PACS-X
New Frontier of Accelerated HPC

Taisuke Boku
Deputy Director / HPC Division Leader
Center for Computational Sciences
University of Tsukuba
Outline

- FPGA for HPC as large scale parallel system
- AiS (Accelerator in Switch) concept
- FPGA for high performance interconnection and computation
- PACS-X Project and PPX
- OpenCL-enabled computation/communication on FPGA
- Application Example: Astrophysics
- Summary
Accelerators in HPC

- Traditionally...
  - Cell Broadband Engine, ClearSpeed, GRAPE. ...
  - then GPU (most popular) and MATRIX-2000 😊

- Is GPU perfect?
  - good for many applications (replacing vector machines)
  - depending on very wide and regular computation
    - large scale SIMD (STMD) mechanism in a chip
    - high bandwidth memory (GDR5, HBM) and local memory
  - bad for
    - not enough parallelism
    - not regular computation (warp splitting)
    - frequent inter-node communication (kernel switch, go back to CPU)
FPGA in HPC

- Goodness of recent FPGA for HPC
  - True codesigning with applications (essential)
  - Programmability improvement: OpenCL, other high level languages
  - High performance interconnect: 40Gb~100Gb
  - Precision control is possible
  - Relatively low power

- Problems
  - Programmability: OpenCL is not enough, not efficient
  - Low standard FLOPS: still cannot catch up to GPU
    -> “never try what GPU works well on”
  - Memory bandwidth: 2-gen older than high end CPU/GPU
    -> be improved by HBM (Stratix10)
## Simple pros/cons

<table>
<thead>
<tr>
<th></th>
<th>performance (FLOPS)</th>
<th>external communication (sec, B/s)</th>
<th>programming cost</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>CPU</strong></td>
<td>△</td>
<td>○</td>
<td>◎</td>
</tr>
<tr>
<td><strong>GPU</strong></td>
<td>◎</td>
<td>△</td>
<td>○</td>
</tr>
<tr>
<td><strong>FPGA</strong></td>
<td>○</td>
<td>◎</td>
<td>× → △?</td>
</tr>
</tbody>
</table>

How to compensate with each other toward large degree of strong scaling?
AiS

AiS: Accelerator in Switch

- Using FPGA not only for computation offloading but also for communication
- Combining computation offloading and communication among FPGAs for ultra-low latency on FPGA computing
- Especially effective on communication-related small/medium computation (such as collective communication)
- Covering GPU non-suited computation by FPGA
- OpenCL-enable programming for application users
invoke GPU/FPGA kernels

data transfer via PCIe
(invoked from FPGA)

CPU

GPU

FPGA

PCIe

comp.

comm.

Ethernet Switch

collective or specialized comp.+comm.

> QSFP+ interconnect
How fast on communication? (FPGA-FPGA link)

- Xilinx XC7VX1140T (Virtex7) with 100Gbps optical interconnect
- up to 96% of theoretical peak
- good scalability up to 3 channels aggregation
- Intel Stratix10 will have up to 4 channels of 100Gbps link
PACS-X (ten) Project at CCS, U. Tsukuba

- PACS (Parallel Advanced system for Computational Sciences)
  - a series of co-design base parallel system development both on system and application at U. Tsukuba (1978~)
  - recent systems focus on accelerators
    - PACS-VIII: HA-PACS (GPU cluster, Fermi+Kepler, PEACH2, 1.1PFLOPS)
    - PACS-IX: COMA (MIC cluster, KNC, 1PFLOPS)
- Next generation of TCA implementation
  - PEACH2 with PCIe is old and with several limitation
  - new generation of GPU and FPGA with high speed interconnection
  - more tightly co-designing with applications
  - system deployment starts from 2018 (?)

PPX: Pre-PACS-X
PPX: testbed under AiS concept (x6~12 nodes)

- 1.6TB NVMe
- Xeon Broadwell
- QPI
- Xeon Broadwell
- GPU: coarse-grain offloading
  NVIDIA P100 x 2
- FPGA: fine-grain partial offloading
  Altera Arria10 (Bitware A10PL4)
- HCA: Mellanox IB/EDR
  100G IB/EDR
- 40Gb Ether x 2
  -> upgrade to 100G x 2
- 1.6TB NVMe
- 100G IB/EDR
- NVIDIA P100 x 2
- Xeon Broadwell
- QPI
PPX (Pre-PACS-X) mini-cluster system

- CPU: BDW x 2
- GPU: P100 x 2
- FPGA: Intel or Xilinx

- InfiniBand/EDR Switch
- x12 nodes
- InfiniBand/EDR (100Gbps)
- 100G Ethernet Switch
- Ethernet (40Gbps x 2)
- GbE switch
- login node
- FPGA: 6 with Intel
  6 with Xilinx
OpenCL-enabled high speed network

- OpenCL environment is available
  - ex) Intel FPGA SDK for OpenCL
  - basic computation can be written in OpenCL without Verilog HDL
- But, current FPGA board is not ready for OpenCL on interconnect access
  - BSP (Board Supporting Package) is not complete for interconnect
    → we developed for OpenCL access
- Our goal
  - enabling OpenCL description by users including inter-FPGA communication
  - providing basic set of HPC applications such as collective communication, basic linear library
  - providing 40G~100G Ethernet access with external switches for large scale systems
BSP (Board Support Package)

- description specifying FPGA chip and board peripherals, configuration and access/control method
  - independent for each board with FPGA
  - a sort of virtualization to enable same kernel development on FPGA
- minimum interface is provided by board vendors
  - we need optical interconnection access method in BSP
Our test bed (BittWare A10PL4 with Intel Arria10 FPGA)
Ethernet IP Controller

OpenCL Kernel
- recv data
- MAC address + etc.
- send data

user specified
send/recv data from OpenCL

joint module for OpenCL and Ethernet IP
creating Ethernet frame
- INSERT
  - MAC addr. etc.
- EXTRACT
  - getting payload data
currently, re-transmission feature is not implemented

Ethernet IP Controller
implemented by Verilog HDL

Ethernet IP

FIFO
payload
EXTRACT

FIFO
payload
INSERT

MAC addr.
payload

joint module for OpenCL and Ethernet IP
creating Ethernet frame
- INSERT
  - MAC addr. etc.
- EXTRACT
  - getting payload data
currently, re-transmission feature is not implemented
OpenCL code example for pingpong

```c
write_channel_intel (SET_SOURCE , source_addr);
write_channel_intel (SET_DEST , (int2)(data_size , dest_addr));

for( i = 0 ; i < data_size ; i ++)
    write_channel_intel (SEND , send_data[i] );

for( i = 0 ; i < data_size ; i ++)
    receive_data[i] = read_channel_intel (RECEIVE);
```

OpenCL Kernel

- sender MAC addr.: source_addr
- receiver MAC addr. : dest_addr
- send data size : data_size
- data pointer : send_data
- data pointer : receive_data

Ethernet IP Controller

- SET_SOURCE
- SET_DEST
- SEND
- RECEIVE
- INSERT
- FIFO
- EXTRACT

Channel_ID
Evaluation test-bed

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

Comp. node
- CPU: Xeon E5-2660 v4
- GPU: NVIDIA P100 x2
- FPGA: Bittware A10PL4
- QSFP+: 40Gbpsx2
- Host OS: CentOS 7.3
- Host Compiler: gcc 4.8.5
- FPGA Compiler: Intel FPGA SDK for OpenCL, Intel Quartus Prime Pro Version 17.0.0 Build 289
- HCA: Mellanox IB/EDR IB/EDR: 100Gbps
Communication paths

IB (InfiniBand) Switch

IB EDR (100Gbps)

PCle Gen.3x16

CPU0

QPI

CPU1

QSFP+ (40Gbps)

FPGA

via-IB

via-Ethernet

Ethernet Switch

NODE

CPU0

IB HCA

IB HCA

CPU1

FPGA

CPU1

NODE

...
Communication latency

- ~1μs latency on Ethernet
- CPU-FPGA comm. occupies comm. latency via IB
  - CPU-FPGA interface by current BSP is not good

![Bar chart](image)

Latency [usec]

- Node-to-node communication latency (1 byte data)
  - FPGA <-> CPU1
  - CPU1 -> CPU1
  - FPGA -> CPU
  - CPU -> FPGAの合計

Break down of Ethernet latency

- Ethernet IP Controller
- Ethernet IP
- Ethernet Switch

Latency [usec] via Ethernet
Communication bandwidth

• 40Gbps Ethernet achieves 4.97GB/s
  - 99.8 % of theoretical peak (w/o error handling)
  - small $N_{1/2}$ by short latency

• via-IB achieves 2.32GB/s
  - non-pipelined
  - no special feature (such as GPUDirect) on FPGA-HCA
AiS application example: ARGOT

- **ARGOT** (Accelerated Radiative transfer on grids using Oct-Tree)
  - Radiative transfer simulation code developed in CCS
  - Two basic computing methods for radiation transfer
    - ARGOT method
      - from a light source
    - ART
      - from spatially spread light sources
- CPU version and GPU version with MPI
- ART method occupies >90% of computation even on GPU, and we need more speedup
  → making FPGA offloading in AiS concept
ART method

- radiative transfer computing on spatially spread light sources
- ray-tracing on 3-D space with grid decomposed partitions
  - rays are in parallel
  - different input angles
  - no reflection nor refraction (different from 3-D graphics ray-tracing)
  - HEALPix algorithm for ray generation
- large scale for parallel processing
  - mesh size: $100^3 \sim 1000^3$
  - ray angles: 768 $\sim$ 1000s
Performance (single FPGA) on ART method

- up to 14.6x faster than CPU, and 5.1x faster than GPU
- 93% of computation time of ARGOT is dominated by ART method
  → 7.48x speedup on entire code is expected
## Circuit resource utilization

<table>
<thead>
<tr>
<th></th>
<th>ALMs</th>
<th>Registers</th>
<th>M20K</th>
<th>MLAB</th>
<th>MLAB size</th>
<th>DSP</th>
<th>Freq.</th>
</tr>
</thead>
<tbody>
<tr>
<td>w/o autorun</td>
<td>228,610</td>
<td>473,747</td>
<td>1,839</td>
<td>4,330</td>
<td>47,968</td>
<td>536</td>
<td>228.57</td>
</tr>
<tr>
<td></td>
<td>(54%)</td>
<td>(55%)</td>
<td>(68%)</td>
<td></td>
<td></td>
<td></td>
<td>MHz</td>
</tr>
<tr>
<td>w/ autorun</td>
<td>228,835</td>
<td>467,225</td>
<td>1,716</td>
<td>7,350</td>
<td>138,288</td>
<td>536</td>
<td>236.11</td>
</tr>
<tr>
<td></td>
<td>(54%)</td>
<td>(55%)</td>
<td>(63%)</td>
<td></td>
<td></td>
<td></td>
<td>MHz</td>
</tr>
<tr>
<td>difference</td>
<td>+225</td>
<td>-6,255</td>
<td>-123</td>
<td>+3,020</td>
<td>+90,320</td>
<td>0</td>
<td>+7.54</td>
</tr>
</tbody>
</table>

- largest resource use is on M20K (63%)
  - actually 53.3% (without BSP use)
- DSP utilization is only 53%
- We can achieve up to 2x more speed
Next Step

- **Precision controlling**
  - for ART and ARGOT, SP is too much, HP is not balanced
  - finding best (e, m, s) combination
    - e=exponent  m=mantissa  s=(exponent digit shift)

- **Combining Communication and Computation**
  - OpenCL computing kernels binding with OpenCL Ethernet communication layer kernels with OpenCL Channel (by Intel SDK)
  - Possibility on network from Ethernet-switch to Direct Link between FPGA (with Stratix10 or similar in Xilinx)

- **Combining GPU and FPGA**
  - GPU global memory access via PCIe from FPGA
    - technology porting from TCA/PEACH2 in HA-PACS project
  - GPU/FPGA offloading control from CPU
    - new programming paradigm is required
High Level Programming Paradigm

- **XcalableACC**
  - under development in collaboration between CCS-Tsukuba and RIKEN-AICS
  - PGAS language XcalableMP is enabled to imply OpenACC for sophisticated coding of distributed memory parallelization with accelerator
  - inter-node communication among FPGA can be implemented by FPGA-Ethernet direct link
  - Data movement between GPU and FPGA

- **OpenACC for FPGA**
  - (plan) research collaboration with ORNL FTG
  - OpenACC -> OpenCL -> FPGA compilation by OpenARC project is under development
  - final goal: XcalableACC with OpenARC compiler and FPGA-Ethernet link
  - (Or OpenMP->OpenCL->FPGA for XcalableMP implementation)
Summary

- FPGA for HPC is very attractive theme for next generation of accelerated platform
- FPGA is usable not only for computing but also for communication
- 360-degree system to cover highly parallel STMD computing by GPU and flexible processing on FPGA with communication feature
- OpenCL-enabled programming including communication for application users
- CCS, U. Tsukuba is moving forward to realize AiS concept on next generation multi-hetero supercomputing toward PACS-X implementation