Automatic Nested Loop Acceleration on FPGAs Using Soft CGRA Overlay

Cheng Liu, Ho-Cheung Ng and Hayden Kwok-Hay So
Department of Electrical and Electronic Engineering
The University of Hong Kong
Email: {liucheng, hcng, 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. To construct such accelerators with high design productivity, researchers have increasingly turned to the use of overlay architectures as an intermediate generation target built on top of off-the-shelf FPGAs. However, achieving the desired performance-overhead trade-off remains a major productivity challenge as complex application-specific customizations over a large design space covering multiple architectural parameters are needed.**

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, the framework 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 the given application. In our experiments, at a cost of 10 to 20 minutes additional tools run time, the proposed customization process resulted in up to 5 times additional speedup over a baseline accelerator generated by the same framework without customization. Overall, when compared to the equivalent software running on the host ARM processor alone on the Zedboard, the resulting accelerators achieved up to 10 times speedup.

I. 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 [1], [2]. However, the relatively low productivity in developing FPGA-based compute applications remains one of the major obstacles that hinder widespread employment of FPGAs [3]. To address this challenge, a number of researchers have turned to the use of virtual FPGA overlay architectures built on top of the physical FPGA configurable fabric to help with improving design productivity through fast compilation, good design portability and debugging support [4], [5], [6], [7], [8], [9], [10], [11].

Despite the great advantages on design productivity, the additional layer on top of the physical FPGA inevitably introduces performance and resource consumption penalty. An overlay must ensure that the overall FPGA acceleration performance remains competitive. Otherwise, mapping the loop kernels to the overlay based FPGA accelerators will not be as useful. Therefore, the capability to customize the overlay specifically to an application or a domain of application becomes essential to the overlay based FPGA accelerator design. However, navigating through a labyrinth of architectural and compilation parameters to fine-tune an accelerator's 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.

We have been developing in-house a soft coarse-grained reconfigurable array (SCGRA) overlay based nested loop acceleration framework targeting a hybrid CPU-FPGA system called QuickDough, which allows rapid compilation from C loops to FPGA with a library of pre-built overlay bitstreams [10]. In this work, we mainly focus on automatically customizing the overlay architectural parameters, exploiting loop unrolling and hardware-software communication in combination with buffer sizing specifically to an application with given high-level resource constraints. In particular, 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. With both the efficient application-specific customization and rapid compilation, the proposed design framework ensures both high design productivity and high performance of FPGA loop acceleration.

From our experiments, it took the framework 10 to 20 minutes to complete the loop accelerator customization using our proposed two-step approach, which was up to 100 times faster than an exhaustive search through the design space. With customization, the resulting accelerators performed up to 5 times faster than a corresponding baseline accelerator before customization. Overall, when compared to the performance of the benchmark executed on the host ARM processor, the resulting FPGA accelerators achieved up to 10× speedup.

II. 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 [4], [6], [5], array-of-FUs [7], [8], [11], soft CGRA [9], [10], soft GPU [12], vector processors[13], [14] to configurable processors or multi-

2nd International Workshop on *FPGAs for Software Programmers (FSP 2015)*, London, United Kingdom, September 1, 2015.

core processors [15], [16], [17], [18], [19], [20] 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 [21], [22]. 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 [10], thus it is particularly promising for high productivity nested loop acceleration.

In addition, customizing the CGRA specifically for an application or a domain of application provides promising performance improvement while saving the hardware resource at the same time as demonstrated in CGRA work targeting ASIC design [24], [25], [26]. While CGRA customization on ASIC is relatively limited due to the tape-out cost, CGRA overlays allow more intensive architectural customization providing just enough hardware to the target application or application domains because of the FPGA's inherent programmability. In [27], Coole and Stitt proposed to provide the overlay with limited flexibility instead of full configurability specifically to a group of design. With this customization, the area overhead was reduced significantly. The authors in [28] developed an SCGRA topology customization method using genetic algorithm and showed the potential benefits of the SCGRA overlay customization. Nevertheless, the rest of the system design parameters were not covered. In [2], the authors formalized the loop acceleration on a regular processing array overlay on FPGA. They focused on the hardware resource constrain, IO bandwidth constrain and the loop parallelism partition while processing architectural design parameters were not included. 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.

III. Nested Loop Accelerator Design Framework

By using a regular SCGRA overlay built on top of the physical FPGA devices, we have developed an automatic nested loop acceleration framework called QuickDough. QuickDough targets hybrid CPU-FPGA computing systems where the FPGA is devoted to accelerating compute intensive loop kernel and CPU handles the rest of the application. Figure 1 depicts an overview of the design framework, highlighting the complementary *accelerator generation* and *accelerator customization* paths.

By design, the steps along the accelerator generation path are short and essential during rapid design iterations. Collectively, they are able to generate FPGA loop accelerators making use of a pre-built bitstream library in the order of seconds [10].

Meanwhile, the focus of this paper is on the accelerator customization path, which is relatively slow but is necessary

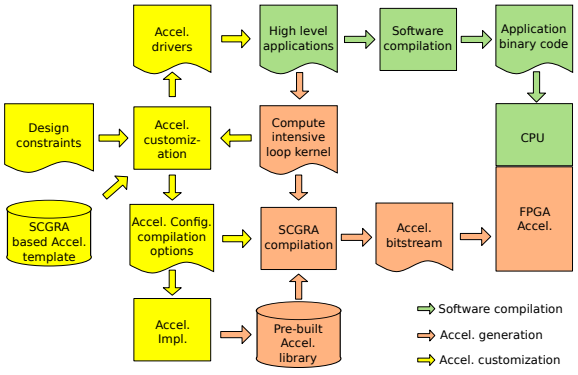


Fig. 1. Automatic nested loop acceleration framework

for improving performance of the resulting accelerators on a per-application basis. These steps 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 user constraints such as hardware resource budgets. With the customized design parameters, HDL models of the corresponding SCGRA overlay and their associated drivers are then generated. Afterwards, the drivers will be used by the software compiler while the FPGA accelerator will be implemented and stored in the accelerator library, which can be reused by the fast accelerator generation path in subsequent compilations.

A. 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. The current implementation of a PE template is also presented in this figure. 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.

B. Loop execution on the 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

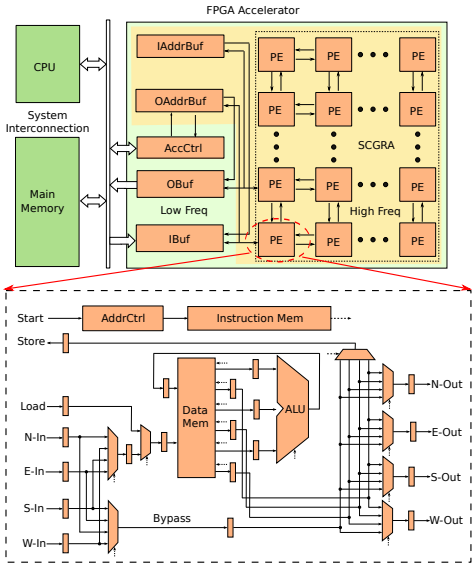


Fig. 2. SCGRA overlay based FPGA accelerator

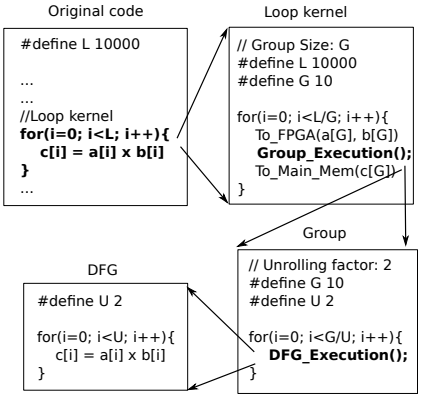


Fig. 3. Loop, group and DFG. The loop will be divided into groups. Each group will be partially unrolled and the unrolled part will be translated to DFG. IO transmission between FPGA and host CPU is performed in the granularity of a group.

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.

C. SCGRA overlay compilation

With pre-built SCGRA overlay library and customized overlay configuration, the corresponding FPGA accelerator can be generated rapidly, which is also the basis of the high-productivity loop accelerator design framework. Figure 4 presents the detailed SCGRA overlay compilation. With the specified loop unrolling and grouping 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

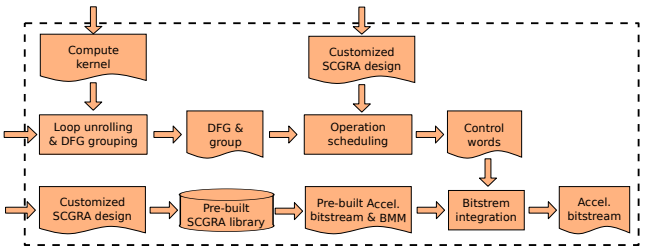


Fig. 4. Rapid SCGRA overlay compilation

as illustrated in [10] which is particularly important during early application development phases.

IV. SCGRA Overlay Based FPGA Accelerator Customization

Application-specific customization provides unique opportunity to reduce the resource consumption and improve performance of the resulting 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. With the customization, optimized application-specific nested loop accelerator can be produced efficiently.

A. 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 consumption can be used while hardware resource consumption is taken as an example here.

TABLE I
Design Parameters of Nested Loop Acceleration [1]

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	2D Torus, fixed
	SCGRA Size	$r \times c$
	Data Width	W_0
	Data Mem	$D_0 \times W_0$
	Input Buffer	$D_1 \times W_0$
	Output Buffer	$D_2 \times W_0$
	Instruction Mem	$D_3 \times W_1$
	Input Address Buffer	$D_4 \times W_2$
	Output Address Buffer	$D_5 \times W_3$
	Operation Set	fixed
	Implementation Frequency	f, fixed
	Pipeline Depth	fixed

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 I. Assume that the loop to be accelerated has n nested levels and loop count can be denoted as

[1]The parameters are all customizable in the proposed design framework except the ones that are clearly identified as fixed.

$l = (l_1, l_2, ..., l_n)$. $R = (R_1, R_2, R_3, R_4)$ stands for the FPGA resource (i.e. BRAM, DSP, LUT and FF) that are available on a target FPGA and $ResConsumption(\boldsymbol{C}, i)$ denotes the four different types of FPGA resource consumption. $In(\boldsymbol{g})$ 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 run time can be formulated as follows:

Minimize

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

subject to

$$\begin{aligned} &ResConsumption(\boldsymbol{C}, i) \leq R_i, i = 1, 2, 3, 4 \\ &In(g) \leq D_1 \\ &Out(g) \leq D_2 \\ &DFGCompuTime(\boldsymbol{C}) \leq D_3 \\ &\prod_{i=1}^{n} \frac{g_i}{u_i} \times In(u) \leq D_4 \\ &\prod_{i=1}^{n} \frac{g_i}{u_i} \times Out(u) \leq D_5 \end{aligned} \tag{2}$$

$RunTime(\boldsymbol{C})$ 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 resource on FPGA mainly includes DSP, LUT, FF and BRAM (block RAM). LUT, FF and DSP consumption can be roughly estimated with a linear function of SCGRA size and can be calculated using Equation 5. BRAM consumption $ResConsumption(\boldsymbol{C}, 1)$ which is slightly different from LUT, FF and DSP consumption can be calculated precisely based on the memory block configurations.

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

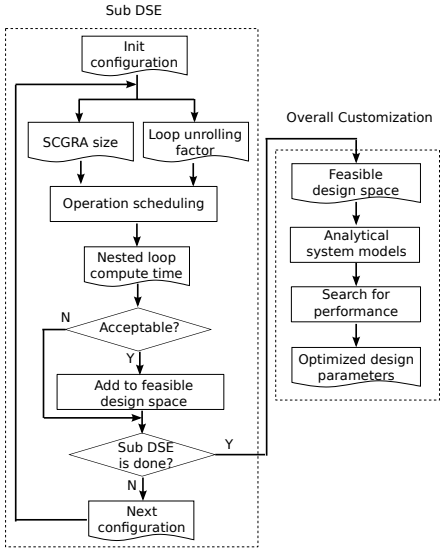


Fig. 5. System customization framework.

B. Customization framework

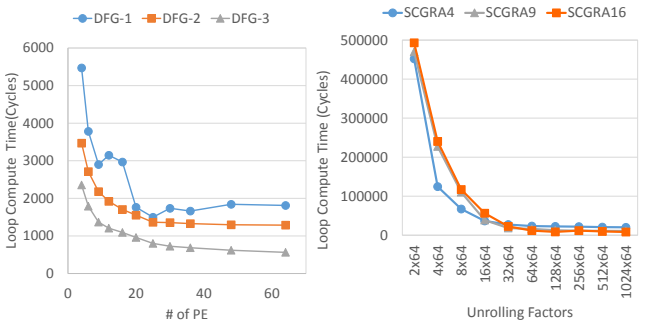
Figure 5 illustrates the overview of the customization framework. It can be roughly divided into two parts. In the first part, a sub DSE targeting loop execution time is performed and the feasible design space can be obtained. Since loop execution time is determined by the operation scheduling which 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 hardware resource consumption. 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 6 and Equation 7.

$$\begin{aligned} &\forall \boldsymbol{C} = (..., \boldsymbol{u}, r, c, ...) \in \Phi, \boldsymbol{C'} = (..., \boldsymbol{u'}, r', c', ...) \in \Phi, \\ &(r + 1 == r' \text{ and } c == c') \text{ or } (r == r' \text{ and } c + 1 == c') : \\ &\frac{CompuTime(\boldsymbol{C}) - CompuTime(\boldsymbol{C'})}{CompuTime(\boldsymbol{C})} > \epsilon \end{aligned} \tag{6}$$

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

Each feasible configuration $\boldsymbol{C} \in \Phi$ must have gone through the scheduling and thus the corresponding scheduling result is



(a) (b)

Fig. 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, the SCGRA topology used are torus with 2×2, 3×2, 3×3, ... while DFG-1, DFG-2 and DFG-3 are DFGs extracted from matrix-matrix multiplication, fir and Kmean respectively. (b) Unrolling Factor, the loop used is a 63-tap Fir with 1024 input.

known. Consequently, 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 in previous section and obtain the optimized configuration through a simple search.

In addition, a series of experiments on Zedboard 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. This observation further helps to simplify the sub DSE with a simple branch and bound algorithm.

V. 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.

A. 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 II.

B. Customization time

Figure 7 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.

TABLE II
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)	$5000 \times 4 \times 2$

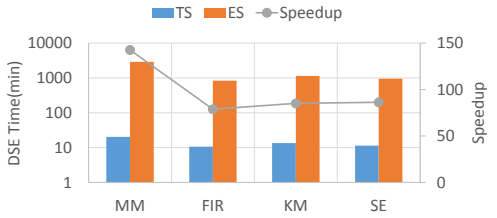


Fig. 7. Benchmark customization time using both TS and ES

TABLE III
Accelerator configurations [2]

MM	Base	$(1 \times 2 \times 100, 4 \times 2 \times 100, 5 \times 5, 1k, 2k)$
	TS	$(1 \times 5 \times 100, 50 \times 5 \times 100, 4 \times 4, 1k, 8k)$
	ES	$(1 \times 5 \times 100, 25 \times 5 \times 100, 5 \times 4, 1k, 8k)$
FIR	Base	$(10 \times 50, 100 \times 50, 3 \times 3, 1k, 2k)$
	TS	$(50 \times 50, 2000 \times 50, 4 \times 4, 1k, 4k)$
	ES	$(100 \times 50, 5000 \times 50, 5 \times 4, 1k, 8k)$
SE	Base	$(4 \times 4 \times 3 \times 3, 128 \times 128 \times 3 \times 3, 3 \times 2, 1k, 8k)$
	TS	$(16 \times 16 \times 3 \times 3, 128 \times 128 \times 3 \times 3, 4 \times 4, 1k, 4k)$
	ES	$(16 \times 16 \times 3 \times 3, 128 \times 128 \times 3 \times 3, 5 \times 4, 1.5k, 4k)$
KM	Base	$(25 \times 4 \times 2, 2500 \times 4 \times 2, 4 \times 3, 1k, 8k)$
	TS	$(125 \times 4 \times 2, 625 \times 4 \times 2, 5 \times 5, 1k, 2k)$
	ES	$(125 \times 4 \times 2, 625 \times 4 \times 2, 5 \times 5, 1k, 2k)$

C. Customized accelerator performance

In order to demonstrate the quality of proposed framework, we compared the performance of the accelerators with a random configuration as well as customized configurations obtained using both TS and ES. The detailed configurations of the accelerators are listed in Table III. The performance comparison is shown in Figure 8. It can be found that the customized accelerators obtained using TS achieve quite close performance to the ones customized through ES. Particularly, the customized accelerator achieves 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.

[2]The configurations include loop unrolling factor, grouping factor, SCGRA array size, instruction memory depth and IO buffer depth

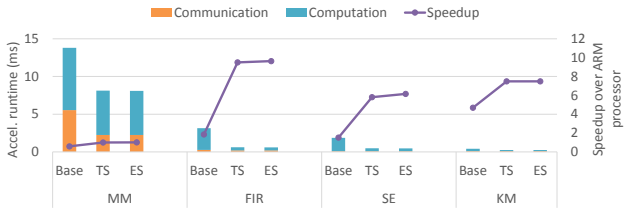


Fig. 8. Customized FPGA loop accelerator performance

VI. Conclusion

In this work, we have presented an automatic nested loop acceleration framework that is based on a soft coarse-grained reconfigurable array overlay. We have demonstrated that by taking advantage of the regularity of the overlay, intensive system customization specific to the given user application can be performed efficiently, resulting in up to 5 times performance improvement over solutions without customization at the cost of 10 to 20 minutes additional tools run time. Overall, the framework is able to generate accelerators that achieve up to 10 times speed up over software running on the host processor, resulting in a high design productivity experience for software programmers.

Acknowledgment

This work was supported in part by the Research Grants Council of Hong Kong project ECS 720012E and the Croucher Innovation Award 2013.

References

[1] E. Chung *et al.*, "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.

[2] U. Bondhugula, J. Ramanujam, and P. Sadayappan, "Automatic mapping of nested loops to FPGAs," in *Proceedings of the 12th ACM SIGPLAN Symposium on Principles and Practice of Parallel Programming, PPOPP 2007, San Jose, California, USA, March 14-17, 2007*, 2007, pp. 101–111. [Online]. Available: http://doi.acm.org/10.1145/1229428.1229446

[3] J. Cong *et al.*, "High-level synthesis for FPGAs: From prototyping to deployment," *Computer-Aided Design of Integrated Circuits and Systems, IEEE Transactions on*, vol. 30, no. 4, pp. 473–491, 2011.

[4] D. Grant, C. Wang, and G. G. Lemieux, "A CAD framework for Malibu: An FPGA with time-multiplexed coarse-grained elements," in *Proceedings of the 19th ACM/SIGDA International Symposium on Field Programmable Gate Arrays*, ser. FPGA '11. New York, NY, USA: ACM, 2011, pp. 123–132. [Online]. Available: http://doi.acm.org/10.1145/1950413.1950441

[5] J. Coole and G. Stitt, "Intermediate fabrics: Virtual architectures for circuit portability and fast placement and routing," in *Hardware/Software Codesign and System Synthesis (CODES+ISSS), 2010 IEEE/ACM/IFIP International Conference on*, Oct 2010, pp. 13–22.

[6] A. Brant and G. Lemieux, "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.

[7] D. Capalija and T. Abdelrahman, "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.

[8] R. Ferreira *et al.*, "An FPGA-based heterogeneous coarse-grained dynamically reconfigurable architecture," in *Proceedings of the 14th international conference on Compilers, architectures and synthesis for embedded systems*. ACM, 2011, pp. 195–204.

[9] D. Kissler *et al.*, "A dynamically reconfigurable weakly programmable processor array architecture template." in *ReCoSoC*, 2006, pp. 31–37.

[10] C. Liu, C. Yu, and H.-H. So, "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*, April 2013, pp. 228–228.

[11] A. K. Jain, S. A. Fahmy, and D. L. Maskell, "Efficient overlay architecture based on DSP blocks," in *Field-Programmable Custom Computing Machines (FCCM), 2015 IEEE 23rd Annual International Symposium on*, May 2015, pp. 25–28.

[12] A. Al-Dujaili *et al.*, "Guppy: A GPU-like soft-core processor," in *Field-Programmable Technology (FPT), 2012 International Conference on*, Dec 2012, pp. 57–60.

[13] P. Yiannacouras, J. G. Steffan, and J. Rose, "Fine-grain performance scaling of soft vector processors," in *Proceedings of the 2009 International Conference on Compilers, Architecture, and Synthesis for Embedded Systems*, ser. CASES '09. New York, NY, USA: ACM, 2009, pp. 97–106. [Online]. Available: http://doi.acm.org/10.1145/1629395.1629411

[14] A. Severance and G. Lemieux, "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.

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

[16] I. Lebedev *et al.*, "MARC: A many-core approach to reconfigurable computing," in *Reconfigurable Computing and FPGAs (ReConFig), 2010 International Conference on*, Dec 2010, pp. 7–12.

[17] P. Yiannacouras, J. Steffan, and J. Rose, "Exploration and customization of FPGA-based soft processors," *Computer-Aided Design of Integrated Circuits and Systems, IEEE Transactions on*, vol. 26, no. 2, pp. 266–277, Feb 2007.

[18] D. Capalija and T. Abdelrahman, "An architecture for exploiting coarse-grain parallelism on FPGAs," in *Field-Programmable Technology, 2009. FPT 2009. International Conference on*, Dec 2009, pp. 285–291.

[19] C. E. LaForest and J. G. Steffan, "OCTAVO: An FPGA-centric processor family," in *Proceedings of the ACM/SIGDA International Symposium on Field Programmable Gate Arrays*, ser. FPGA '12. New York, NY, USA: ACM, 2012, pp. 219–228. [Online]. Available: http://doi.acm.org/10.1145/2145694.2145731

[20] H. Y. Cheah, S. Fahmy, and D. Maskell, "iDEA: A DSP block based FPGA soft processor," in *Field-Programmable Technology (FPT), 2012 International Conference on*, Dec 2012, pp. 151–158.

[21] R. Tessier and W. Burleson, "Reconfigurable computing for digital signal processing: A survey," *Journal of VLSI signal processing systems for signal, image and video technology*, vol. 28, no. 1-2, pp. 7–27, 2001.

[22] K. Compton and S. Hauck, "Reconfigurable computing: a survey of systems and software," *ACM Computing Surveys (csuR)*, vol. 34, no. 2, pp. 171–210, 2002.

[23] M. X. Yue, D. Koch, and G. G. Lemieux, "Rapid overlay builder for Xilinx FPGAs," in *Field-Programmable Custom Computing Machines (FCCM), 2015 IEEE 23rd Annual International Symposium on*, May 2015, pp. 17–20.

[24] K. Compton and S. Hauck, "Totem: Custom reconfigurable array generation," in *Field-Programmable Custom Computing Machines, 2001. FCCM '01. The 9th Annual IEEE Symposium on*, March 2001, pp. 111–119.

[25] L. Zhou *et al.*, "Application-specific coarse-grained reconfigurable array: architecture and design methodology," *International Journal of Electronics*, no. ahead-of-print, pp. 1–14, 2014.

[26] N. R. Miniskar *et al.*, "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*. IEEE, 2014, pp. 1–9.

[27] J. Coole and G. Stitt, "Adjustable-cost overlays for runtime compilation," in *Field-Programmable Custom Computing Machines (FCCM), 2015 IEEE 23rd Annual International Symposium on*, May 2015, pp. 21–24.

[28] C. Y. Lin and H. K.-H. So, "Energy-efficient dataflow computations on FPGAs using application-specific coarse-grain architecture synthesis," *SIGARCH Comput. Archit. News*, vol. 40, no. 5, pp. 58–63, Mar. 2012. [Online]. Available: http://doi.acm.org/10.1145/2460216.2460227