Interconnect Synthesis of Heterogeneous Accelerators in a Shared Memory Architecture

Yu-Ting Chen and Jason Cong
Computer Science Department, University of California, Los Angeles, CA, USA
E-mail: {ytchen, cong}@cs.ucla.edu

***Abstract*—An accelerator-rich architecture (ARA) is composed of heterogeneous accelerators with an on-chip memory system. Compared to the general-purpose processors, an accelerator demands short and predictable latency to its local on-chip memory to satisfy its performance target. Moreover, an accelerator requires a much higher off-chip memory bandwidth than a CPU since it consumes much more data in a given time period. Therefore, a customized on-chip memory system design is one of the keys to an efficient ARA. In this work we provide a two-layer interconnect synthesis method. We first provide an optimal layer of partial crossbar that connects the heterogeneous accelerators and shared memory banks with a minimum number of switches. The second layer of interconnect tries to interleave possible conflicting long-burst memory requests for prefetching data from off-chip memory. The experimental results show that we can reduce more than 45% of the switches of the partial crossbar compared to the best known method. This further leads to 53% reduction of LUTs and 34% reduction of slice utilization on a 30-accelerator FPGA prototype. Furthermore, the performance of an ARA can be improved by 36% - 52% with a well-designed interleaved network in a real ARA prototype for medical imaging applications. This prototype also shows a 7.44x energy efficiency gain over the state-of-the-art Xeon processors.**

***Keywords*—Interconnect synthesis, partial crossbar, heterogeneous accelerators, shared memory architecture**

I. Introduction

In the age of "dark silicon" [1][2], multi-core scaling is no longer sufficient for achieving the expected speedup levels while meeting the power limit. Compared to the general-purpose processors, application-specific accelerators can provide 10 - 1000x speedup by exploiting the application parallelism [3][4]. By 2022 we will be able to place hundreds to thousands of customized accelerators in an SoC [5]. Therefore, integrating general-purpose processor cores with a large number of customized accelerators is an attractive solution to multi-core processors. For the rest of our discussion, we use the term "accelerator-rich architecture (ARA)" when referring to this kind of architecture. Recently, accelerators have been integrated into real processors, such as the IBM wire-speed [6] and Intel Haswell processors. Meanwhile, many research efforts also demonstrate that ARAs can provide significant speedup and energy savings over conventional processors in various applications [2][3][7][8][9][10].

Accelerators improve performance by exploiting the application parallelism and data locality. An accelerator utilizes customized and deep pipelines to process a series of data. To maximize performance, the accelerator is usually designed with its initiation interval (II) to be one, i.e., fully pipelined to maximize its throughput. To enable a fully pipelined design, an accelerator must be able to simultaneously fetch multiple data elements from all its owned memory banks within one cycle. This can be easily achieved if all accelerators have their own private memory banks. However, as the number of accelerators in an ARA increases, the on-chip memory resource needs to be shared. The interconnects between the accelerators and shared memory banks have to provide (1) sufficient connectivity, and (2) the fixed one-cycle latency.

Figure 1 shows three conventional on-chip shared memory architectures. Multi-level caches (Figure 1(a)) are commonly used in multi-core processors. However, the fixed latency demand cannot be met since a cache miss can lead to uncertain access latency. Furthermore, a cache may not be able to service a number of simultaneous requests efficiently, even if interval banking is performed. A conventional SoC uses a system bus (Figure 1(b)) to share memory resource. However, when the number of devices increases, significant arbitration latency and area overhead to synthesize memory interfaces become the bottleneck [11]. Other interconnect topologies, such as ring and mesh, usually cannot meet the one-cycle latency constraint. A partial crossbar, shown in Figure 1(c), can provide sufficient connectivity and the one-cycle latency with moderate area overhead. In this paper we conduct optimization for the partial crossbar for heterogeneous accelerators.

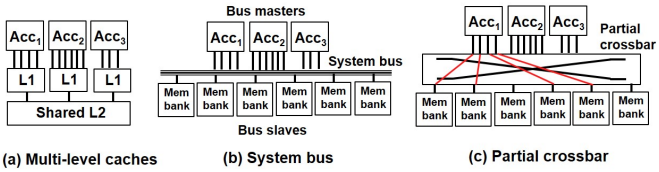


Fig. 1. Three conventional on-chip shared memory architectures.

Another important issue is how to efficiently fetch the data from off-chip memory. In CPU cores, the off-chip memory accesses are issued when misses occur in the last-level caches. An accelerator improves performance by grouping a series of memory accesses together into a burst request and then issuing multiple long burst requests to prefetch data into its own on-chip memory banks. In an ARA, it is common to have multiple memory ports to fully use the available memory bandwidth. However, if multiple burst requests are issued to the same interface, the outstanding requests must wait.

In this paper we aim to design the interconnects to satisfy the need for sufficient connectivity, fixed latency, and efficient off-chip prefetching capabilities for the shared ARA memory system. Our contributions can be summarized as follows:

- **An Optimal Partial Crossbar Between Heterogeneous Accelerators and Memory Banks:** Assuming the number of accelerators that can be simultaneously powered on is bounded, we provide a novel algorithm to synthesize the interconnect between heterogeneous accelerators and memory banks as a partial crossbar with the minimum number of switches. Compared to the state-of-the-art synthesis algorithm [11], we further generalize the optimal solution for accelerators with heterogeneous memory bank demands.
- **Interleaved Network Between Memory Banks and Memory Interfaces:** We generate the interleaved network to interleave the simultaneous long burst requests to limited memory interfaces based on the proposed optimal partial crossbar topology. Performance can thus be significantly improved when request conflicts are reduced.

We first evaluate our synthesis algorithm by comparing the number of switches in the partial crossbar with [11], the full-capacity crossbar and full crossbar. Based on experimental results, we can reduce more

than 45% of switches compared to the work [11]. To validate the effectiveness of the reduction of switches, we synthesize our partial crossbar design with 30 accelerators on the Xilinx Zynq platform. Our method can reduce 47% of switches, 53% of LUTs and 34% of slices when compared to [11]. We further demonstrate the efficiency of our interleaved network through a real ARA prototype with heterogeneous medical imaging accelerators. The interleaved network can reduce the execution time by 36% - 52% by improving the prefetching process. The ARA prototype also shows a 7.44x energy efficiency gain over the state-of-the-art Xeon processors.

II. Preliminary

A. Accelerator-Rich Architectures

An ARA is composed of the general-purpose cores, heterogeneous accelerators, on-chip memories, and interconnects [2][8][9][12]. Figure 2 demonstrates the accelerator plane of an ARA, which can be decomposed into the following components: (1) heterogeneous accelerators, (2) shared memory banks, (3) direct memory access controllers (DMACs), (4) physical memory ports (interfaces), and (5) two layers of interconnects [12]. The heterogeneous accelerators can have a different number of memory bank demands. For example, Acc_1 needs four memory banks, while Acc_2 requires six. These heterogeneous demands make the design of interconnects more difficult than that of the "homogeneous" demands.

In the dark silicon era, the power wall limits the number of transistors that can be powered on simultaneously. For an accelerator island, the power budget is determined by (1) the number of powered-on accelerators at a certain time period, and (2) the number of memory banks in this island. The number of powered-on accelerators leads to the dynamic power consumption [11], while the number of memory banks significantly contributes to leakage power [8]. Therefore, the number of on-chip memory banks and the maximum number of powered-on accelerators should be limited under a given power budget.

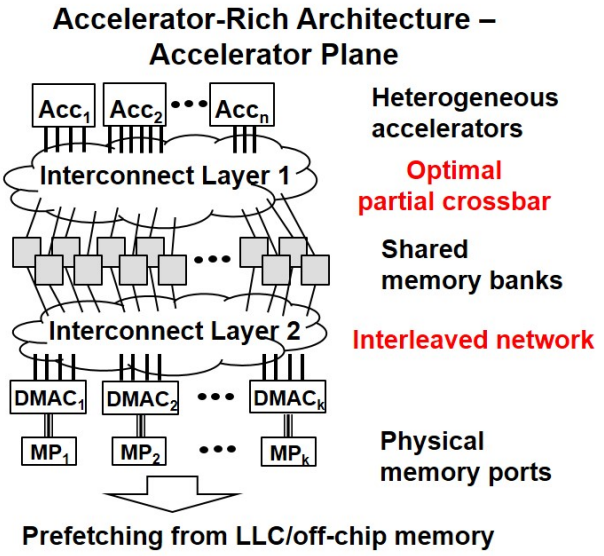


Fig. 2. An accelerator-rich architecture (ARA) – the accelerator plane.

B. Limitations of Existing Methods and Motivation

The authors in [13] summarized the complexity for several types of crossbar designs, such as the full-capacity crossbar [14], for general signal routing in FPGA. However, these methods are not efficient enough when the constraint on the number of powered-on accelerators is considered. The work in [11] first investigated the partial crossbar synthesis when the power budget is limited, which is the work that is most relevant to ours. However, the method proposed in [11] can only generate the minimum partial crossbar for accelerators with homogeneous memory bank demands, as shown in Figure 3(a). The homogeneous bank demands do not match the heterogeneity nature of accelerators. Therefore, we are motivated to synthesize the minimum partial crossbar for accelerators with heterogeneous bank demands, as shown in Figure 3(b).

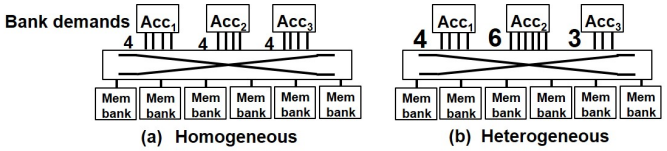


Fig. 3. Limitation of the synthesis method developed in [11].

The crossbar (bus matrix) is also used to provide sufficient connectivity for buses in high-performance systems [15]. The crossbar network is usually designed in a cascaded fashion. However, this design style cannot meet the demand that accelerators can fetch multiple data every cycle to maintain high throughput. Similarly, the network-on-chip (NoC) topologies [16][17] and the combination of buses and NoC [18] for large-scale multi-core processors cannot satisfy the high-throughput need.

In this paper the first important question that we address is how to synthesize the interconnects between heterogeneous accelerators and shared memory banks. We use the configurable partial crossbar to provide one-cycle fixed latency and guaranteed connectivity. The second question is how to design an interleaved network between memory banks and DMACs based on the optimal partial crossbar topology. The topology synthesis in this layer has not been considered together with a given partial crossbar topology from the existing work.

III. Optimal Partial Crossbar Design

In this section we first discuss the crossbar configurability and then define the problem formulation of the partial crossbar synthesis. Following that, we propose a novel algorithm to synthesize the optimal partial crossbar between the accelerators and the shared memory banks. The algorithm guarantees that the number of switches in the partial crossbar is minimum, while supporting at least any c accelerators in the island that can be simultaneously powered on. For simplicity of discussion, we summarize the key notations used in this paper in Table I.

TABLE I. Major Notations

Notation	Explanation
n	the number of heterogeneous accelerators in the island
a_i	accelerator i, $1 \leq i \leq n$ $\{a_1, a_2, ..., a_n\}$ are sorted in descending order based on the memory bank demand
d_i	the number of memory bank demand of a_i
c	the number of simultaneous powered-on accelerators
m	the number of shared memory banks
b_i	memory bank i, $1 \leq i \leq m$
k	the number of DMACs and MPs

A. Crossbar Configurability

Figure 4 is an example of the partial crossbar design between the heterogeneous accelerators and the shared memory banks. In this example, banks 1 to 4 are assigned to Acc_1, while banks 3

to 8 are assigned to Acc_2. Acc_1 and Acc_2 cannot be powered on simultaneously since they share bank 3 and bank 4. In our assumption, one memory bank cannot be simultaneously used by two accelerators or any two ports in one accelerator. This is because a memory bank only has two ports. One is connected to an accelerator while the other is connect to a DMAC. Also, an accelerator accesses its own memory banks every cycle to achieve high throughput. The crossbar is designed to be configurable for sharing these two memory banks for Acc_1 and Acc_2. We assume that two-port memory banks are used. One port of a memory bank is connected to the accelerators while the other port is connected to one DMAC.

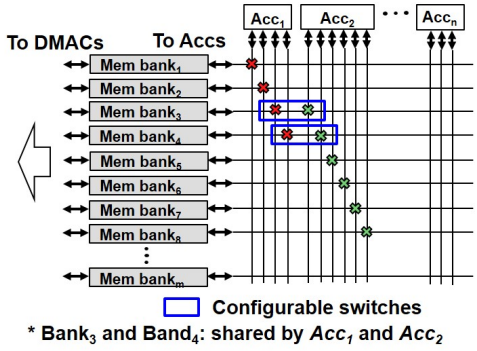


Fig. 4. An example of a configurable crossbar.

Algorithm 1 Optimal partial crossbar synthesis

```
1: port_map: the mapping between the accelerator ports and shared memory banks; 1st dimension: the accelerator index; 2nd dimension: the accelerator's port index
2: d: the array recording the memory bank demands for all accelerators
3: n: the number of accelerators
4: m: the number of memory banks
5: c: the number of simultaneous powered-on accelerators
6: procedure OptCrossbar(d, n, m, c)
7:     bank_index ← 0
8:     for i ← 1 to c do ▷ 1st nested loop: for the largest c accelerators
9:         for j ← 1 to d[i] do ▷ Assign consecutive memory banks
10:            port_map[i][j] ← bank_index
11:            bank_index ← bank_index + 1
12:        end for
13:    end for
14:    for i ← 1 to c do ▷ 2nd nested loop: for the rest n - c accelerators
15:        bank_index ← port_map[i][1]
16:        for j ← c + 1 to n do
17:            if bank_index + d[j] > port_map[i][d[i]] then
18:                bank_index ← port_map[i][1]
19:            end if
20:            for k ← 1 to d[j] do ▷ Assign consecutive memory banks
21:                port_map[j][k] ← bank_index
22:                bank_index ← bank_index + 1
23:            end for
24:        end for
25:    end for
26:    return port_map
27: end procedure
```

B. Minimum Required Memory Banks

Suppose that the number of allowed simultaneous powered-on accelerators within the power budget is c, the primary goal is to provide a configurable partial crossbar which makes any c accelerators in this island simultaneously work together. The first question we try to answer is what is the minimum number of required memory banks to support c accelerators to be powered on simultaneously.

Lemma 1 *Given a set of accelerators, $\{a_1, a_2, ..., a_n\}$, with non-increasing memory band demands $d_1 \geq d_2 \geq ... \geq d_n$, and c, the minimum number of required memory banks is at least $\sum_{i=1}^{c} d_i$.*

Proof: Omitted. ■

C. Problem Formulation

Given the number of simultaneous powered-on accelerators c, the number of shared memory banks m ($m = \sum_{i=1}^{c} d_i$), and n accelerators, our goal is to minimize the total number of switches of the partial crossbar.

D. Optimal Partial Crossbar Synthesis

We propose Algorithm 1 to synthesize the partial crossbar, which meets these two requirements: (1) the minimum number of memory banks shown in Lemma 1, and (2) the minimum number of switches equal to the lower bound shown in Theorem 1. In this section we illustrate the high-level concepts of the proposed algorithm. We will discuss (1) the lower bound of the minimum required number of switches and (2) the optimality analysis of the proposed algorithm in Section III-E and Section III-F, respectively.

We have n heterogeneous accelerators, and only c accelerators can be powered on simultaneously. In Algorithm 1, we first assign the crossbar switches for the c accelerators, $\{a_1, a_2, ..., a_c\}$, with the largest memory bank demand. For these c accelerators, one memory port exactly maps to one memory bank (lines 8 - 13). Next, we assign the switches for the remaining $n-c$ accelerators (lines 14 - 25). Each port in accelerators, $\{a_c+1, ..., a_n\}$, is mapped to c memory banks. The number of switches generated from Algorithm 1 is $m + c \times \sum_{i=c+1}^{n} d_i$ (Theorem 2), which is the minimum number of required switches (Theorem 1). Note that the memory ports of an accelerator are assigned in a contiguous way to one *Region* (Definition 1) for accelerators $\{a_1, a_2, ..., a_c\}$ or to c *Regions* for accelerators $\{a_c+1, ..., a_n\}$. Figure 5(a) shows an example of the partial crossbar topology generated from Algorithm 1.

E. The Lower Bound of The Required Switches

To design the minimum (optimal) partial crossbar, we first find the lower bound of the minimum required switches and then provide a proof for this lower bound.

Theorem 1 *The lower bound for the number of switches required in the partial crossbar is $m + c \times \sum_{i=c+1}^{n} d_i$, where m is equal to $\sum_{i=1}^{c} d_i$.*

Proof: Omitted. ■

Theorem 1 can be easily used to find the minimum number of switches when all accelerators have the same number of memory bank demands, as described in [11]. We use r to denote the homogeneous memory bank demand. The minimum required memory banks, m, is equal to $c \times r$. As demonstrated in Equation 1, Theorem 1 can further generalize the theorem derived from [11] for synthesizing accelerators with homogeneous bank demands.

$$\begin{aligned} m + c \times \sum_{i=c+1}^{n} d_i &= m + c \times (n - (c+1) + 1) \times r \\ &= c \times r + c \times (n-c) \times r = m \times (1 + n - c) \end{aligned} \quad (1)$$

F. Algorithm Optimality Analysis

In this section we first prove that Algorithm 1 can synthesize a partial crossbar with a minimum number of switches by Theorem 2.

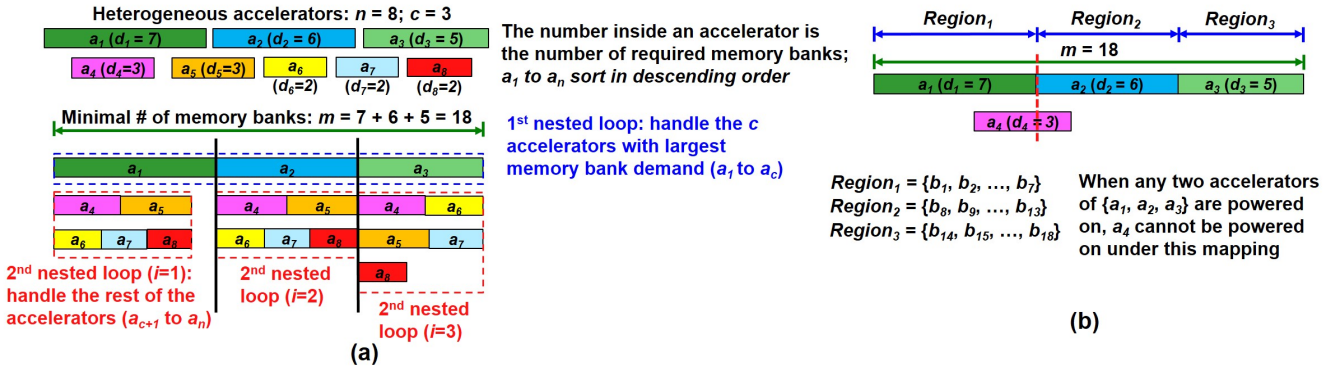


Fig. 5. (a) An example of partial crossbar synthesis using Algorithm 1. (b) An example demonstrating the insight of Algorithm 1 design.

Theorem 2 *Algorithm 1 synthesizes the partial crossbar with* $m + c \times \sum_{i=c+1}^{n} d_i$ *switches, which is equal to the lower bound described in Theorem 1.*

Proof: Omitted. ■

To further prove that the crossbar generated from Algorithm 1 can power on any c accelerators, we first define the term $Region$.

Definition 1 *"Regions" are the ranges for the contiguous memory bank assignments for the c accelerators with the largest memory bank demands. All the c Regions are not overlapped with one another based on the Algorithm 1.*

Lemma 2 *For any two accelerators, if their bank assignments reside at two different Regions, i.e., non-overlapped Regions, the two accelerators can be powered on simultaneously. (We suppose the other accelerators are currently powered off.)*

Lemma 3 *Based on Lemma 2, a partial crossbar with c Regions can support at least c accelerators that are powered on simultaneously.*

Based on Definition 1, we have three $Regions$ for the synthesized partial crossbar shown in Figure 5(a). We can further deduce Lemma 2 and Lemma 3 based on the definition. The key insight of Algorithm 1 is to avoid the case of the cross-Region bank assignment such as a_4, shown in Figure 5(b). By using Algorithm 1, the bank assignment of the accelerators is aligned well inside the $Regions$, which avoids the cross-Region assignment. We show that Algorithm 1 can provably power on any c accelerators by Theorem 3.

Theorem 3 *The crossbar generated from Algorithm 1 can power on any c heterogeneous accelerators simultaneously.*

Proof: Omitted. ■

Due to the space limit of the paper, we will put all the detailed proofs in a technical report as a reference for the readers.

IV. Interleaved Network

A. Conflicts of Burst Prefetch Requests

In an ARA, the off-chip data prefetching memory access patterns have the following properties. First, the prefetch requests issued from accelerators are usually long memory bursts for performance concerns. In our design, it is at the page granularity (4KB), ranging from one to four pages (4 - 16KB). Second, an accelerator issues multiple burst requests simultaneously. The number of simultaneous burst requests depends on the memory bank demands of an accelerator; this ranges from 5 to 12 in our design. Third, the partial crossbar described in Section III maps the ports of an accelerator to contiguous memory banks. The interconnects between memory banks and DMACs need to be carefully designed considering the partial crossbar topology.

Figure 6 shows a real topology synthesized using Algorithm 1. In this example, a_1 is powered on and six simultaneous burst requests are issued from a_1 to prefetch the required input data into $\{b_1, b_2, ..., b_6\}$. If the interconnect layer between memory banks and DMACs is not designed carefully, request conflicts will arise. Figure 6(a) shows that four burst requests are sent to $DMAC_1$, while two requests are sent to $DMAC_2$. The four requests issued to $DMAC_1$ will become the bottleneck since MP_1 services the requests in a sequential way. Each request is a 4KB to 16KB long burst request, which leads to a significant performance degradation.

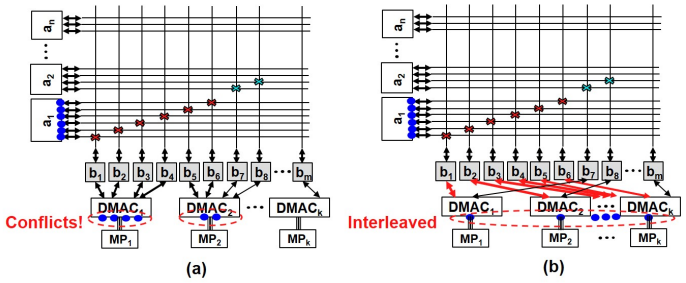


Fig. 6. (a) Burst requests conflict at $DMAC_1$. (b) Burst requests are interleaved to different DMACs.

B. Interleaved Network Design

We consider the following important properties for interleaved network design in order to resolve possible conflicts.

1. The partial crossbar topology According to Algorithm 1, the memory banks assigned to an accelerator are contiguous. We design the mapping function described in Equation 2 to map the memory banks to DMACs. The mapping function can guarantee the distribution of simultaneous burst requests distributed uniformly to different DMACs. Figure 6(b) shows an example our interleaved network design. In this example ($k = 6$), the memory banks $\{b_1, b_2, ..., b_6\}$ are mapped to $\{DMAC_1, DMAC_2, ..., DMAC_6\}$, respectively. The six interleaved burst requests can be serviced in parallel with available memory bandwidth without conflicts.

$$MemBankToDMAC(i) = i \bmod k, \quad i \in 1..m \tag{2}$$

2. Accelerator usage pattern We believe that the priority of interleaving requests within an accelerator is more important than that of interleaving requests across accelerators. This is because not all of the accelerators can start simultaneously with limited power budgets. The heterogeneous nature of accelerators also reduces the possibility that accelerators launch simultaneously. Even if multiple accelerators are running simultaneously, the requests from multiple

accelerators may still interleave. However, a single accelerator cannot start to work until all data are prefetched and ready. Therefore, we choose the topology generated in Equation 2 to ensure that the burst requests of a specific accelerator can be distributed uniformly across DMACs.

V. Experimental Results

A. Case Study: A Real Medical Imaging ARA Prototype

1) Appliations and Prototyping Platform: We are interested in accelerating the medical imaging processing pipeline for computerized tomography (CT) images [19]. First, the noise and blur need to be removed. Second, the process would align the current image with previous images of an individual. Third, a region of interest for diagnosis is segmented. To accelerate the pipeline, we include four accelerator kernels—$gradient$, $gaussian$, $rician$, and $segmentation$ in our ARA design.

We use the Xilinx Zynq ZC706 as our prototyping platform. The Zynq SoC, which is composed of a dual-core ARM Cortex-A9 and FPGA fabrics, is used to prototype an ARA (Figure 2). We use ARACompiler [12][20] to prototype the ARA for the medical imaging pipeline. In this ARA, we have five heterogeneous accelerators: (1) two $gradient$, (2) one $gaussian$, (3) one $rician$, and (4) one $segmentation$. The memory bank demand of each accelerator is shown in Table II. The shared memory banks are synthesized using the on-chip BRAMs, while both interconnect layers are realized using FPGA LUTs and routing resources. In Zynq, there are four physical memory ports (k = 4) for accelerators to prefetch data from off-chip DRAM.

TABLE II. Memory bank demands (i.e., the number of ports of each accelerator)

Type	gradient	gaussian	rician	segmentation
Bank demand	6	5	8	12

2) Optimal Partial Crossbar: Figure 7(a) shows the minimum number of required memory banks, while Figure 7(b) shows the number of switches generated from [11] and Algorithm 1. We set c to four in the ARA prototype, and thus 8.8% of switches are saved.

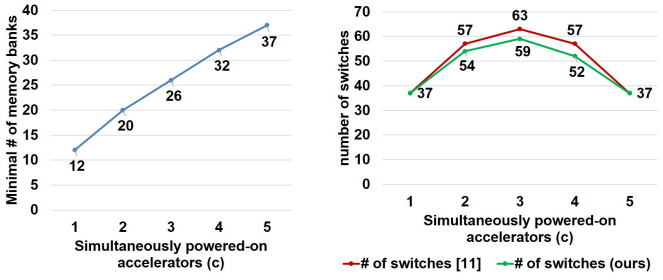


Fig. 7. (a) The minimum number of required memory banks for this ARA. (b) The number of switches generated from [11] and our algorithm.

3) Effectiveness of Interleaved Network: To evaluate the effectiveness of the interleaved network, we measure performance from our FPGA prototype. This is because the request conflicts occur during runtime based on the accelerator utilization, which is difficult to model in an analytical way. We compare our interleaved network design from Equation 2 to an non-interleaved design described in Equation 3. The non-interleaved mapping is similar to the case demonstrated in Figure 6(a). The interleaved network can reduce the runtime from 36% to 52%, as shown in Figure 8. Note that the x-axis represents the powered-on accelerators.

$$MemBankToDMAC(i) = i \text{ div } k, \quad i \in 1..m \tag{3}$$

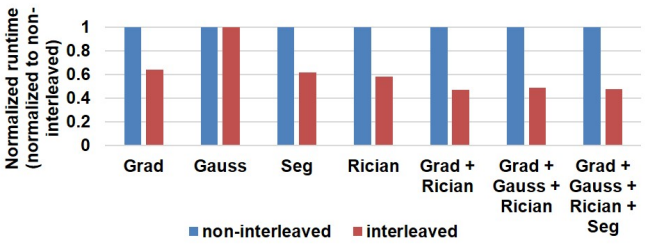


Fig. 8. Effectiveness of interleaved network

4) Energy-Efficiency of the ARA prototype: Table III shows the performance and power results of the $denoise$ application in our ARA prototype and the state-of-the-art processors. $denoise$ is composed of $gradient$ and $rician$ kernels and executes the two kernels sequentially for 10 iterations. We use OpenMP to implement $denoise$ for evaluating Xeon processors. The result of Cortex-A9 uses single thread. $denoise$ is compiled using gcc with -O2 option. Table III shows that our prototype can achieve 7.44x and 2.22x energy efficiency over Xeon and ARM, respectively.

TABLE III. Performance and power comparison over (1)ARM Cortex-A9, (2)Intel Xeon (Haswell), and (3)ARA

	Cortex-A9	Xeon (24 threads)	ARA
Freq.	667MHz	1.9GHz	Acc@100MHz CPU@667MHz
Runtime(s)	28.34	0.55	4.53
Power	1.1W	190W(TDP)	3.1W
Total Energy	2.22x	7.44x	1x

B. Scalability Study of the Optimal Crossbars

To evaluate our crossbar design, we compare the number of switches of (1) full crossbar, (2) full-capacity crossbar [14], (3) the crossbar generated from [11], and (4) the crossbar generated from Algorithm 1 over different numbers of powered-on accelerators. A full-capacity crossbar can only connect any m inputs to m outputs, which is less flexible than a full crossbar. In our case, m is the number of memory banks. The algorithm in [11] can only guarantee the minimum design when all accelerators have the same memory bank demands. Note that the number of switches is measured based on the minimum required memory banks derived from Lemma 1 for all the cases in Figure 9. We generate the memory demand of each accelerator at random, ranging from 4 to 16.

We evaluate Algorithm 1 using three sizes of designs: (1) 10 accelerators, (2) 50 accelerators, and (3) 100 accelerators, as shown in Figure 9. For larger designs, Algorithm 1 generates only about 1% to 10% switches of full crossbar and full-capacity crossbar, respectively. Compared to the state-of-the-art method [11], we can reduce the number of switches by more than 45% in larger designs, as shown in Figure 9(b)(c). This means that the method in [11] is still far from the optimal solution for many cases.

Note that the method in [11] can only generate optimal results when c = 1 and $c = n$, where n is the number of accelerators in the ARA. For all the other cases, only Algorithm 1 can generate a crossbar with the minimum number of switches and outperform [11]. When c is equal to 1, the accelerator with the largest memory demand shares its memory banks with all the other accelerators. When c is equal to n, each accelerator has its own memory banks. In both cases, the number of switches is equal to the sum of the memory bank demands of all accelerators.

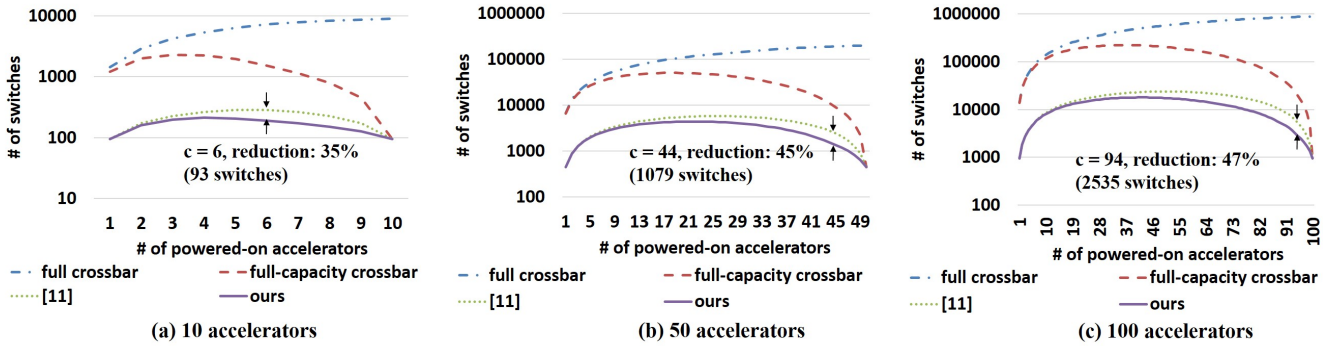


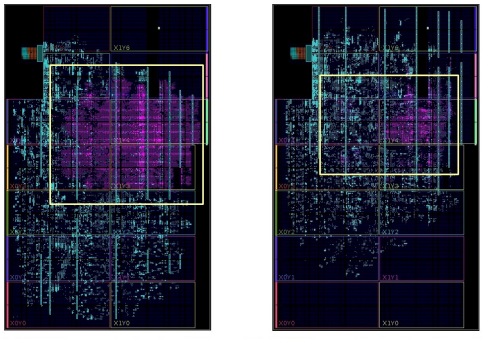
Fig. 9. The number of switches of (1) the full crossbar, (2) the full-capacity crossbar, (3) the crossbar generated using [11], and (4) the crossbar generated using Algorithm 1, over different numbers of powered-on accelerators.

C. FPGA Validation of the Partial Crossbars

To further validate the effectiveness of switch reduction, we prototype the partial crossbar designs generated from Algorithm 1 and [11] on the Xilinx ZC706. We generate a large design with 30 heterogeneous accelerators with 156 memory banks and only 20 accelerators can be powered on simultaneously (n=30, c=20, m=156). In order to save the FPGA resource for synthesizing the partial crossbar, we use dummy accelerators instead of real ones. Figure 10 shows the resource utilization of the FPGA of the generated partial crossbars of Algorithm 1 and [11]. Algorithm 1 can generate a much less congested partial crossbar by reducing 47% of switches, which leads to 53% LUT and 34% slice reduction when compared to [11].

VI. Conclusions

In this work we first provide an optimal partial crossbar synthesis algorithm that guarantees the required number of switches to be minimum while supporting at least a given number of accelerators that can be powered on simultaneously. Second, we improve the data prefetching efficiency by interleaving multiple simultaneous long burst requests into different physical memory ports for better bandwidth utilization. With the optimal synthesis algorithm, we can reduce 45% of switches when compared to previous work. Our prototyping results show that we can improve performance by 36% - 52% using the interleaved network for a real ARA.



(a) FPGA snapshot using [11] (b) FPGA snapshot using Algorithm 1

Fig. 10. Snapshots of the Zynq FPGA using (a) [11] and (b) Algorithm 1 for partial crossbar synthesis for the configuration (n=30, c=20, m=156). The purple color represents the resources used by the partial crossbar while the blue one represents the rest of utilized resources.

VII. Acknowledgement

This work is partially supported by the Center for Domain- Specific Computing under the Intel Award 20134321 and NSF Award CCF-1436827. It is also supported in part by C-FAR, one of six centers of STARnet, a Semiconductor Research Corporation program sponsored by MARCO and DARPA.

References

[1] H. Esmaeilzadeh, E. Blem, R. St. Amant, K. Sankaralingam, and D. Burger, "Dark silicon and the end of multicore scaling," in *ISCA*, 2011, pp. 365–376.

[2] G. Venkatesh, J. Sampson, N. Goulding, S. Garcia, V. Bryksin, J. Lugo-Martinez, S. Swanson, and M. B. Taylor, "Conservation cores: Reducing the energy of mature computations," in *ASPLOS*, 2010, pp. 205–218.

[3] R. Hameed, W. Qadeer, M. Wachs, O. Azizi, A. Solomatnikov, B. C. Lee, S. Richardson, C. Kozyrakis, and M. Horowitz, "Understanding sources of inefficiency in general-purpose chips," in *ISCA*, 2010, pp. 37–47.

[4] T.-C. Chen, S.-Y. Chien, Y.-W. Huang, C.-H. Tsai, C.-Y. Chen, T.-W. Chen, and L.-G. Chen, "Analysis and architecture design of an hdtv720p 30 frames/s h.264/avc encoder," *IEEE Trans. Cir. and Sys. for Video Technol.*, vol. 16, no. 6, pp. 673–688, Sep. 2006.

[5] "The international technology roadmap for semiconductors (ITRS), system drivers," http://www.itrs.net/, 2007.

[6] H. Franke, J. Xenidis, C. Basso, B. M. Bass, S. S. Woodward, J. D. Brown, and C. L. Johnson, "Introduction to the wire-speed processor and architecture," *IBM J. Res. Dev.*, vol. 54, no. 1, pp. 27–37, Jan. 2010.

[7] J. Cong, M. A. Ghodrat, M. Gill, B. Grigorian, and G. Reinman, "Architecture support for accelerator-rich cmps," in *DAC*, 2012, pp. 843–849.

[8] M. J. Lyons, M. Hempstead, G.-Y. Wei, and D. Brooks, "The accelerator store: A shared memory framework for accelerator-based systems," *ACM Trans. Archit. Code Optim.*, vol. 8, no. 4, pp. 48:1–48:22, Jan. 2012.

[9] J. Cong, M. A. Ghodrat, M. Gill, B. Grigorian, and G. Reinman, "Charm: A composable heterogeneous accelerator-rich microprocessor," in *ISLPED*, 2012, pp. 379–384.

[10] J. Cong, M. A. Ghodrat, M. Gill, B. Grigorian, K. Gururaj, and G. Reinman, "Accelerator-rich architectures: Opportunities and progresses," in *DAC*, 2014, pp. 1–6.

[11] J. Cong and B. Xiao, "Optimization of interconnects between accelerators and shared memories in dark silicon," in *ICCAD*, 2013, pp. 630–637.

[12] Y.-T. Chen, J. Cong, and B. Xiao, "Aracompiler: a prototyping flow and evaluation framework for accelerator-rich architectures," in *ISPASS*, 2015, pp. 157–158.

[13] G. Lemieux, P. Leventis, and D. Lewis, "Generating highly-routable sparse crossbars for plds," in *FPGA*, 2000, pp. 155–164.

[14] S. Nakamura and G. M. Masson, "Lower bounds on crosspoints in concentrators," *IEEE Trans. Comput.*, vol. 31, no. 12, pp. 1173–1179, Dec. 1982.

[15] M. Jun, D. Woo, and E.-Y. Chung, "Partial connection-aware topology synthesis for on-chip cascaded crossbar network," *IEEE Trans. Comput.*, vol. 61, no. 1, pp. 73–86, Jan. 2012.

[16] J. Balfour and W. J. Dally, "Design tradeoffs for tiled cmp on-chip networks," in *ICS*, 2006, pp. 187–198.

[17] D. Sanchez, G. Michelogiannakis, and C. Kozyrakis, "An analysis of on-chip interconnection networks for large-scale chip multiprocessors," *ACM Trans. Archit. Code Optim.*, vol. 7, no. 1, pp. 4:1–4:28, May 2010.

[18] R. Das, S. Eachempati, A. Mishra, V. Narayanan, and C. Das, "Design and evaluation of a hierarchical on-chip interconnect for next-generation cmps," in *HPCA*, 2009, pp. 175–186.

[19] A. Bui, K.-T. Cheng, J. Cong, L. Vese, Y.-C. Wang, B. Yuan, and Y. Zou, "Platform characterization for domain-specific computing," in *ASP-DAC*, 2012, pp. 94–99.

[20] Y.-T. Chen, J. Cong, M. Ghodrat, M. Huang, C. Liu, B. Xiao, and Y. Zou, "Accelerator-rich cmps: From concept to real hardware," in *ICCD*, 2013, pp. 169–176.