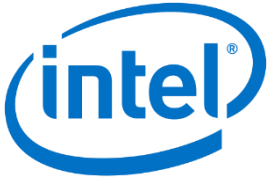


HBM Connect: High-Performance HLS Interconnect for FPGA HBM

Young-kyu Choi, Yuze Chi, Weikang Qiao,
Nikola Samardzic, and Jason Cong

University of California, Los Angeles

High Bandwidth Memory (HBM) becoming increasingly popular in FPGA boards:

Xilinx Establishes FPGA Adaptive Compute Clusters at Leading Universities

May 5, 2020 by Rich Brueckner

HPC wire

Since 1987 - Covering the Fastest Computers in the World and the People Who Run Them

- Home
- Technologies
- Sectors
- COVID-19

Intel Launches Stratix 10 NX FPGAs Targeting AI Workloads

By John Russell

June 18, 2020

Intel shipping FPGA with HBM

Intel is shipping Stratix 10 MX FPGAs with integrated High Bandwidth Memory DRAM (HBM2).

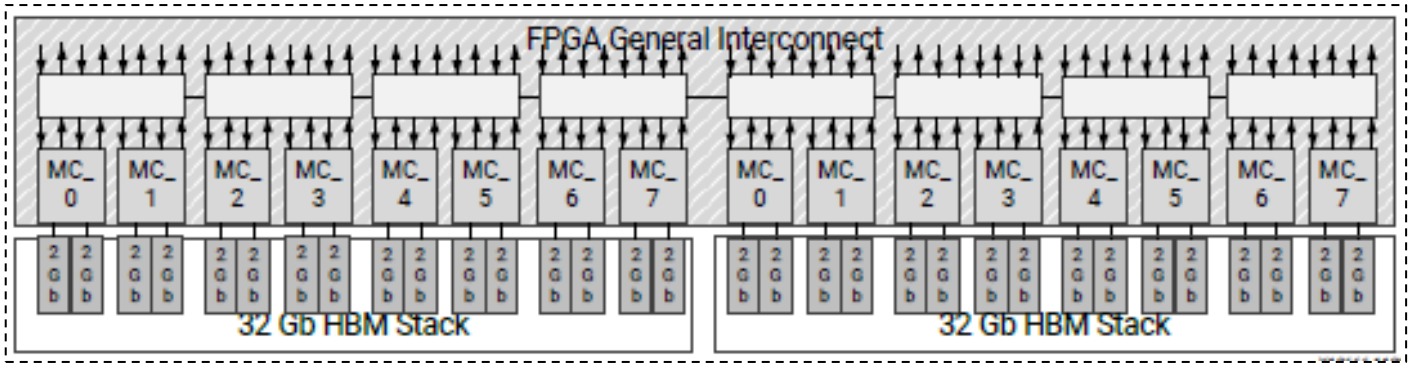
Xilinx Alveo U50 Accelerator

HPC wire

Xilinx Adds High-Bandwidth Memory Capabilities to its Virtex UltraScale+ Portfolio

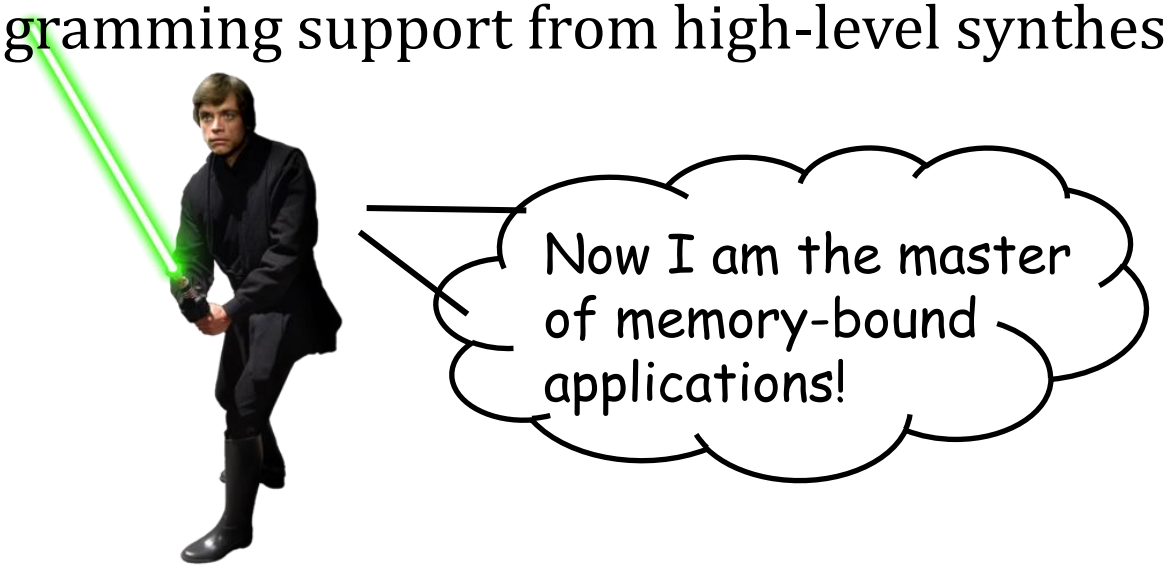
June 26, 2020

- Stratix 10 MX / Alveo U280 HBM boards provides 32 pseudo channels (PCs), each with ~13GB/s BW



+ Reconfigurability and energy-efficiency of FPGAs

+ Programming support from high-level synthesis (HLS) tools



...Really?

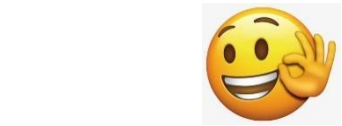
<FPGA + HBM + HLS>

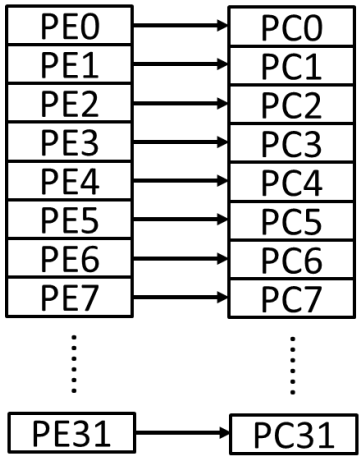
https://the-stuingtion-and-hiatt-grey-cinematic-universe.fandom.com/wiki/Luke_Skywalker

- Implementation result of memory-bound applications on Alveo U280

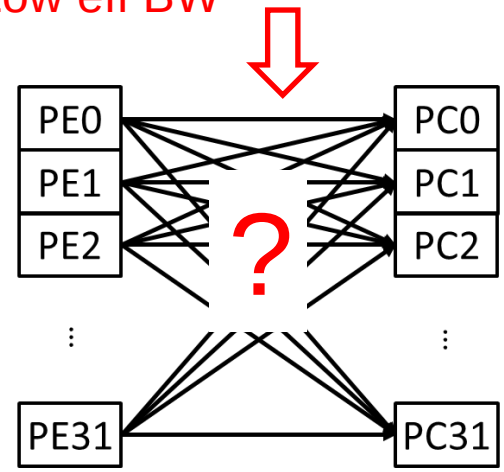
Appli-cation	PC #	KClk (MHz)	EffBW (GB/s)	EffBW/PC (GB/s)
MV Mult	16	300	211	13.2
Stencil	16	293	206	12.9
Bucket sort	16	277	65	4.1
Merge sort	16	196	9.4	0.59

PEs compute data in each PC independently
-> High eff BW



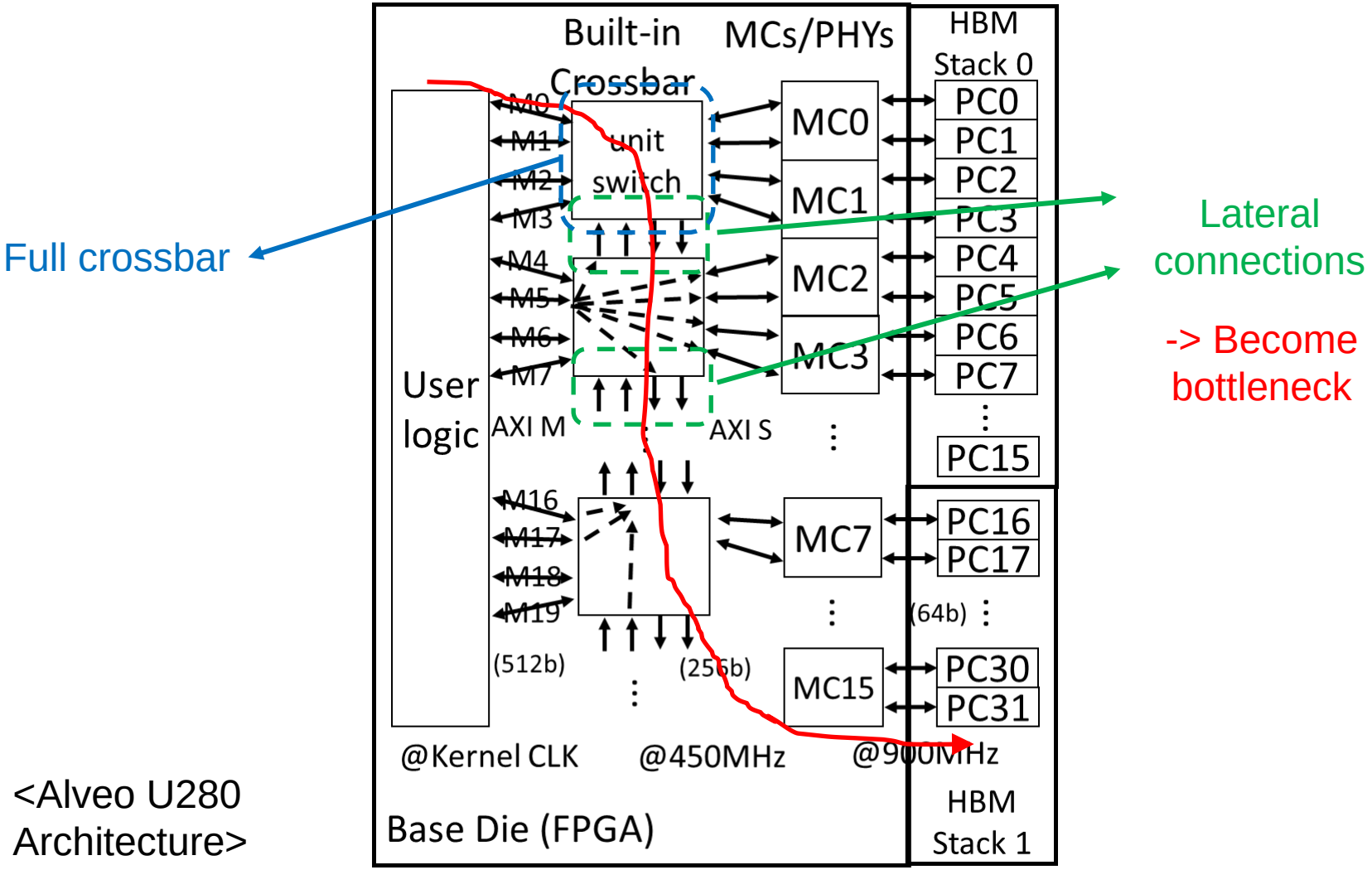


PEs need to read/write from multiple PCs
-> Low eff BW



More evaluation result in Y. Choi, et al., "When HLS Meets FPGA HBM: Benchmarking and Bandwidth Optimization." arXiv preprint arXiv:2010.06075 (2020).

- Reason for the low effective BW:
  - 1. Shared links among the built-in switches become a bottleneck



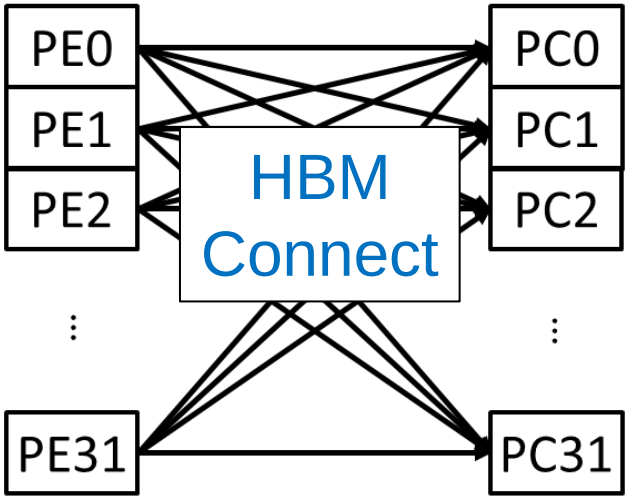
<Alveo U280 Architecture>

- Reason for low effective BW:
  - 2. Difficult to infer burst access to multiple PCs in current HLS programming environment

```
11 void key_write( hls::stream< ... > & in_fifo,
12             ap_uint<512>* pe0_pc0, pe0_pc1, ... pe0_pc15){
13   while(...){
14 #pragma HLS pipeline II=1
15     {data, bucket_id} = in_fifo.read();
16     switch( bucket_id ){
17       case 0 : pe0_pc0[addr0++] = data; break;
18       case 1 : pe0_pc1[addr1++] = data; break;
19       ...
20       case 15 : pe0_pc15[addr15++] = data; break;
21 } }
```

- Data comes in random order
  -> Data may be sent to different PC in next iteration
  -> AXI burst length of one

- How to solve this problem?

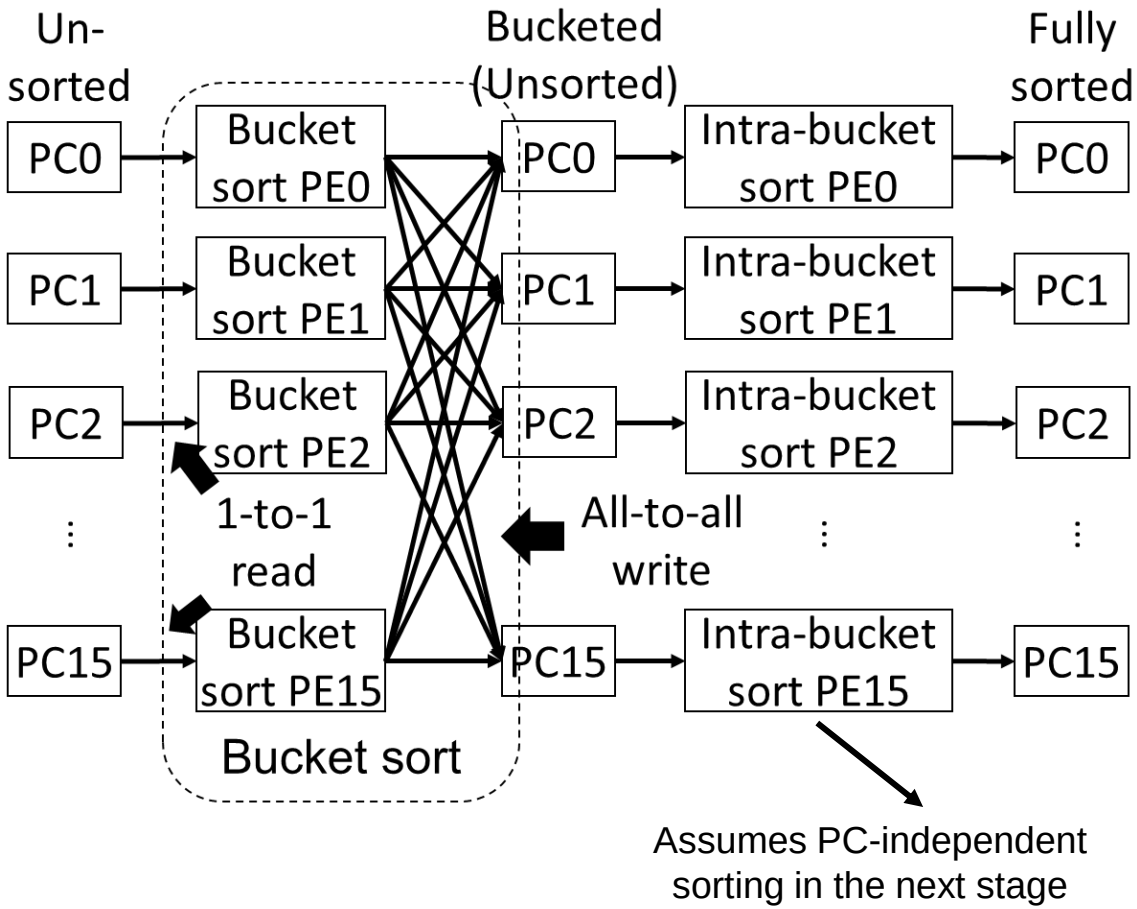


  - We propose HBM Connect: a high-performance customized interconnect (between PEs and the HBM) for HBM FPGA board
    - HLS-based optimization techniques to increase the throughput of AXI bus masters and switching elements
    - High-performance customized crossbar
    - Finds the design point with best BW-resource tradeoff

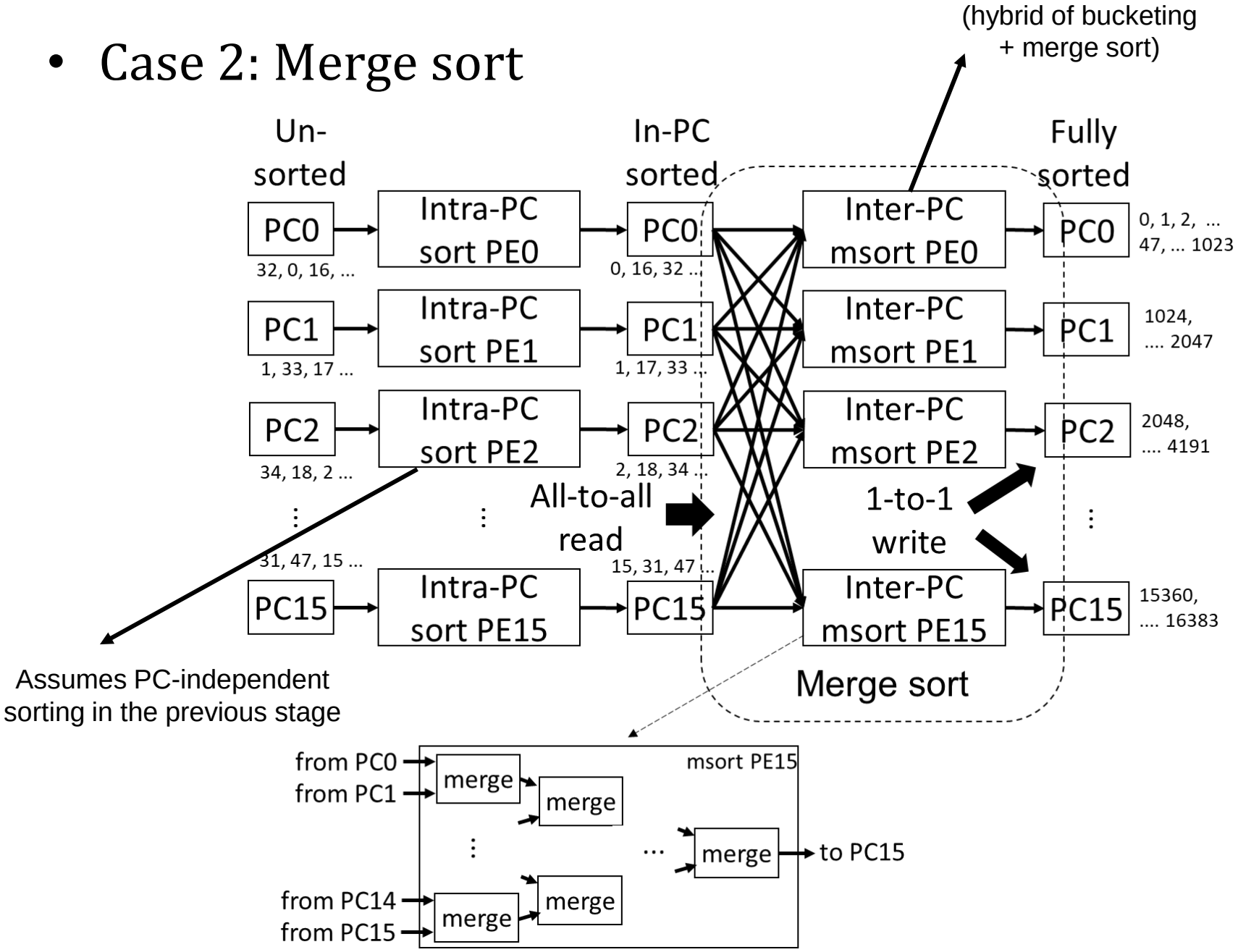
- Introduction
- Case Studies
- Design Space and Problem Formulation
- Built-in Crossbar and Custom Crossbar
- AXI Burst Buffer
- Experimental Result

Case Studies

- Case 1: Bucket sort

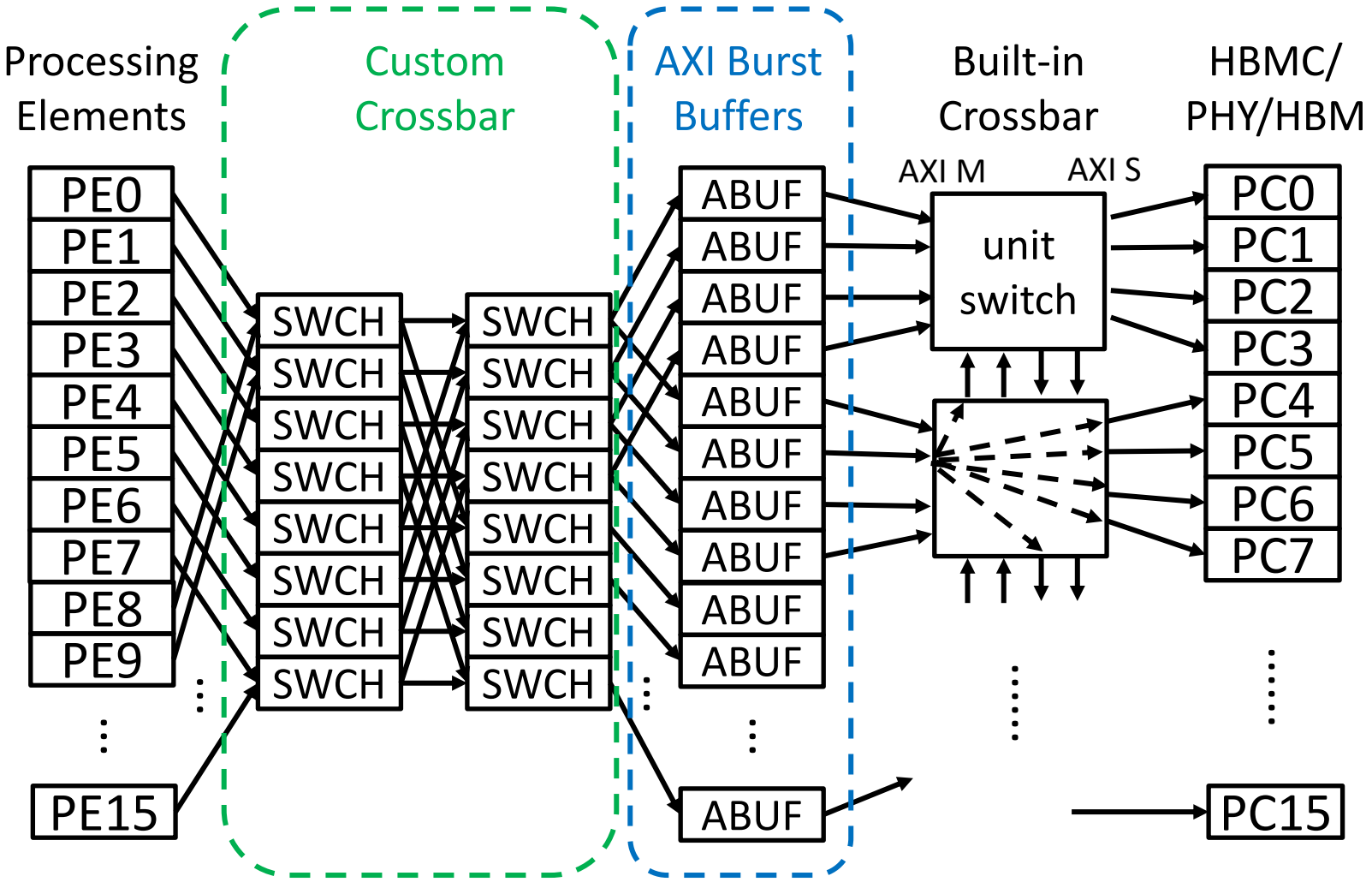


- Case 2: Merge sort



- Introduction
- Case Studies
- Design Space and Problem Formulation
- Built-in Crossbar and Custom Crossbar
- AXI Burst Buffer
- Experimental Result

Design Space of HBM Connect



CXBAR= {4,3,2,1,0}
Reduces traffic in built-in xbar

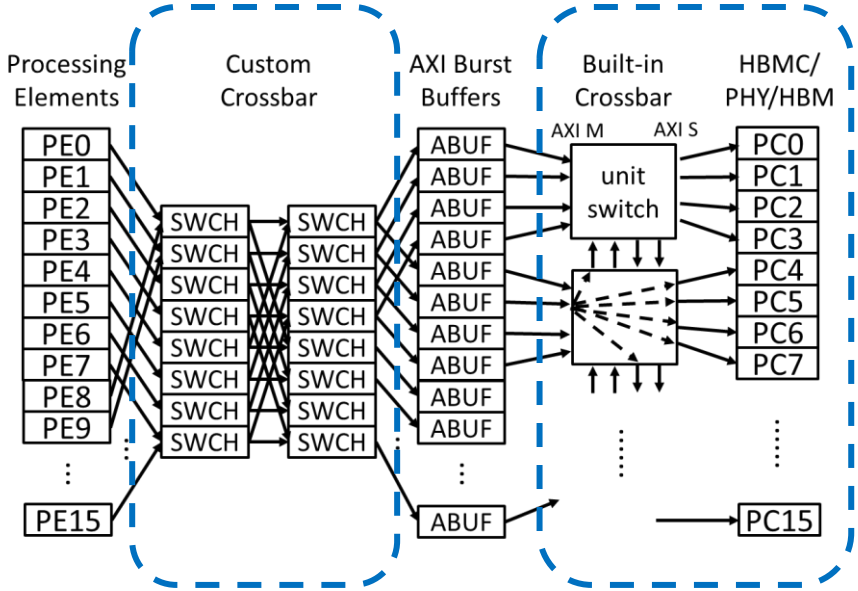
ABUF= {256, …, 2, 1, 0}
Increases AXI burst length

Problem Formulation

- Assumptions
  - Memory-bound application
  - HLS kernel in dataflow style (with streaming FIFOs)
  - Data is read/written in sequential address to HBM

- Problem :
  - Given the data size between all PEs and PCs, find a design space (CXBAR, ABUF) that maximizes BW^2/LUT
    - AT^2
    - Maximize effective bandwidth while using small resource as possible
    - BW term is more important than resource term (memory bound application)
    - Metric BW^2/LUT may be replaced with BW^2/FF or $BW^2/BRAM$

- Introduction
- Case Studies
- Design Space and Problem Formulation
- Built-in Crossbar and Custom Crossbar
- AXI Burst Buffer
- Experimental Result



Built-in crossbar and HBM

- Single PC benchmarking result

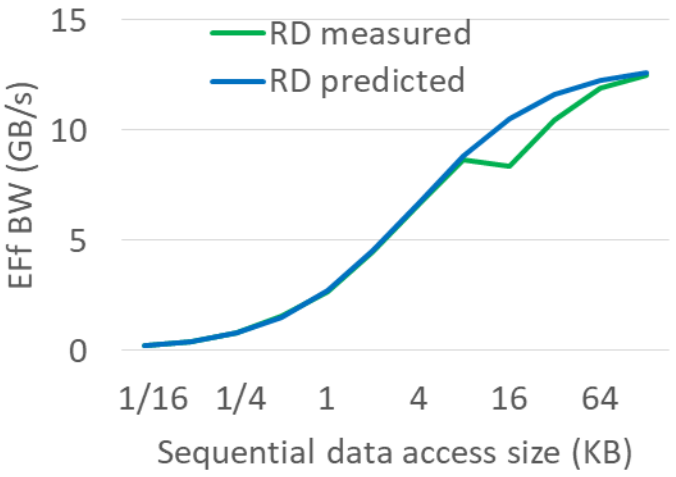
Maximum BW:

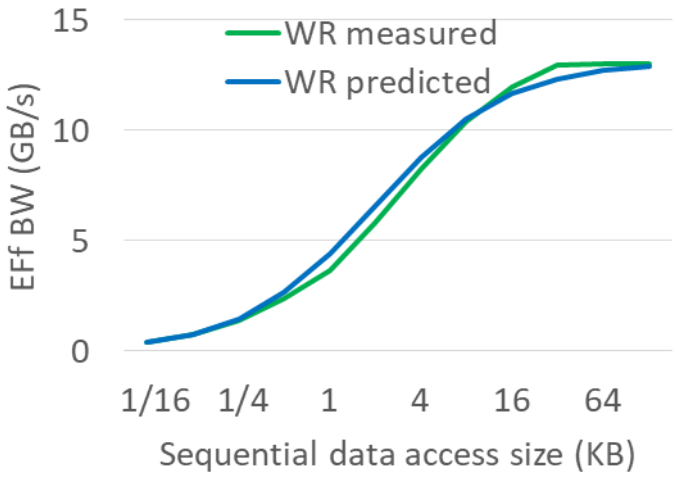
Read & Write	Read only	Write only	Ideal
12.9	13.0	13.1	14.4

Latency:

	Read lat	Write lat
Total	289 ns	151 ns

$$t_{BLEN_{ij}} = BLEN_{ij}/BW_{max} + LAT \rightarrow BW_{ij} = BLEN_{ij}/t_{BLEN_{ij}}$$



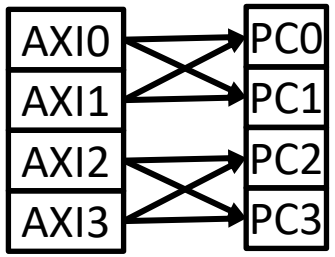


* Our HLS HBM benchmarking work is open-sourced at https://github.com/UCLA-VAST/hbmbench

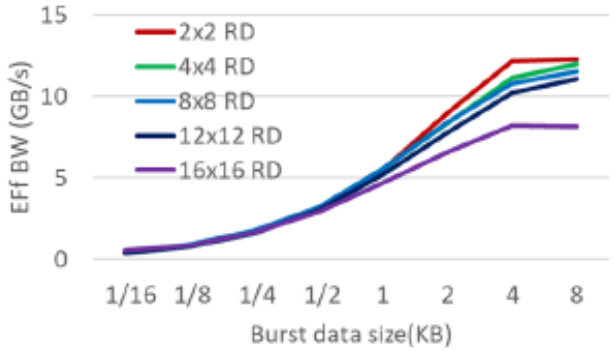
Limitation of Built-in Crossbar

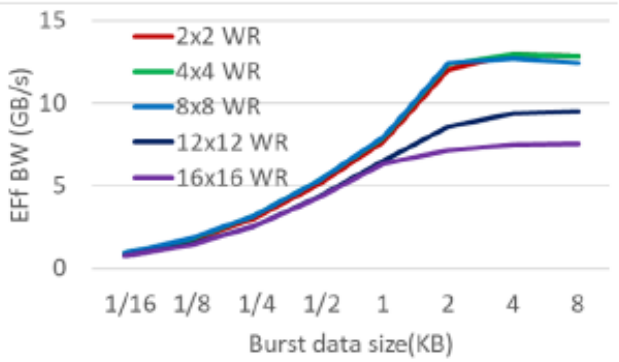
- Many-to-many unicasting BW
  - Test configuration: 2x2~16x16 AXI masters x PCs RD/WR

Ex) 2x2 WR:



  - Test result:

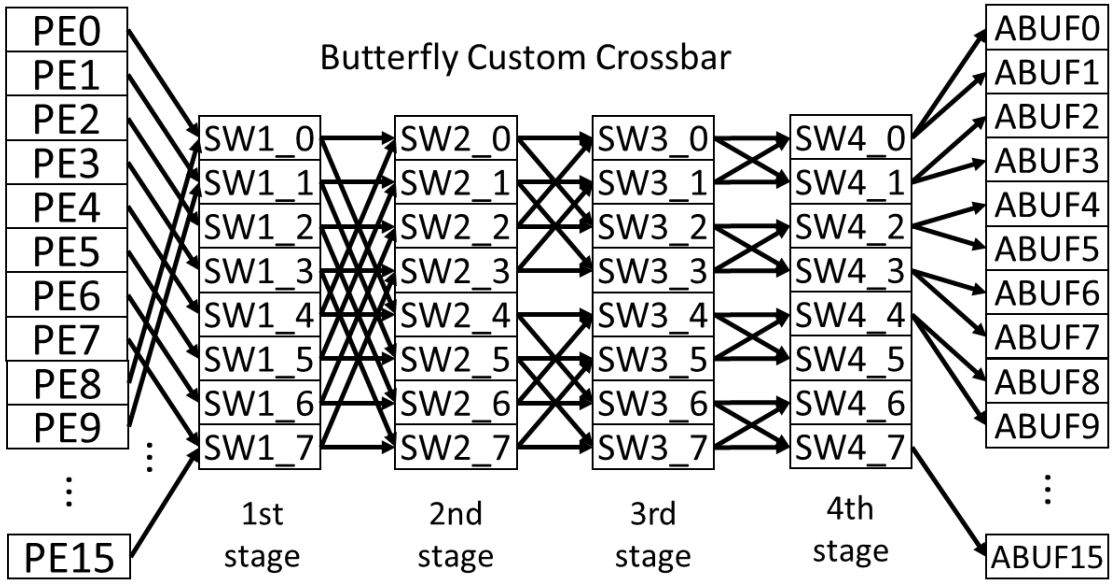


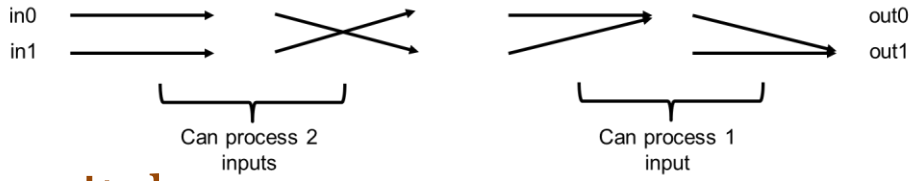


    - Severe effective BW reduction in 16x16

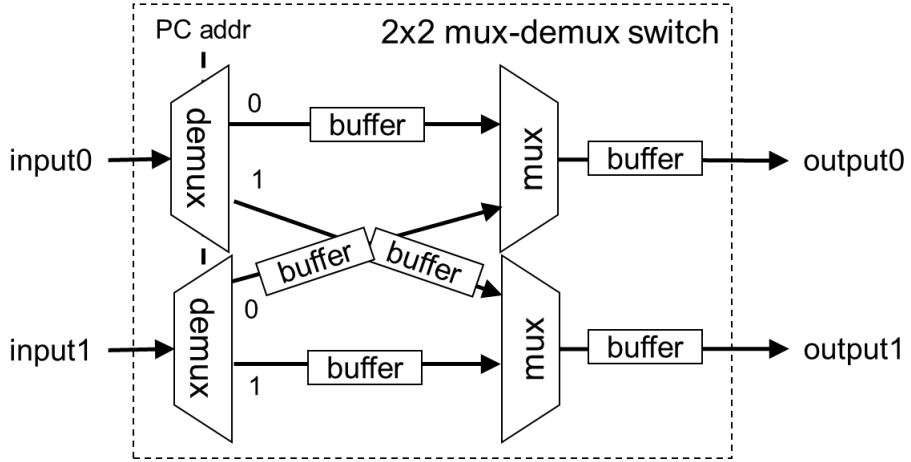
Customized crossbar

- Topology of crossbar
  - Difficult to route fully-connected crossbar
  - Decided to use multistage crossbar composed of 2x2 switches
    - Several topologies exist – Omega, Clos, Benes, butterfly, etc
  - Chose butterfly network
    - Reason: Sends data across many hops of AXI masters in its early stage -> good LUT-BW tradeoff with just few custom xbar stages





- 2x2 switch in HLS
  - Problem with typical 2x2 switch
    - Can send both input data to output if the data's output ports are different
    - Average throughput for random input : 1.5 elements per cycle
  - Proposed solution: mux-demux switch
    - Idea: Decompose a 2x2 switch into simple operations to be performed in parallel



Note:
- Buffer implemented as FIFO

- Can produce 2 outputs per cycle,

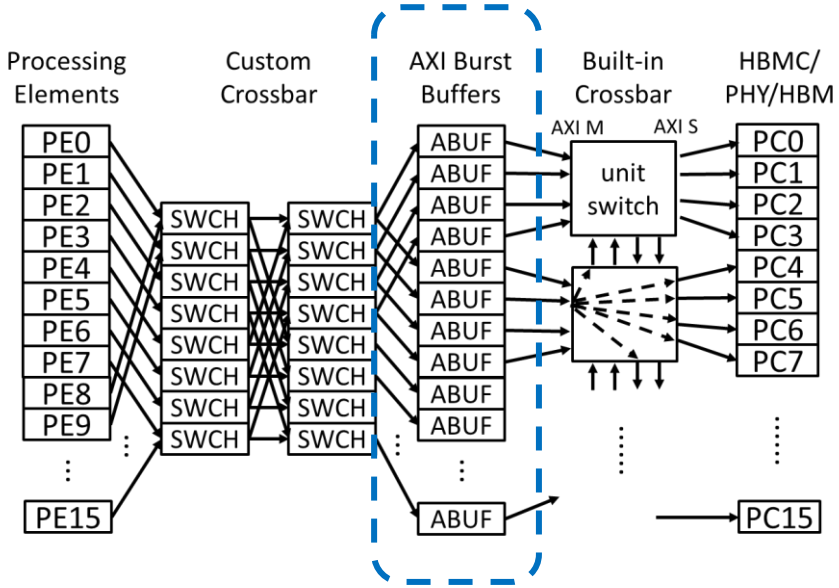
  if consecutive length of data for output < buffer size

- Mux-demux switch (…continued)
  - Resource (post PnR) & throughput comparison
    - Tested with random input in a standalone Vivado HLS test
    - Comparable resource between typical SW vs mux-demux SW
      - Complex control in Typ SW
    - Better throughput than Typ SW

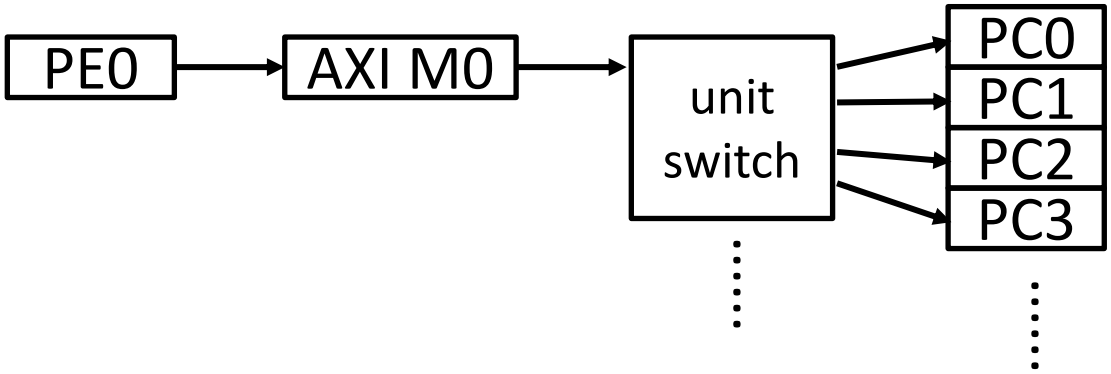
	Typ SW	Mux-Demux SW		
Buffer size	-	4	8	16
LUT	3184	3732	3738	3748
FF	4135	2118	2124	2130
Thr (Exp.)	1.49	1.74	1.86	1.93
Thr (Est.)	1.5	1.74	1.88	1.94

HBM Connect default configuration

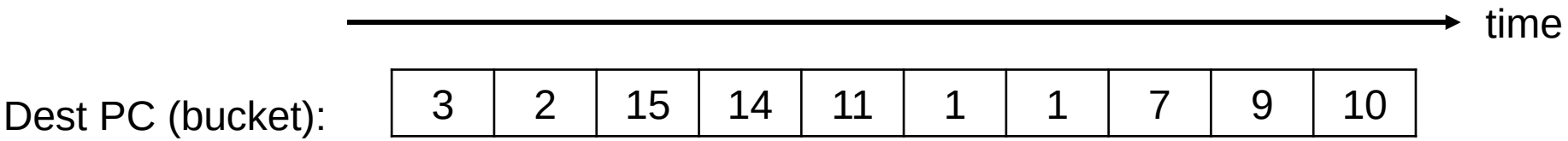
- Introduction
- Case Studies
- Design Space and Problem Formulation
- Built-in Crossbar and Custom Crossbar
- AXI Burst Buffer
- Experimental Result



- Conventional HLS coding style: Direct access from PE to AXI master

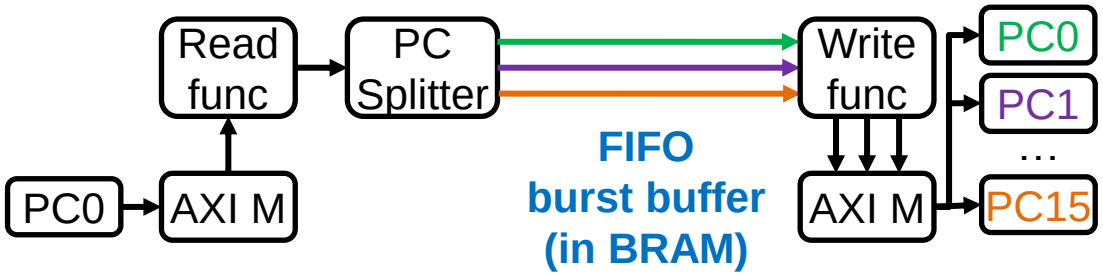


  - Problem: In bucket sort, two consecutive keys may have different destination PCs (random order)



  - Existing HLS tools do NOT automatically infer burst access to different PCs
  - AXI burst length of one

- Intuitive solution:
  - FIFO-based burst buffer**
    - Instantiate burst buffer for each destination PC



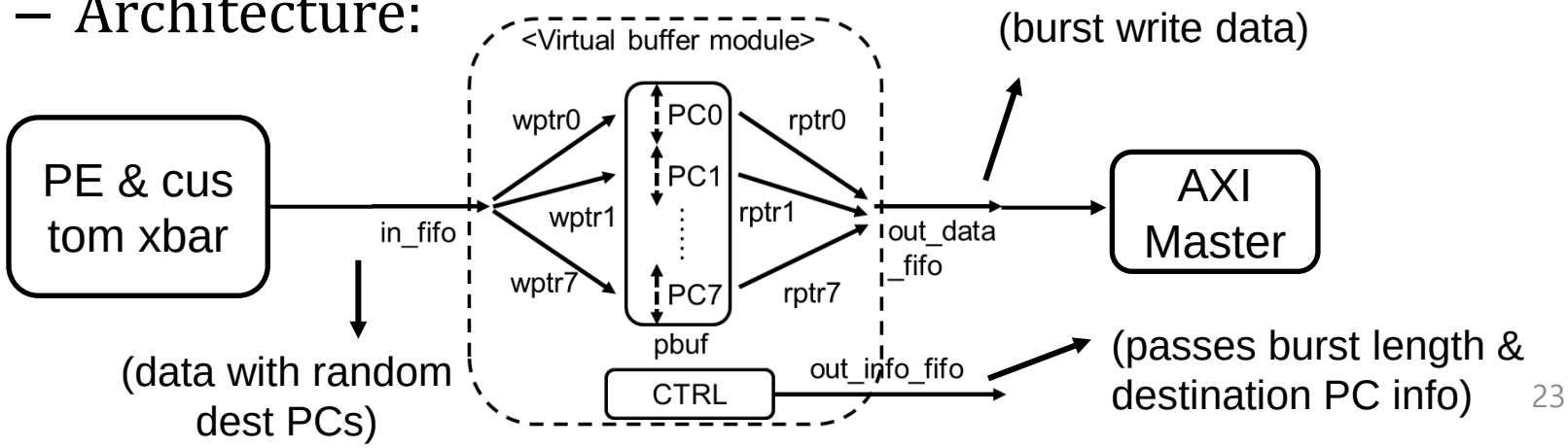
    - Problem 1: Underutilized BRAM
      - Only requires ~32 length burst ↔ BRAM min depth is 512
    - Problem 2: Complex routing
      - Scatters data to multiple FIFOs and again gathers data to a single AXI master
    - Result: PnR failed

Buf Sch	CX bar	Bur Len	FPGA Resource LUT/FF/DSP/BRAM	KClk (MHz)	EffBW (GB/s)
Direct access			126K / 238K / 0 / 248	178	56
FIFO	2	16	195K / 335K / 0 / 728	PnR failed	
Burst	2	32	193K / 335K / 0 / 728	PnR failed	
Buf	2	64	195K / 335K / 0 / 728	PnR failed	

** Called “BICA” in Y. Choi, et al., "When HLS Meets FPGA HBM: Benchmarking and Bandwidth Optimization." arXiv preprint arXiv:2010.06075 (2020).

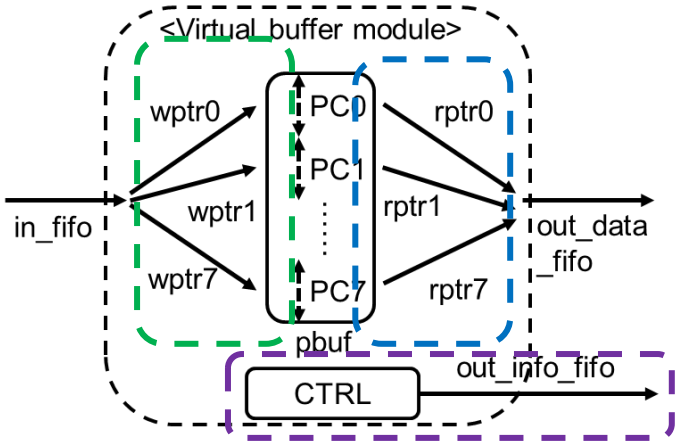
- 3 problems at hand:
  - AXI burst access problem
  - BRAM under-utilization problem
  - FIFO scatter/gather problem

- Proposed solution: HLS Virtual Buffer (HVB)
  - Idea: Share the BRAM as a burst buffer for many different destination PCs
    - Single physical FIFO is shared among multiple virtual channels
  - Architecture:



• HLS Virtual Buffer (....continued)

<Architecture>



<HLS code>

```
01 ap_uint<512> pbuf[8*ABUF];                  //single physical buffer
02
03 while(...){
04 #pragma HLS pipeline II=1
05 #pragma HLS dependency variable=pbuf inter false
06   if( write_in_progress = 1 ){
07     write_data = pbuf[64*write_ch + rptr[write_ch]];
08                                  //read data from physical buffer
09     out_data_fifo.write(write_data);    //write to out_data_fifo
10     rptr[write_ch]++;                 //increases the read pointer
11     write_num--;                      //keep track of data written
12     if( write_num == 0 ){
13       write_in_progress = 0;  //stop write when data is all sent
14     }
15   }
16   else{          //initiate a write transfer for a new target PC
17     write_ch++;
18     write_num = wptr[write_ch] - rptr[write_ch];
19           //number of write determined by the existing elements
20     out_info_fifo.write((write_ch, write_num)); //pass ctrl info
21     write_in_progress = 1;
22   }
23
24   read_success = in_fifo.read_nb(read_data);
25   if( read_success ){
26     targetPC = read_data(……); //find targetPC from the read_data
27     pbuf[64*targetPC + wptr[targetPC]] = read_data;
28                                     //write to physical buffer
29     wptr[targetPC]++;               //increases the write pointer
30 } }
```

Read data from buffer,
burst write to each PC

Burst length &
destination PC info

Reads input data (random PC),
write to physical buffer

- HLS Virtual Buffer (….continued)
  - Comparison

Buf Sch	Bur Len	CX bar	FPGA Resource LUT/FF/DSP/BRAM	KClk (MHz)	EffBW (GB/s)
Direct access		2	126K / 238K / 0 / 248	178	56
FIFO	16	2	195K / 335K / 0 / 728	PnR failed	
Burst	32	2	193K / 335K / 0 / 728	PnR failed	
Buf	64	2	195K / 335K / 0 / 728	PnR failed	
HLS	16	2	134K / 233K / 0 / 368	283	116
Virt	32	2	134K / 233K / 0 / 368	286	185
Buf	64	2	134K / 233K / 0 / 368	300	180

Eff BW higher than direct access with AXI burst access

Smaller LUT/FF usage by sharing physical FIFO

Smaller BRAM usage than FIFO burst buffer with sharing buffer space

  - HVB Abstraction (S2S transformation)

```
vir_ch0 = 0;
for(i=0; i<BURST_LEN; i++){
#pragma HLS pipeline II=1
  data = pfifo.vfifo_read(vir_ch0);
  pe0_pc0[i] = data;
}
```

Virtual buffer tag

- Introduction
- Case Studies
- Design Space and Problem Formulation
- Built-in Crossbar and Custom Crossbar
- AXI Burst Buffer
- Experimental Result

Experimental Result

- Case study 1: Bucket sort
  - Varying number of custom crossbar stages

	Cus Xbar	AXI Xbar	A BUF	FPGA Resource LUT/FF/DSP/BRAM	KClk (MHz)	EffBW (GB/s)	BW^2/Resource Metrics BW^2/LU	BW^2/FF	BW^2/BR
(baseline)	0	4	0	102K / 243K / 0 / 248	277	65	1.0	1.0	1.0
	0	4	64	122K / 243K / 0 / 480	166	108	2.3	2.7	1.4
	1	3	64	121K / 231K / 0 / 368	281	160	5.1	6.4	4.1
	2	2	64	134K / 233K / 0 / 368	300	180	5.8	8.0	5.2
	3	1	64	155K / 243K / 0 / 368	299	195	5.9	9.0	6.1
	4	0	0	189K / 305K / 0 / 248	207	203	5.3	7.8	9.8

Best

Best

CXBAR=4
-> 1-1 connection between AXI master and PC
-> Approaches the maximum BW achievable (206GB/s = 16PCs * 12.9 GB/s)

More custom crossbar stages
-> less contention on lateral connections
-> higher effective BW
-> more LUT/FF usage (but less than BW improvement)
-> fewer output PCs per AXI master -> less BRAM usage

– BW^2/resource metrics

ABUF → , CXBAR ↓

(BW^2/LUT)

	0	16	32	64	128
0	1.0	0.9	2.6	2.3	NA
1	1.0	2.8	**6.5**	5.1	3.5
2	0.6	2.4	**6.2**	5.8	4.7
3	0.8	2.3	3.8	5.9	5.3
4	5.3	-	-	-	-

Adding few stages adds only few LUTs while greatly improving BW

Low effective BW when burst length is too short

$(BW^2/BRAM)$

	0	16	32	64	128
0	1.0	0.7	2.0	1.4	NA
1	1.1	2.2	5.2	4.1	2.2
2	0.7	2.1	5.5	5.2	4.2
3	1.1	2.3	3.9	**6.1**	5.5
4	**9.8**	-	-	-	-

Long burst length is excessive since both reading & writing

AXI master only communicates with 1 PC -> Low BRAM usage

- Case study 2: Merge sort

(BW^2/LUT)

	0	32	64	128	256
0	1.0	64	52	NA	NA
1	1.8	82	120	100	114
2	1.7	88	149	119	168
3	1.5	86	141	154	**211**
4	12	85	137	181	**191**

Max achieved at 128-256 (compared to 32-64 in write), since longer burst needed for read

$(BW^2/BRAM)$

	0	32	64	128	256
0	1.0	57	34	NA	NA
1	1.6	62	66	36	25
2	1.6	66	81	42	35
3	1.6	70	**84**	60	48
4	15	70	**85**	73	46

Peak reached at shorter ABUF, since larger ABUF requires more BRAM

Note: Overall values higher in merge sort, since baseline read BW is 6X slower than baseline write BW

On-going Work

- In the process of extending to other benchmarks and making it user-friendly
- HBM Connect will be released as an automatic HLS C++ component generator based on template functions
  - Coming soon in https://github.com/UCLA-VAST/

https://image.freepik.com/free-vector/hand-drawn-construction-background_23-2147738793.jpg

Summary

- How to fully exploit FPGA HBM boards with HLS?
  - When multiple PEs access multiple HBM PCs?
- HLS Connect

Thank you!

(Please reach me at: ykchoi@cs.ucla.edu)

Supported by: