HA-PACS/TCA
GPU Direct Communication

Taisuke Boku

Deputy Director, HPC Division

Center for Computational Sciences

University of Tsukuba

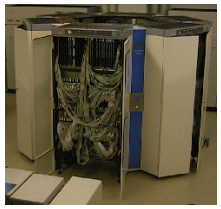
Hisotry of PAX (PACS) MPP series

- **Launched in 1977 (Prof. Hoshino and Prof. Kawai)**
- **First machine was completed in 1979**
- **6th generation machine CP-PACS was ranked #1 in TOP500 in Nov. 1996**

1978
1st PACS-9

1980
2nd PAXS-32

1989
5th QCDPAX



1996
6th CP-PACS as world fastest machine

2006
7th PACS-CS
bandwidth aware

2012
8th HA-PACS
GPU accelerated

Year	Name	Performance
1978年	PACS-9	7 KFLOPS
1980年	PACS-32	500 KFLOPS
1983年	PAX-128	4 MFLOPS
1984年	PAX-32J	3 MFLOPS
1989年	QCDPAX	14 GFLOPS
1996年	CP-PACS	614 GFLOPS
2006年	PACS-CS	14.3 TFLOPS
2012年	HA-PACS	802 TFLOPS

- High end supercomputer based on MPP architecture towards “practical machine” under collaboration with computational scientists and computer scientists
- Development in Application-driven
- Continuation of R & D by an organization

PAX (PACS) Series

- MPP system R&D continued at U. Tsukuba for more than 30 years
- Coupling of need from applications and seeds from the latest HPC technology, the machines have been developed and operated with the effort by application users on programming
  → a sort of application oriented machine
  (not for a single application)
- HA-PACS is the first system in the series to introduce accelerating devices (GPUs)
- CCS has been focusing on the accelerating devices for ultra high performance to provide to “high-end” users who require extreme computing facilities

Project plan of HA-PACS

- **HA-PACS (Highly Accelerated Parallel Advanced system for Computational Sciences)**
- Accelerating critical problems on various scientific fields in Center for Computational Sciences, University of Tsukuba
  - The target application fields will be partially limited
  - Especially interested argets: QCD, Astro, QM/MM (quantum mechanics / molecular mechanics, for life science), Climate
- Two parts
  - HA-PACS *base cluster*:
    - for development of GPU-accelerated code for target fields, and performing product-run of them
  - HA-PACS/*TCA*: (TCA = Tightly Coupled Accelerators)
    - for elementary research on new technology for accelerated computing
    - Our original communication system based on PCI-Express named “PEARL”, and a prototype communication chip named “PEACH2”

GPU Computing: current trend of HPC

- **Major GPU base systems in TOP500 on Nov. 2013**
  - Titan, ORNL
  - Tienha-1A, NUDT
  - Nebulae, CSA
  - TSUBAME2.5

- **Features**
  - high peak performance / cost ratio
  - high peak performance / power ratio
  - large scale applications with GPU acceleration don't run yet in production on GPU cluster

  ⇒ **Our First target is to develop large scale applications accelerated by GPU in real computational sciences**

Issues of GPU Cluster

- **Problems of GPGPU for HPC**
  - Data I/O performance limitation
    - Ex) GPGPU: PCIe gen2 x16
    - Peak Performance： 8GB/s (I/O) ⇔ 665 GFLOPS (NVIDIA M2090)
  - Memory size limitation
    - Ex) M2090: 6GByte vs CPU: 4 – 128 GByte
  - Communication between accelerators: no direct path (external)
    ⇒ communication latency via CPU becomes large
    - Ex) GPGPU:
      GPU mem ⇒ CPU mem ⇒ (MPI) ⇒ CPU mem ⇒ GPU mem
- **Researches for direct communication between GPUs are required**

Our another target is developing a direct communication system between external GPUs for a feasibility study for future accelerated computing

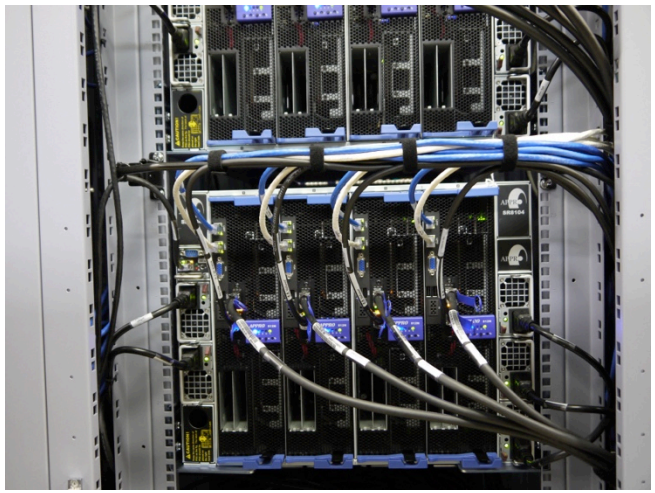
HA-PACS base cluster (Feb. 2012)

HA-PACS base cluster

Front view

Side view

HA-PACS base cluster

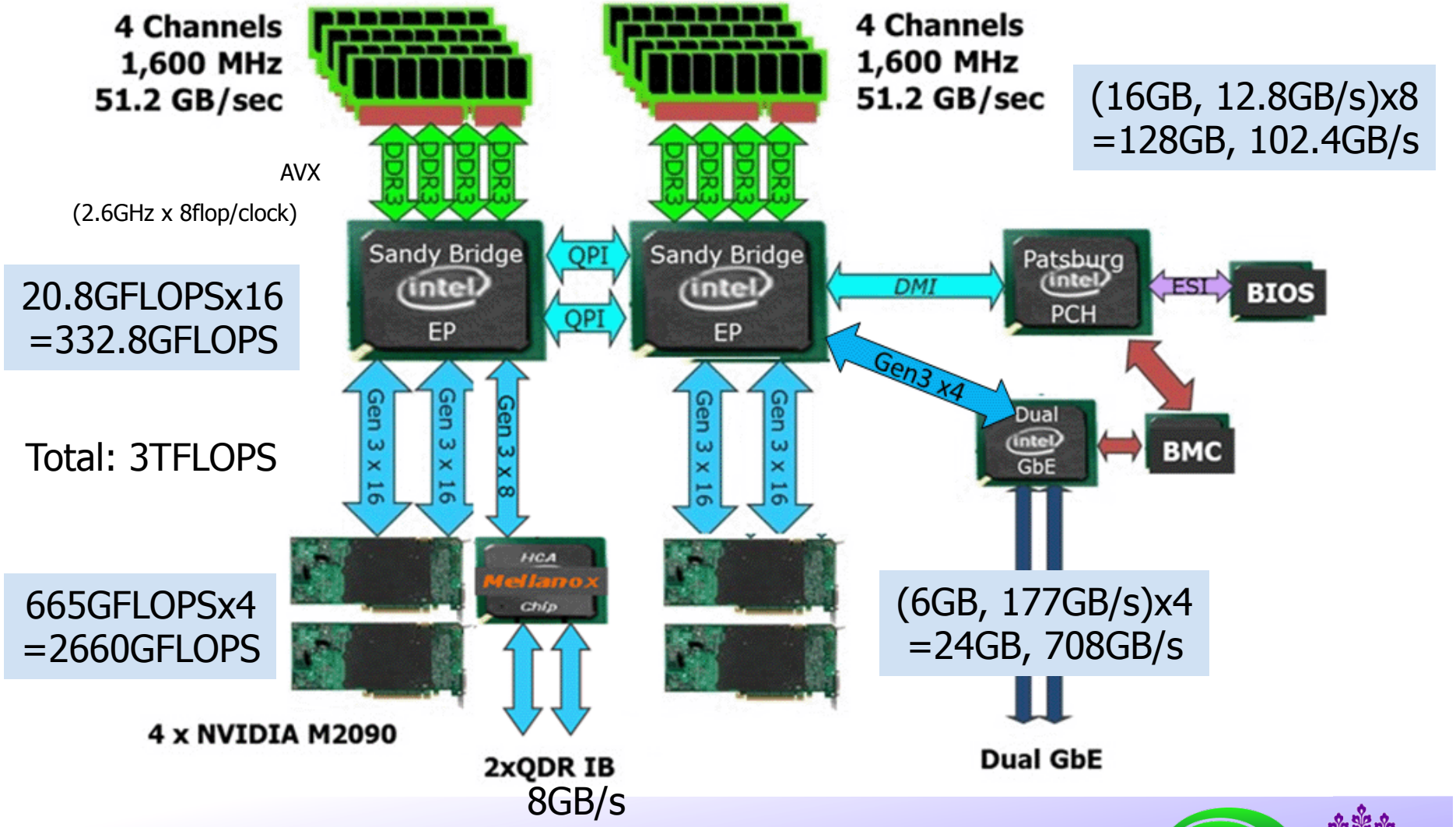


Rear view of one blade chassis with 4 blades

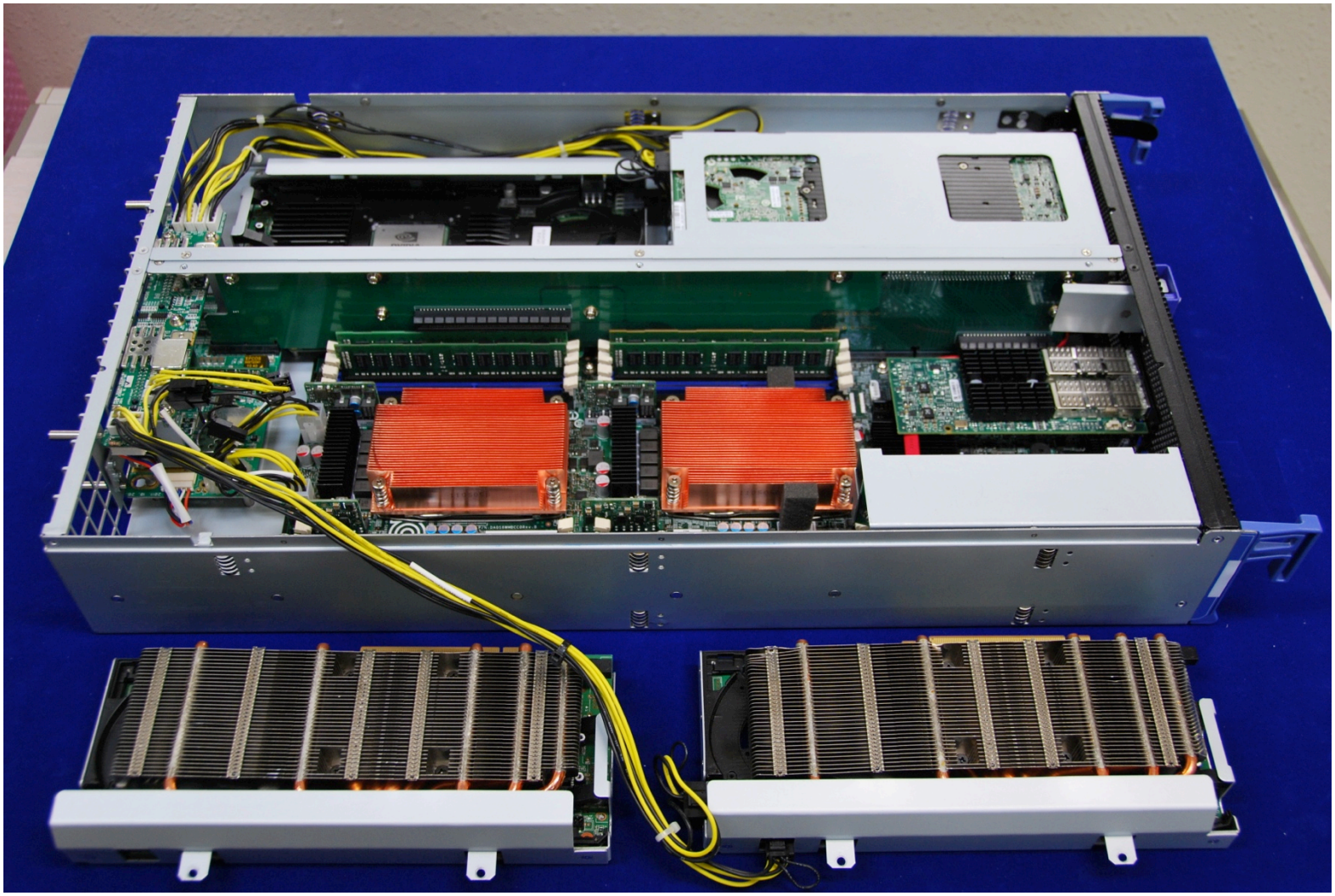
Front view of 3 blade chassis

Rear view of Infiniband switch and cables
(yellow=fibre, black=copper)

HA-PACS: base cluster (computation node)



Computation node of base cluster



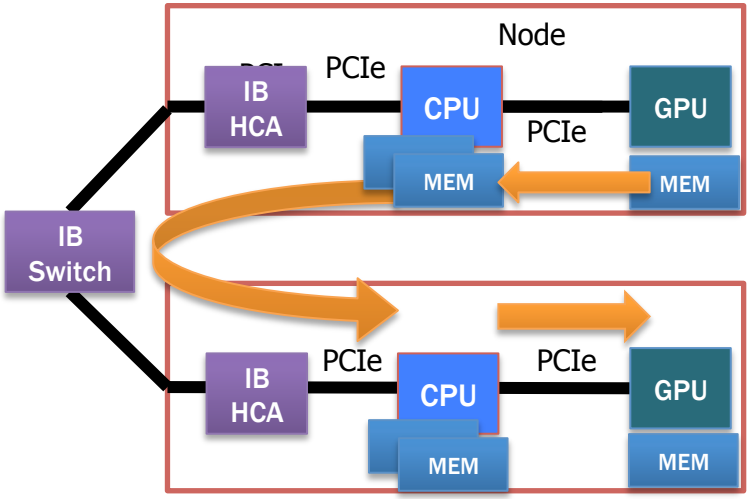
HA-PACS: TCA

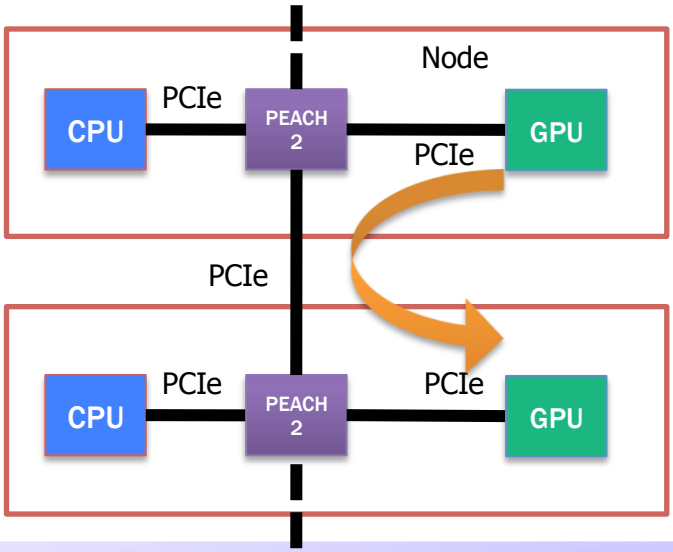
- **TCA: Tightly Coupled Accelerator**
  - Direct connection between accelerators (GPUs)
  - Using PCIe as a communication device between accelerator
    - Most acceleration device and other I/O device are connected by PCIe as PCIe end-point (slave device)
    - An intelligent PCIe device logically enables an end-point device to directly communicate with other end-point devices
- **PEACH: PCI Express Adaptive Communication Hub**
  - We already developed such PCIe device on JST-CREST project “low power and dependable network for embedded system”
  - It enables direct connection between nodes by PCIe Gen2 x4 link

⇒ **Improving PEACH for HPC to realize TCA**

HA-PACS/TCA (Tightly Coupled Accelerator)

- **True GPU-direct**
  - current GPU clusters require 3-hop communication (3-5 times memory copy)
  - For strong scaling, Inter-GPU direct communication protocol is needed for lower latency and higher throughput
- Enhanced version of PEACH
  ⇒ ***PEACH2***
  - x4 lanes -> x8 lanes
  - hardwired on main data path and PCIe interface fabric



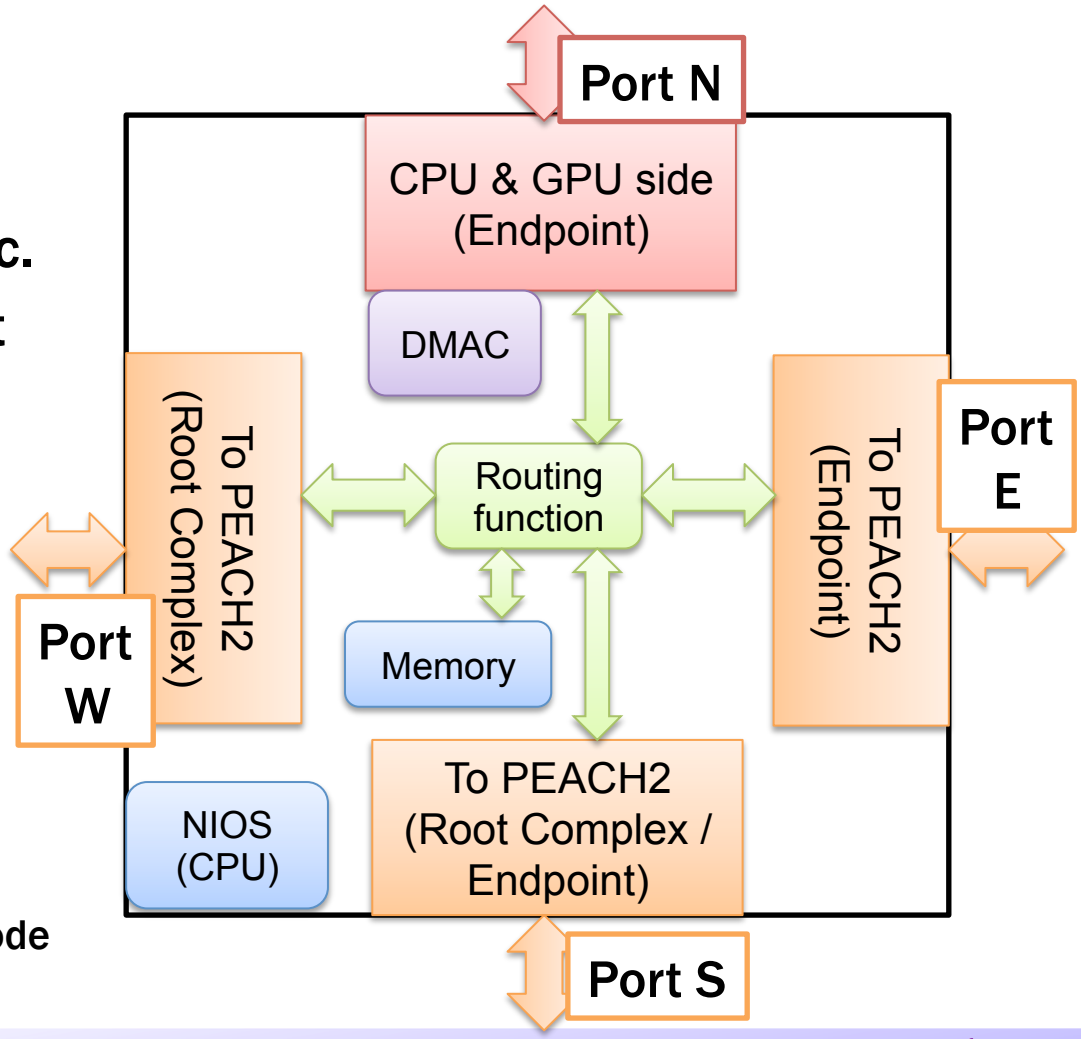


Implementation of PEACH2: FPGA solution

- **FPGA based implementation**
  - today's advanced FPGA allows to use PCIe hub with multiple ports
  - currently gen2 x 8 lanes x 4 ports are available
    ⇒ soon gen3 will be available (?)
  - easy modification and enhancement
  - fits to standard (full-size) PCIe board
  - internal multi-core general purpose CPU with programmability is available
    ⇒ easily split hardwired/firmware partitioning on certain level on control layer
- **Controlling PEACH2 for GPU communication protocol**
  - collaboration with NVIDIA for information sharing and discussion
  - based on CUDA4.0 device to device direct memory copy protocol and CUDA5.0 PCIe RDMA feature

Overview of PEACH2 chip

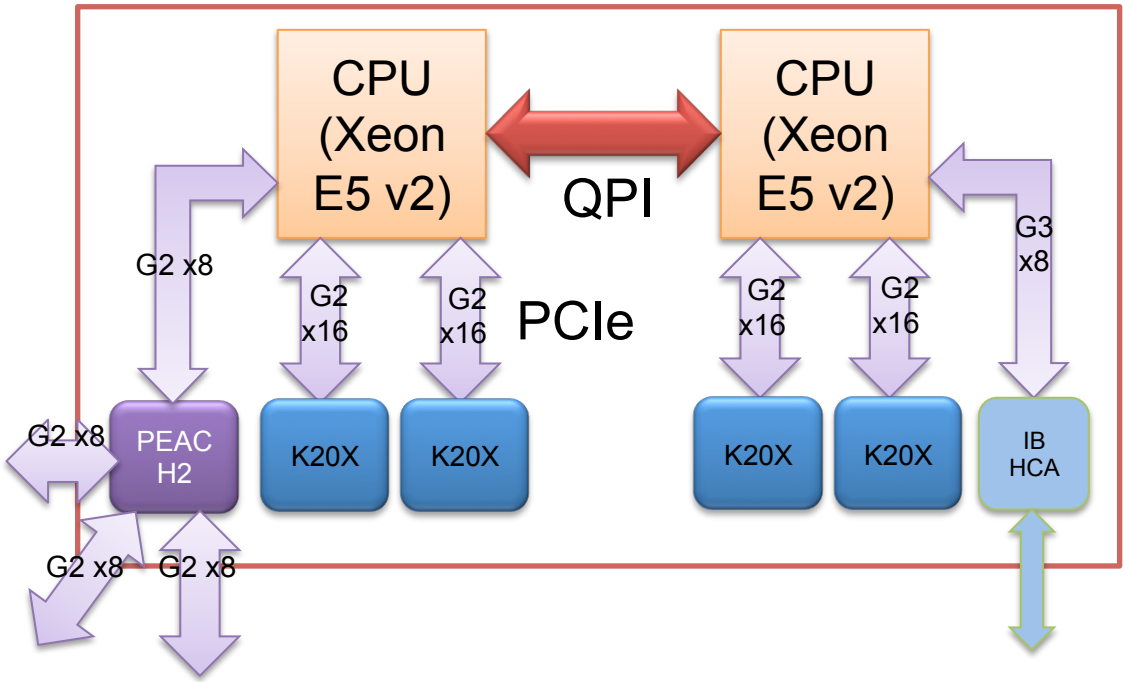
- Fully compatible with PCIe Gen2 spec.
- Root and EndPoint must be paired according to PCIe spec.
- Port N: connected to the host and GPUs
- Port E and W: form the ring topology
- Port S: connected to the other ring
  - Selectable between Root and Endpoint
- Write only except Port N
  - Instead, “Proxy write” on remote node realizes pseudo-read.



TCA node structure

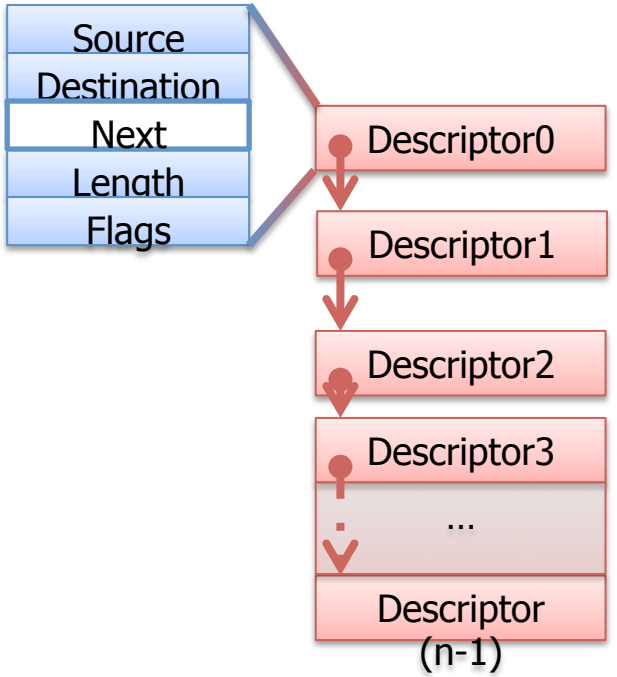
- CPU can uniformly access to GPUs.
- PEACH2 can access every GPUs
  - Kepler architecture + CUDA 5.0 “GPUDirect Support for RDMA”
  - Performance over QPI is quite bad.
    => support only for two GPUs on the same socket
- Connect among 3 nodes

- This configuration is similar to HA-PACS base cluster except PEACH2.
  - All the PCIe lanes (80 lanes) embedded in CPUs are used.



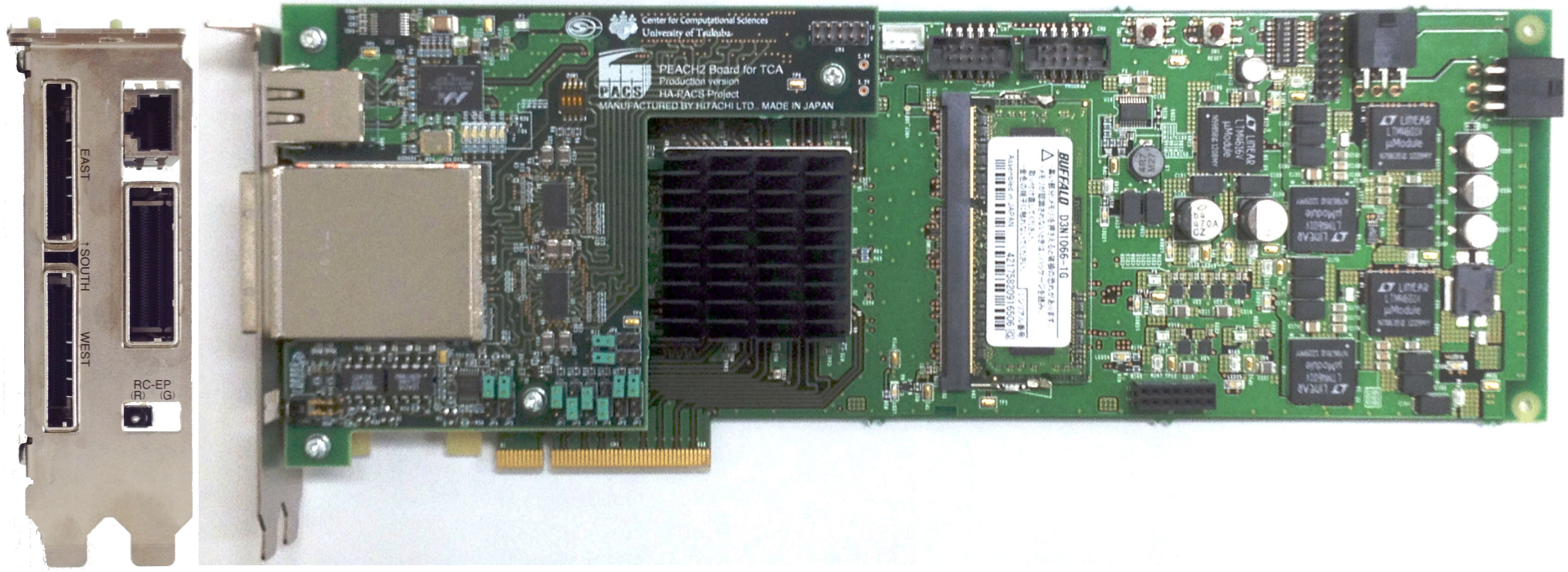
Communication by PEACH2

- PIO
  - CPU can store the data to remote node directly using mmap.
- DMA
  - Chaining mode
    - DMA requests are prepared as the DMA descriptors chained in the host memory.
    - DMA transactions are operated automatically according to the DMA descriptors by hardware.
  - Register mode
    - DMA requests are registered into the PEACH2 by up to 16.
    - Lower overhead than chaining mode by omitting transfer for descriptors from host
  - Block stride transfer function



PEACH2 board

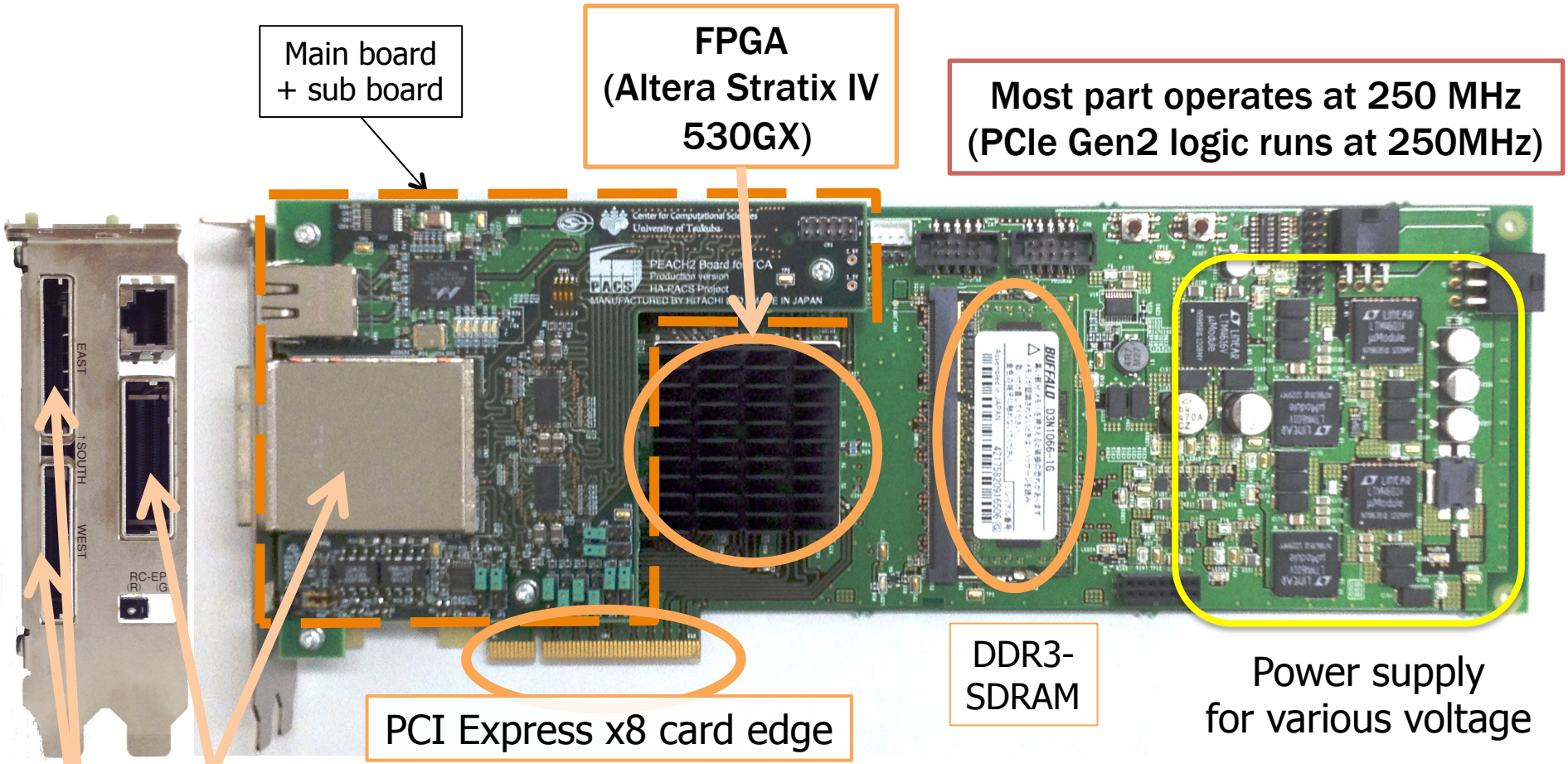
- **PCI Express Gen2 x8 peripheral board**
  - **Compatible with PCIe Spec.**



Side View

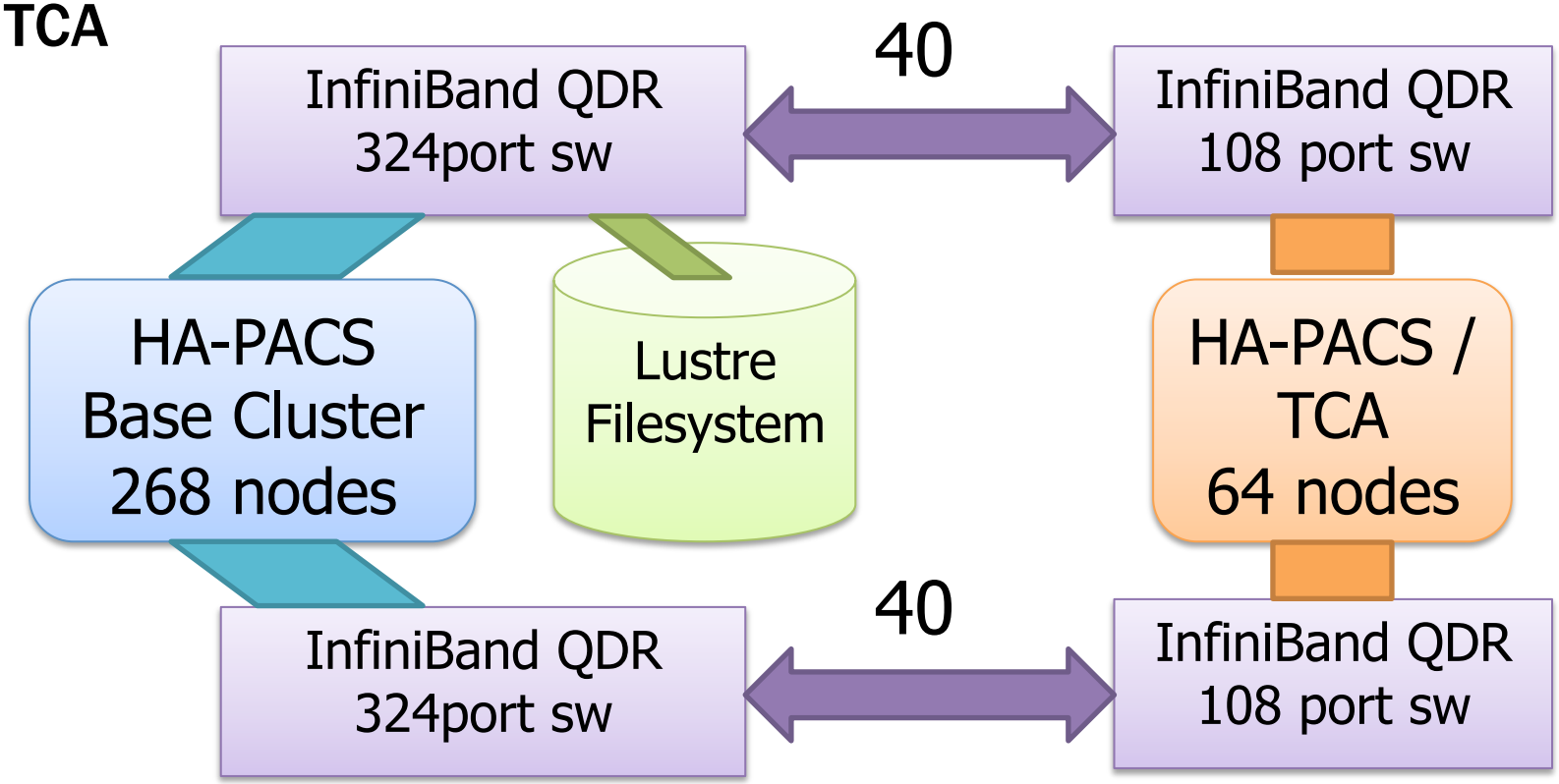
Top View

PEACH2 board

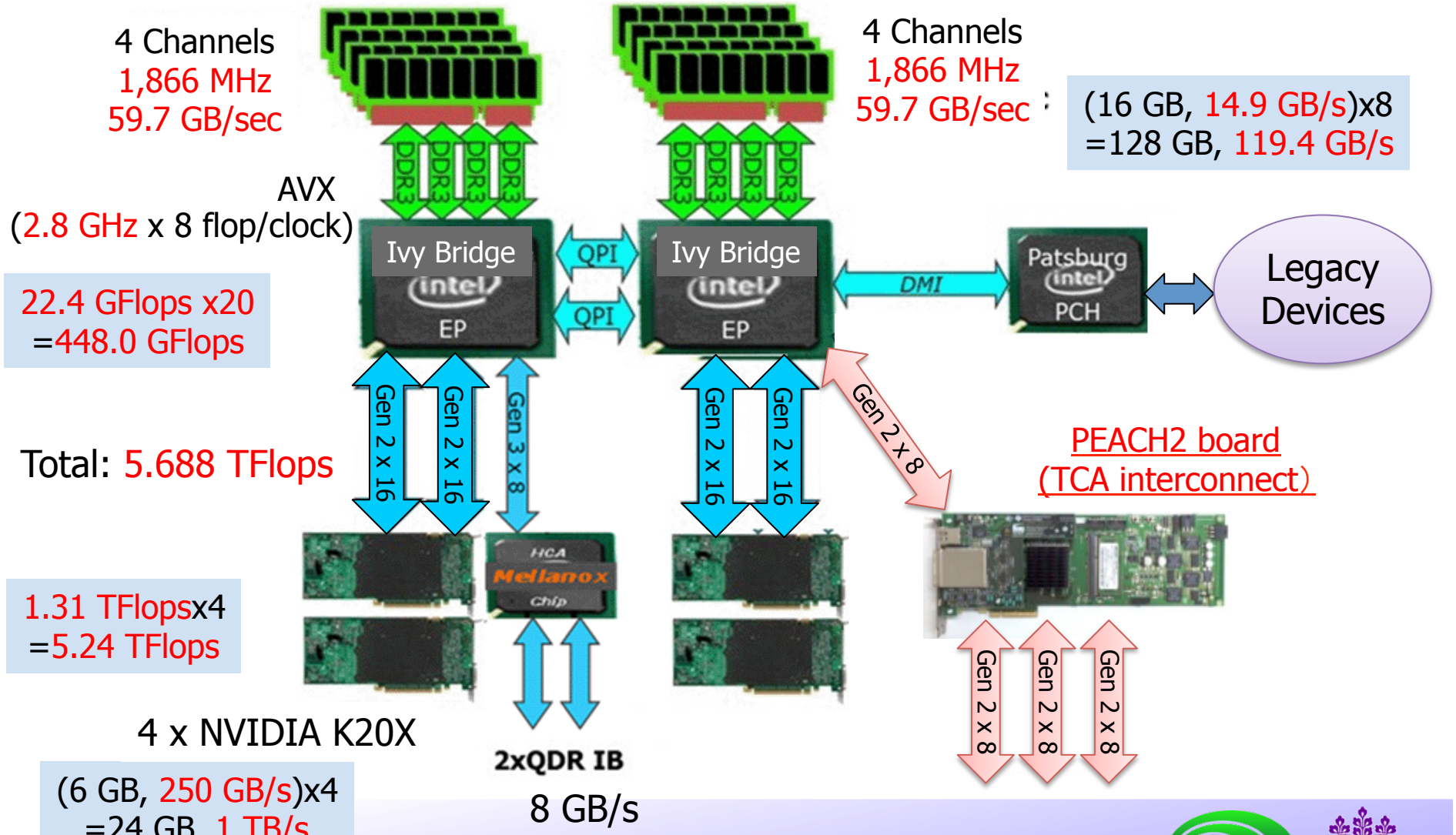


HA-PACS Total System

- **InfiniBand QDR 40port x 2ch between base cluster and TCA**



HA-PACS/TCA (computation node)



HA-PACS Base Cluster + TCA
(TCA part starts operation on Nov. 1st 2013)

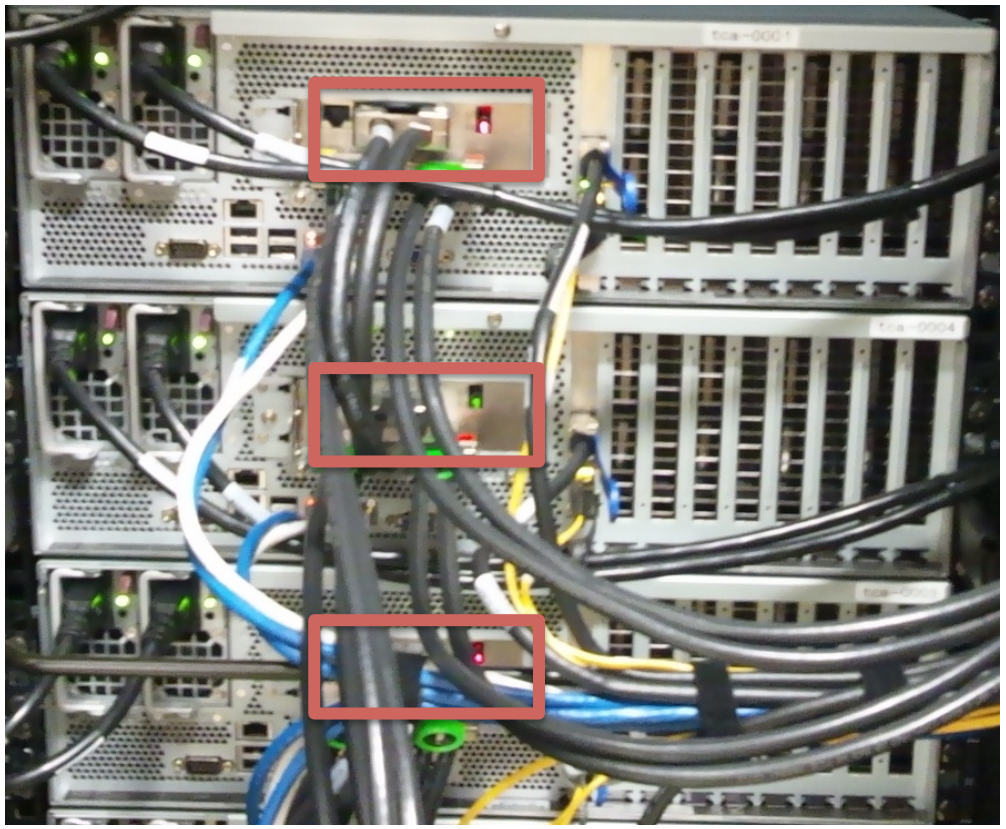
- HA-PACS Base Cluster = 2.99 TFlops x 268 node = 802 TFlops
- HA-PACS/TCA = 5.69 TFlops x 64 node = 364 TFlops
- TOTAL: 1.166 PFlops

HA-PACS/TCA computation node inside

HA-PACS/TCA

PEACH2 boards are installed with PCI-e external cables

front view
(8 node/rack)
3U height



rear view

TOP500 and Green500

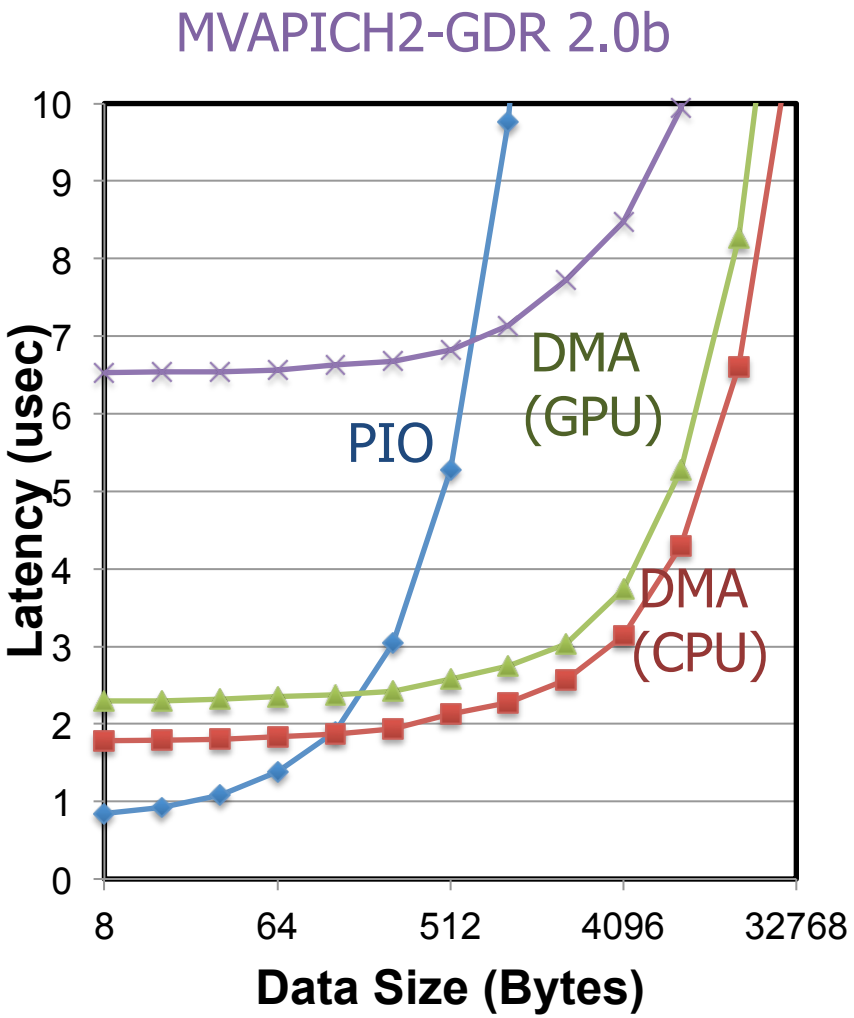
- TOP500 (HPL) on Base Cluster
  - 421.6 TFLOPS (ranked #41 in TOP500, June 2012)
  - #7 as GPU cluster
  - Computing efficiency: 54.2% of theoretical peak
- Green500 on Base Cluster
  - 1151.91 MFLOPS/W (ranked #24 in Green500, June 2012)
  - #3 as GPU cluster
  - #1 as “large scale” (within TOP50) GPU cluster
- Green500 on TCA Part (without TCA feature)
  - 3518 MFLOPS/W (ranked #3 in Green500, November 2013)
  - 76% of HPL efficiency : quite high as GPU cluster
  - ranked #134 (277 TFLOPS) in TOP500 Nov. 2013

Ping-pong Latency

Minimum Latency (nearest neighbor comm.)

- **PIO: CPU to CPU: 0.8 us**
- **DMA:CPU to CPU: 1.8 us**
  GPU to GPU: 2.3 us

cf. MV2-GDR 2.0b: 6.5 us (w/ GDR), 17 us (w/o GDR)

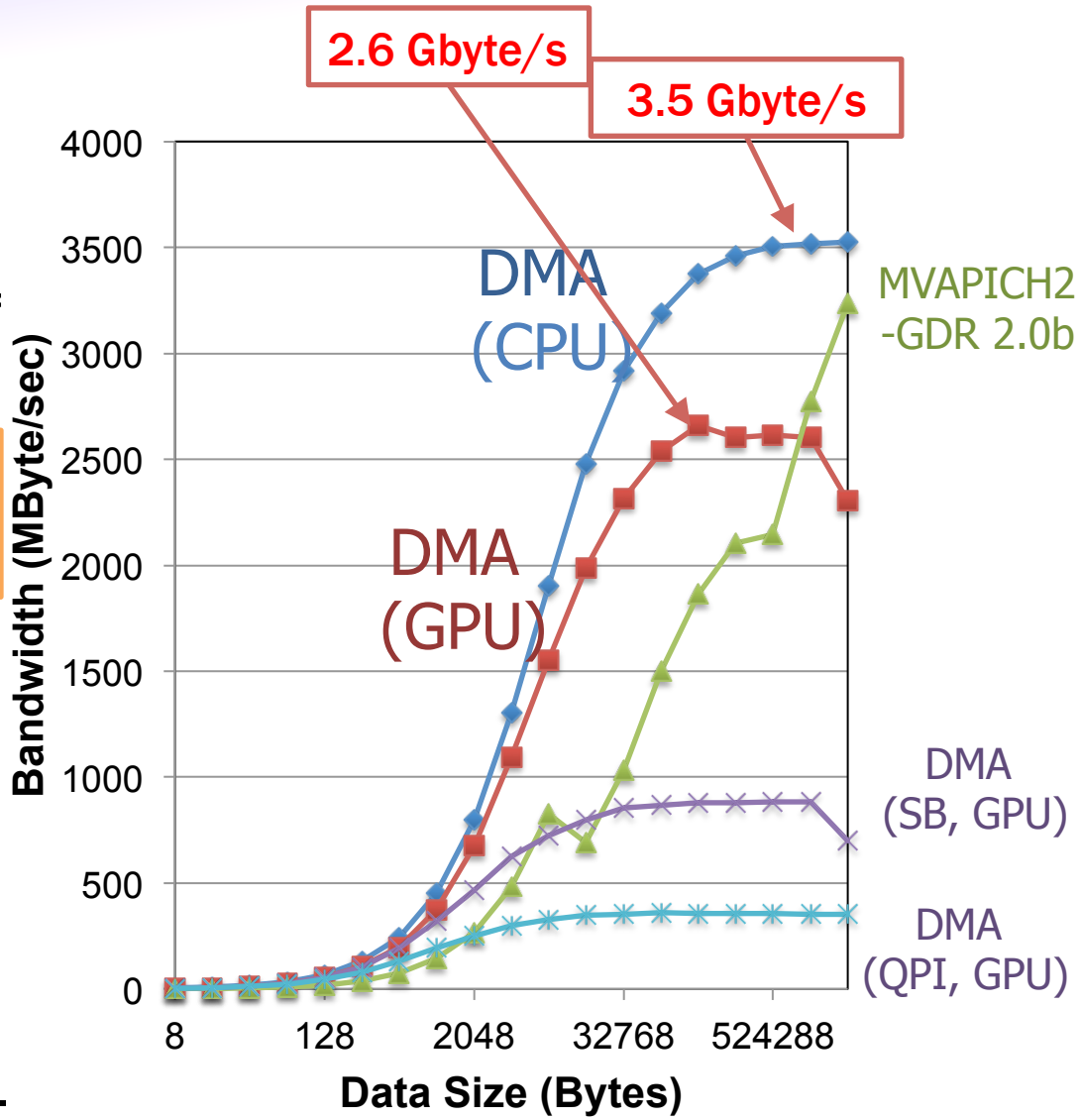


Ping-pong Bandwidth

- **Max. 3.5 GByte/sec**
  - **95% of theoretical peak**
  - **Converge to the same peak if hop count increases**

Max Payload Size = 256byte
Theoretical peak (detailed):
4GB/sec × 256 / (256 + 24) = 3.66 GB/s

- **GPU - GPU DMA performance is up to 2.6 GByte/sec.**
  - better than MV2GDR under < 1MB
  - Over QPI: limited to 360MB/s
  - SB(SandyBridge): limited to 880MB/s due to PCIe sw perf.



Ping-pong Latency

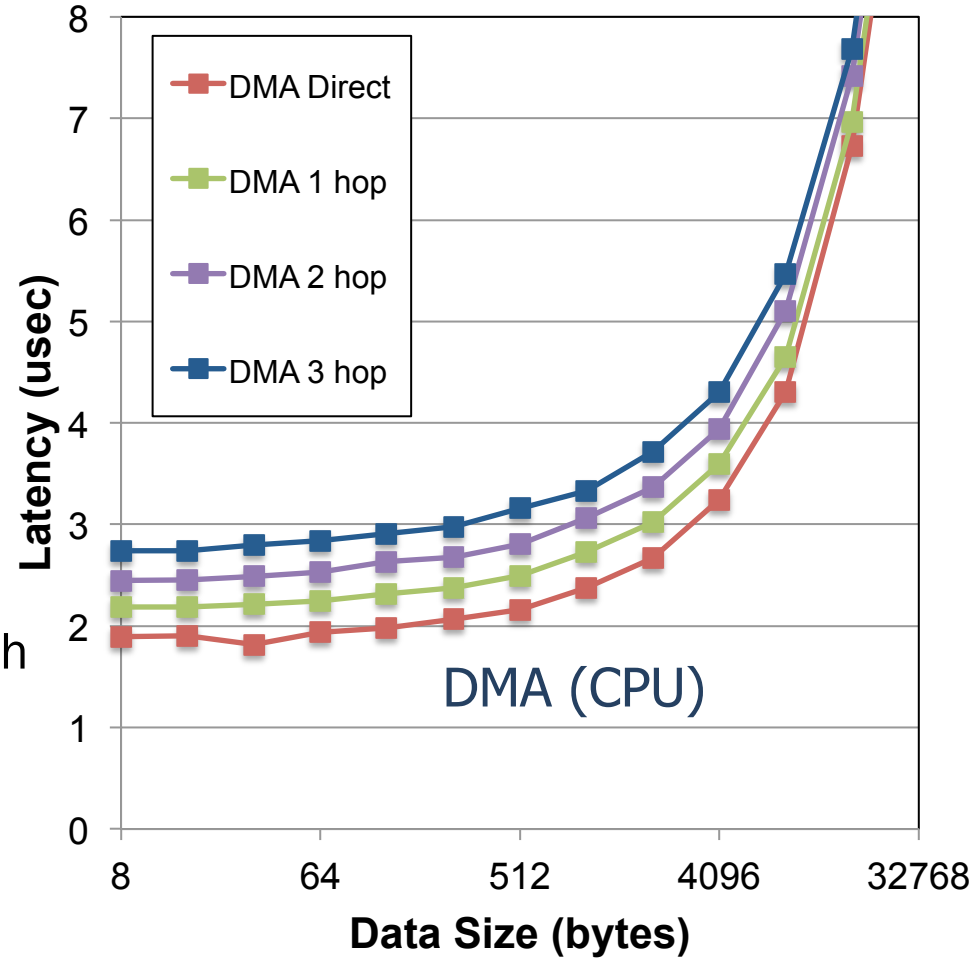
Minimum Latency

(nearest neighbor comm.)

- **PIO: CPU to CPU: 0.8 us**
- **DMA:CPU to CPU: 1.8 us**
  GPU to GPU: 2.3 us

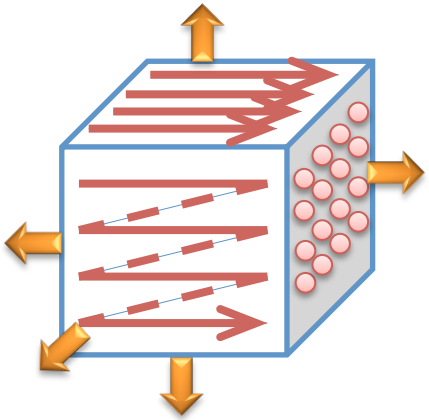
Forwarding overhead

- **200-300 nsec**
- BW converges to the same peak with various hop counts



Programming for TCA cluster

- Data transfer to remote GPU within TCA can be treated like multi-GPU in a node.
- In particular, suitable for stencil computation
  - Good performance at nearest neighbor communication due to direct network
  - Chaining DMA can bundle data transfers for every “Halo” planes
    - XY-plane: contiguous array
    - XZ-plane: block stride
    - YZ-plane: stride
  - In each iteration, DMA descriptors can be reused and only a DMA kick operation is needed



Bundle to 1 DMA

=> <u>Improve strong scaling with small data size</u>

Current activities

- Develop API for user programming
  - similar to CudaMemcpy API. It enables use GPUs in a sub cluster seamlessly as same as Multi-GPUs in a node using CudaMemcpy API.
- XMP for TCA
  - cooperating with RIKEN AICS, we develop XMP for TCA.
- Function offloading on TCA
  - a reduction mechanism between GPUs in a sub cluster will be offloaded on TCA cooperating with Keio-Univ. Amano lab. and astrophysics group in CCS
- QUDA (QCD libraries for CUDA)
  - TCA feature will be added to QUDA cooperating with NVIDIA.

COMA (PACS-IX)

- New supercomputer at CCS, starting operation on April 15th 2014
- A follow-up system after T2K-Tsukuba
- System configuration
  - Computation node: general CPU + many-core (MIC)
  - Node configuration
    - CPU x 2: Intel Xeon E5-2670v2
    - MIC x 2: Intel Xeon Phi 7110P
    - Memory: CPU=64GB MIC=16GB
    - Network: IB FDR Full-bisection b/w Fat Tree
  - # of nodes: 393
  - peak perf.: CPU=157.2 TFlops MIC=843.8 TFlops
    TOTAL: 1001 TFlops = **1.001 PFLOPS**
- System vendor: Cray Inc.

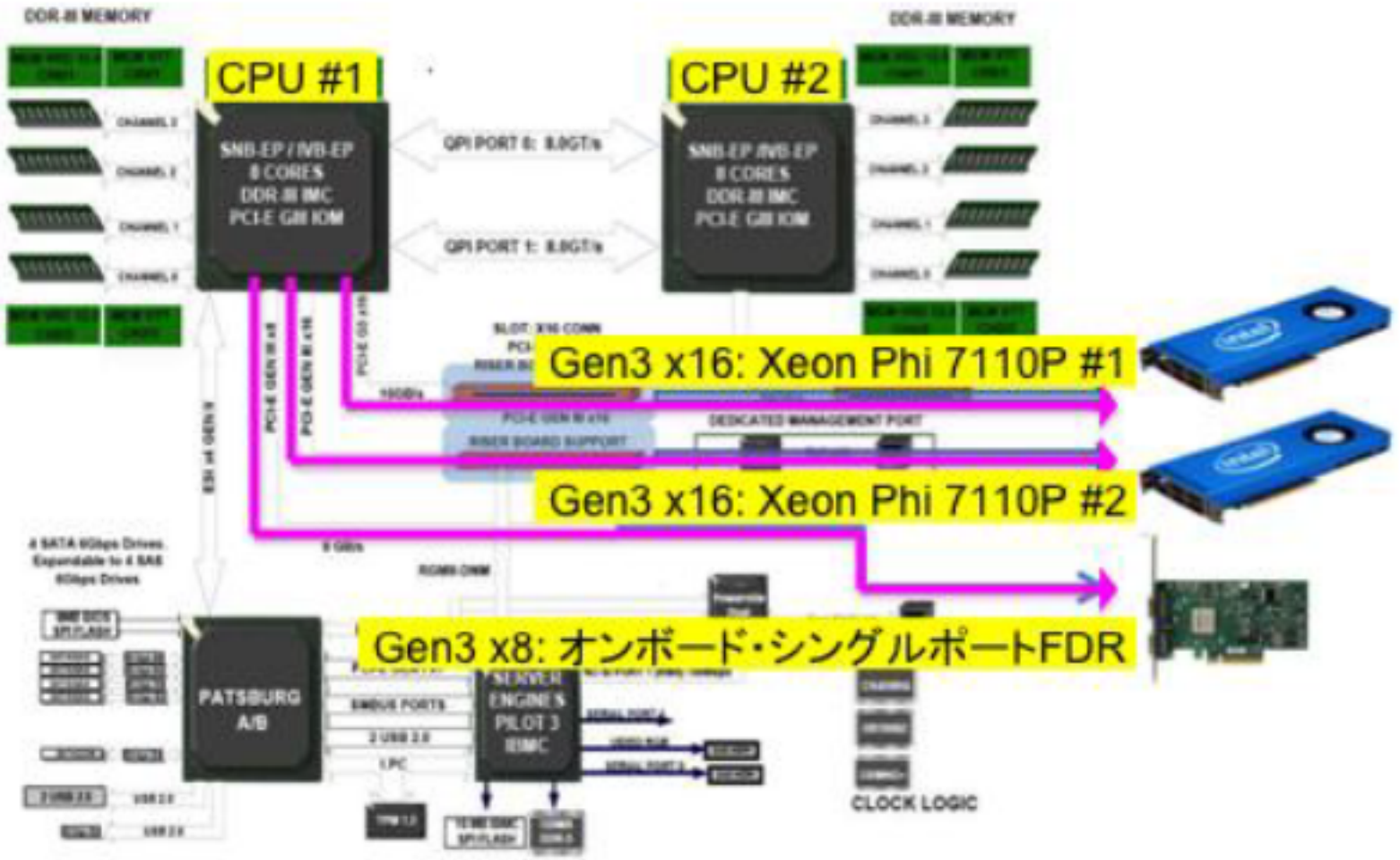
What is COMA ?

- Cluster of Many-core Architecture processor
- COMA
  - A famous cluster of galaxies
  - galaxy = cluster of stars (= Many Core)
  - cluster of galaxies = cluster of many-core
- It is also the 9th generation machine of PACS/PAX series
  -> PACS-IX as project series number


Computation node of COMA

- CPU (2 sockets): Intel Xeon E5-2670v2 (Ivy Bridge)
  - 10 core/CPU, 2.5GHz
  - 200GFLOSP x 2 = 400GFLOPS
  - memory: 64GB
    DDR3 1866MHz x 4chan x 2CPU = 119.4 GB/s
- MIC (2 boards): Intel Xeon Phi 7110P
  - 61 core/MIC, 1.1GHz
  - 1.0736 TFLOPS x 2 = 2.1472 TFLOPS
  - memory: 8GB x 2 = 16GB
    GDR5 352GB/s x 2 = 704 GB/s
- Interconnection: InfiniBand FDR (on-board)
  - Mellanox Connect-X3
- Local HDD: 1TB x 2 (RAID-1)

COMA node block diagram



COMA system facts

- # of nodes: 393
  - Peak performance
    - CPU: 157.2 TFLOPS
    - MIC: 843.8 TFLOPS
    - TOTAL: 1001 TFLOPS = 1.001 PFLOPS
  - Memory capacity
    - CPU: 25.1 TB
    - MIC: 6.3 TB
- Interconnection: Fat-Tree full-bisection b/w
  - Bisection bandwidth: 2.75 TB/s
- Shared file system: Lustre
  - 1.5 PB of shared file system directly accessed by all nodes via InfiniBand FDR
  - Data Direct Network, Lustre, RAID-6


Software

- OS: Red Hat Enterprise Linux (login server)
- OS: CentOS (compute node)
- Cluster management: ACE
- Job scheduler: SLURM
- Programming environment:
  - Intel Cluster Studio XE2013 (Composer/XE)
  - Fortran95/C/C++
  - Intel MPI


Programming of MIC

- MIC (KNC) runs x86-based Linux on itself
- Three programming models
  - Native Mode: running program on Linux on MIC directly
    - OpenMP (,etc.) multi-threaded application
    - Up to 240 of threads on 60 of cores (up to 4 h/w threads / core)
    - No I/O, mounting host node's HDD on NFS
  - Offload Mode: running partial code of host to offload on MIC
    - supported by Composer XE (Intel compiler for MIC)
    - "device" directive shows the part to be offload
    - also describe OpenMP on offload-part to utilize parallelism on each MIC
    - host code may be executed with MPI also
  - Symmetric Mode: flat-model to treat all cores on host and MIC as the same to run big MPI program, even over multiple nodes