A Sorting Library for FPGA Implementation in OpenCL Programming

Ryohei Kobayashi*†, Kento Miura†, Norihisa Fujita*†, Taisuke Boku*†, and Toshiyuki Amagasa*†

*Center for Computational Sciences,
† Graduate School of Systems and Information Engineering,
University of Tsukuba, Japan

HEART 2021
International Symposium on Highly Efficient Accelerators and Reconfigurable Technologies
Monday, 21 June, 2021

OpenCL programming model for FPGAs

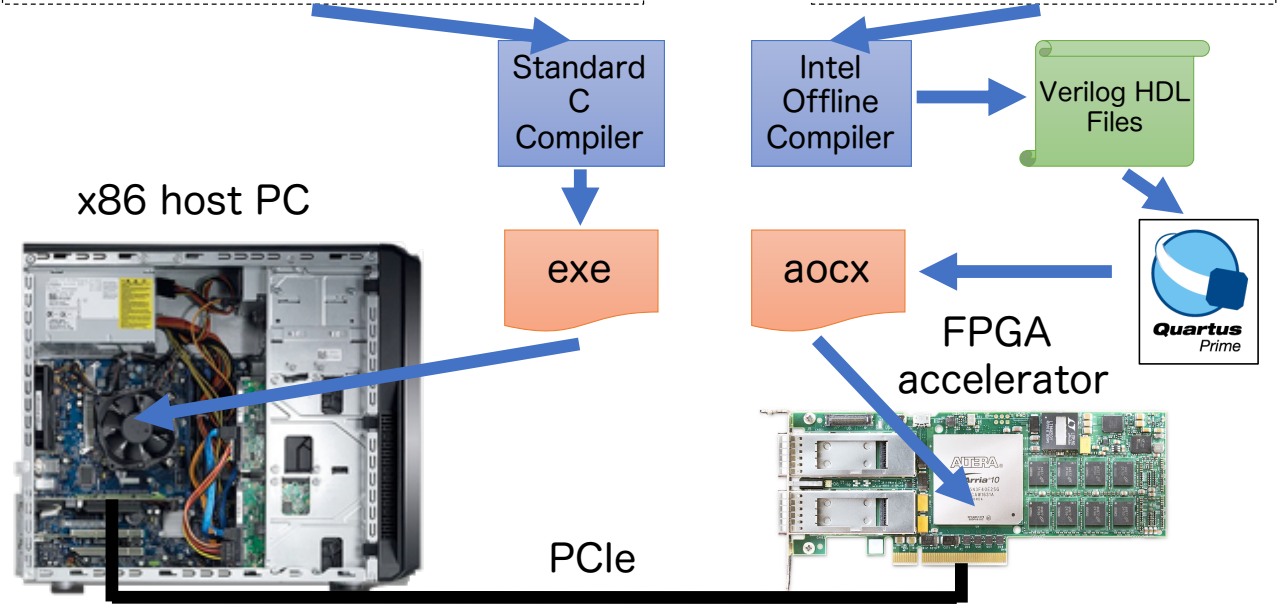
- enabling implementation of FPGA applications without the HDL
  - Pros
    - ✓ hiding from the user the fundamental parts of FPGA implementation (PCIe, DDR, etc.)
      - All-in-one dev. environment
  - Cons
    - ✓ limited in their expressiveness
      - Programming is easy, but optimization is not
    - ✓ suffering from place and route problems that limit the maximum frequency

OpenCL host code

```
int main(int argc, char *argv[]) {
  init();
  clEnqueueWriteBuffer(...);
  clEnqueueNDRangeKernel(...,vecadd,...);
  clEnqueueReadBuffer(...);
  display_result(...);
  return 0;
}
```

OpenCL kernel code

```
__kernel void vecadd
        (__global float *a,
         __global float *b,
         __global float *c)
{
  int gid = get_global_id(0);
  c[gid] = a[gid] + b[gid];
}
```



Intel FPGA SDK for OpenCL's programming model

What we are aiming is...

- to optimize general-purpose computation kernels for practical application development for FPGAs

  and

- to make them available to users

- Related work: OpenCL implementation of matrix multiplication [2]
  - assuming that it could serve as a building block for algorithms that are built upon the base functionality of matrix multiplications.

TABLE II

DESIGN CONFIGURATIONS AND SYNTHESIS RESULTS (RESOURCE USAGE EXCLUDING BSP), EFFECTIVE PERFORMANCE BASED ON MEASUREMENTS, COMPARISON TO PERFORMANCE MODEL, PEAK PERFORMANCE OF COMPUTE PIPELINE BASED ON CLOCK FREQUENCY AND CYCLE EFFICIENCY OF COMPUTE PIPELINE BASED ON TILE SIZE AND GLOBAL MEMORY BANDWIDTH

#CUs	block size d_1	#DSPs (% of avail.)	#MK20s (% of avail.)	fmax [MHz]	effective GFLOPS	model GFLOPS	area efficiency	frequency efficiency	cycle efficiency
1	1024	645 (14%)	6242 (70%)	363	344	349	0.115	0.955	0.925
2	720	1289 (29%)	6340 (71%)	354	648	665	0.229	0.931	0.894
3	512	1934 (43%)	4902 (55%)	319	863	869	0.344	0.839	0.880
4	512	2578 (58%)	6536 (73%)	315	1104	1144	0.458	0.829	0.856
5	512	3223 (72%)	8170 (91%)	294	1324	1335	0.573	0.774	0.880
6	360	3867 (86%)	5196 (58%)	224	1120	1167	0.688	0.589	0.814

[2]: P. Gorlani et al., “OpenCL Implementation of Cannon's Matrix Multiplication Algorithm on Intel Stratix 10 FPGAs”, ICFPT19, pp.99-107

Target kernel: sorting

- Sorting is everywhere
  - basic arithmetic operation
    - ✓ SNS
    - ✓ Gene analysis
    - ✓ Database
    - ✓ Etc...

SNS

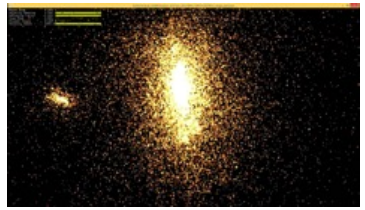
Gene analysis

Distributed computing frameworks

database

Reference network

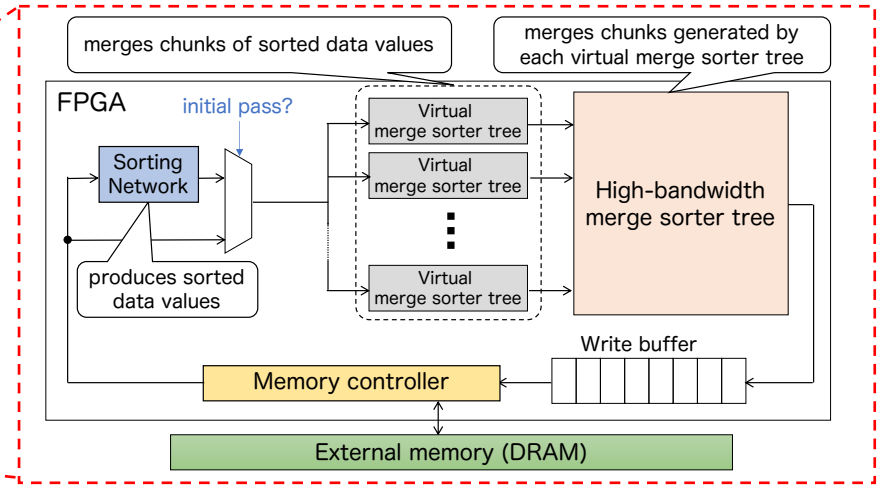
N body problem



- Our proposal
  - Providing a sorting library for FPGA implementation in OpenCL programming

```
1  #include "fpga_sort.h"
2
3  __kernel void fpga_sort_test(
4                               __global uint *restrict tmp,
5                               __global uint *restrict src,
6                               const uint numdata,
7                               __global uint *restrict ret
8                              )
9  {
10   // Do sort
11   *ret = fpga_sort(tmp, src, numdata);
12 }
```

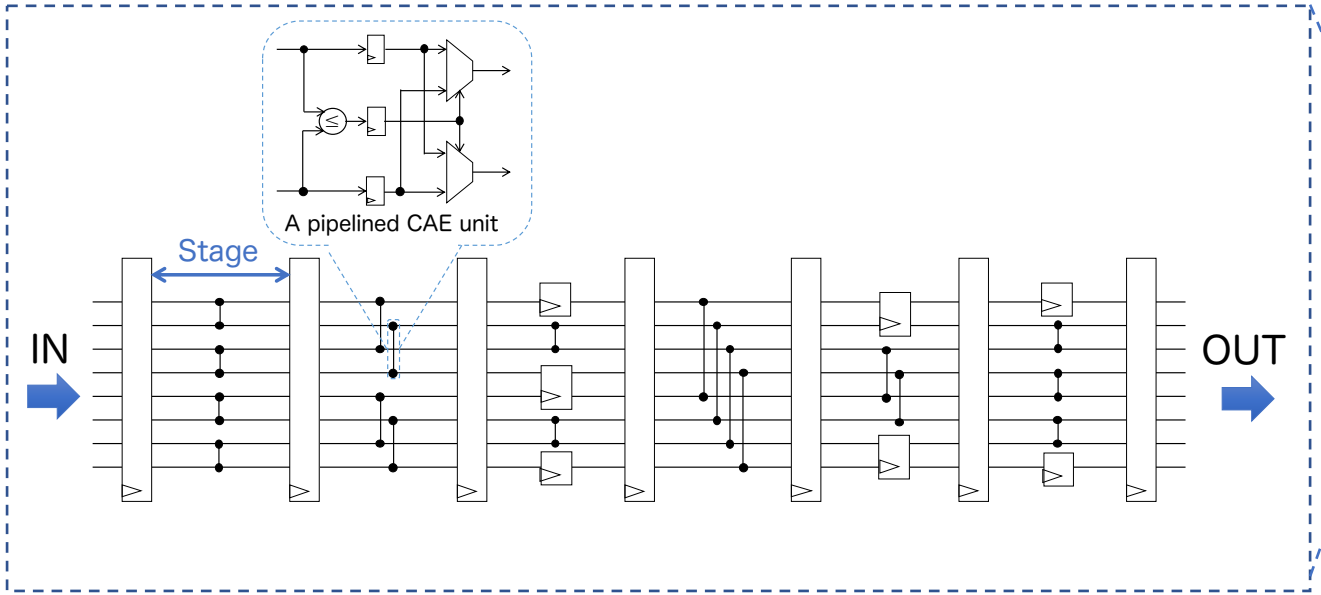
OpenCL kernel code with sorting function



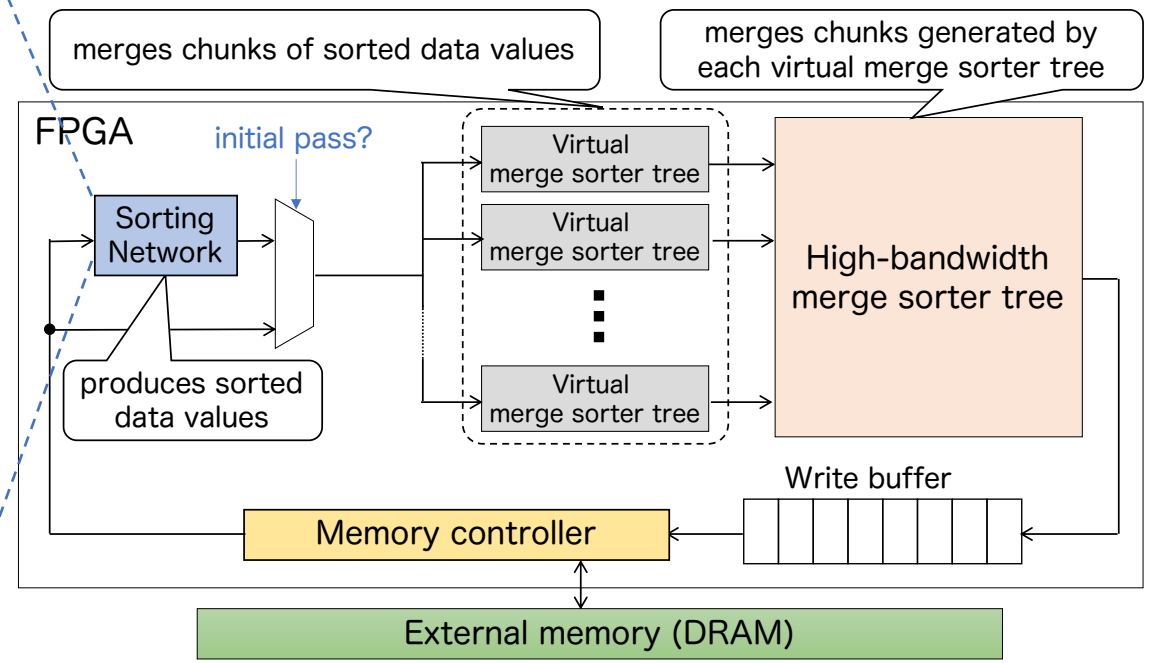
sorting engine implemented in HDL

Batcher's Odd-Even Sorting Network

- Output sorted values in pipeline manner
  - two-stage pipelined CAE (Compare-and-exchange) units are used
    - ✓ critical path: one comparator
      → increasing the operating frequency

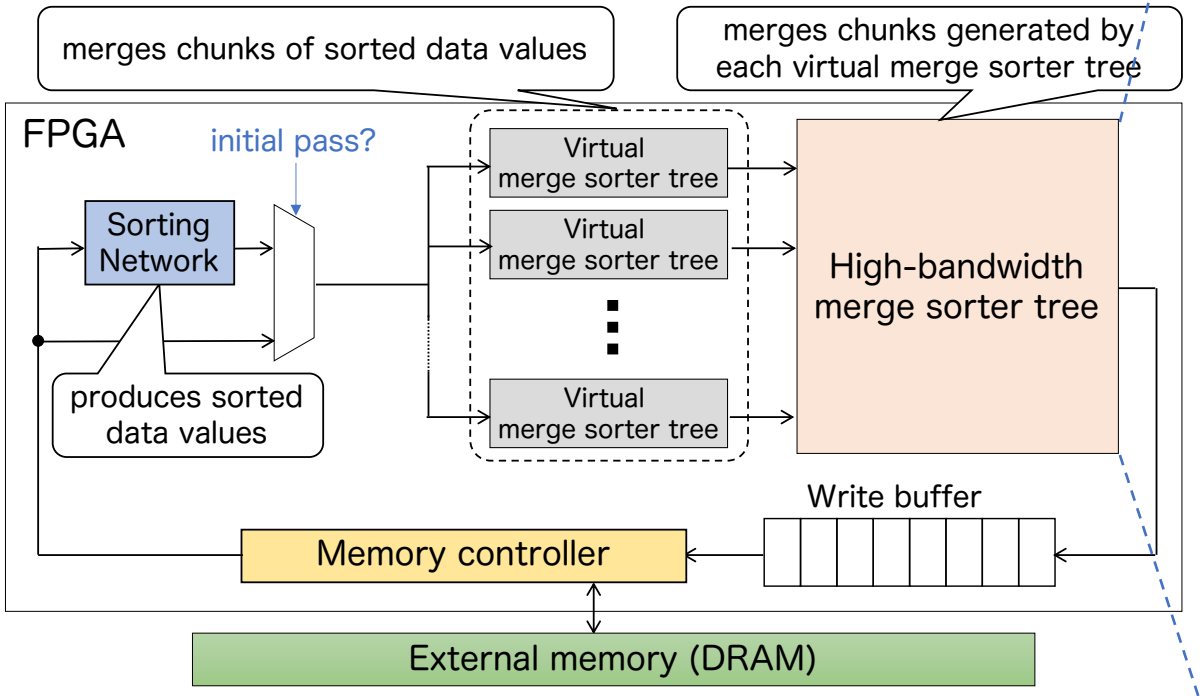


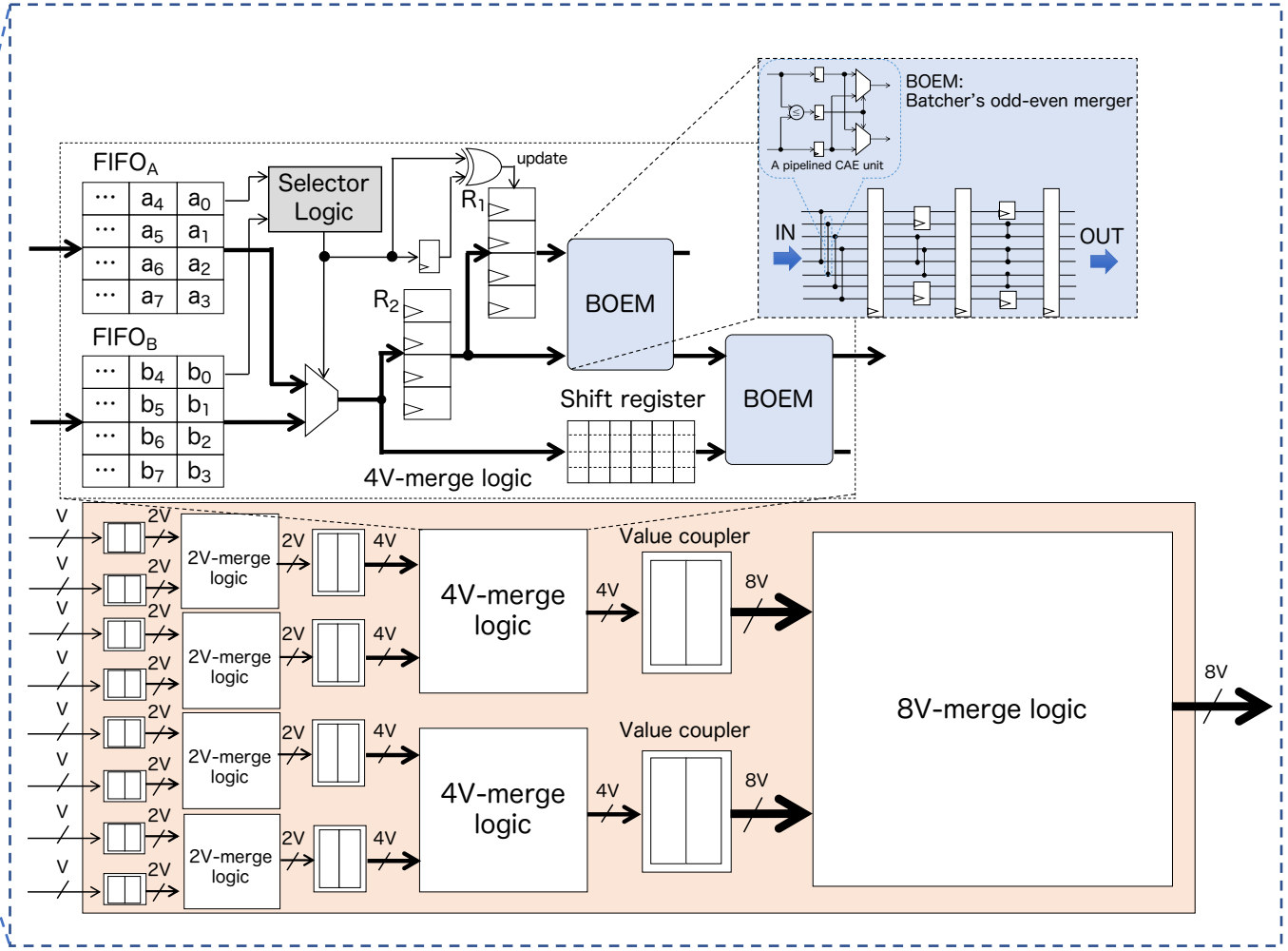
A pipelined Batcher's odd-even sorting network with 8 inputs and 8 outputs
(the width of the sorting network is parameterized)



High-bandwidth Merge Sorter Tree [7]

- Merging sorted values with a throughput of a maximum of N data values
  - In this figure, N is 8



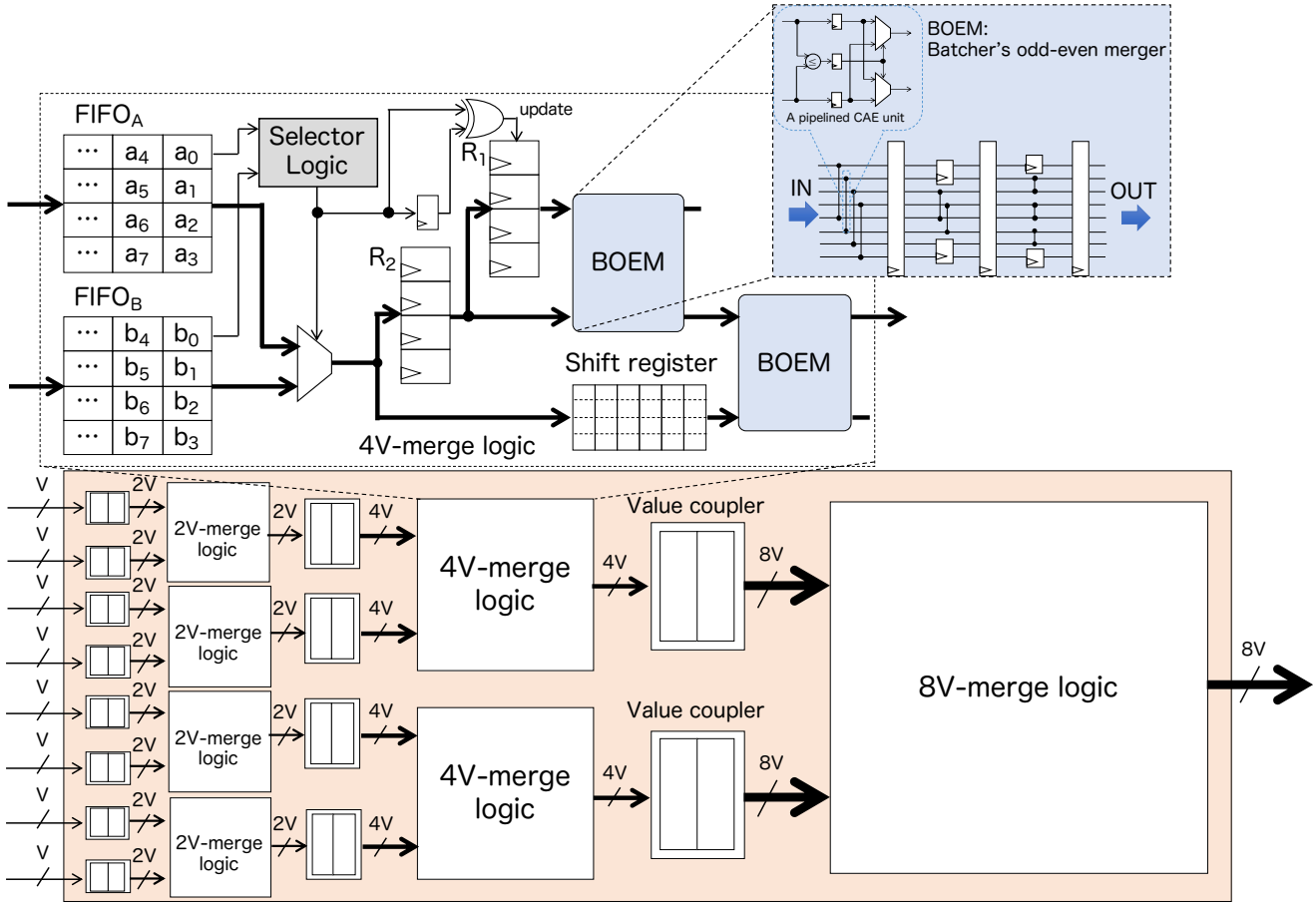


A high-bandwidth merge sorter tree with 8 input ports (the number of input ports is parameterized)

[7]: M. Saitoh et al., "Very Massive Hardware Merge Sorter", ICFPT18, pp.86-93

Issue of High-bandwidth Merge Sorter Tree

- As the number of input ports of the tree is increased, the hardware resource usage increases linearly
  - the size of the tree that can be implemented in an FPGA is limited to a certain extent
  - the number of sorted data values in the output data stream generated by passing through the tree does not scale



Solution

connecting a merge sorter tree with wide input width to each input port of the high-bandwidth merge sorter tree

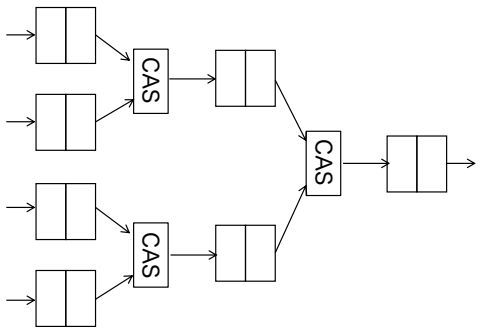
→ a merged sort tree that has both high throughput and wide input can be built

Building a merge sorter tree with wide input width

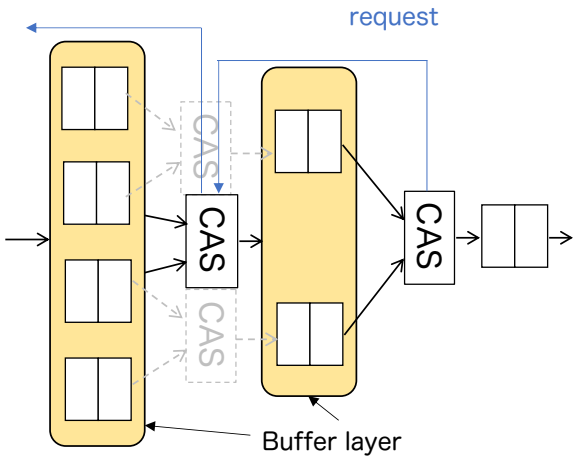
- Solution: Virtual Merge Sorter Tree [9]
  - placing only one CAS (Compare -and-select) unit in each stage
  - Integrating the FIFOs into a single buffer layer implemented in BlockRAM

Hardware-efficient implementation

[9]: K. Manev et al., “Large Utility Sorting on FPGAs”, ICFPT18, pp.334–337



A traditional merge sorter tree



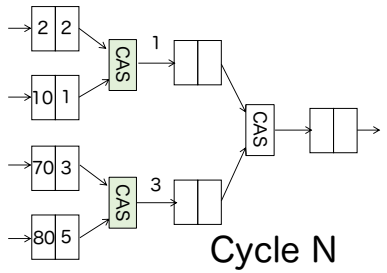
A virtual merge sorter tree

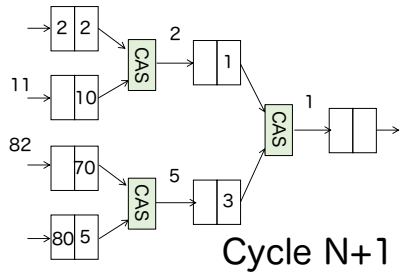
- How it can be done?

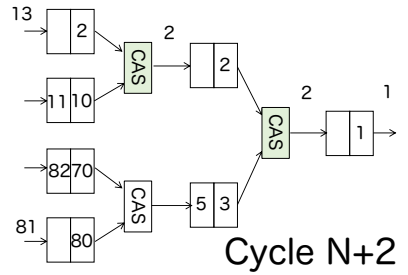
In the traditional merge sorter tree, the CAS unit located at the root selects and outputs only one value on either side at a time.

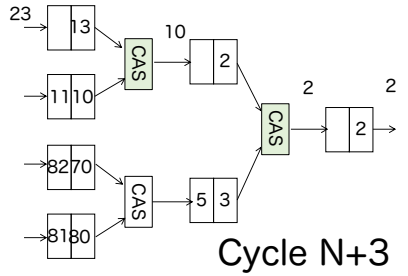
↓

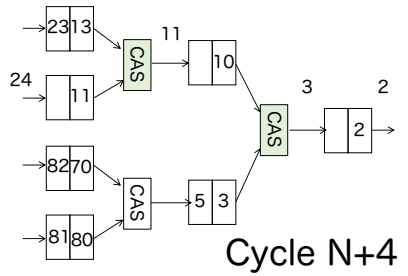
Therefore, in each stage of the tree, only one CAS unit is active as well as one FIFO’s dequeue request and one FIFO’s enqueue request, which is highlighted in the figure

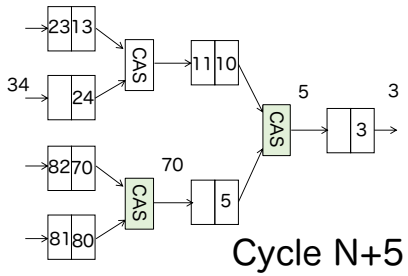












Virtual Merge Sorter Tree [9]'s issue

- Operating frequency is too low

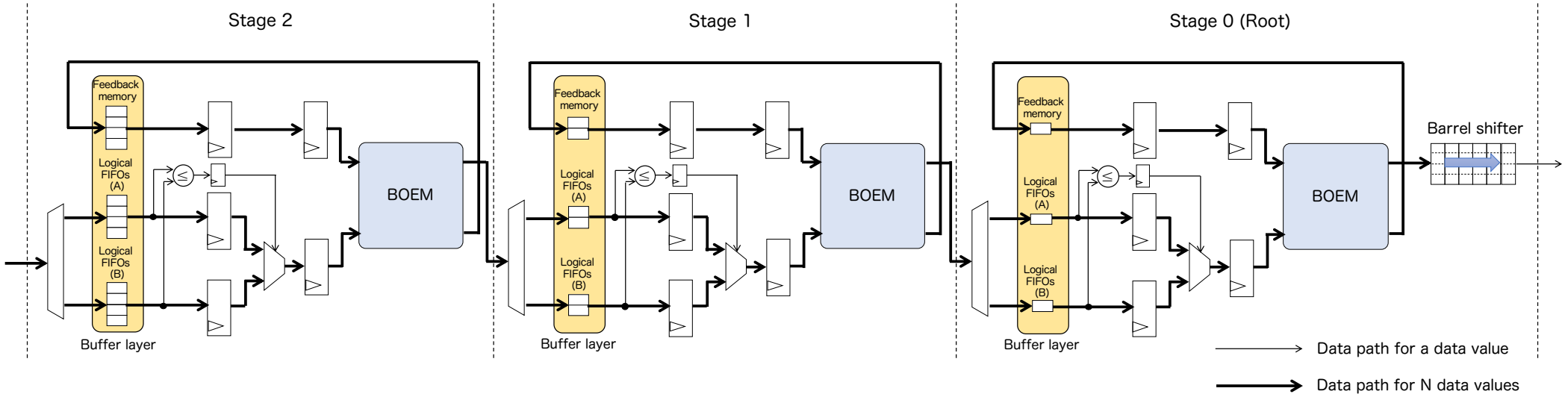
[9]: K. Manev et al., "Large Utility Sorting on FPGAs", ICFPT18, pp.334–337

  - less than half that of the high-bandwidth merge sorter

Not good...

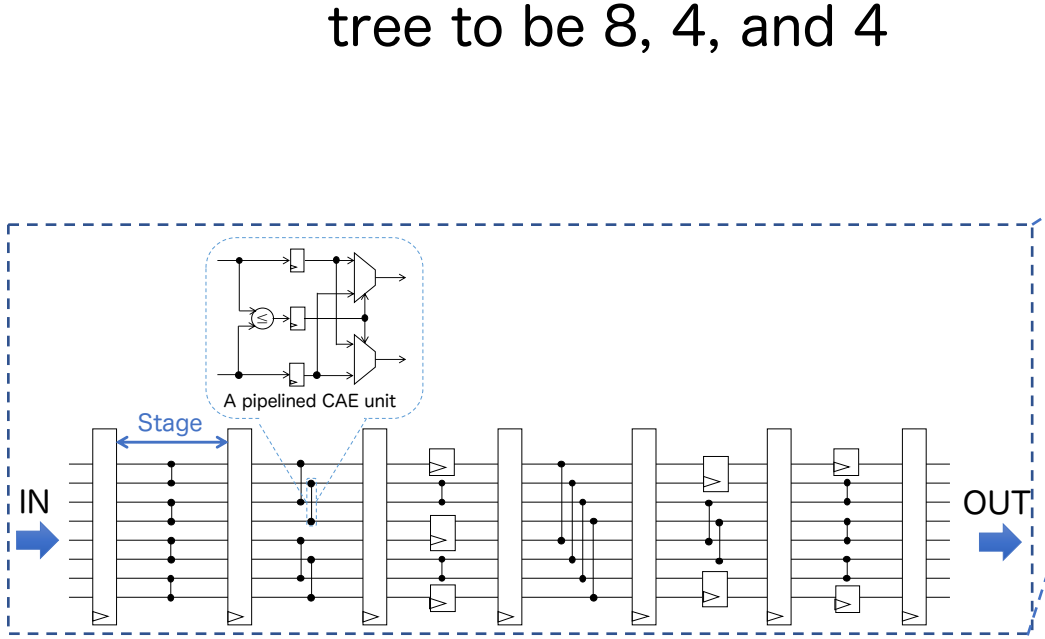
the performance of the high-bandwidth merge sorter tree cannot be maximized by simply connecting the virtual merge sorter tree

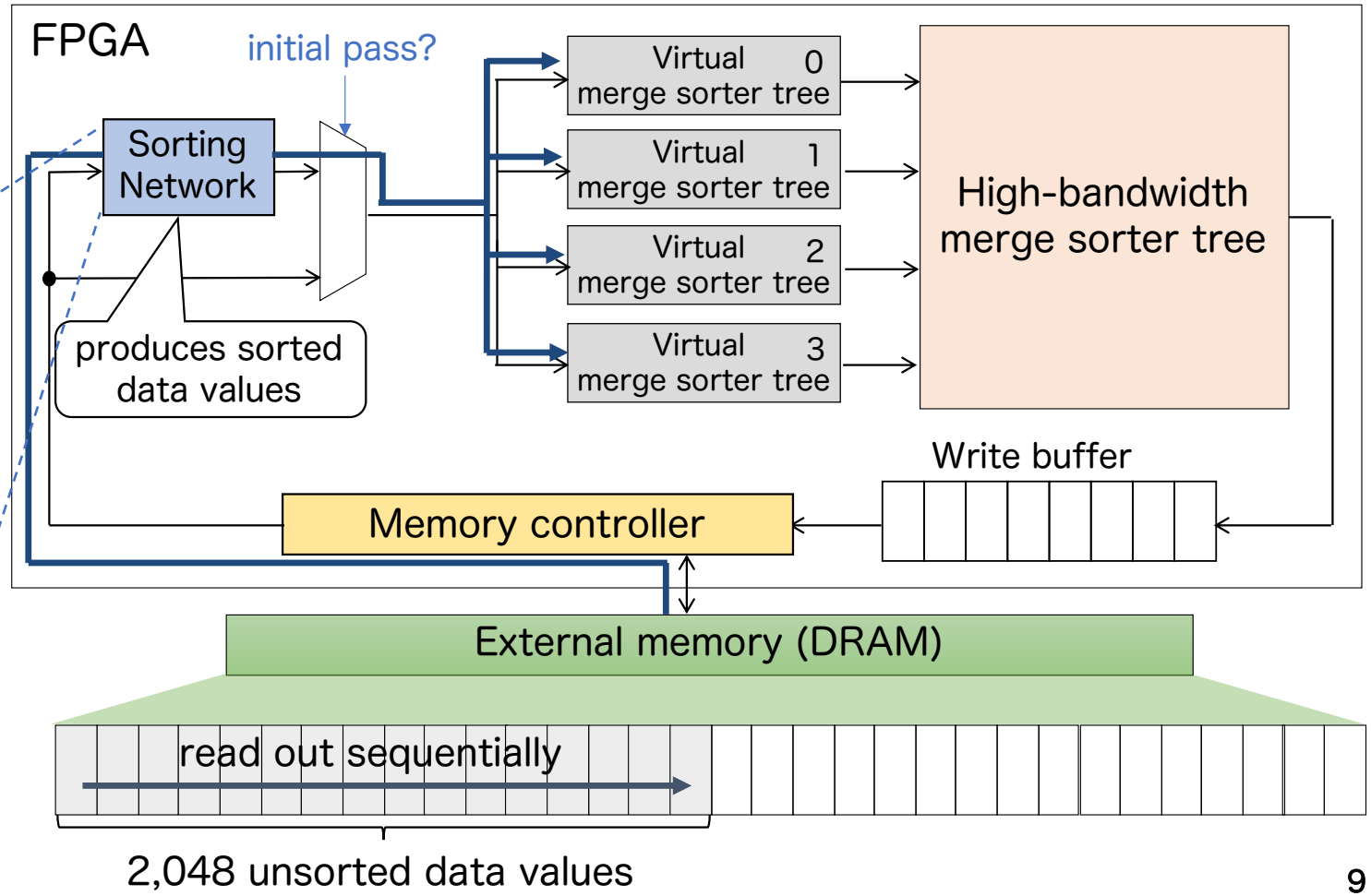
- Our solution: A multi-cycle virtual merge sorter tree
  - outputs N data values per N cycles (throughput is same as [9])
  - critical path: one comparator
    → increasing the operating frequency compared to [9]



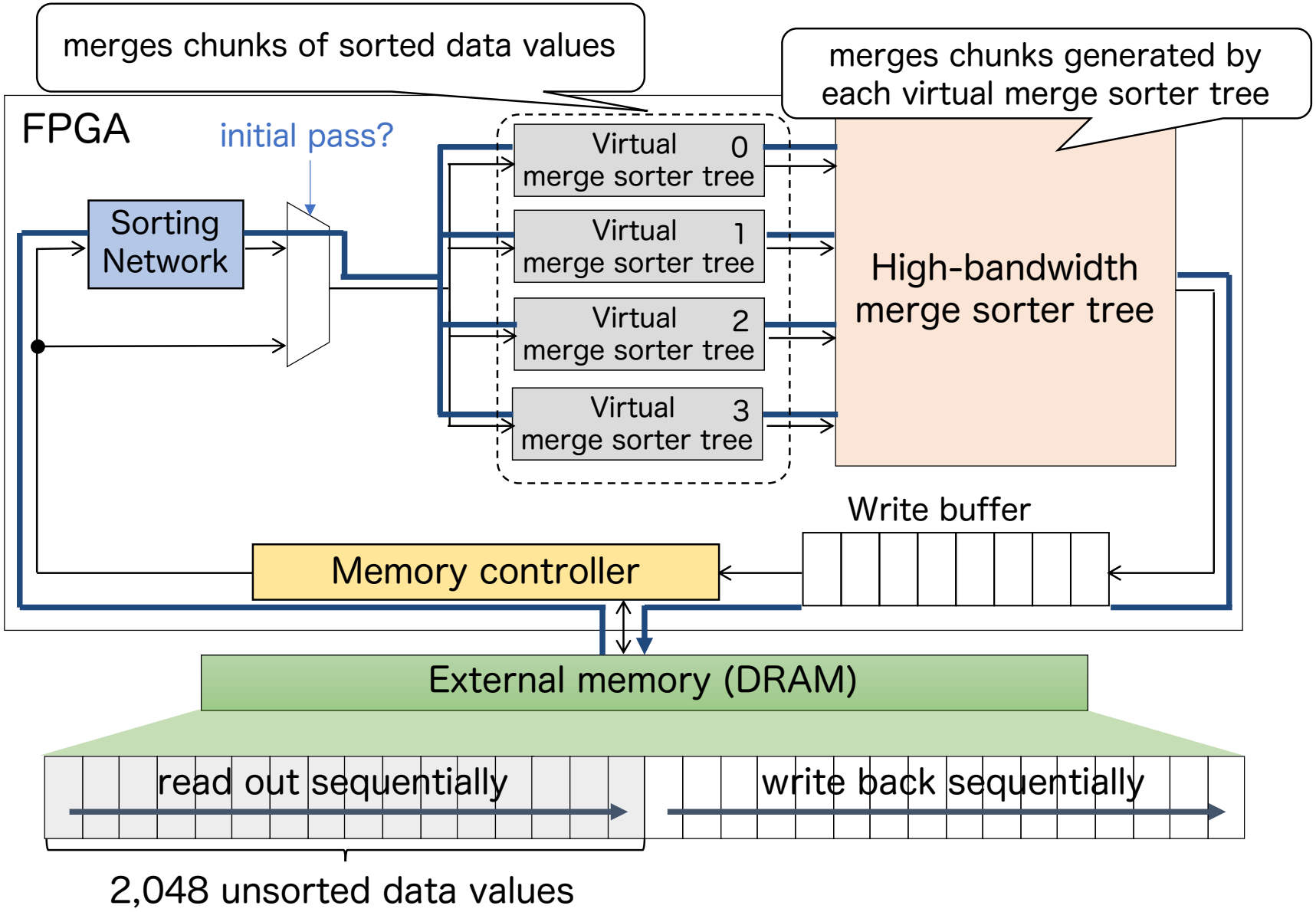
Sorting Engine's Behavior

- Example: sorting of 2,048 data values
  - the width of the sorting network, number of leaves in the virtual merge sorter tree, and number of input ports in the high-bandwidth merge sorter tree to be 8, 4, and 4

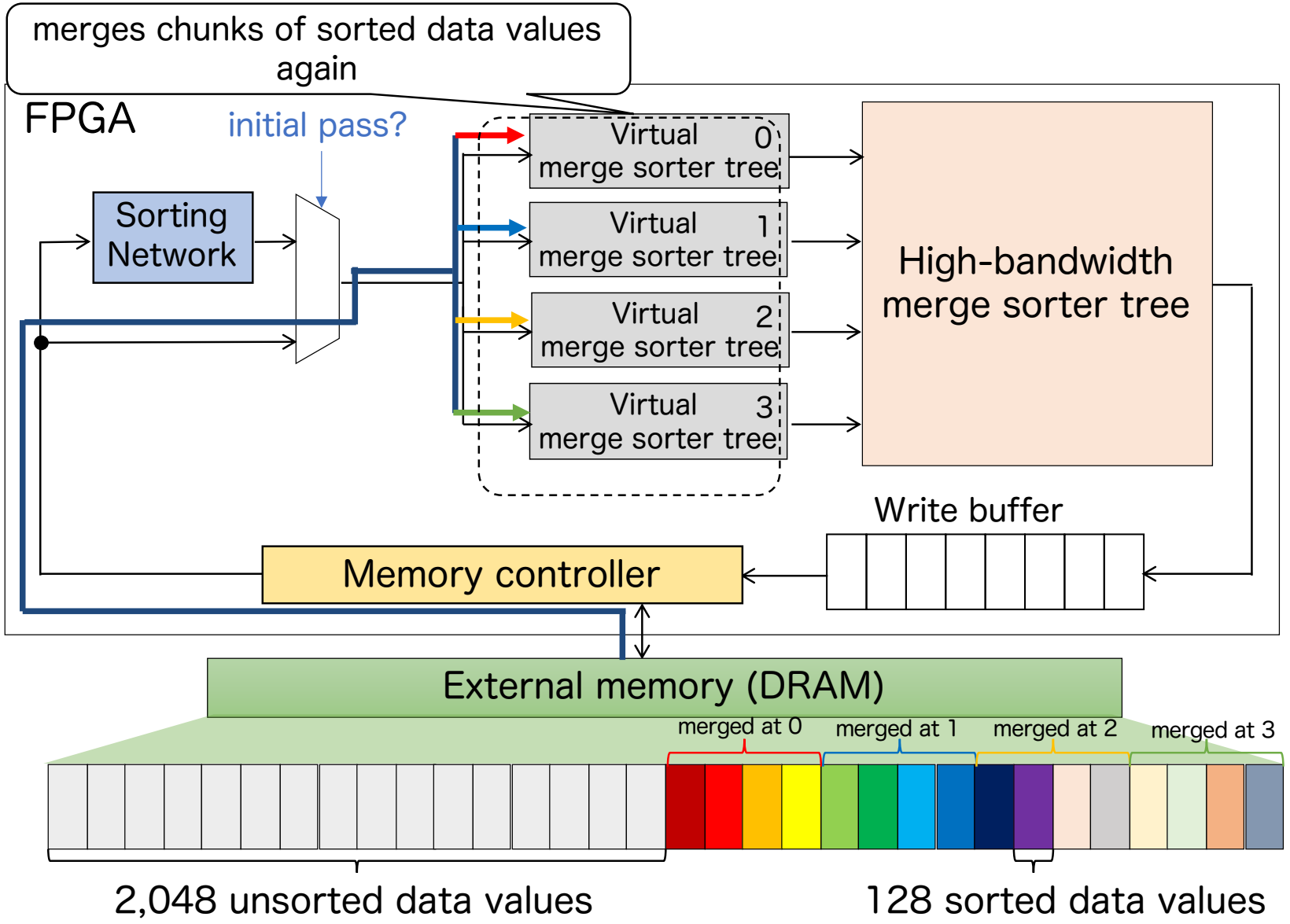




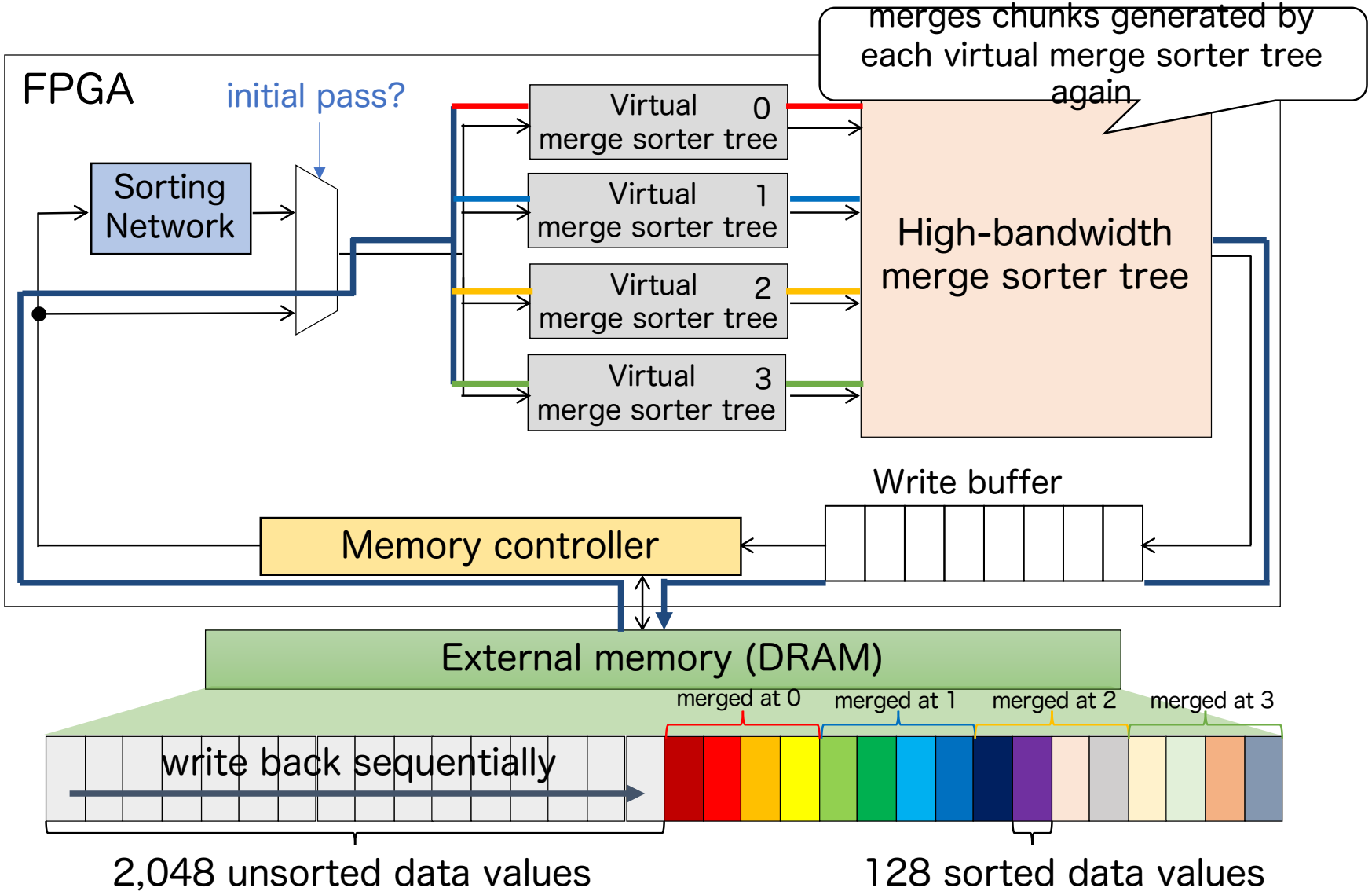
Sorting Engine's Behavior



Sorting Engine's Behavior

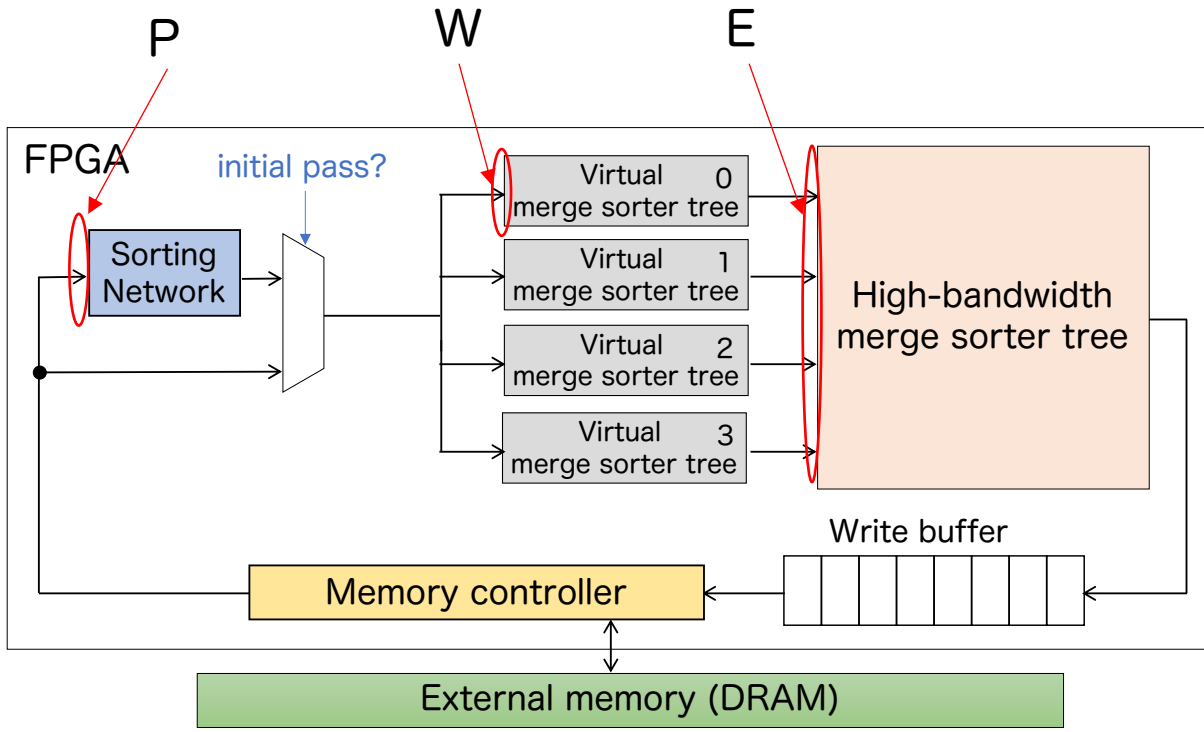


Sorting Engine's Behavior



Theoretical performance

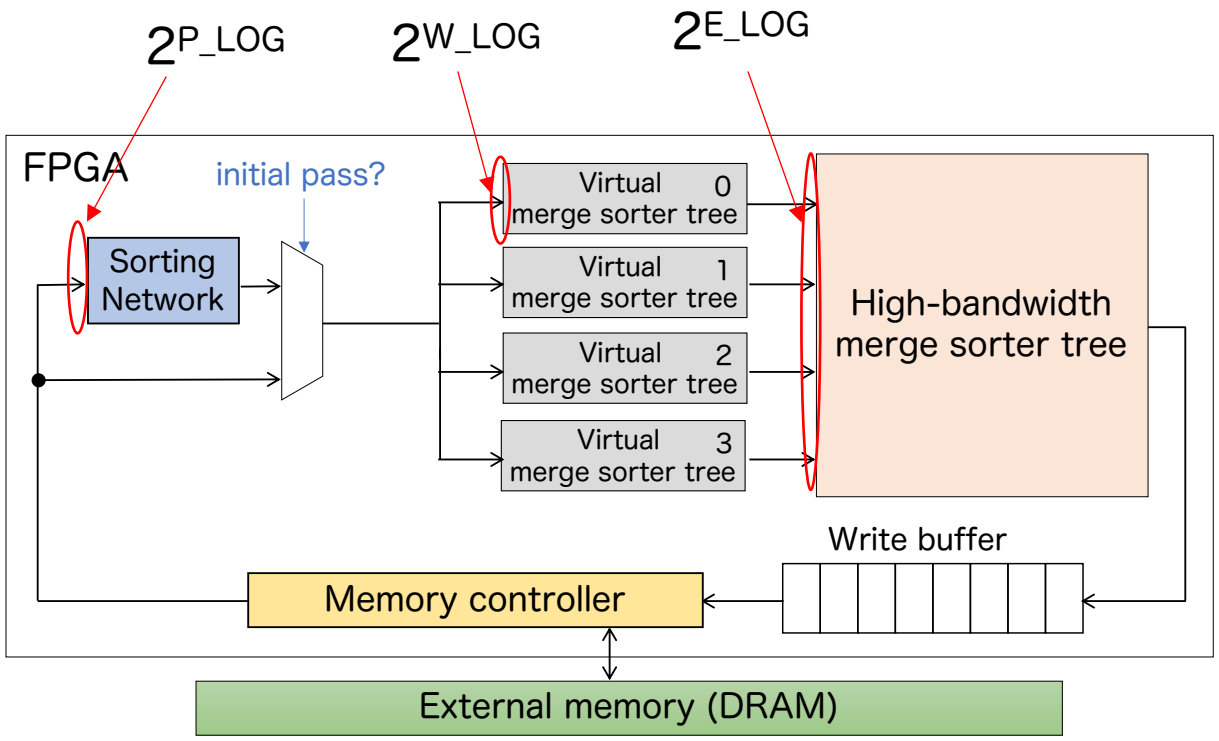
- # of pass: $\lceil \log_{KxE}(N/P) \rceil$
  - W: the number of leaves in the virtual merge sorter tree
  - E: the number of input ports in the high-bandwidth merge sorter
  - N: the number of data values to be sorted
  - P: the width of the sorting network
- Example: sorting 2,048 data values
  - W = 4, E = 4, N = 2048, P = 8
    - ✓ # of pass to be 2
- Throughput: E x B x F / (# of pass) [Byte / s]
  - B: data size of a data value
  - F: Operating Frequency

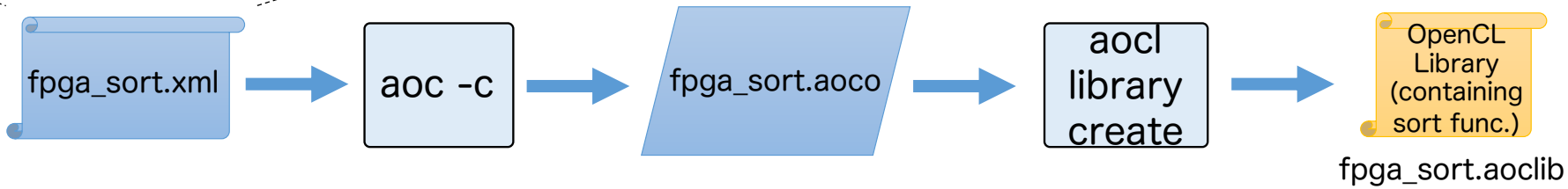


Creating an OpenCL library for data sorting

```
<RTL_SPEC>
 <FUNCTION name="fpga_sort" module="fpga_sort">
  <ATTRIBUTES>
   <IS_STALL_FREE value="no"/>
   <IS_FIXED_LATENCY value="no"/>
   <EXPECTED_LATENCY value="10"/>
   <CAPACITY value="1" />
   <HAS_SIDE_EFFECTS value="yes"/>
   <ALLOW_MERGING value="no"/>
   <!-- memory access parameters -->
   <PARAMETER name="MAXBURST_LOG" value="4"/>
   <PARAMETER name="WRITENUM_SIZE" value="5"/>
   <PARAMETER name="DRAM_ADDRSPACE" value="64"/>
   <PARAMETER name="DRAM_DATAWIDTH" value="512"/>
   <!-- parameters for hybrid sorter's configuration -->
   <PARAMETER name="W_LOG" value="2"/>
   <PARAMETER name="P_LOG" value="3"/>
   <PARAMETER name="E_LOG" value="2"/>
   ⋮
```

specifying sorting engine's configuration



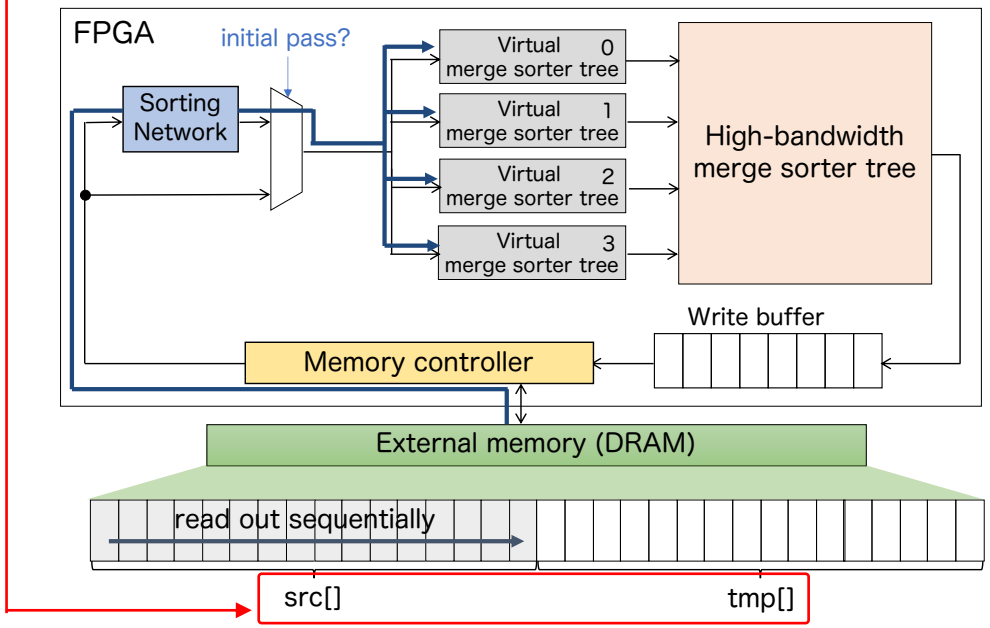


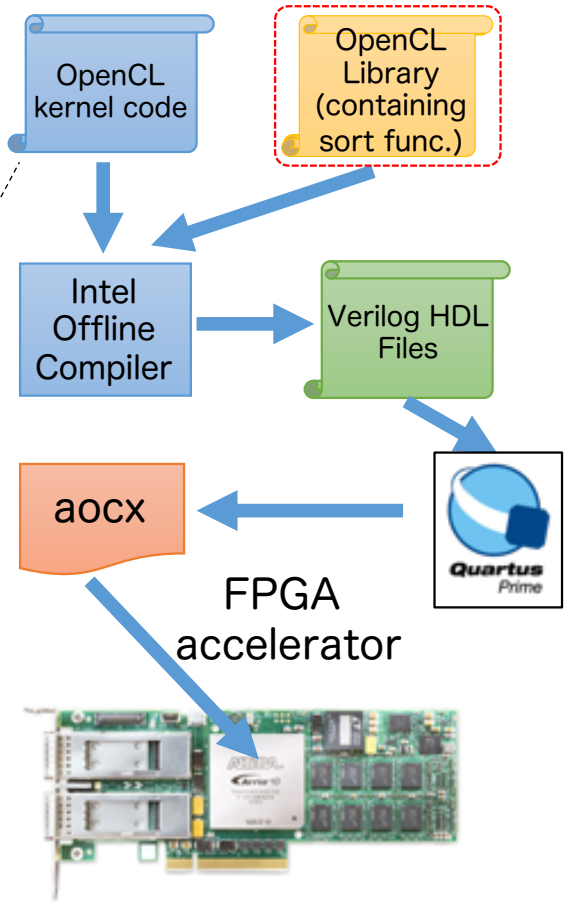
Overview of OpenCL library creation that contains the sorting function

Intel FPGA SDK for OpenCL programming model with our sorting library

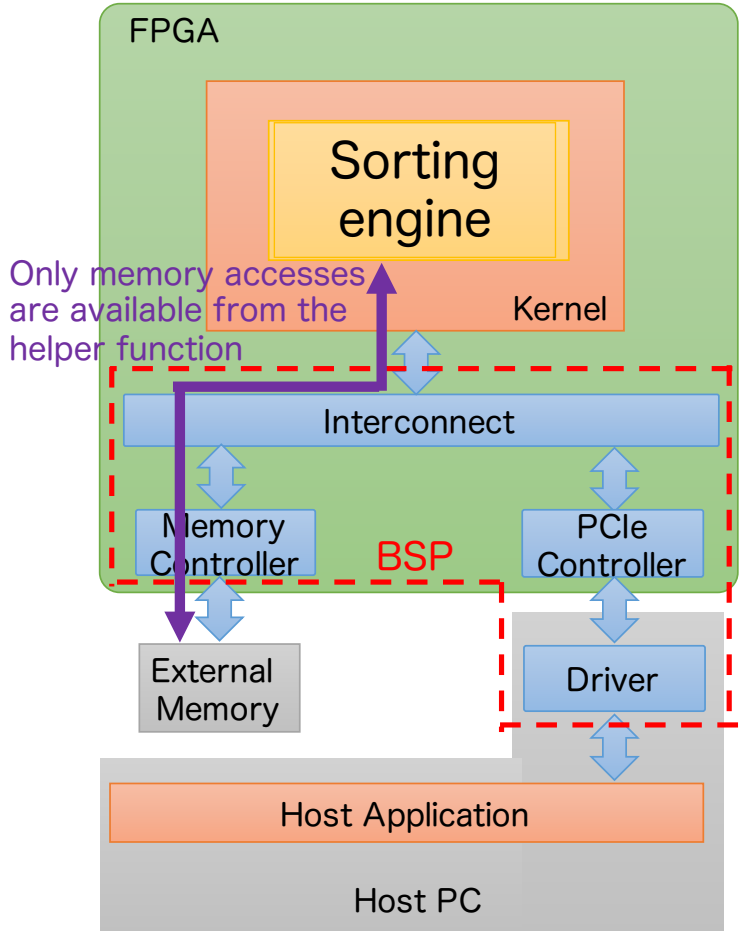
```
1  #include "fpga_sort.h"
2
3  __kernel void fpga_sort_test(
4                               __global uint *restrict tmp,
5                               __global uint *restrict src,
6                               const uint numdata,
7                               __global uint *restrict ret
8                              )
9  {
10   // Do sort
11   *ret = fpga_sort(tmp, src, numdata);
12 }
```

OpenCL kernel code with sorting function





Compilation flow



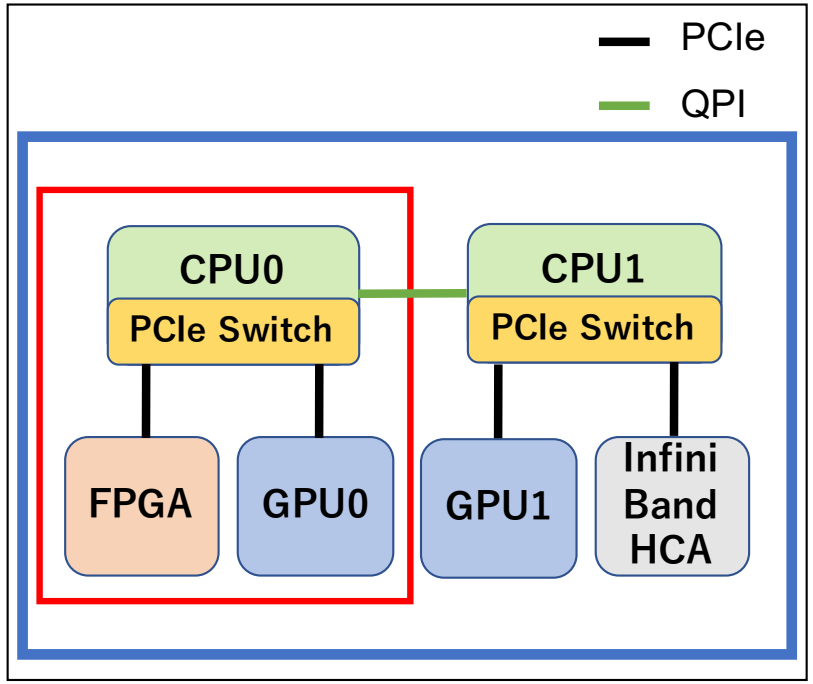
Schematic of the Intel FPGA SDK for OpenCL platform

Evaluation testbed

●Pre-PACS version X (PPX)

➢ working at Center for Computational Sciences, University of Tsukuba.

Hardware specification	
CPU	Intel Xeon E5-2690 v4 x2
Host Memory	DDR4-2400 8GB x8
GPU	NVIDIA Tesla V100 (PCIe Gen3 x16)
GPU Memory	32 GiB CoWoS HBM2 @ 900 GB/s with ECC
FPGA	BittWare 520N (Intel Stratix 10 1SG280HN2F43E2VG)
FPGA Memory	DDR4 2400MHz 32 GB (8GB × 4)
Software specification	
OS	CentOS 7.3
Host Compiler	Intel C++ Compiler 18.0.1
GPU Compiler	CUDA 9.2.148
OpenCL SDK	Intel FPGA SDK for OpenCL 19.4.0 Build 64 Pro Edition



A computation node of PPX

Performance comparison to OpenCL kernel for sorting

- For comparison, we restructured merge sort algorithm [12] for 2^29 data elements (data: 64 bit [32-bit key, 32-bit payload])
- Sorting engine's configuration: W = 4, P = 8, E = 4

Comparison result. data: 64 bit (32-bit key, 32-bit payload)

three orders of magnitude greater

	ALMs	(%)	Registers	(%)	M20Ks	(%)	DSPs	(%)	fmax [MHz]	Throughput [MB/s]
Our sorting library	86,856	9%	244,697	7%	486	4%	2	0.03%	309.98	116
The merge sort [12]	35,455	4%	73,736	2%	231	2%	0	0%	283.60	0.26

- Operating frequency: 309.98 MHz
  - Critical path: a control logic for global memory interleave
    - ✓ can be removed by specifying -no-interleaving=DDR -global-ring at compilation

      → 378.21 MHz (380 MHz is the upper limit of the achievable operating freq.)

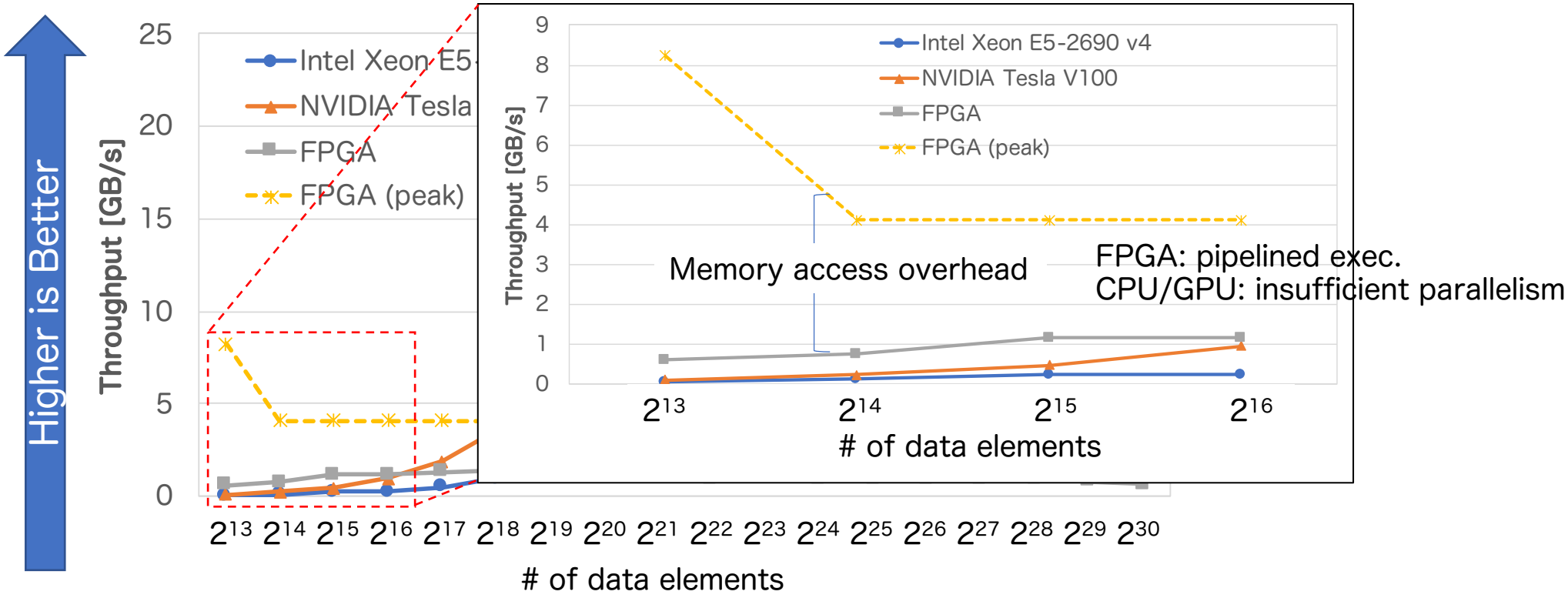
the sorting performance of the library is further improved

[12]: R. Kastner, J. Matai, and S. Neuendorffer. Parallel Programming for FPGAs. ArXiv e-prints, May 2018.

Performance comparison between CPU and GPU

- CPU: OpenMP-versioned radix sort based on [14]
- GPU: Thrust library (CUDA 9.2.148)
- Sorting engine's configuration (FPGA) : W = 64, P = 32, E = 4

[14]: Nadathur Satish et al., "Fast Sort on CPUs and GPUs: A case for bandwidth oblivious SIMD Sort" SIGMOD10, pp.351–362

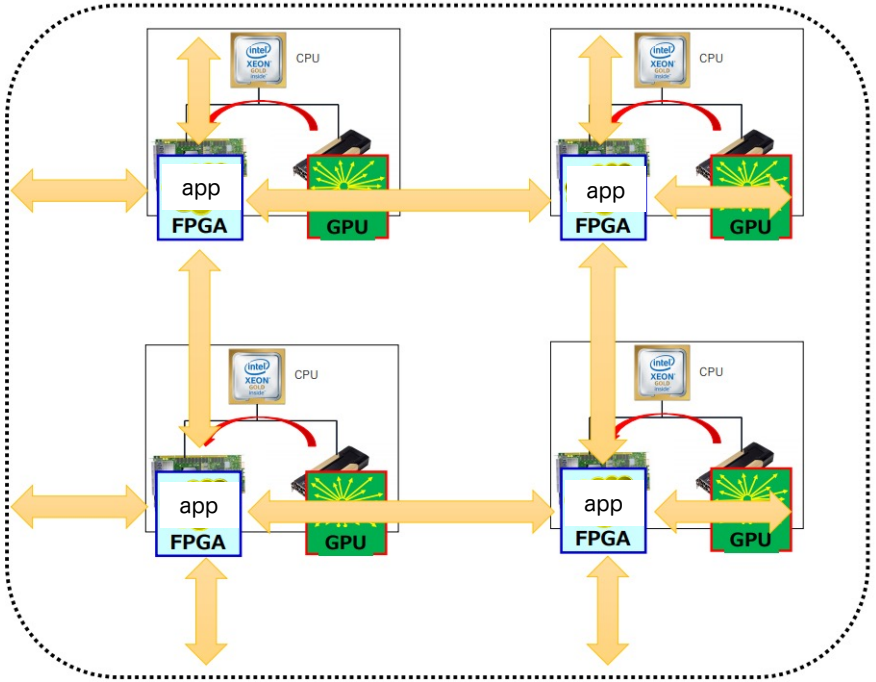


Comparison of sorting performances based on data size
Data: 64 bit (32-bit key, 32-bit payload)

The ability to process small problem sizes

- necessary to achieve strong scaling for FPGA-centric applications
  - # of FPGAs is increased → problem size per FPGA becomes smaller
- sorting is required in nearly all algorithms, particularly in data intensive applications
  - statistics (percentile)
  - search (k-nearest neighbor query)
  - index construction (inverted index)
  - graph processing
  - etc...

CYGNUS



Our study is worthwhile for developing a hardware logic to accelerate that fundamental process and enabling its ease of use as a library.

Conclusion

- A sorting library that can be used with the OpenCL programming model for FPGA
  - which is built by combining the three hardware sorting algorithms.

- Our sorting library consumes more than twice the overall hardware resources compared to a merge sort restructured for the OpenCL programming model for FPGA, but its operating frequency is 1.09x higher and its sorting throughput is three orders of magnitude better

- We also derived the performance model to estimate for data sorting and application developers can determine the optimal configuration of the sorting engine, taking into account the amount of target FPGA resources and the required sorting performance