QuickDough: A Rapid FPGA Loop Accelerator Design Framework Using Soft CGRA Overlay

***Abstract*—The use of CGRAs as compute accelerators has been demonstrated by numerous researchers as an effective solution to meet the performance requirement across many application domains. However, the design productivity of developing FPGA accelerators remains much lower compared to the use of a typical software development flow. Although the use of high-level synthesis (HLS) partly alleviates the shortcoming, the lengthy low-level FPGA implementation process including synthesis, placing and routing dramatically limits the number of compile-debug-edit cycles per day and hinders the widespread adoption of FPGAs.**

To address this design productivity problem, we have developed a rapid FPGA loop accelerator generation framework called QuickDough. By utilizing a soft coarse-grained reconfigurable array (SCGRA) overlay built on top of off-the-shelf FPGAs, it compiles an high-level loop to the overlay through a rapid operation scheduling first and then generates the FPGA accelerator bitstream through a rapid integration of the scheduling result and a pre-built overlay bitstream. According to the experiments, QuickDough is able to produce accelerators in the order of seconds while maintaining acceleration performance 1X-10X over the execution of the same software running on a hard ARM processor.

I. Introduction

Recent years have witnessed a tremendous growth in the use of accelerators in computer systems to improve the systems' performance and power-efficiency. Among these accelerators, GPUs and the Xeon Phi accelerators have stood out as two of the most popular choices in spite of their relatively short history as accelerators — 5 of the top 10 systems on the top500 list take advantage of them. On the other hand, despite the long and successful track record of FPGA accelerators [1], the use of FPGA accelerators in main-stream systems remains limited and has yet to receive widespread adoption beyond highly skilled hardware engineers.

We argue that there are 3 major challenges faced by software developers when using FPGAs as accelerators: (1) the unfamiliar hardware development methodology with low-level hardware description languages; (2) the lack of hardware-software support during run-time and compile-time; and (3) the lack of efficient implementation and debugging facilities.

To address challenge 1, recent advances in high-level synthesis tools have significantly raised the abstraction level for FPGA design entry, allowing users to effectively express hardware designs using familiar software languages such as C/C++, Java, Python and Scala. At the same time, to address challenge 2, researchers have also explored various facilities to support mixed hardware-software designs in a unified language [2], [3] and run-time environment [4], [5], [6]. Challenge 3, however, remains a major productivity hurdle to most software developers. Unlike compiling software programs, implementing a hardware design on to an FPGA using standard hardware design tools can take upward of days with some of the largest designs. This disproportionally long run time greatly limits the number of debug-edit-implement-cycle per day possible and hinders the designer's productivity.

The focus of this work is therefore to address challenge 3. In particular, we are interested in significantly improving the speed of generating hardware accelerators on FPGAs for compute intensive loops expressed in high-level languages, while maintaining a competitive overall performance for the resulting system.

To that end, we have developed QuickDough, a design framework that rapidly generates loop accelerators and their associated software-hardware data I/O facilities on FPGAs. By using a soft coarse-grain reconfigurable array (SCGRA) *overlay* as an intermediate architecture implemented on top of the physical FPGAs, QuickDough partitions the complex accelerator development flow into two paths. Along the rapid and common path, QuickDough generates loop accelerators by selecting an overlay configuration from a library of partially implemented FPGA bitstream, schedules the compute operations from the user-provided loop onto the overlay, and finally updates the bitstream configuring the target FPGA. On the other hand, QuickDough can also update the overlay bitstream library upon user request. This process is considerably slower as a number of accelerators may need to be implemented and added to the library. In these cases, QuickDough helps to expedite the library update process by providing a minimum representative set of accelerator configurations to be implemented and added to the library. Moreover, to ensure the system accessible to application developers, the pre-built accelerator library will be automatically generated using a template based system and implemented accordingly.

Experiments show that despite the use of an overlay, the accelerators generated by QuickDough has promising performance speedup over the software executed on a hard ARM processor, yet the accelerator generation completes in seconds achieving near-software compilation speed and enhancing software designers' productivity in designing, developing and debugging their applications with accelerators.

In next section, we will elaborate on the FPGA loop accelerator design framework – QuickDough. Then we will present the experimental results in Section III. Finally, we will compare QuickDough with related works in Section IV and conclude in Section V.

II. QuickDough Framework

QuickDough is an FPGA loop accelerator generation framework. It generates FPGA accelerators for compute intensive loop kernels in the order of seconds through the use of an SCGRA overlay. It also generates the drivers of the accelerators automatically, integrating both software and hardware generation in a unified framework.

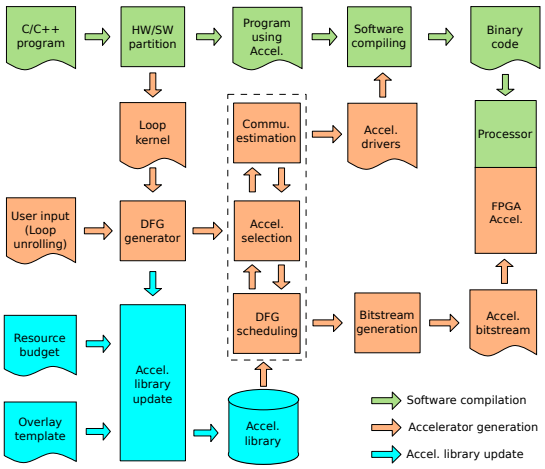

Fig. 1. QuickDough: FPGA loop accelerator design framework using SCGRA overlay. The compute intensive loop kernel of an application is compiled to the SCGRA overlay based FPGA accelerator while the rest is compiled to the host processor.

A. *QuickDough Overview*

Figure 1 illustrates the proposed FPGA loop accelerator design framework named QuickDough. It roughly consists of a conventional software compilation path, a fast and common FPGA loop accelerator generation path and a slow yet rare accelerator library update path. The first path of QuickDough is to compile the overall software application to the host processor after replacing the compute kernels with calls to the accelerator drivers which mainly control and transfer data to and from the accelerator.

The second path of QuickDough is a rapid and common route for loop accelerator generation. To begin, the compute intensive loop kernel is statically transformed to the corresponding data flow graph (DFG) with specified loop unrolling factor. Then the accelerator selection process selects an accelerator from a pre-built accelerator library based on the scheduling performance and the communication overhead. The scheduling performance is obtained from the SCGRA scheduling process which schedules the generated DFG to the SCGRA overlay included in the selected accelerator. The communication overhead is obtained through a communication estimation based on the communication requirements and on-chip buffer size. After the accelerator selection process, the specified accelerator on-chip buffer is decided and the accelerator drivers can be generated accordingly. Meanwhile, the selected pre-built accelerator and the corresponding scheduling result are integrated to create the final FPGA configuration bitstream. This bitstream, in combination with the software created in the first path, forms the final application that will be executed on a target CPU-FPGA system.

The third path of QuickDough is to update the accelerator library upon users' request. Users may simply provide the hardware resource budget. Then target operations to be supported will be decided automatically by analyzing the DFGs produced by the DFG generator. With the resource budget and the supported operation set, a set of accelerator HDL models will be generated by utilizing the overlay template. Finally, the accelerator HDL models are implemented on target FPGA platforms and further updated to the accelerator library.

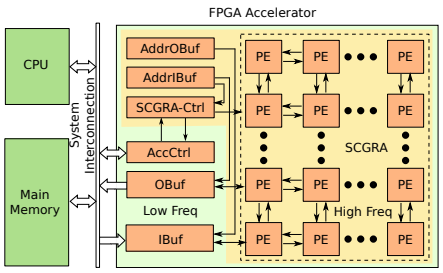

Fig. 2. SCGRA Overlay Based FPGA Accelerator

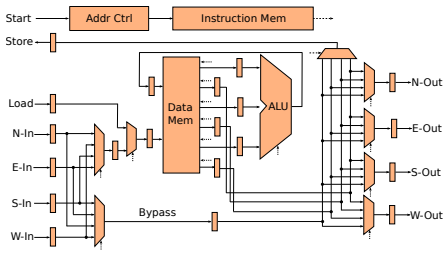

Fig. 3. Fully pipelined PE structure. Each PE can be connected to at most 4 neighbours.

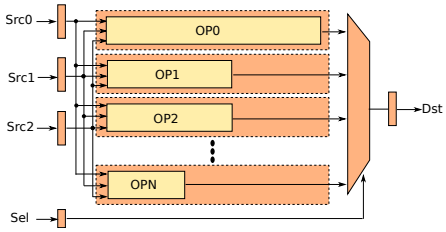

Fig. 4. The QuickDough ALU. It supports up to 16 fully pipelined 3-input operations.

B. *SCGRA overlay based FPGA accelerator*

The QuickDough overlay consists of an array of simple processing elements (PEs) connected by a direct network executing synchronously as shown in Figure 2. Each PE computes and forwards data in lock steps, allowing deterministic multi-hop data communication that overlaps with computations. The action of each PE in each cycle is controlled by an instruction ROM that is populated with instructions generated by the design framework. Finally, a data memory is featured on each PE to serve as a temporary storage for run-time data that may be reused in the same PE or be forwarded in subsequent steps.

Communication between the accelerator and the host processor is carried through a pair of input/output buffers. Accesses to these I/O buffers from the SCGRA array take place in lock step with the rest of the system. The exact buffer location to be accessed is control by the AddrIBuf and AddrOBuf blocks. Both of them are ROM populated with address information generated from the QuickDough compiler.

1) PE template: Figure 3 shows the current implementation of a QuickDough 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. Three of the data memory's read ports are connected to the ALU as inputs, while the remaining ports are sent to the output multiplexors for connection to neighboring PEs and the optional store path to OBuf external to the PE. At the same

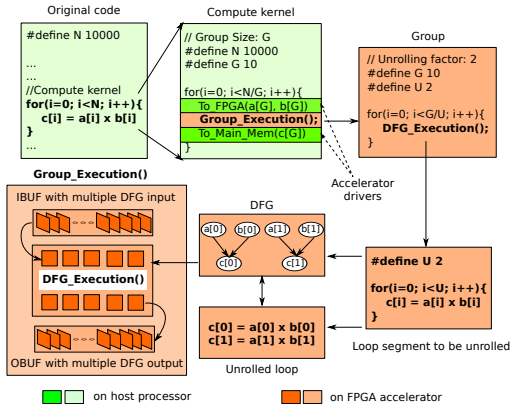

Fig. 5. Loop execution on an SCGRA overlay based FPGA accelerator

time, this data memory takes input from the ALU output, data arriving from neighboring PEs, as well as from the optional IBuf loading path. The action of the PE is controlled by the AddrCtrl unit that reads from the instruction memory. Finally, a global signal from the AccCtrl block controls the start/stop of all PEs in the array.

2) ALU template: At the heart of the proposed PE is the ALU and it can easily be customized with different operations specifically for any given user applications. Figure 4 shows the ALU template used in the QuickDough overlay. These operators in the ALU may execute concurrently in a pipelined fashion and must complete in a deterministic number of cycle. Given the deterministic nature of the operators, the QuickDough scheduler will ensure that there is never conflict at the output multiplexor.

C. Loop execution on the accelerator

The loop kernels are mostly partially unrolled, transformed to DFG and scheduled to the SCGRA overlay of the accelerator. A straightforward way to complete the whole loop computation on top of the SCGRA overlay is to repeat the same DFG computation until the end of the loop. Nevertheless, this may require data transfer between host processor and I/O buffer for each DFG computation. As a result, the communication cost increases dramatically especially when the amount of each data transfer is small. Worse still, input data of the consecutive DFGs may be reused and the straightforward data transfer strategy may greatly increase the total amount of data transfer through out the loop computation.

To alleviate this problem, we have proposed to batch data transfers for multiple executions of the same DFG into groups as shown in Figure 5. Specifically, after the loop is unrolled U times, G of them are grouped together for each data transfer. This group strategy helps to amortize the initial communication cost between host processor and the accelerator. In addition, it allows input data to be reused for different DFG computation in the same group and the group size is mainly limited by the I/O buffer depth. Meanwhile, the accelerator communicates with host processor for each group execution, and thus the accelerator drive that handles the communication depends on the I/O buffer depth as well. Clearly, accelerator with larger I/O buffer is preferable when the rest part of the accelerator configuration fulfills the requirements.

D. FPGA loop accelerator generation

The FPGA loop accelerator generation path is a common path and is critical for QuickDough to produce FPGA loop accelerator rapidly. The major processes on the path are detailed in this section.

1) DFG generation: In order to produce an FPGA loop accelerator using SCGRA overlay, DFGs are extracted from the kernel that is often expressed as inner loop body. The users may further unroll the loops multiple times to increase the amount of operation parallelism in the generated DFG. In this work, we have developed a C++ library to help automate the DFG generation with specified loop unrolling factor.

2) Accelerator selection: Accelerator selection process selects an accelerator from the accelerator library based on the performance of the resulting accelerator which mainly depends on the computation latency and communication latency. The computation latency of the loop kernel can be calculated using Equation 1 where DFG_Lat stands for the number of cycles needed to complete the SCGRA scheduling and mostly depends on the SCGRA overlay size and $Freq$ stands for the pre-built accelerator implementation frequency. The communication latency can be calculated using (2) where $Trans()$ represents the data transfer latency function of the target platform and $GpIn$ and $GpOut$ represent the amount of data transfer of a group which is determined by the capacity of the I/O buffers.

$$CompLat = DFG\_per\_Loop \times DFG\_Lat/Freq \quad (1)$$

$$CommLat = Gp\_per\_Loop \times (Trans(GpIn) + Trans(GpOut)) \quad (2)$$

In summary, the performance of the accelerator can be estimated with analytical models when the scheduling performance is obtained through the DFG scheduling while the scheduling performance is mostly determined by the SCGRA overlay size. The analytical estimation is fast while the scheduling process is relatively slow. Therefore, the accelerator selection process essentially centers the SCGRA overlay size selection and then explores all the accelerator configurations with the same SCGRA overlay size.

To compromise the loop accelerator generation time and performance, three different levels of accelerator selection optimization options are provided in this framework namely O0, O1 and O2 centering the SCGRA overlay size selection. O0 doesn't provide any optimization, and it selects an accelerator with the smallest SCGRA overlay. O1 estimates three typical accelerators with the smallest SCGRA overlay, a medium one and the largest SCGRA overlay. Then the one that provides the best performance will be adopted. O3 explores all the accelerators in the library and searches for the best accelerator configuration. With the increase of the optimization level, the accelerator selection process spends more efforts in searching the accelerator library for better performance and thus results in longer acceleration generation time.

3) DFG Scheduling: When a DFG is extracted from the loop kernel, it can be then scheduled to execute on the SCGRA overlay of the accelerator. Since the target DFGs typically include thousands of nodes, a classical list scheduling algorithm [7] was adopted. A scheduling metric as presented in [8], considering both load balancing and communication cost was adopted in our current implementation.

4) Accelerator bitstream generation: The final step of the accelerator generation is to generate the instructions for each PE and the address sequences for the I/O buffers according to the scheduler's result, which will subsequently be incorporated into the configuration bitstream of the overlay produced from previous steps. Then we take advantage of the reconfigurability of SRAM based FPGAs and store the cycle-by-cycle configuration words using on-chip ROMs. The content of the ROMs are embedded in the bitstream and the `data2mem` tool from Xilinx [9] is used to update the ROM content of the pre-built bitstream directly. To complete the bitstream integration, `BMM` file that describes the organization and placements of the ROMs in the overlay is extracted from `XDL` file corresponding to the overlay implementation [10]. This bitstream integration process costs only a few seconds of the compilation time.

E. Accelerator library update

Accelerator library consists of a number of pre-built SCGRA overlay based accelerators with different configurations. It is the basis for the proposed rapid FPGA loop accelerator generation framework. In this section, we will illustrate how the accelerator library is updated given the hardware resource budget and target loop kernels.

Accelerator library update is essentially to pre-implement a group of SCGRA overlay based FPGA accelerators upon users' request, which may either target a specified application or a domain of applications. Since QuickDough design framework is developed to enhance the designers' design productivity and to make FPGA accelerator design accessible to high-level application designers, the library update which involves low-level overlay design and implementation must be automated so that it will not become a new barrier to the application developers.

Figure 6 presents the proposed automatic accelerator library update flow. It roughly consists of four steps i.e. DFG generation, common operation analysis, minimum accelerator configuration set analysis, and accelerator HDL model generation and implementation. Since DFG generation has been discussed in previous section, we will mainly detail the rest three steps in this section.

1) Common operation analysis: Assume that the operations used to construct the DFG is up to the DFG generation process, the common operation analysis step mainly decides the minimum operation set that is needed to support the target applications. It is possible to co-optimize the DFG generation and common operation set analysis, but it is beyond the discussion of this work. Currently, we just perform a union of the operation types included in the DFGs. It is trivial, but the minimum operation set can be decided automatically and rapidly.

2) Minimum accelerator configuration set analysis: Although the library can be implemented off-line, it does takes a long time to complete. Therefore, we try to find out the minimum set of accelerator configurations that need to be pre-implemented as the library and maintain the application coverage of the library at the same time.

The proposed SCGRA overlay based FPGA accelerator utilizes block RAM to implement the instruction memory, data memory, on-chip buffer as well as the address buffer, and block RAM is the hardware resource bottleneck. As a result, the library depends on how the block RAM budget is allocated to different components of the accelerators. Therefore, the minimum library can be obtained using equation Equation 3. Row and Col stand for the SCGRA overlay size and they are integers. IM, DM, $AIOB$ and $DIOB$ stand for the instruction memory capacity, data memory capacity, address IO buffer capacity and IO buffer capacity. They can only increase with the granularity of a primitive block RAM. B stands for the user specified block RAM budget.

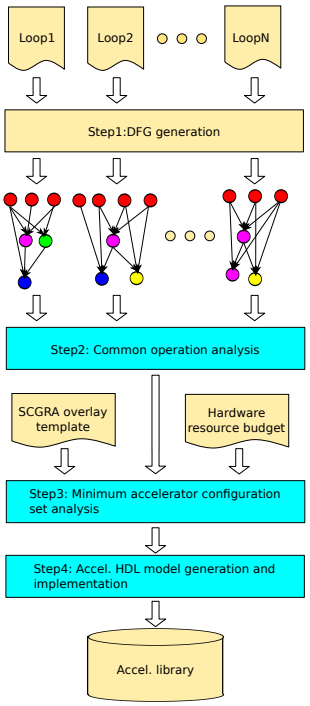

Fig. 6. Automatic SCGRA overlay based FPGA accelerator library update

$$Row \times Col \times (IM + DM) + AIOB + IOB \leq B \tag{3}$$

Moreover, empirical settings such as limiting data memory in each PE to a single primitive block RAM (i.e. $DM = 1$), constraining the difference between SCGRA row size and column size (i.e. $Col \leq Row \leq (Col + Gap)$, Gap is an integer) and setting $AIOB = IOB$ are employed to further reduce the number of accelerators pre-built in the library. Eventually, the accelerators in the library differ on Row, Col, IM and IOB.

3) Accelerator HDL model generation and implementation: With the proposed SCGRA overlay template and the accelerator configurations to be pre-built in the library, corresponding HDL models of the SCGRA overlay based FPGA accelerators are generated with a python script. Then the library can be implemented using the conventional hardware implementation tools. The lengthy implementations can be done in parallel. Moreover, the regular tiling structure even allows the implementations to be accelerated using macro based implementation techniques as presented in [11], which can be up to 20X faster than a standard HDL implementation with negligible timing and overhead penalty. After the implementation, implementation frequency is added to the corresponding accelerator configuration, which completes the whole library update process.

III. EXPERIMENTS

With an objective to improve designers' productivity in developing FPGA accelerators, the key goal of QuickDough is to reduce FPGA loop accelerator development time for a hybrid CPU-FPGA system. By using four typical loop kernels as the benchmark, we have evaluated the FPGA accelerator generation time with QuickDough. Meanwhile, to warrant the merit of such framework, the performance of the generated acceleration system should remain competitive. For that purpose, the performance is then compared against to that of software executed on an ARM processor. Finally, the pre-built accelerator library that affects both the design productivity and overhead of the resulting accelerators is also discussed.

The experiment section is organized as follows. We will first briefly introduce the benchmark programs in the following subsection and explain the basic experiment setup in Section III-B. Then we will discuss the accelerator library update in Section III-C. Finally, we will elaborate the loop accelerator generation time, performance and implementation overhead in Section III-D, Section III-E and Section III-F respectively.

A. Benchmark

Four applications were used as benchmark in this work, namely, a matrix-matrix multiplication (MM), a finite impulse response (FIR) filter, a K-mean clustering algorithm (KM) and a Sobel edge detector (SE). The basic parameters and configurations of the benchmark are illustrated in Table I.

TABLE I. DETAILED CONFIGURATIONS OF THE BENCHMARK

MM	FIR	SE	KM
Matrix Size	# of Input/ # of Taps+1	# of Vertical Pixels/ # of Horizontal Pixels	# of Nodes/Centroids/ Dimension
100	10000/50	128/128	5000/4/2
1000	100000/50	1024/1024	50000/4/2

B. Experiment Setup

The Xilinx implementation tools were run on a computer with Intel Core i5-3230M CPU and 8 GB of RAM. The resulting hardware-software platform was targeted at the Zedboard with both a hard ARM processor and an XC7Z020 FPGA. Software runtime was obtained from the ARM processor with -O3 compiling option. The accelerators were implemented on the FPGA of Zedboard. ISE 14.7 was used to implement the overlay based FPGA accelerators.

The acceleration system handles the input data loading, accelerator computation and output data storing sequentially. The performance of the accelerators is calculated using Equation 1 and Equation 2 in Section II. The data transfer latency used in Equation 2 is estimated based on Zedboard DMA between main memory and FPGA on-chip buffer through AXI high-performance port. The transfer latency is detailed in Table II. When the transfer size is not included in the table, a simple linear model is used to estimate its latency. Fmax and the number of cycles were extracted from the ISE14.7 and SCGRA scheduler respectively.

Loop unrolling is a critical design parameter for FPGA loop accelerators developed using QuickDough. Table III shows the loop unrolling factor that is used for the loop accelerator generation.

C. Accelerator library update

To ensure the rapid FPGA accelerator generation, we have implemented a group of SCGRAs based accelerators as the pre-built library by using the SCGRA overlay template. The library is developed to support all the four loop kernels, and it includes 12 3-source-1-destination operations as presented in Table IV. In addition, the pre-built accelerators share the same basic configurations as listed in Table V.

In addition, empirical settings are adopted to reduce the number of accelerators to be built in library. Input and output buffer depth are set to be the same. The depth of the address buffers are set to be twice with that of the I/O buffer depth. The data memory in each PE consumes only one primitive block RAM. The SCGRA overlay adopts a torus topology, and the row size is set to be equal to the column size or larger by one for the sake of performance. Eventually, different accelerator configurations merely differ on the on-chip I/O buffer depth, SCGRA size and instruction memory depth when the data width is determined.

To explore the library update efficiency, we have evaluated the number of accelerators included in the accelerator library and the time consumption to implement the library when different block RAM budgets ranging from 70, 140, 210, 280 to 350, different instruction memory depth exploration steps ranging from 1K(i.e. a single primitive block RAM36 on Zedboard), 2K to 4K are provided (Note that there are 140 RAMB36 on Zedboard FPGA) and different I/O buffer depth exploration steps ranging from 1K, 2K, 4K to 8K. As presented in Figure 7, when the BRAM budget increases, the number of accelerators in the library increases linearly. The library implementation time increases slightly faster as the SCGRA overlay size gets larger for larger resource budget. It can also be found that coarser instruction memory exploration step helps to reduce the number of accelerators in the library as well as the library implementation time. With this observation, coarser instruction memory depth exploration step will be adopted for larger BRAM budget to restrict the library implementation time while

TABLE II. DMA TRANSFER LATENCY ON ZEDBOARD THROUGH AXI HIGH PERFORMANCE PORT

transfer size (word, 32bit)	$\geq$512	256	128	64	32	16	$\leq$8
Latency per word (ns)	10.08	11.28	13.32	15.18	21.45	36.24	63

TABLE III. QUICKDOUGH UNROLLING SETUP

	MM	FIR	SE	KM
Unrolling	$1 \times 5 \times 100$	50×50	$16 \times 16 \times 3 \times 3$	$125 \times 4 \times 2$
DFG size	750	2500	9720	5768
Full Loop	$100 \times 100 \times 100$	10000×50	$128 \times 128 \times 3 \times 3$	$5000 \times 4 \times 2$

TABLE IV. OPERATION SET. IT COVERS ALL THE FOUR APPLICATIONS USED IN THE EXPERIMENTS.

Type	Opcode	Expression
MULADD	0001	Dst = (Src0 $\times$ Src1) + Src2
MULSUB	0010	Dst = (Src0 $\times$ Src1) - Src2
ADDADD	0011	Dst = (Src0 + Src1) + Src2
ADDSUB	0100	Dst = (Src0 + Src1) - Src2
SUBSUB	0101	Dst = (Src0 - Src1) - Src2
PHI	0110	Dst = Src0 ? Src1 : Src2
RSFAND	0111	Dst = (Src0 $\gg$ Src1) & Src2
LSFADD	1000	Dst = (Src0 $\ll$ Src1) + Src2
ABS	1001	Dst = abs(Src0)
GT	1010	Dst = (Src0 $>$ Src1) ? 1 : 0
LET	1011	Dst = (Src0 $\leq$ Src1) ? 1 : 0
ANDAND	1100	Dst = (Src0 & Src1) & Src2

TABLE V. SCGRA CONFIGURATION

SCGRA Topology	Instruction Memory Width	Data Memory	I/O Data Buffer Width	I/O Addr Buffer Width
Torus	72 bits	256×32 bits	32 bits	18 bits

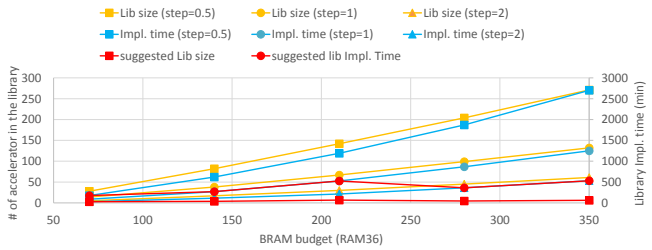

Fig. 7. Accelerator library size and implementation time given different BRAM budgets.

TABLE VI. ACCELERATORS GENERATED USING QUICKDOUGH

Opt. option	Resulting Accel. Config.	MM	FIR	SE	KM
O0	SCGRA size	2×2	2×2	2×2	2×2
	Inst. Mem depth	1K	2K	4K	3K
	I/O buffer depth	30K	28K	24K	26K
O1	SCGRA size	4×3	4×3	4×3	5×5
	Inst. Mem depth	1K	1K	2K	1K
	I/O buffer depth	20K	20K	14K	3K
O2	SCGRA size	3×2	4×4	4×4	5×5
	Inst. Mem depth	1K	1K	2K	1K
	I/O buffer depth	27K	15K	7K	3K

finer exploration step will be used for a smaller BRAM budget to maintain the coverage of the library. With this strategy, the suggested accelerator library update for different resource budgets is shown in Figure 7 as well. The accelerator library implementation time ranges from 170 minutes to 530 minutes.

D. Accelerator generation time

In this section, the loop accelerator generation time of QuickDough is evaluated. It is used as an indicator on the designer's productivity as it greatly limits the number of debug-edit-implementation cycles achievable per day.

In order to evaluate the loop accelerator generation time, we took the FPGA resource on Zedboard as the resource budget and pre-built the accelerator library with 1K instruction memory depth step. Then FPGA loop accelerators are generated for each application in the benchmark using the three different accelerator selection options respectively.

Table VI shows the configurations of the resulting FPGA accelerators.

Every implementation iterations in QuickDough involves 3 steps:

- DFG generation: The compute kernel is translated to corresponding DFG.
- DFG scheduling: Select an accelerator configuration and schedule the DFG to it through an operation scheduling.
- Bitstream generation: The scheduling result is embedded into a pre-built accelerator bitstream to produce the final FPGA bitstream of the compute kernel.

Figure 8 shows loop accelerator generation time of QuickDough. DFG generation step is very fast and it is almost negligible compared to the rest two steps. The DFG scheduling is relatively slower, but it usually completes in a few seconds. It is possible that the DFG scheduling process must be repeated and thus the time consumption increases accordingly when QuickDough can't find the accelerator with the user specified SCGRA overlay size in the accelerator library. This explains the relatively longer accelerator generation time for SE with 5×5 SCGRA overlay input. Fortunately, it is not a frequent

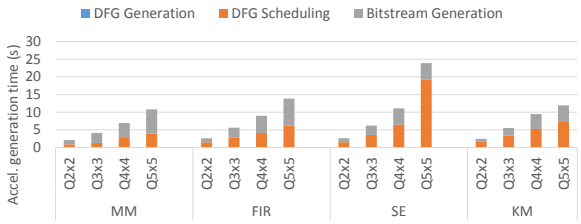

Fig. 8. Time consumption of loop accelerator generation using QuickDough.

situation and the worse case finishes in 25 seconds. The bitstream generation step typically takes a few seconds, and accelerators with larger SCGRA overlay are relatively slower.

Typically QuickDough produces an loop accelerator in seconds to a dozen seconds. Even though QuickDough may fail to provide accelerators with specified SCGRA overlay size sometimes, a smaller SCGRA overlay will be able to find immediately and the overall loop accelerator generation is still able to complete within a minute.

Clearly, the designer must spend the time to physically pre-implement the overlay architecture on the target FPGA, spending considerable time on the implementation tools. However, it can be reused by the whole benchmark. Moreover, the designer may iterate via the above rapid steps during design and debugging phases using an initial overlay implementation. Once the functionality is frozen, the designer may then opt to further optimize performance through more intensive overlay customization and update the library. We argue that the ability to separate functionality and optimization concern, and the possibility of performing rapid debug-edit-implement iterations in QuickDough are crucial factors that contribute to a high-productivity design experience.

E. Performance

While improving designers' productivity is the primary goal of QuickDough, the FPGA accelerators it generates must remain competitive in performance to software executed on general purposed processors. Therefore, execution time of the loop kernels executed on ARM processor of Zedboard and FPGA accelerators generated using QuickDough are compared.

Figure 9 shows the accelerator performance speedup over software execution on the ARM processor and execution time decomposition of the 4 benchmark programs. The reported loop execution time on the accelerators includes time spent on I/O data communication between FPGA and the ARM processor as well as FPGA computation.

The results in Figure 9 show that the accelerators generated using QuickDough are capable to provide 1X-10X performance speedup over software executed on ARM processor. For FIR, SE as well as KM which have abundant parallelism and moderate I/O requirements, the maximum speedup goes up to 10X, 6X and 7X respectively. Even when smaller SCGRA overlays are specified, clear performance speedup can be observed. MM optimized by simple loop unrolling is eventually reduced to a matrix-vector multiplication, so the compute kernel has low compute-to-IO rate and the single port connection between compute logic and input/output buffers becomes the bottleneck hindering the performance of the accelerator.

According to Figure 9, accelerators with larger SCGRA

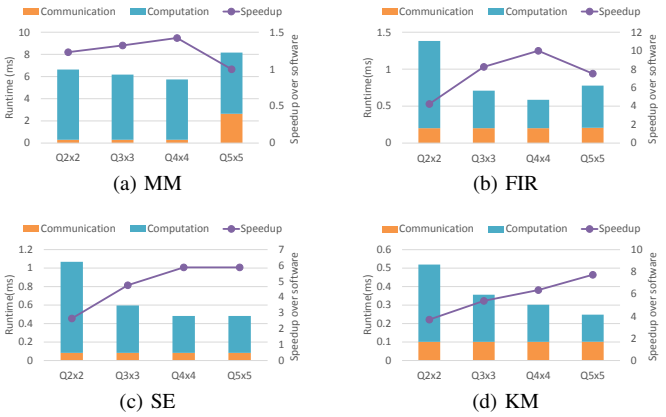

Fig. 9. Benchmark performance speedup over software executed on ARM processor and execution time decomposition of loop accelerators generated using QuickDough.

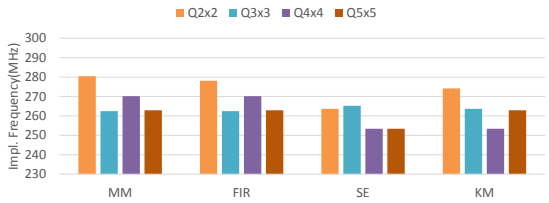

Fig. 10. Implementation frequency of the accelerators using QuickDough

overlay size typically achieve better performance than the ones with smaller overlay size. However, larger SCGRA overlay will not guarantee better performance for a few reasons. First of all, accelerators with larger overlay size consume more block RAM for instruction memory leaving less block RAM for I/O buffer. As a result, the I/O buffer may limit the transfer size between main memory and FPGA on-chip I/O buffer and reduce the chance of data reuse between DFGs included in a single group. This increased number of transfer between main memory and FPGA significantly limits the overall performance accordingly. This explains the performance degradation for the accelerator of MM with 5×5 overlay size. Secondly, accelerator with larger SCGRA overlay may confront scheduling problem as larger SCGRA overlay requires larger average cost between PEs and the compute performance may degrade as well. This is the major reason that accelerator performance of FIR with 5×5 SCGRA overlay degrades.

F. Implementation frequency and hardware overhead

One advantage of employing a simple and regular overlay architecture allows highly pipelined implementations with much higher frequencies as shown in Figure 10. The increased running frequency in turns results in higher overall performance of the system. Though both larger SCGRA overlay size and deeper instruction memory may degrade the implementation frequency, they can typically runs at more than 250MHz on Zedboard FPGA which is much higher than random logic synthesis on Zedboard.

Block RAM is the resource bottleneck for the SCGRA overlay based FPGA accelerators and it is almost fully utilized. LUT, FF and DSP48 overhead mainly depends on the SCGRA overlay size and only a portion of them are utilized. Figure 11 presents the detailed hardware overhead utilization of the

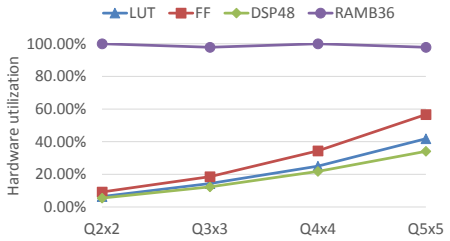

Fig. 11. Hardware overhead utilization of accelerators for MM

accelerators for MM. Hardware overhead utilization of the rest accelerators is quite similar to that of MM accelerators and is not presented here in case of redundancy.

IV. Related Work

Researchers have approached the challenge of lengthy hardware implementation tool run time from many angles. Focusing on the low-level FPGA EDA tools, researchers have looked improving their run time by making quality-runtime trade-offs [12] and by parallelizing the tools themselves [13], [14], [15], [16]. Other researchers take advantages of the dynamic partial reconfiguration capabilities of modern FPGAs to shorten run time by effectively reducing the user design size [17]. Yet another group of researchers approach the problem from a higher level, innovating on how these tools are being used from a design methodology's point of view. The use of modular design flow and by using pre-built hard macros [18], [19] have thus been explored. While these approaches have significantly reduced the hardware implementation time, they remain at least 2 orders of magnitude slower when compared to a the software compilation experience.

In recent years, there has been an increased interest in applying the concept of *overlay architectures* as a way to address this productivity challenge. An overlay architecture is a virtual intermediate architecture that is overlaid on top of the physical configurable fabric of an FPGA. Overlays with different granularity ranging from virtual FPGAs [20], [21], [22], [23], CGRA overlays [24], [25], [26], [27], [28] to soft processors [29] and GPU-like overlays [30] have been developed.

Among these overlays, CGRA overlays are particularly suitable for compute intensive loop acceleration as demonstrated by numerous prior works [31], [32]. A large number of CGRAs with different features have been developed and prototyped on FPGAs. A VLIW architecture based CGRA overlay was developed [24], which support dynamic topology customization. A heterogeneous CGRA overlay was proposed [25] that utilized a global multi-stage interconnection to achieve topology customization to adapt to different applications. A customized CGRA overlay called QUKU [26] was developed for DSP algorithms and it supported fast configuration for similarly to this work applications and slow configuration for distinct applications. Finally High-speed CGRA overlays were built in [27] and [28] by using the elastic pipeline technique and smart DSP reuse respectively to achieve better performance and higher throughput.

Our proposed fully pipelined synchronous coarse-grained reconfigurable array overlay continues this trend of exploiting coarse-grain reconfigurability to improve both design productivity and the resulting accelerator speed. In addition, QuickDough pays particular attention to data I/O between the host

and the accelerator, which often becomes the bottleneck both in terms of performance and designer productivity. For that, QuickDough provides buffer optimization and communication scheduling between hardware and software automatically, creating a seamless hardware-software codesign experience for the user. Finally, our overlay was designed to be *soft* from the beginning, featuring a template system to allow for rapid overlay generation with different CGRA topology and compute operations. An extensive application-specific customization framework that is able to automatically update the overlay library is in progress and is left as future work.

V. Conclusions

In this work, we have presented the QuickDough compilation framework for high productivity development of FPGA-based accelerators. QuickDough makes use of a soft coarse-grained reconfigurable array as an overlay architecture to greatly improve the designer's compilation experience. The QuickDough overlay is simple, regular, deterministic, and is highly scalable to future devices. Taking advantage of the overlay, the lengthy low-level implementation tool flow is reduced to a rapid operation scheduling problem and the compilation time of QuickDough is reduced to seconds, which contributes directly into higher application designers' productivity. Despite the use of an additional layer of overlay architecture on the target FPGA, the overall application performance remains competitive in many cases.

References

[1] A. Putnam *et al.*, "A reconfigurable fabric for accelerating large-scale datacenter services," in *Computer Architecture (ISCA), 2014 ACM/IEEE 41st International Symposium on*. IEEE, 2014, pp. 13–24.

[2] Khronos, "Opencl," https://www.khronos.org/opencl, 2015, [Online; accessed 9-May-2015].

[3] Xilinx, "Sdaccel," http://www.xilinx.com/products/design-tools/sdx/sdaccel.html, 2015, [Online; accessed 9-May-2015].

[4] H. K.-H. So and R. Brodersen, "A unified hardware/software runtime environment for FPGA-based reconfigurable computers using BORPH," *Transactions on Embedded Computing Systems*, vol. 7, no. 2, pp. 1–28, 2008.

[5] E. Lübbers and M. Platzner, "ReconOS: Multithreaded programming for reconfigurable computers," *ACM Trans. Embed. Comput. Syst.*, vol. 9, no. 1, pp. 8:1–8:33, Oct. 2009. [Online]. Available: http://doi.acm.org/10.1145/1596532.1596540

[6] A. Ismail and L. Shannon, "FUSE: front-end user framework for O/S abstraction of hardware accelerators," in *Field-Programmable Custom Computing Machines (FCCM), 2011 IEEE 19th Annual International Symposium on*, 2011, pp. 170–177.

[7] J. Schutten, "List scheduling revisited," *Operations Research Letters*, vol. 18, no. 4, pp. 167–170, 1996.

[8] C. L. Yu and H. K.-H. So, "Energy-efficient dataflow computations on FPGAs using application-specific coarse-grain architecture synthesis," in *Highly Efficient Accelerators and Reconfigurable Technologies, The 4th International Workshop on*. IEEE, 2012.

[9] Xilinx, "data2mem," http://www.xilinx.com/support/documentation/sw_manuals/xilinx11/data2mem.pdf, 2012, [Online; accessed 19-September-2012].

[10] C. Beckhoff, D. Koch, and J. Torresen, "The Xilinx design language (XDL): Tutorial and use cases," in *Reconfigurable Communication-centric Systems-on-Chip (ReCoSoC), 2011 6th International Workshop on*. IEEE, 2011, pp. 1–8.

[11] X. Yue, "Rapid overlay builder for xilinx fpgas," 2014.

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

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

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

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

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

[17] T. Frangieh *et al.*, "PATIS: Using partial configuration to improve static FPGA design productivity," in *Parallel Distributed Processing, Workshops and Phd Forum (IPDPSW), 2010 IEEE International Symposium on*, april 2010, pp. 1–8.

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

[19] S. Korf *et al.*, "Automatic hdl-based generation of homogeneous hard macros for fpgas," in *Field-Programmable Custom Computing Machines (FCCM), 2011 IEEE 19th Annual International Symposium on*. IEEE, 2011, pp. 125–132.

[20] 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*, 2012, pp. 93–96.

[21] 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

[22] 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.

[23] D. Koch, C. Beckhoff, and G. Lemieux, "An efficient FPGA overlay for portable custom instruction set extensions," in *Field Programmable Logic and Applications (FPL), 2013 23rd International Conference on*, Sept 2013, pp. 1–8.

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

[25] 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.

[26] S. Shukla, N. Bergmann, and J. Becker, "QUKU: a two-level reconfigurable architecture," in *Emerging VLSI Technologies and Architectures, 2006. IEEE Computer Society Annual Symposium on*, March 2006.

[27] 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.

[28] A. K. Jain, S. A. Fahmy, and D. L. Maskell, "Efficient overlay architecture based on dsp blocks," in *IEEE Symposium on Field-Programmable Custom Computing Machines (FCCM)*, 2015.

[29] 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.

[30] J. Kingyens and J. G. Steffan, "The potential for a GPU-Like overlay architecture for FPGAs," *Int. J. Reconfig. Comp.*, vol. 2011, 2011.

[31] R. Tessier and W. Burleson, "Reconfigurable computing for digital signal processing: A survey," *The Journal of VLSI Signal Processing*, vol. 28, no. 1, pp. 7–27, 2001.

[32] 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.