Mini-workshop of Japan Korea HPC
Winter School 2018 @ CCS, U. Tsukuba
14:30 - 15:00, February 22, 2018

筑波大学
University of Tsukuba

OpenCL-ready FPGA Programming for High Performance Computing

Ryohei Kobayashi

Assistant Professor, CCS

Background

Accelerators in HPC

- The most popular one: GPU
- GPU-based HPC clusters
  - HA-PACS/TCA, TSUBAME3, ABCI, etc.
- GPUs are good at
  - parallel applications depending on very wide and regular computation
    - ✓ large scale SIMD (STMD) fabric in a chip
    - ✓ high bandwidth memory (GDR5, HBM) and local memory
- GPUs do not work well on applications that employ
  - partially poor parallelism
  - non-regular computation (warp divergence)
  - frequent inter-node communication (kernel switch, go back to CPU)

FPGAs have been emerging in HPC

- Strengths of today's FPGA for HPC
  - true co-designing with applications (**indispensable**)
  - programmability improvement
    - ✓ OpenCL-based FPGA development toolchains are available
  - high bandwidth interconnect: 40 ~ 100 Gbps/link
  - implementing computation engines with specialized precision is possible
  - relatively low power (it depends on implemented logics)
- Problems
  - OpenCL is not good enough! Programming effort is still high!
    - ✓ Programmers must consider carefully how application's algorithms are implemented on an FPGA in order to exploit desired performance
  - FPGAs still cannot beat GPU in terms of absolute performance (FLOPS)
    - ✓ Don't try what GPU can perform well
  - Memory bandwidth is two generation older than high-end CPU/GPU
    - ✓ Good news: this is improved by HBM (Stratix10 MX)

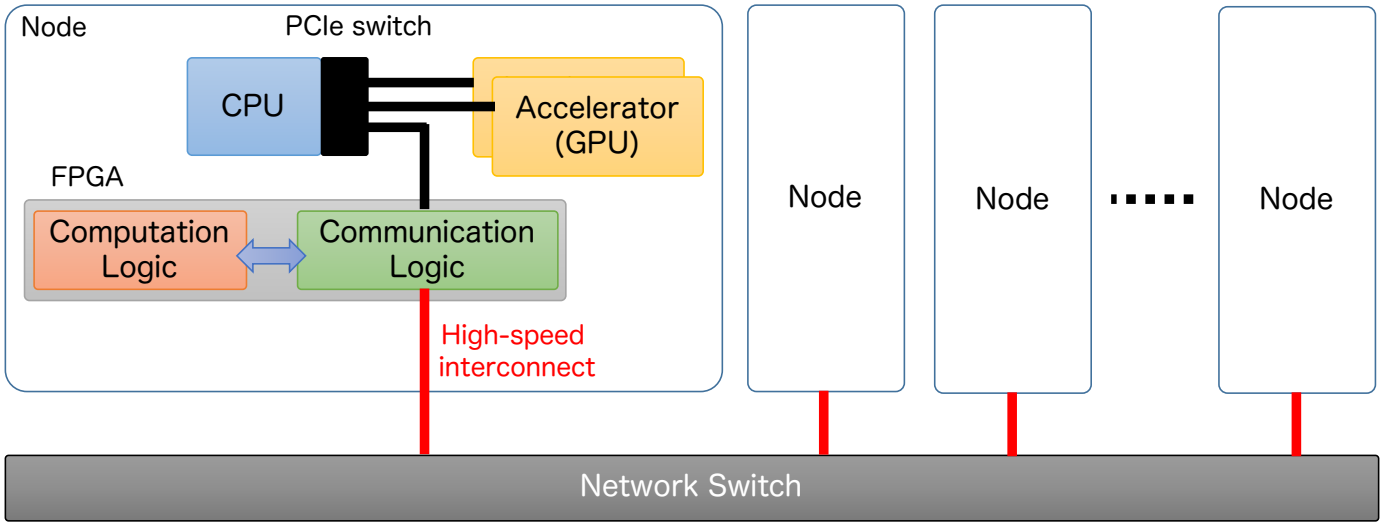
Each device's pros and cons

	performance (FLOPS)	external comm.	programming effort
CPU	△	○	◎
GPU	◎	△	○
FPGA	○	◎	× -> △? recently getting better

- Each device's strength and weakness are different
- A technology to compensate with each other is needed for more driving HPC forward
  - offering large degree of strong scalability

Accelerator in Switch (AiS) concept

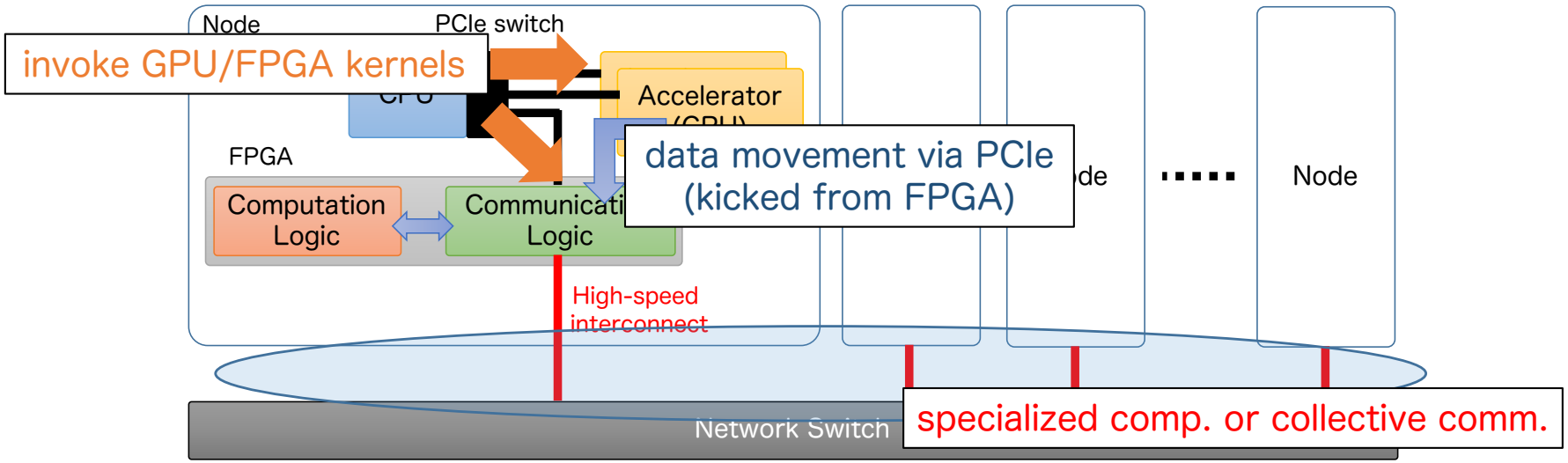
- What's this?
  - using FPGA for not only computation offloading but also communication
  - covering GPU non-suited computation by FPGA
  - combining computation offloading and ultra-low latency communication among FPGAs
  - especially effective on communication-related small/medium computation (such as collective communication)
  - OpenCL-enable programming for application users <- currently we are working on



Accelerator in Switch (AiS) concept

- What's this?
  - using FPGA for not only computation offloading but also communication
  - covering GPU non-suited computation by FPGA
  - combining computation offloading and ultra-low latency communication among FPGAs
  - especially effective on communication-related small/medium computation (such as collective communication)
  - OpenCL-enable programming for application users <- currently we are working on

Computation model



Previous studies about FPGA for HPC

- with OpenCL
  - Zohouri et al., “Evaluating and optimizing OpenCL kernels for high performance computing with FPGAs”, SC’16, pp.35:1 - 35:12
  - Weller et al., “Energy Efficient Scientific Computing on FPGAs using OpenCL”, FPGA2017, pp.247 - 256
  - Lee et al., “OpenACC to FPGA: A Framework for Directive-Based High-Performance Reconfigurable Computing”, IPDPS2016, pp.544 - 554

・focusing on how to implement and optimize high-performance and energy-efficient computation units with OpenCL capabilities and a single FPGA

- with multi-FPGA based system
  - Putnam et al., “A Reconfigurable Fabric for Accelerating Large-Scale Datacenter Services”, ISCA’14, pp.13 - 24
  - Sano et al., “Multi-FPGA Accelerator for Scalable Stencil Computation with Constant Memory Bandwidth”, IEEE TPDS, Vol. 25, No. 3, pp. 695–705, March 2014
  - Weerasinghe et al., “Network-attached FPGAs for data center applications”, FPT2016, pp.36 - 43

・focusing on building a huge pipelined computation unit across FPGAs
・not assuming OpenCL utilization
・basically point-to-point comm. (hard to be scaled to hundred or thousand nodes)

Previous studies about FPGA for HPC

- with OpenCL
  - Zohouri et al., “Evaluating and optimizing OpenCL kernels for high performance computing with FPGAs”, SC’16, pp.35:1 - 35:12
  - Weller et al., “Energy Efficient Scientific Computing on FPGAs using OpenCL”, FPGA2017, pp.247 - 256
  - Lee et al., “OpenACC to FPGA: A Framework for Directive Based High

All of people including us don’t know well
how a combination (OpenCL + FPGA + comm.) is
-> then let’s do it!!

- fo cient co
-
  - Putnam et al., “A Reconfigurable Fabric for Accelerating Large-Scale Datacenter Services”, ISCA’14, pp.13 - 24
  - Sano et al., “Multi-FPGA Accelerator for Scalable Stencil Computation with Constant Memory Bandwidth”, IEEE TPDS, Vol. 25, No. 3, pp. 695–705, March 2014
  - Weerasinghe et al., “Network-attached FPGAs for data center applications”, FPT2016, pp.36 - 43
- focusing on building a huge pipelined computation unit across FPGAs
- not assuming OpenCL utilization
- basically point-to-point comm. (hard to be scaled to hundred or thousand nodes)

Today's main topic: OpenCL-ready high speed network

- OpenCL environment is available
  - e.g. Intel FPGA SDK for OpenCL
  - basic computation can be written in OpenCL without Hardware Description Languages (HDLs)
- But, current FPGA board is not ready for OpenCL on interconnect access
- Proposal
  - **a method of high-performance FPGA-to-FPGA data movement that can be controlled using OpenCL code**
- Our goal
  - enabling OpenCL description by users including inter-FPGA communication
  - providing basic set of HPC applications such as collective communication, basic liner library
  - providing 40 ~ 100 Gbps Ethernet access with external switches for large scale systems

Intel FPGA SDK for OpenCL

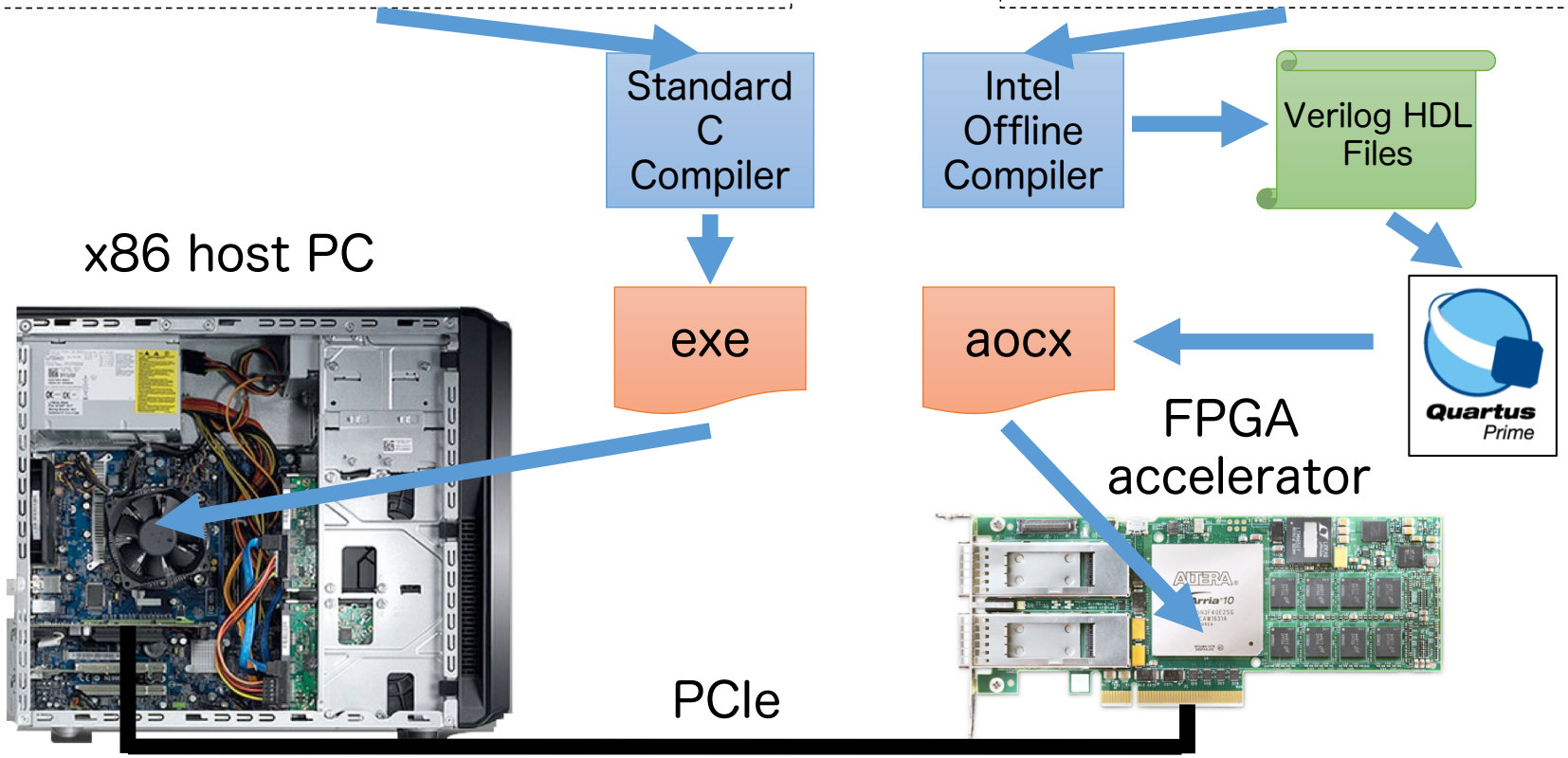
Programming model

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];
}
```



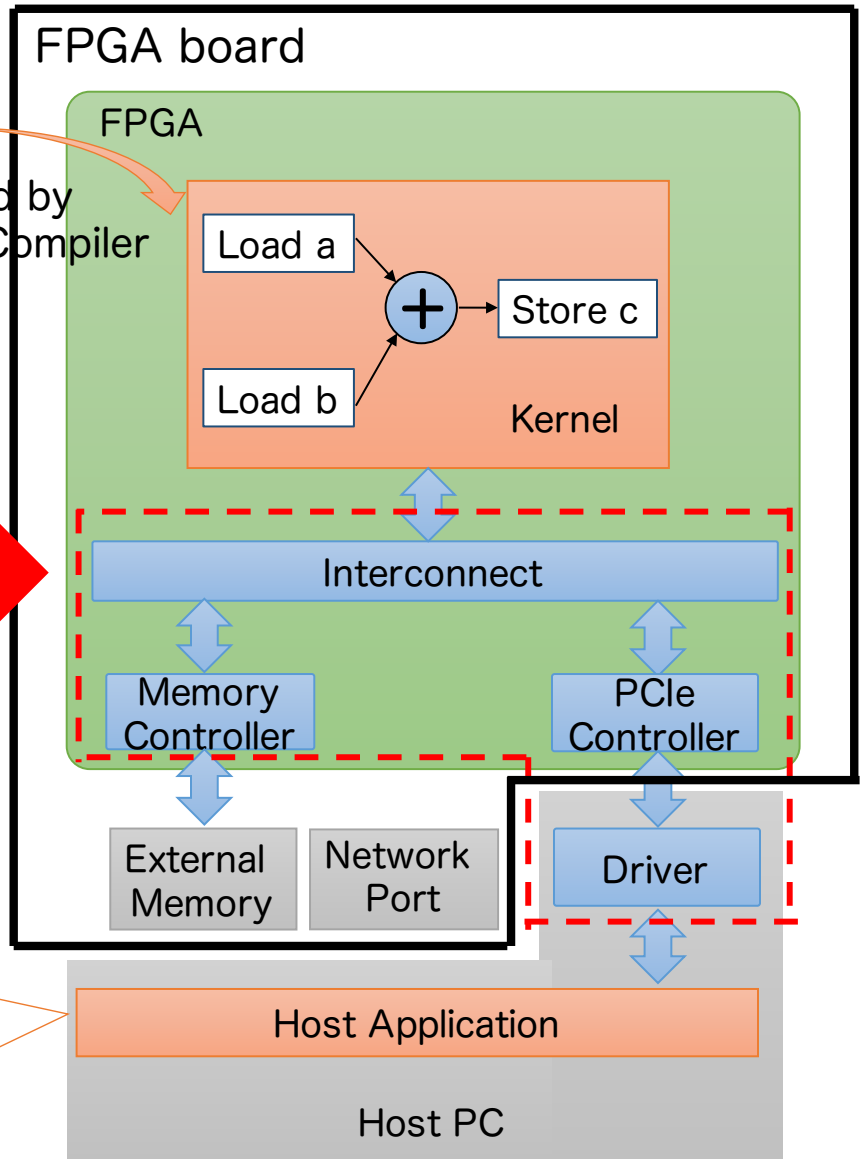
Schematic of the Intel FPGA SDK for an OpenCL platform

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];
}
```

Translated by Intel Offline Compiler

These features like peripheral controllers are provided from Board Support Package (BSP)

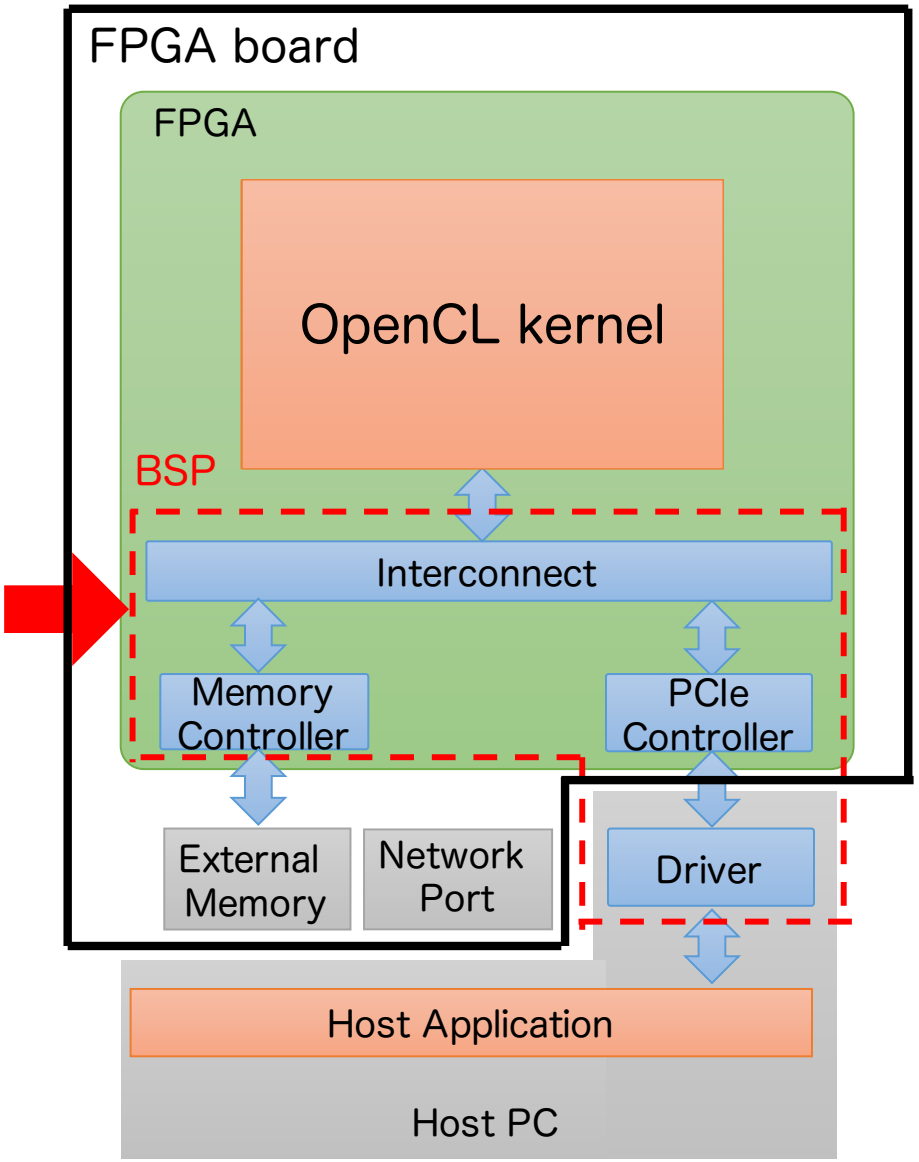


OpenCL host code

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

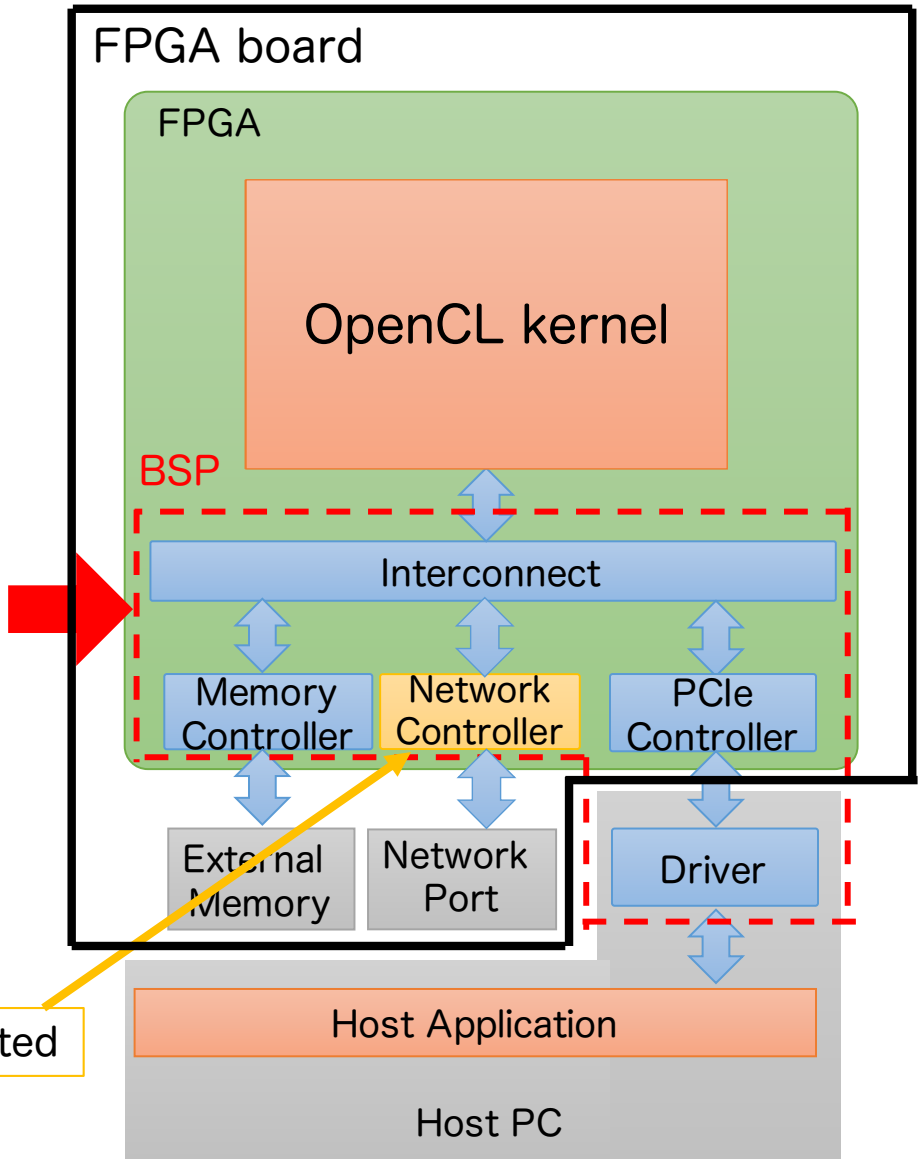
Schematic of the Intel FPGA SDK for an OpenCL platform

- description specifying FPGA chip and board peripherals configuration and access/control method
  - a sort of virtualization to enable same kernel development on an FPGA
  - independent for each board with FPGA
- Basically, only minimum interface is supported
  - minimum interface:
    external (DDR) memory and PCIe



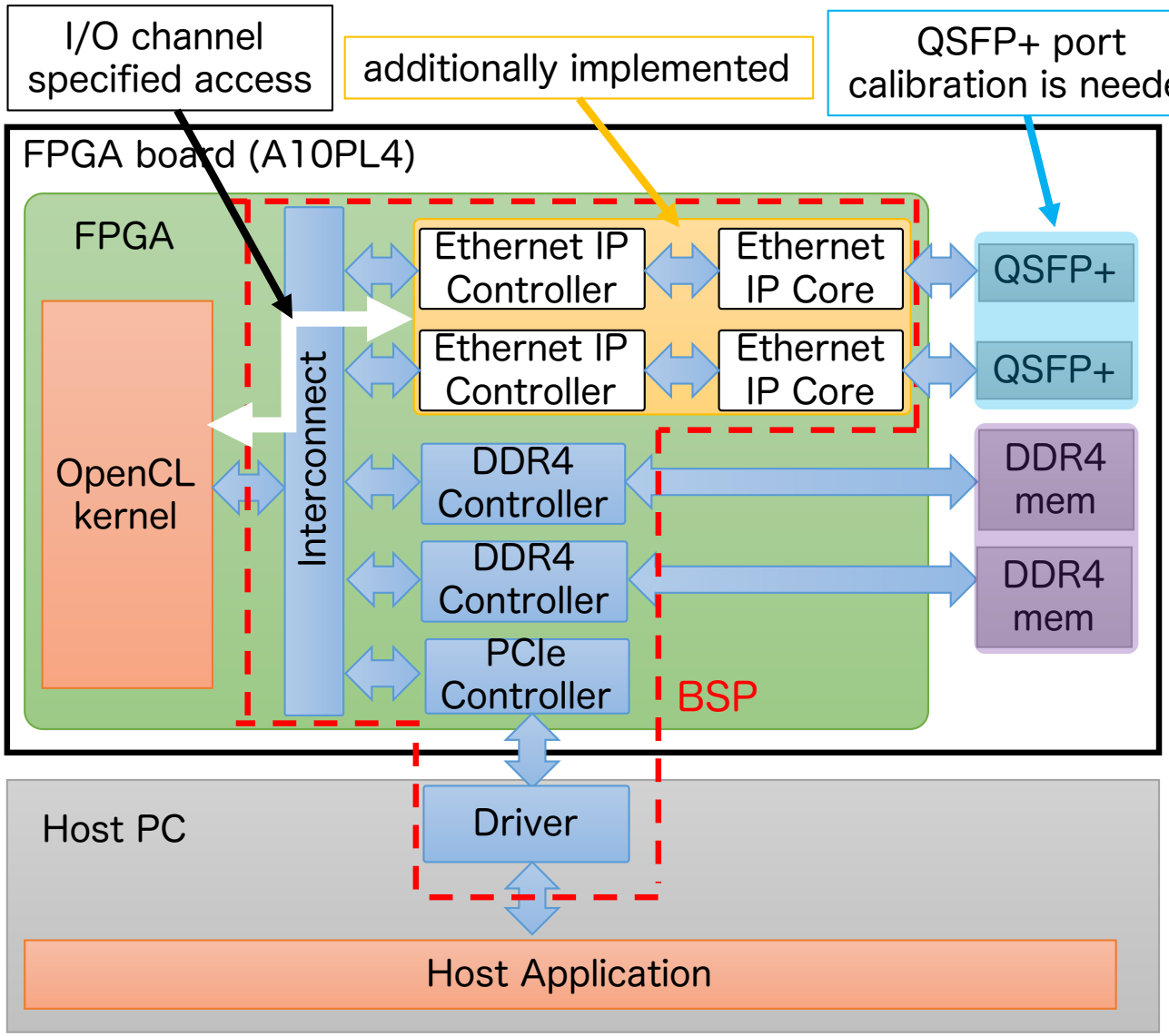
Schematic of the Intel FPGA SDK for an OpenCL platform

- description specifying FPGA chip and board peripherals configuration and access/control method
  - a sort of virtualization to enable same kernel development on an FPGA
  - independent for each board with FPGA
- Basically, only minimum interface is supported
  - minimum interface: external (DDR) memory and PCIe
  - to perform inter-FPGA comm., implementing network controller and integrating it into the BSP are required

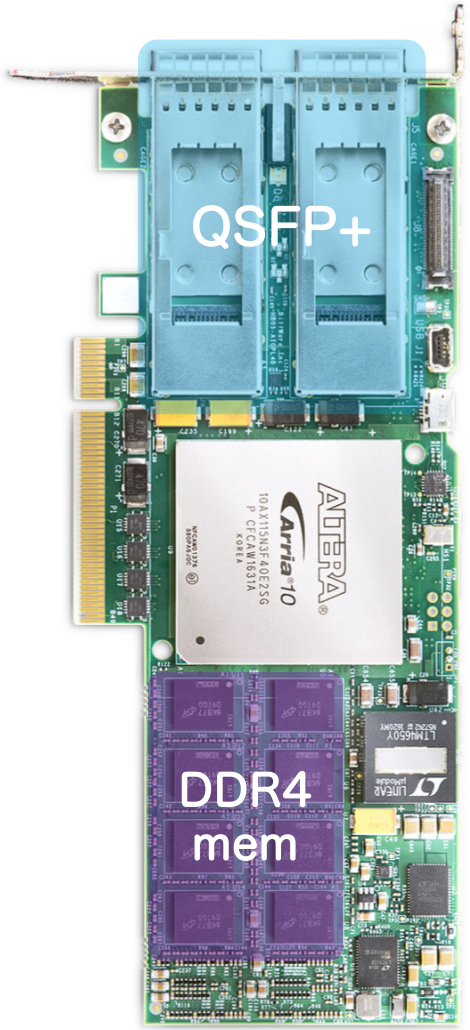


OpenCL-enabled inter-FPGA data movement

Implementation overview

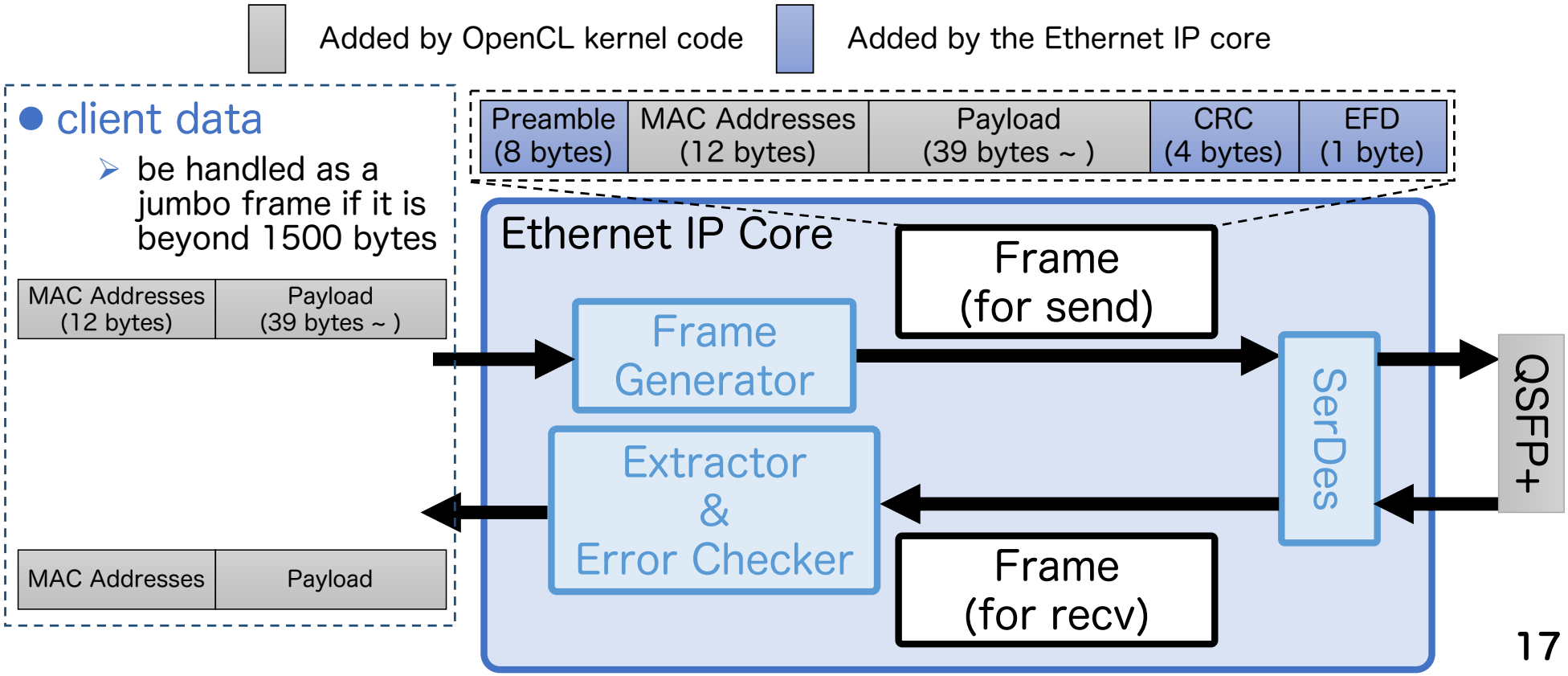


Our FPGA board (BittWare A10PL4)



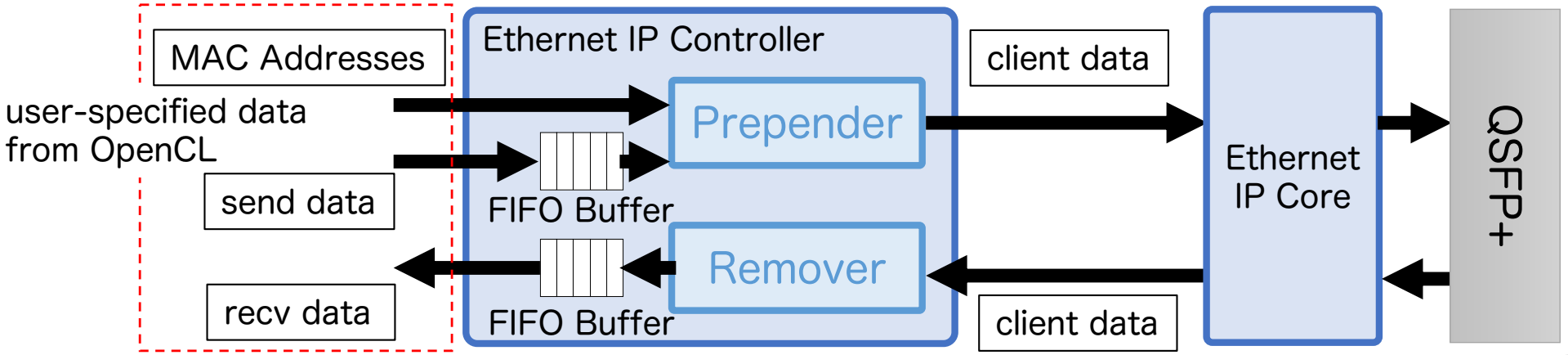
Ethernet Intellectual Property (IP) Core

- Low-latency 40- and 100-Gbps Ethernet MAC and PHY MegaCore function
  - provided from Intel
  - offering essential features of the physical and media access control layers for Ethernet comm.



Ethernet IP Controller

- a joint module for OpenCL and Ethernet IP core
  - our homemade hardware unit implemented with Verilog HDL
- managing the client data
  - Prepender
    - ✓ sends the MAC addresses to the IP core at first, and then extracts data from the client payload stored in the FIFO buffer
  - Remover
    - ✓ getting payload data
- currently, error-handling logic such as a retransmission feature is not supported



OpenCL code snippets for ping-pong

- Data movement is performed with I/O channel API

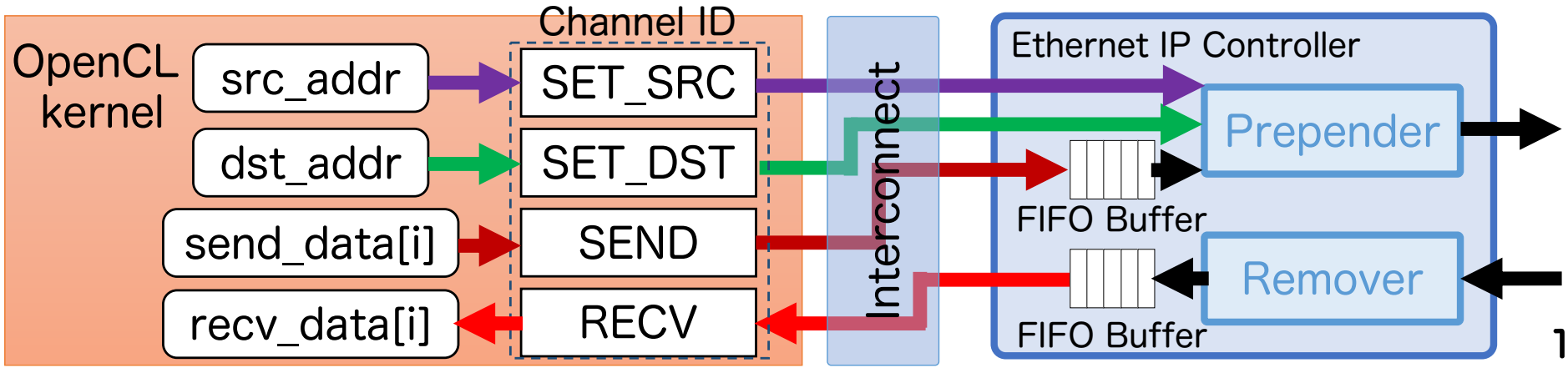
sender

```
// Set MAC Addresses
write_channel_intel(SET_SRC , src_addr);
write_channel_intel(SET_DST, dst_addr);

// Set send data
for (i = 0 ; i < data_size ; i++) write_channel_intel(SEND, send_data[i]);
```

receiver

```
// Get recv data
for (i = 0 ; i < data_size ; i++) recv_data[i] = read_channel_intel(RECV);
```



OpenCL code for ping-pong (ping side)

```
/***** I/O channel id definition *****/
channel int  set_src_addr __attribute__((depth(0))) __attribute__((io("kernel_send_sadr")));
channel int  set_dst_addr __attribute__((depth(0))) __attribute__((io("kernel_send_dadr")));
channel int8 set_data     __attribute__((depth(0))) __attribute__((io("kernel_send_data")));
channel int8 get_data     __attribute__((depth(0))) __attribute__((io("kernel_recv_data")));

__kernel void ether_test(
    const global int8 *restrict send_data,
    global int8 *restrict recv_data,
    const int src_addr,
    const int dst_addr,
    const int datanum)
{
  int i;

  // Set MAC Addresses
  write_channel_intel(set_src_addr, src_addr);
  write_channel_intel(set_dst_addr, dst_addr);

  // Set payload for sending
  for (i = 0; i < datanum; i++) write_channel_intel(set_data, send_data[i]);

  // Get payload recieved
  for (i = 0; i < datanum; i++) recv_data[i] = read_channel_intel(get_data);
}
```

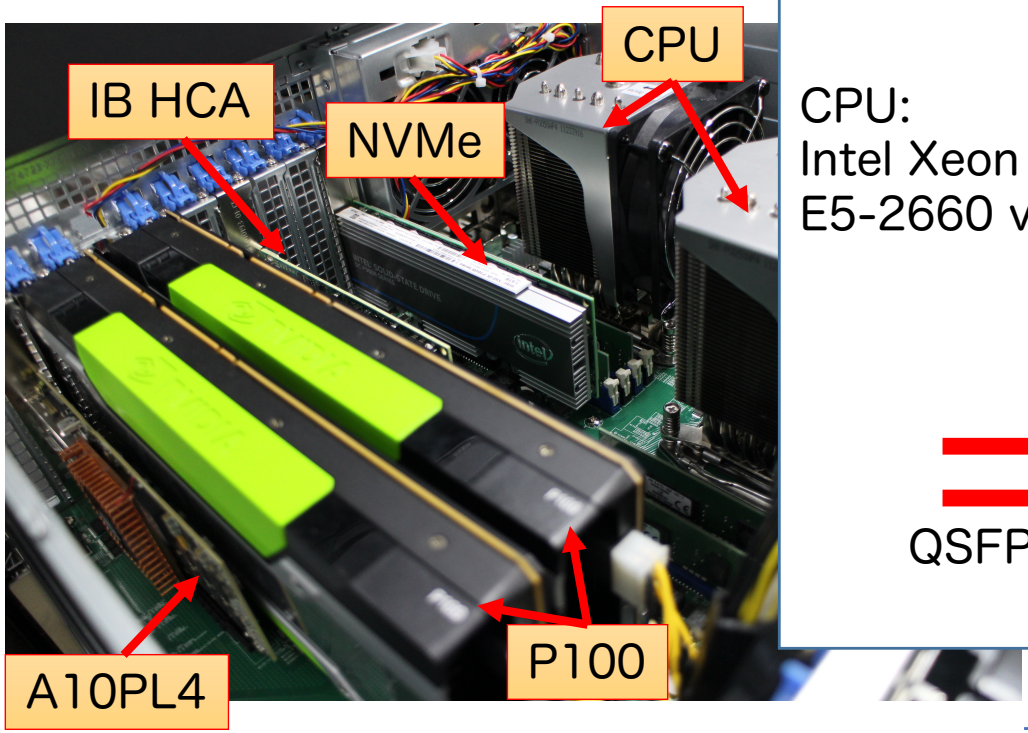
MAC Addresses are sent from OpenCL host code

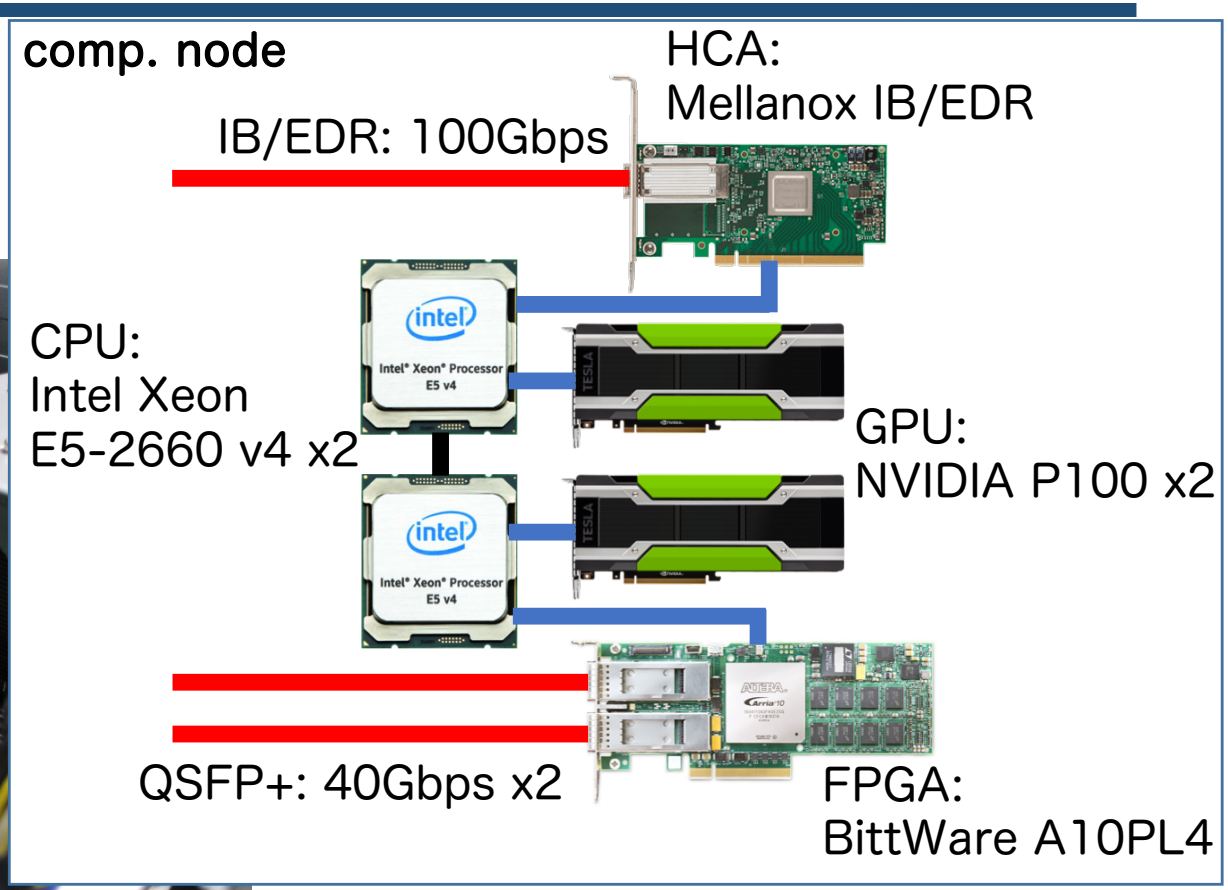
specifying how many data chunks are sent and received

Evaluation

Evaluation testbed

- Pre-PACS-X (PPX)
  - PACS-X prototype
  - CCS, U. Tsukuba

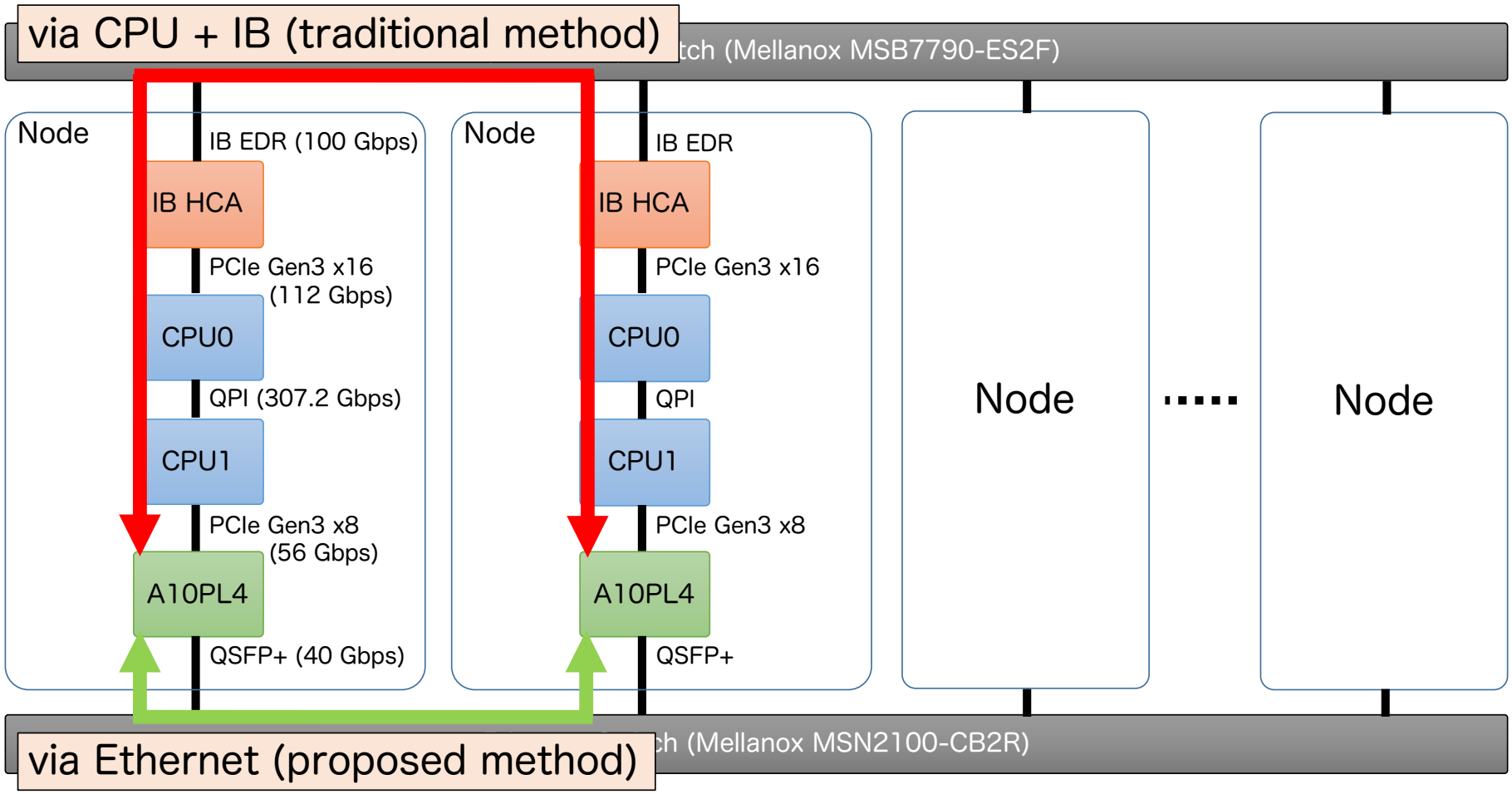




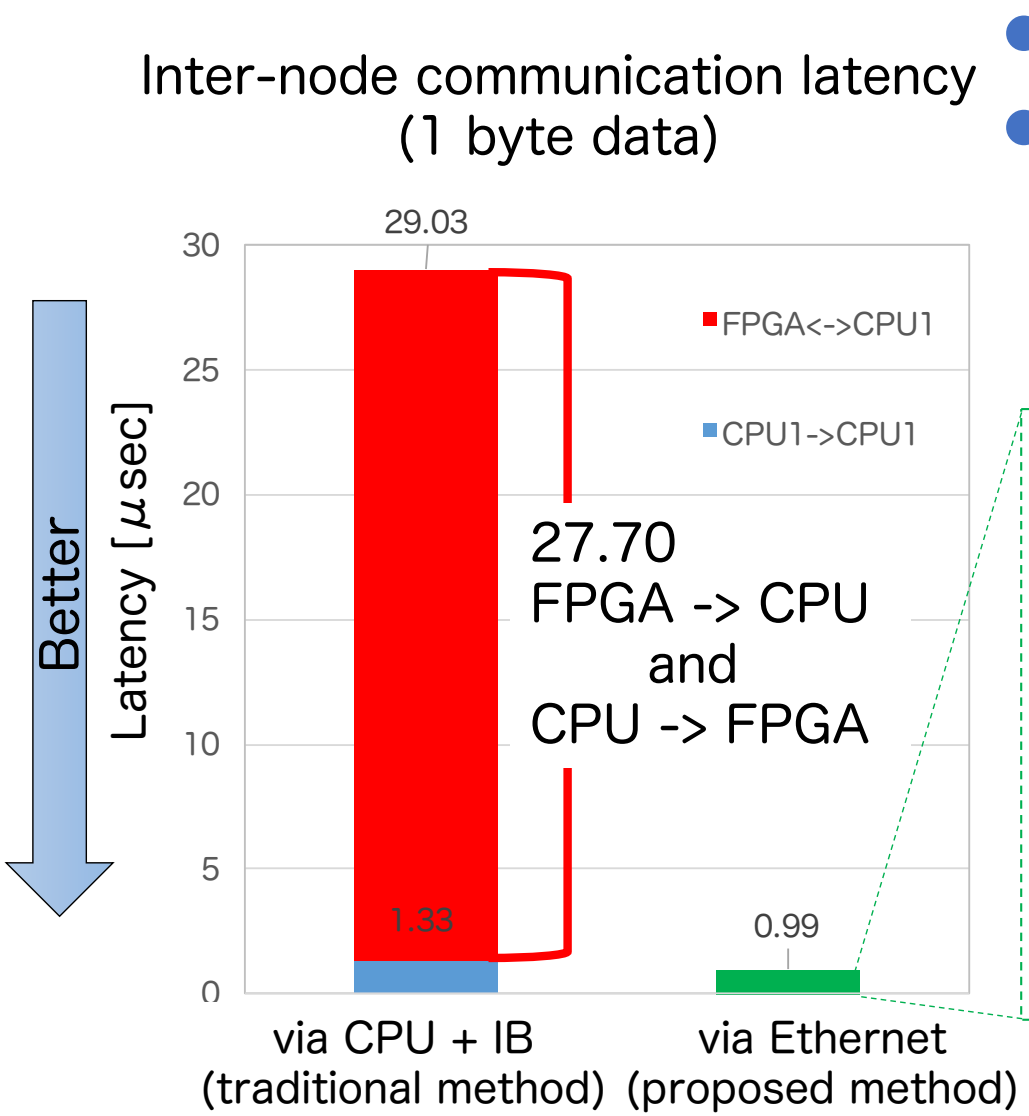
Host OS	CentOS 7.3
Host Compiler	gcc 4.8.5
FPGA toolchain	Intel FPGA SDK for OpenCL, Intel Quartus Prime Pro Version 17.0.0 Build 289

For more detail, please refer to our paper

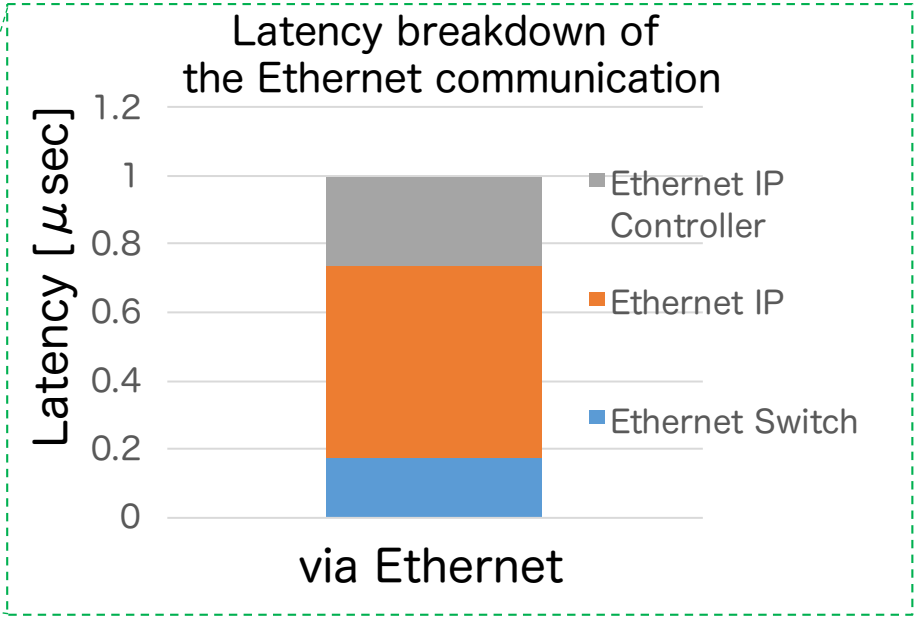
Communication paths for FPGA-to-FPGA data movement



Communication latency

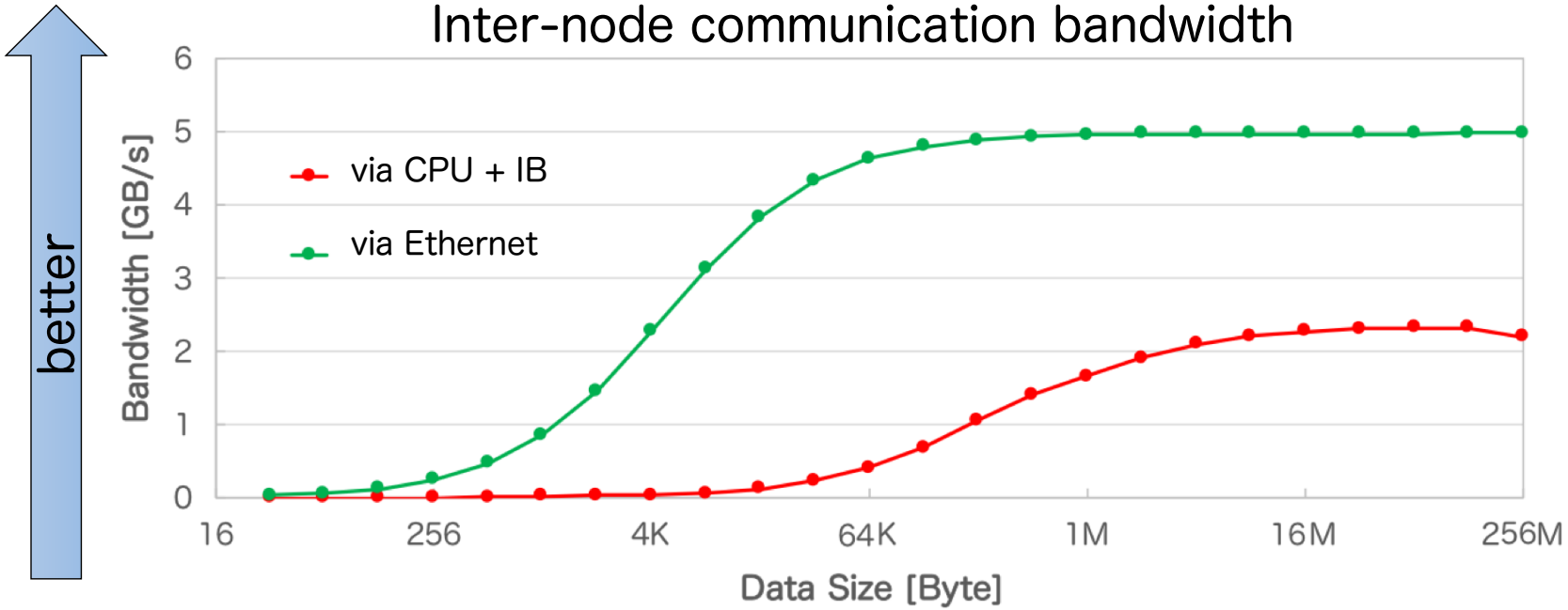


- via Ethernet: ~1 μsec
- via CPU + IB: 29.03 μsec
  - ➢ CPU-FPGA comm. is dominant
    - ✓ CPU-FPGA interface offered by current BSP is not good



Communication bandwidth

- Our proposed method (via Ethernet) achieves 4.97 GB/s
  - 99.4 % of theoretical peak (w/o error handling)
  - the maximum effective bandwidth is achieved at the earlier phase by short latency, compared the traditional method (via CPU + IB)
- The traditional method achieves 2.32 GB/s
  - non-pipelined communication (store-and-forward manner)
  - no special feature (e.g. GPUDirect RMA) between FPGA and IB HCA



FPGA resource usage

	ALMs	Registers	M20K memory blocks	Transceivers
Ethernet IP Core	12,685 (3.0 %)	23,132 (1.3 %)	13 (0.5 %)	8 (8.3 %)
Ethernet IP Controller	417 (0.1 %)	998 (0.1 %)	28 (1.0 %)	0 (0.0 %)
OpenCL Kernel	9,707 (2.3 %)	23,664 (1.4 %)	89 (3.3 %)	0 (0.0 %)
BSP itself	27,741 (6.5 %)	52,852 (3.1 %)	228 (8.4 %)	16 (16.7 %)
Total	50,550 (11.9 %)	100,646 (5.9 %)	358 (13.2 %)	24 (25.0 %)

- resource increase for point-to-point data movement using a single channel
  - ALMs: 3.1 % | Registers: 1.4 % | M20K memory blocks: 1.5 % | Transceivers: 8.0 %
  - implementation overhead for Ethernet comm. is not so big (w/o error-handling)

- how to use the remaining resources
  - for error-handling logics
    - ✓ retransmission, flow control, etc.
  - for supporting collective comm.

Implementing comp. logics depends on the above
-> saving their resource usage is important

- ALM, Register
  - basic elements to implement hardware modules
- M20K
  - an built-in memory block (size: 20k bits / block)
- Transceiver
  - a built-in hardware component for communication
  - using 8 transceivers for QSFP+, 16 for PCIe Gen3 x8

Conclusion and future work

Conclusion

- Proposal
  - OpenCL-ready FPGA-to-FPGA Ethernet communication
    - ✓ through a QSFP+ optical interconnect
    - ✓ using I/O Channel API to perform data movement
      - be realized by implementing QSFP+ controller and integrating it into the BSP
- Evaluation: ping-pong communication
  - latency: ~1 μsec, bandwidth: 4.97 GB/s
    - ✓ Low latency and high bandwidth comm. can be achieved
  - FPGA resource usage (total)
    ALMs: 11.9 % | Registers: 5.9 % | M20K: 13.2 % | Transceivers: 25.0 %
    - ✓ plenty of resources are still available, but we must consider carefully how to use them
  - The results suggest that our proposed approach is a promising means to realize the AiS concept

Future work

- Implementing communication error-handling logics
  - e.g. retransmission, flow control
- Usability improvement
  - e.g. supporting collective communication
- Performing evaluation using more practical apps
  - astrophysics is mainly targeted

We will promote the reconfigurable HPC to offer strong scalability!

Thank you for your attention!!