time

Figure 8 shows the customization time of both the proposed two step (TS) customization and an exhaustive search based customization (ES). TS typically completes the customization in 10 minutes to 20 minutes and it is around 100x faster than the ES on average. In particular, ES is extremely slow on MM which has three levels of loop with relatively large loop count and thus larger design space. Though TS also needs longer time to complete the customization, it skips most of the unfeasible configurations and the runtime is less sensitive to the size of the design space.

5.3 Customized accelerator performance

In order to demonstrate the quality of proposed framework, we compared the performance of the customized accelerators to that of an ARM processor on Zedboard. As shown in Figure 9, the customized accelerators achieve up to 10X speedup over the ARM processor on the benchmark. For FIR, SE and KM, the speedup is promising. MM has relatively low compute-IO rate and the single input and output between the on-chip buffer and the SCGRA overlay limits the performance of the accelerator. This problem can hopefully be alleviated by appropriate on-chip buffer partition, which will be supported in the proposed framework in future.

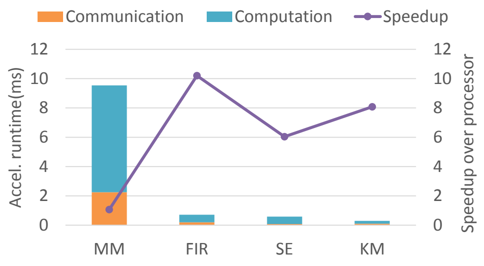


Figure 9: Customized FPGA loop accelerator performance

6. 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, the framework performs an intensive system customization specifically to a nested loop and provides a complete loop acceleration on a hybrid CPU-FPGA system. Particularly, by taking advantage of the regularity of the SCGRA overlay, the complex customization problem is simplified and typically completes in 10 minutes to 20 minutes. According to the experiments, the resulting accelerators achieve up to 10X speedup over a hard ARM processor on Zedboard.

7. 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] 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.

[5] 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.

[6] 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.

[7] 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.

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

[9] 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.

[10] 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.

[11] 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.

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

[13] 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.

[14] 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.

[15] 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.

[16] removed. removed for blind review.

[17] 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.

[18] 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.

[19] 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.

[20] 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.

[21] 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.

[22] 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.

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

[24] 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.