Tightly Coupled Accelerators with Proprietary Interconnect and Its Programming and Applications

Toshihiro Hanawa
Information Technology Center, The University of Tokyo
Taisuke Boku
Center for Computational Sciences, University of Tsukuba

Collaboration with
Yuetsu Kodama, Mitsuhisa Sato, Masayuki Umemura @ CCS, Univ. of Tsukuba
Hitoshi Murai @ Riken AICS, Hideharu Amano @ Keio Univ.

Agenda

- Background
- HA-PACS / AC-Crest Project
- Introduction of HA-PACS / TCA
  - Organization of TCA
  - PEACH2 Board designed for TCA
- Evaluation of Basic Performance
- Collective Communications
  - Implementation Examples
  - Performance Evaluation
- Application Examples
  - QUDA (QCD)
  - FFTE (FFT)
- Introduction of XcalableACC
  - Concept
  - Code Examples
  - Evaluations
- Summary

Current Trend of HPC using GPU Computing

- Advantageous Features
  - High peak performance / cost ratio
  - High peak performance / power ratio
- Examples of HPC System:
  - GPU Clusters and MPPs in TOP500 (Nov. 2014)
    - 2nd: Titan (NVIDIA K20X, 27 PF)
    - 6th: Piz Daint (NVIDIA K20X, 7.8 PF)
    - 10th: Cray CS-Storm (NVIDIA K40, 6.1 PF)
    - 15th: TSUBAME2.5 (NVIDIA K20X, 5.6 PF)
    - 48 systems use NVIDIA GPUs.
  - GPU Clusters in Green500 (Nov. 2014) (“Greenest” Supercomputers ranked in Top500)
    - 3rd: TSUBAME-KFC (NVIDIA K20X, 4.4 GF/W)
    - 4th: Cray Storm1 (NVIDIA K40, 3.9 GF / W)
    - 7th: HA-PACS/TCA (NVIDIA K20X, 3.5 GF/W)
    - 8 systems of Top10 use NVIDIA GPUs.

Issues of GPU Computing

- **Data I/O performance limitation**
  - Ex) K20X: PCIe gen2 x16
    Peak Performance: 8GB/s (I/O) ⇔ 1.3 TFLOPS (Computation)
  - Communication bottleneck becomes significant on multi GPU application
- **Strong-scaling on GPU cluster**
  - Important to shorten Turn-Around Time of production-run
  - Heavy impact of communication latency
- **Ultra-low latency between GPUs is important for next generation's HPC**

Our target is developing a direct communication system between external GPUs for a feasibility study for future accelerated computing.
⇒ "Tightly Coupled Accelerators (TCA)" architecture

HA-PACS Project

- HA-PACS (Highly Accelerated Parallel Advanced system for Computational Sciences)
  - 8th generation of PAX/PACS series supercomputer in University of Tsukuba
  - FY2011-2013, operation until FY2016(?)
- Promotion of computational science applications in key areas in CCS-Tsukuba
  - Target field: QCD, astrophysics, QM/MM (quantum mechanics / molecular mechanics, bioscience)

HA-PACS is not only a "commodity GPU cluster" but also experiment platform

- HA-PACS *base cluster*
  - for development of GPU-accelerated code for target fields, and performing product-run
  - Now in operation since Feb. 2012
- HA-PACS/TCA (TCA = Tightly Coupled Accelerators)
  - for elementary research on direct communication technology for accelerated computing
  - Our original communication chip named "PEACH2" was installed in each node.
  - Now in operation since Nov. 2013

AC-CREST project

- Project "**Research and Development on Unified Environment of Accelerated Computing and Interconnection for Post-Petascale Era**" (**AC-CREST**)
  - Supported by JST-CREST "**Development of System Software Technologies for post-Peta Scale High Performance Computing**" program

- **Objectives**
  - Realization of high-performance (direct) communication among accelerators
  - Development of system software supporting communication system among accelerators
  - Development of parallel language and compilers
    - Higher productivity
    - Highly optimized (offload, communication)
  - Development of practical applications

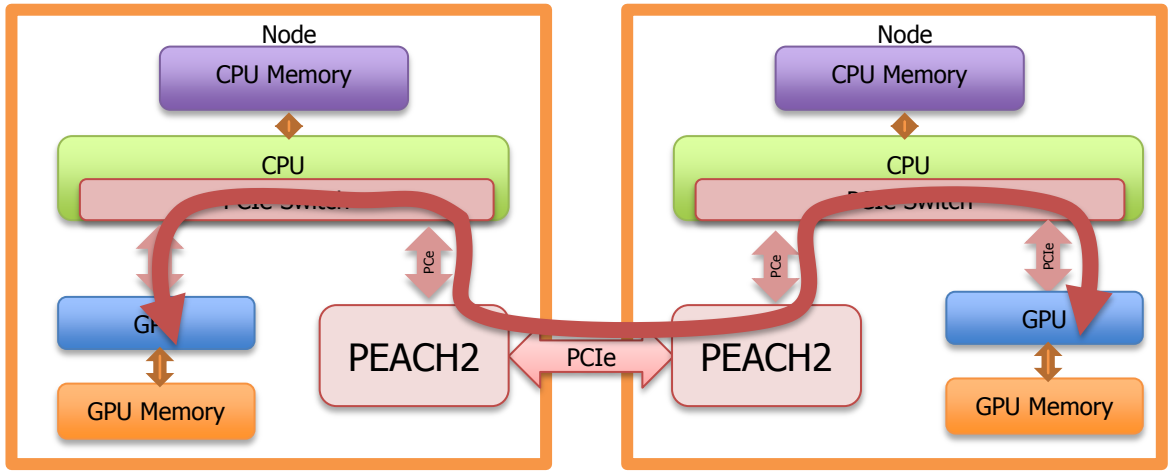
What is “Tightly Coupled Accelerators (TCA)” ?

Concept:

- Direct connection between accelerators (GPUs) over the nodes without CPU assistance
  - Eliminate extra memory copies to the host
  - Reduce latency, improve strong scaling with small data size
  - Enable hardware support for complicated communication patterns

Communication on TCA Architecture

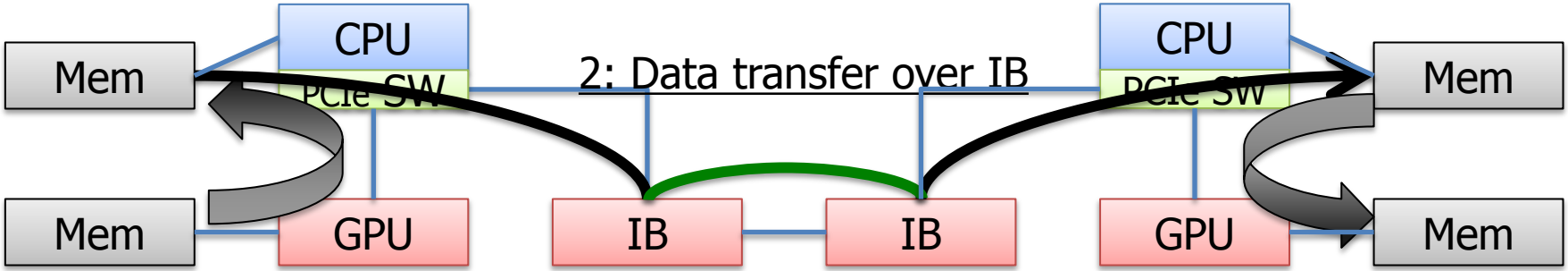
- **Using PCIe as a communication link between accelerators over the nodes**
  - Direct device P2P communication is available thru PCIe.



- **PEACH2:**
  PCI Express Adaptive Communication Hub ver. 2
  - Implementation of the interface and data transfer engine for TCA

GPU Communication with traditional MPI

- Traditional MPI using InfiniBand requires data copy 3 times
  - Data copy between CPU and GPU (1 and 3) have to perform manually

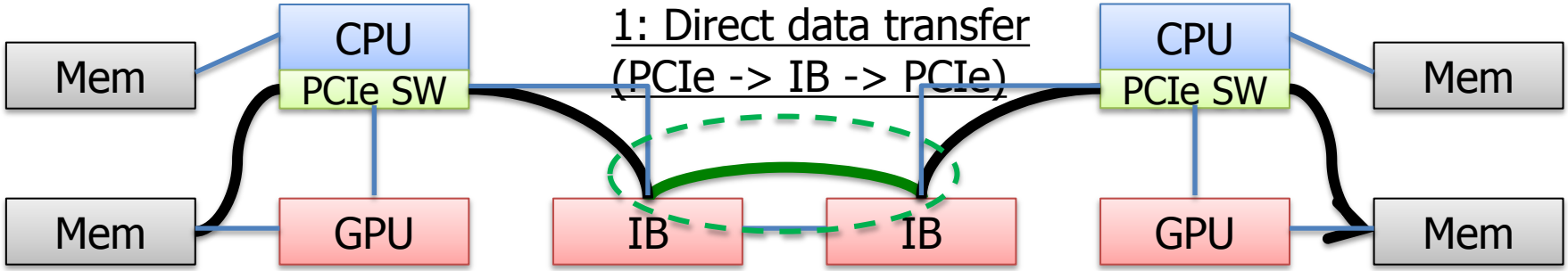


1: Copy from GPU mem to CPU mem through PCI Express (PCIe)

3: Copy from CPU mem to GPU mem through PCIe

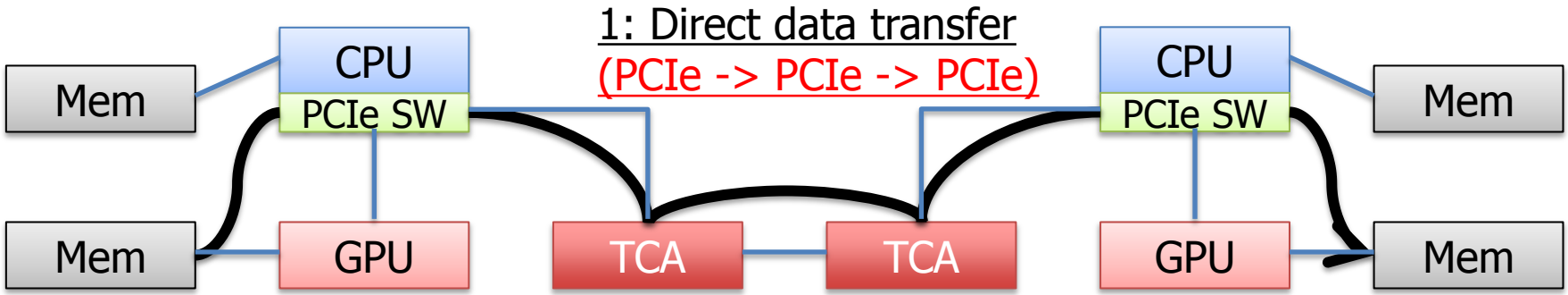
GPU Communication with IB/GDR

- The InfiniBand controller read and write GPU memory directly (with GDR)
  - Temporal data copy is eliminated
  - Lower latency than the previous method
  - Protocol conversion is still needed



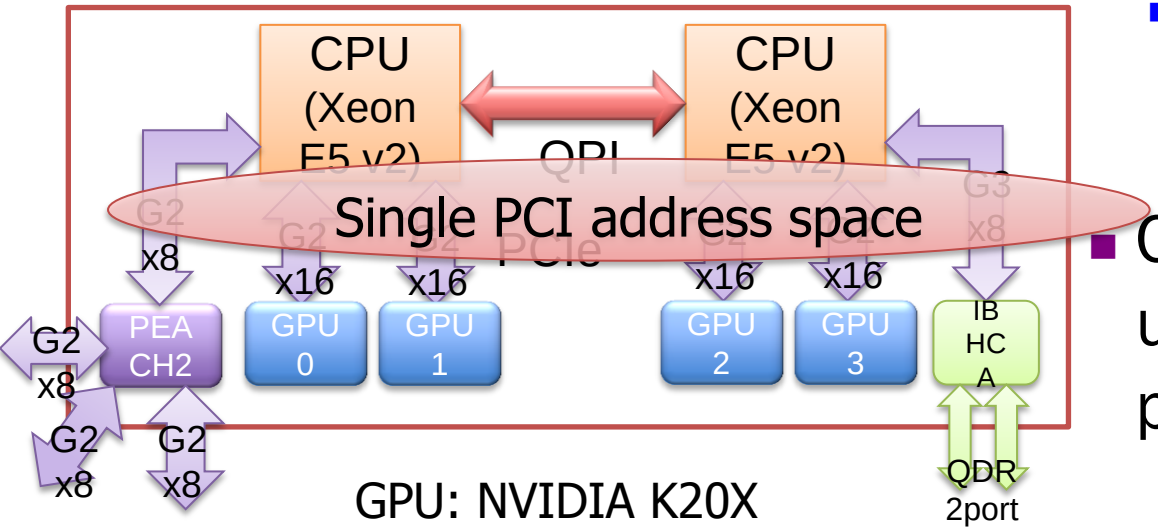
GPU Communication with TCA (PEACH2)

- TCA does not need protocol conversion
  - direct data copy using GDR
  - much lower latency than InfiniBand



TCA node structure example

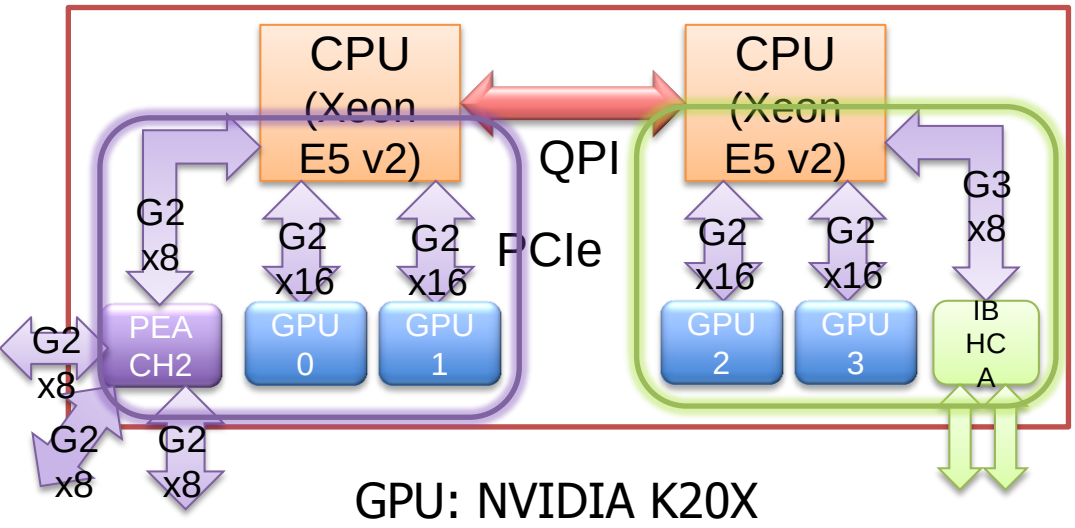
- Similar to ordinary GPU cluster configuration except PEACH2
- 80 PCIe lanes are required



- PEACH2 can access all GPUs
  - **NVIDIA Kepler architecture + "GPUDirect Support for RDMA" are required.**
- Connect among 3 nodes using remaining PEACH2 port

TCA node structure example

- Similar to ordinary GPU cluster configuration except PEACH2
- 80 PCIe lanes are required



Actually,

- **Performance over QPI is miserable.**

- **PEACH2** is available **for <u>GPU0, GPU1.</u>**
- Note that <u>InfiniBand with GPU Direct</u> for RDMA is available only for <u>GPU2, GPU3.</u>

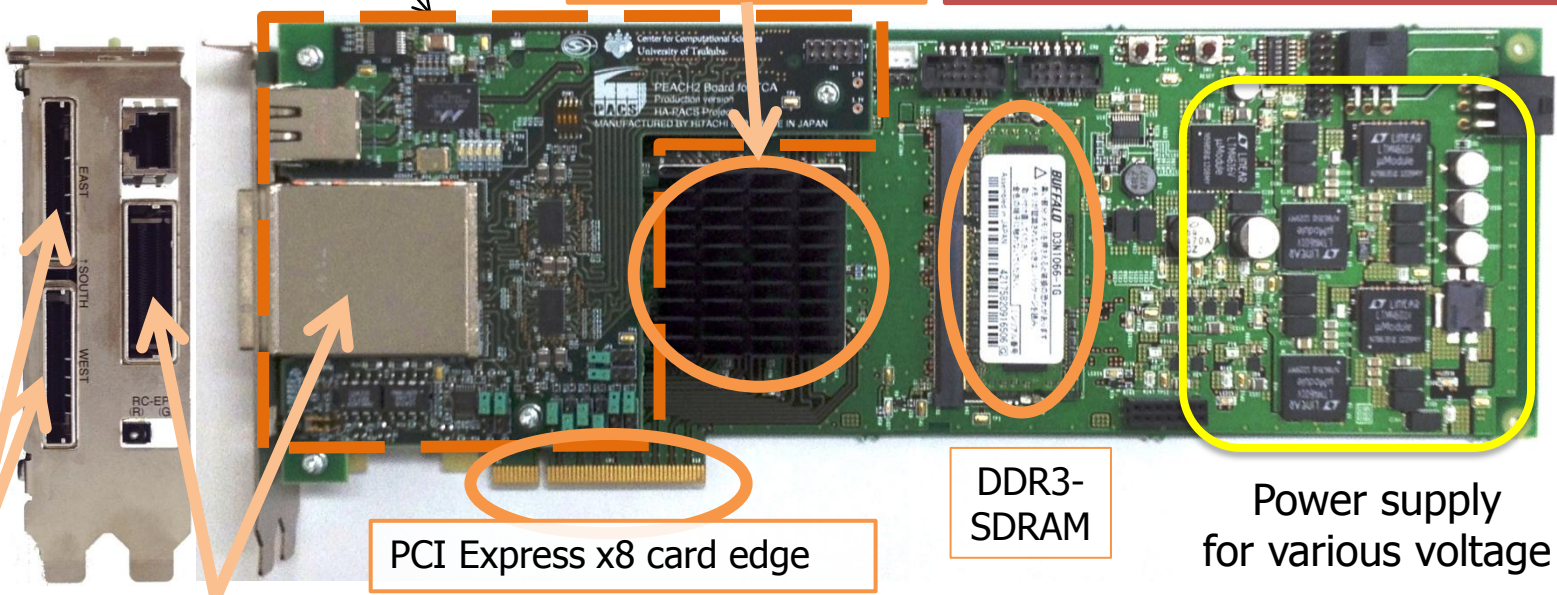
Design of PEACH2

- Implement by FPGA with four PCIe Gen.2 IPs
  - Altera Stratix IV GX
  - Prototyping, flexible enhancement
- Sufficient communication bandwidth
  - PCI Express Gen2 x8 for each port (40Gbps = IB QDR)
  - Sophisticated DMA controller
    - Chaining DMA, Block-stride transfer function

- Latency reduction
  - Hardwired logic
  - Low-overhead routing mechanism
    - Efficient address mapping in PCIe address area using unused bits
    - Simple comparator for decision of output port

It is not only a proof-of-concept implementation, but it will also be available for product-run in GPU cluster.

PEACH2 board (Production version for HA-PACS/TCA)



Main board + sub board

FPGA (Altera Stratix IV 530GX)

Most part operates at 250 MHz (PCIe Gen2 logic runs at 250MHz)

DDR3-SDRAM

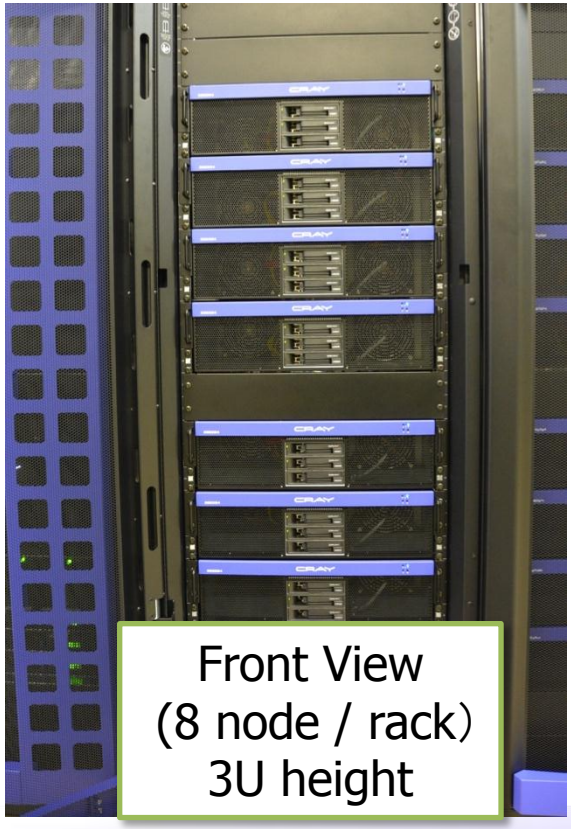
Power supply for various voltage

PCI Express x8 card edge

PCIe x8 cable connecter

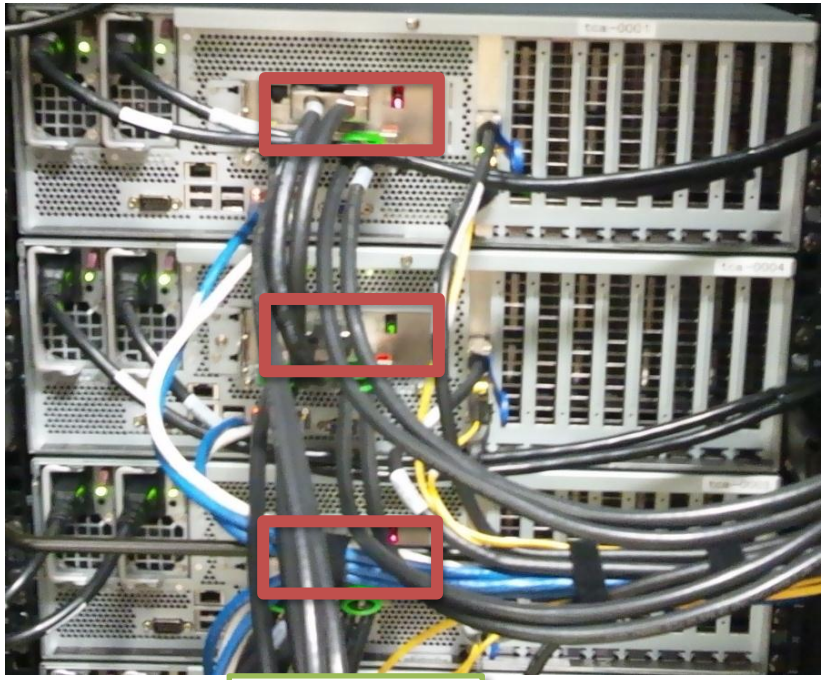
PCIe x16 cable connecter

HA-PACS/TCA Compute Node



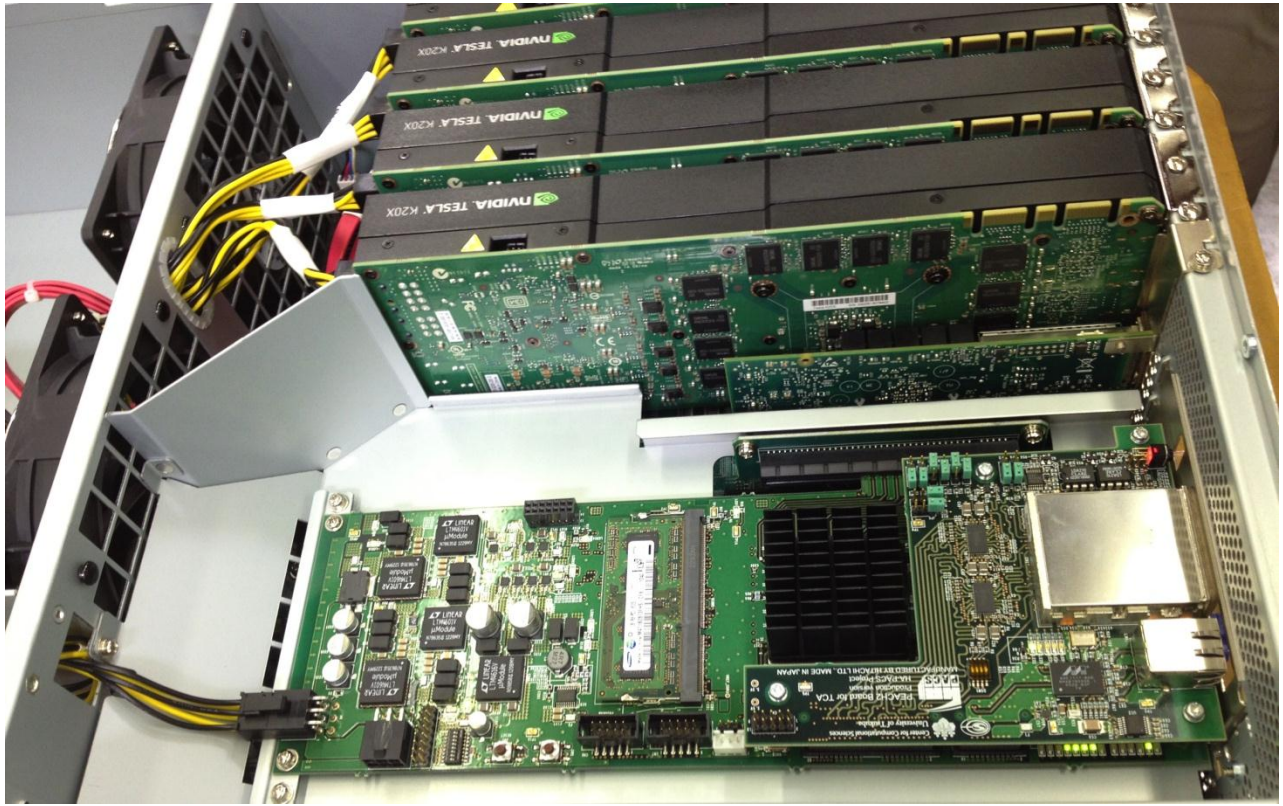
Front View
(8 node / rack)
3U height

PEACH2 Board is installed here!



Rear View

Inside of HA-PACS/TCA Compute Node



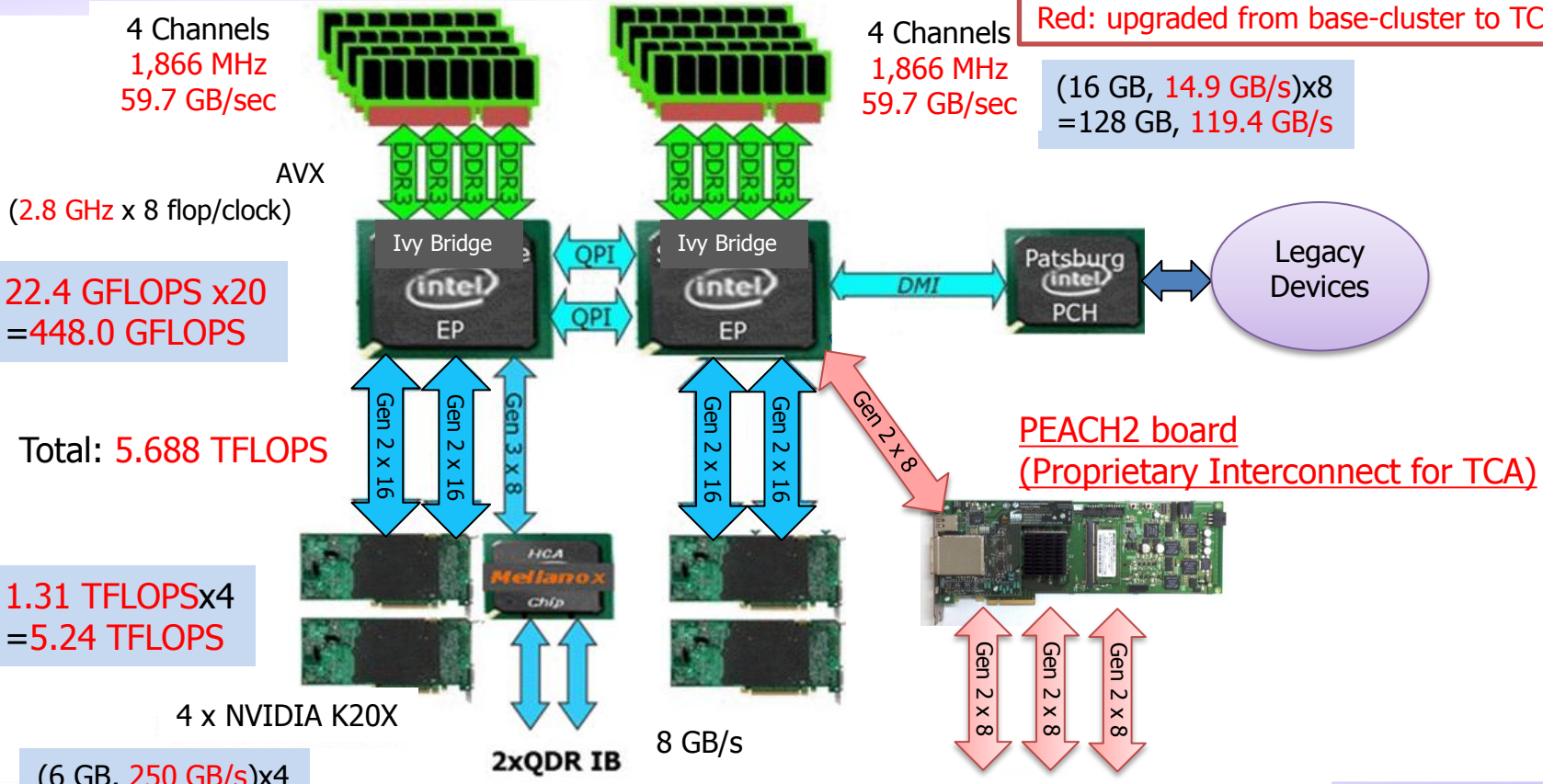
Spec. of HA-PACS base cluster & HA-PACS/TCA

	Base cluster (Feb. 2012)	TCA (Nov. 2013)
Node	CRAY GreenBlade 8204	CRAY 3623G4-SM
MotherBoard	Intel Washington Pass	SuperMicro X9DRG-QF
CPU	Intel Xeon E5-2670 x 2 socket (SandyBridge-EP, 2.6GHz 8 core) x2	Intel Xeon E5-2680 v2 x 2 socket (IvyBridge-EP, 2.8GHz 10 core) x2
Memory	DDR3-1600 128 GB	DDR3-1866 128 GB
GPU	NVIDIA M2090 x4	NVIDIA K20X x 4
# of Nodes (Racks)	268 (26)	64 (10)
Interconnect	Mellanox InfiniBand QDR x2 (Connect X-3)	Mellanox InfiniBand QDR x2 + PEACH2
Peak Perf.	802 TFlops	364 TFlops
Power	408 kW	99.3 kW

Totally, HA-PACS is over 1PFlops system !

HA-PACS/TCA (Compute Node)

Red: upgraded from base-cluster to TCA



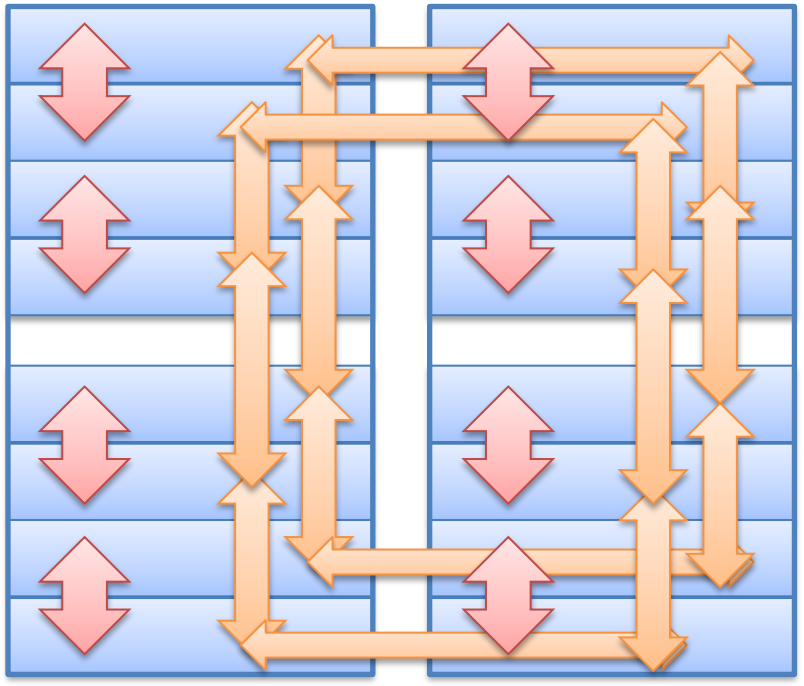
HA-PACS/TCA (since Nov. 2013) + Base cluster

LINPACK: 277 Tflops (Efficiency 76%)

3.52GFLOPS/W #3 Green500 at Nov. 2013

Configuration of TCA Sub-cluster (16 nodes/group)

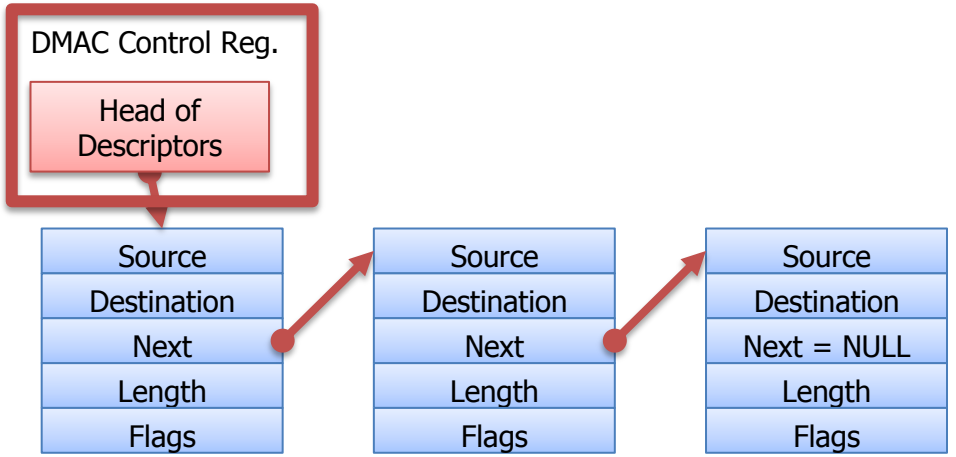
- Each group consists of 2 racks, 16nodes. HA-PACS/TCA includes 4 TCA groups.
  - Orange: Ring
  - Red: Cross link between 2 rings
- In TCA sub-cluster, 32 GPUs can be treated seamlessly.
  - limited to 2 GPUs under the same socket per node



Communication on TCA

- TCA provides two types of communications.
- **DMA controller function**
  - Chaining
    - Multiple DMA descriptors chained on memory
    - DMA transactions are automatically operated by HW
  - Block-stride support
  - DMA Engine with 4ch

Comm. type	Min. Latency	Band width	How working	Comm. patterns
DMA	Low (< 2us)	High	DMA controller	Any CPU or GPU
PIO	Very low (< 1us)	Low	CPU's write operation	CPU-CPU



Evaluation Results

- Ping-pong performance between nodes
  - Latency and bandwidth
  - Written as application
- Comparison
  - MVAPICH2-GDR 2.0b (with/without GPU Direct support) for GPU-GPU communication on TCA nodes
    - A InfiniBand QDR link (40Gbps) is used, which has the same performance as PEACH2.
  - Performance over QPI on TCA nodes

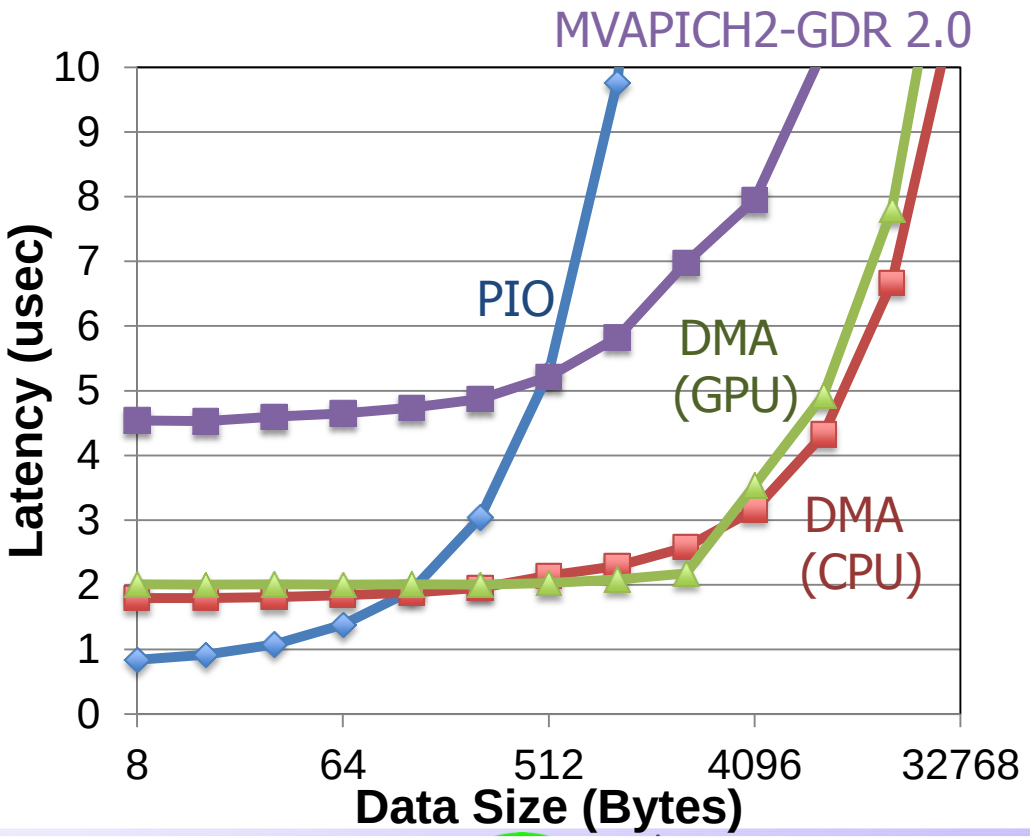
- In order to access GPU memory by the other device, “GPU Direct support for RDMA” in CUDA5 API is used.
  - Special driver named “TCA p2p driver” to enable memory mapping is developed.
- “PEACH2 driver” to control the board is also developed.

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: 2.0 us

cf. MV2-GDR 2.0: 4.5 us (w/ GDR), 17 us (w/o GDR)



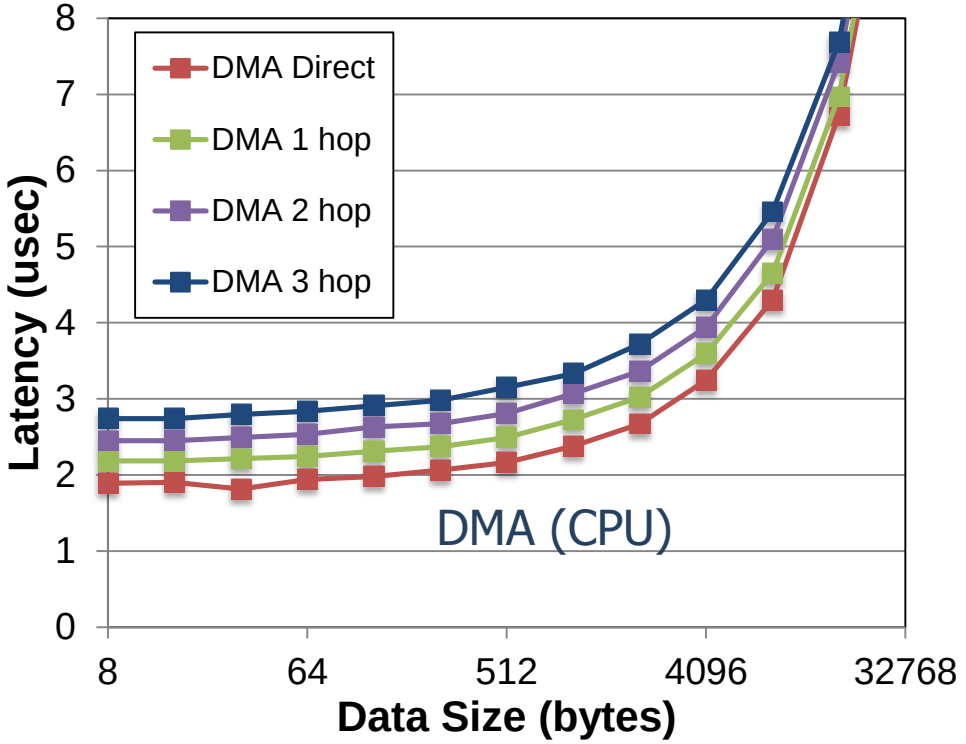
Ping-pong Latency

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

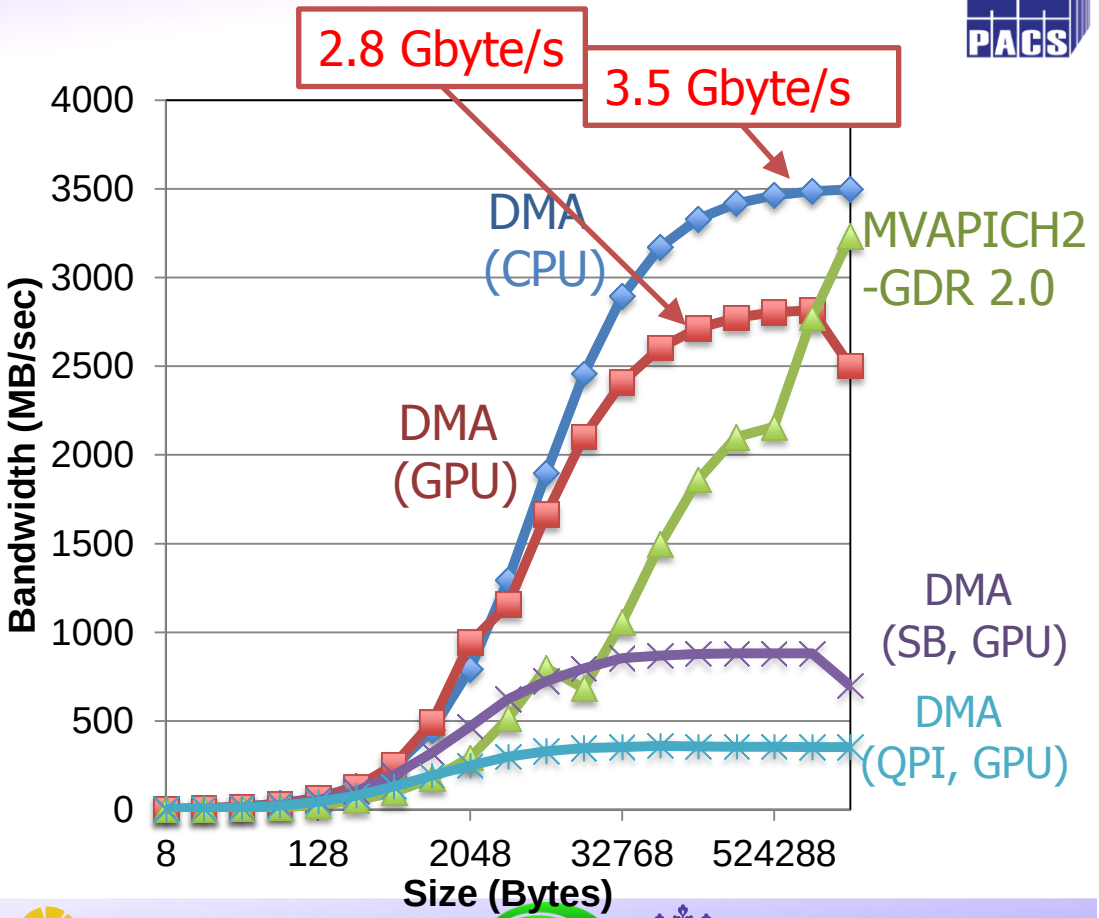


Ping-pong Bandwidth

- 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 2.8 GByte/sec.
  - better than MV2GDR under < 1MB
  - Over QPI: limited to 360MB/s
  - SB(SandyBridge): limited to 880MB/s due to PCIe sw perf.



GPUDirect behavior in MVAPICH2-GDR

From README

- MV2_GPUDIRECT_LIMIT
  * Default: 8192
- MV2_USE_GPUDIRECT_RECEIVE_LIMIT
  * Default: 131072

- X <= 8KB:
  GDR read + GDR write
- 8KB < X <= 128KB:
  memcpy H2D + GDR write
- X > 128KB
  memcpy H2D + memcpy D2H

Collective Communications

- Allgather
  - All processes gather data of each process.
  - Communication bandwidth as well as latency is important.
  - GPU-GPU DMA
- Allreduce
  - Conduct specified operation among data arrays on each process and store the results on all processes.
  - Latency decides the performance.
  - CPU-CPU PIO with host copy
- Alltoall
  - All processes exchange specific data of each process (transpose).
  - Communication bandwidth is important.
  - GPU-GPU DMA, all requests to every nodes are chained

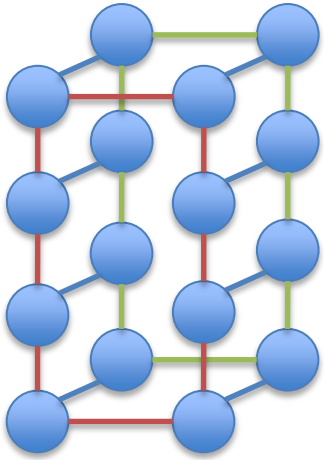
Packet contention might occur on ring, optimization should be required.

[AsHES2015] (To be appeared)

Allgather Implementation: Recursive Doubling

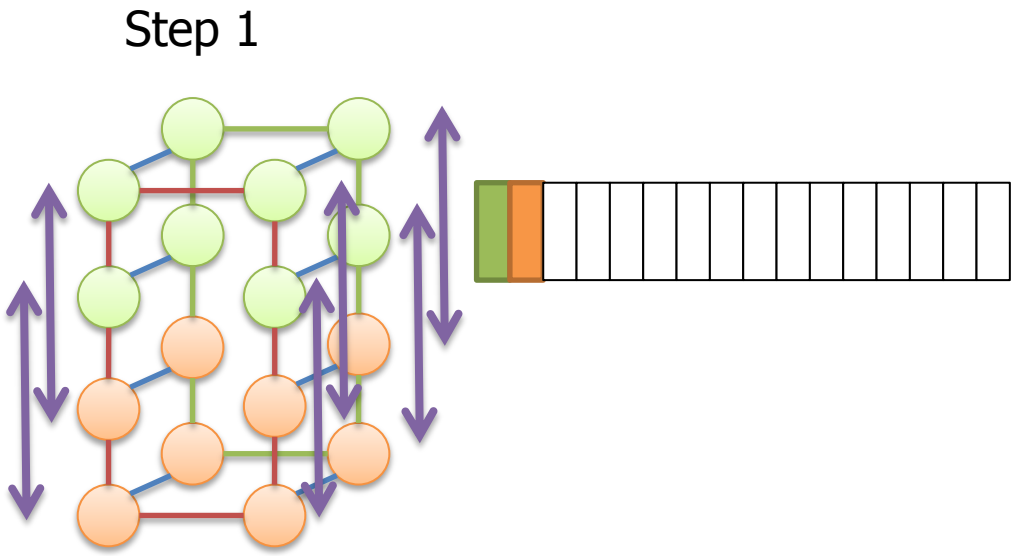
- Requires ($\log_2 p$) steps
  - Ex. p=16 => 4 steps
- Node mapping optimization
  1. Same hop counts between any nodes in every step
  2. Communicate data with neighbor node in the last step

Initial State



Allgather Implementation: Recursive Doubling

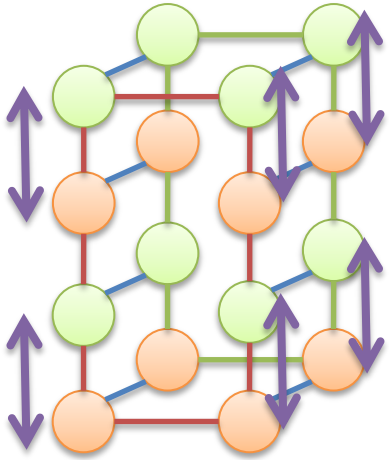
- Requires ($\log_2 p$) steps
  - Ex. p=16 => 4 steps
- Node mapping optimization
  1. Same hop counts between any nodes in every step
  2. Communicate data with neighbor node in the last step



Allgather Implementation: Recursive Doubling

- Requires ($\log_2 p$) steps
  - Ex. p=16 => 4 steps
- Node mapping optimization
  1. Same hop counts between any nodes in every step
  2. Communicate data with neighbor node in the last step

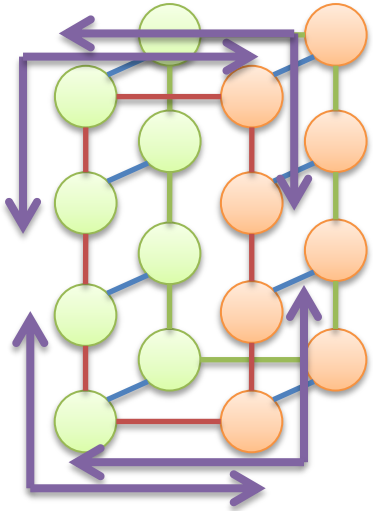
Step 2

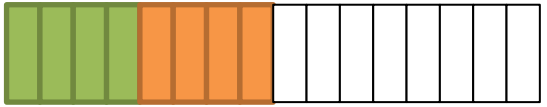


Allgather Implementation: Recursive Doubling

- Requires ($\log_2 p$) steps
  - Ex. p=16 => 4 steps
- Node mapping optimization
  1. Same hop counts between any nodes in every step
  2. Communicate data with neighbor node in the last step

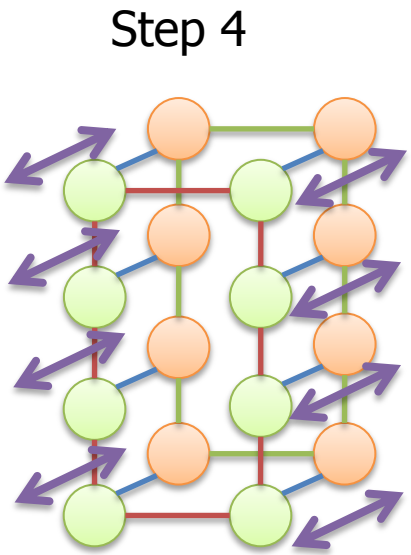
Step 3





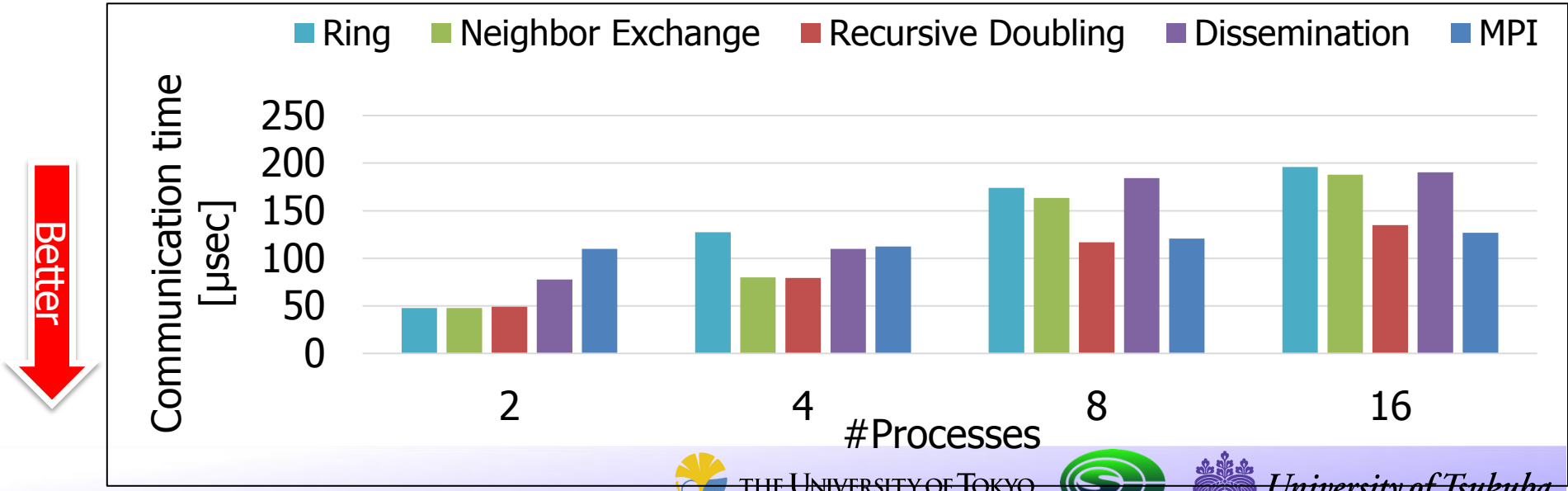
Allgather Implementation: Recursive Doubling

- Requires ($\log_2 p$) steps
  - Ex. p=16 => 4 steps
- Node mapping optimization
  1. Same hop counts between any nodes in every step
  2. Communicate data with neighbor node in the last step



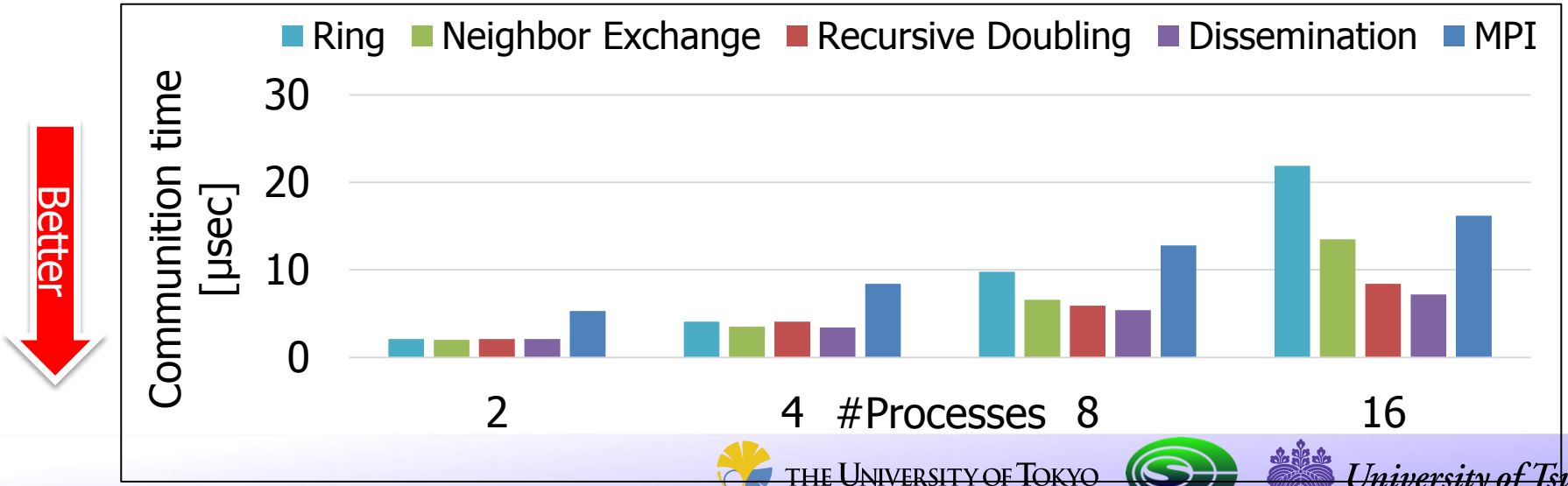
Allgather Performance Comparison among Various Algorithms

- Time for all-gathering 128 KB data
  - N=16384 case in CG method
- Recursive Doubling shows good performance
  - However, when p=16, TCA is slower than MPI in this size



Allreduce Performance

- Allreduce time for 8 Bytes scalar data
- **Dissemination is the fastest.**
- TCA is more than **twice faster** than MPI
  - Low latency of TCA works effectively



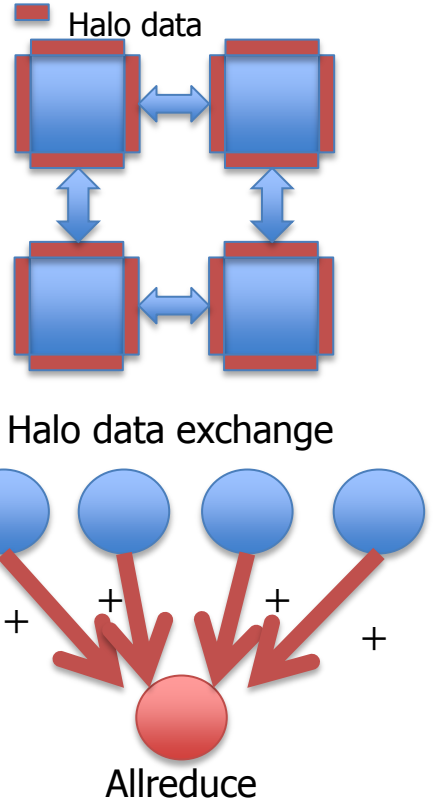
QUDA

- QUDA: The open source Lattice QCD library
  - widely used as a LQCD library for NVIDIA GPUs
    - Optimized for NVIDIA GPUs
    - All calculation run on GPUs
  - Solves a liner equation using CG method
  - inter-node parallelism support
  - supports multiple GPUs in a node
  - source code is available at github
    - https://github.com/lattice/quda

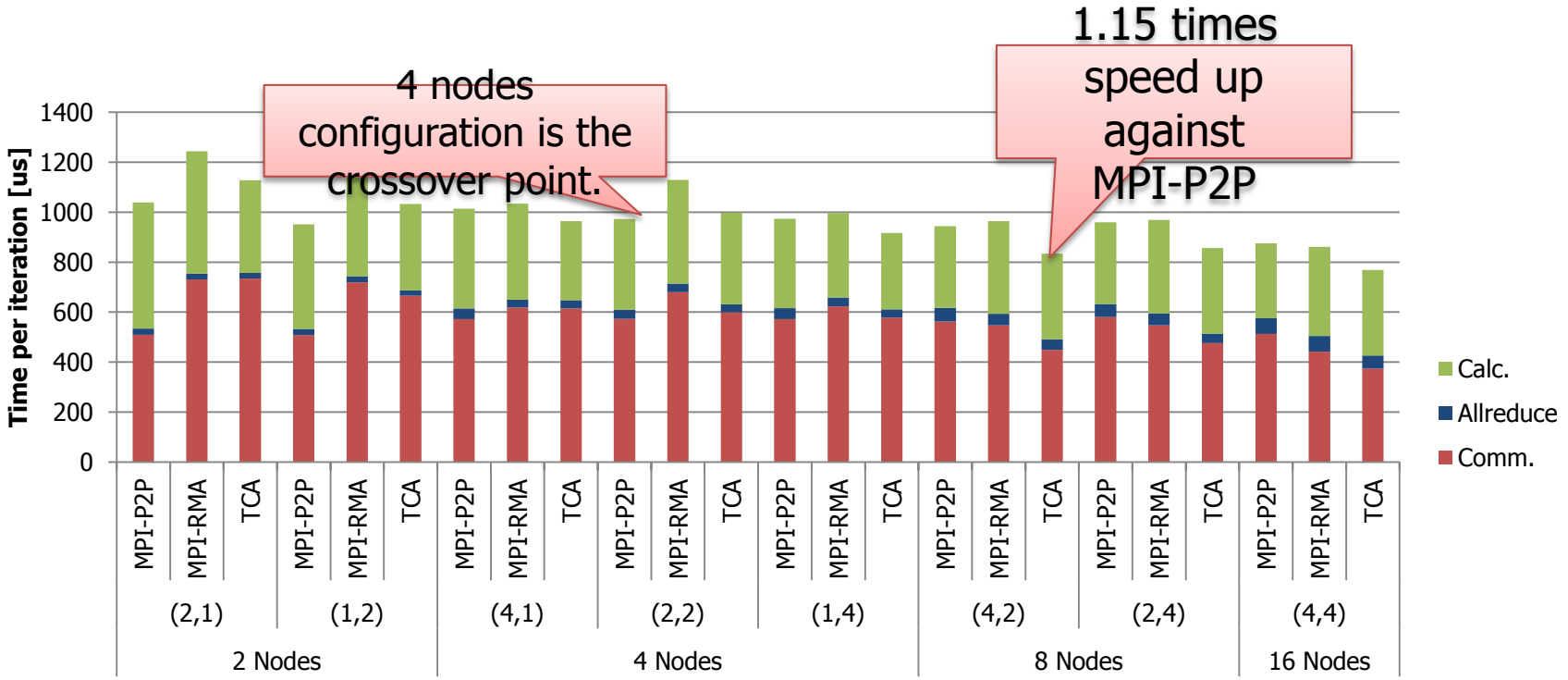
[HeteroPar2014]

Communication in QUDA

- Halo data exchange with RMA is dominant
  - Write data to neighbor processes' memory region
- Allreduce communication in CG
  - latency is important



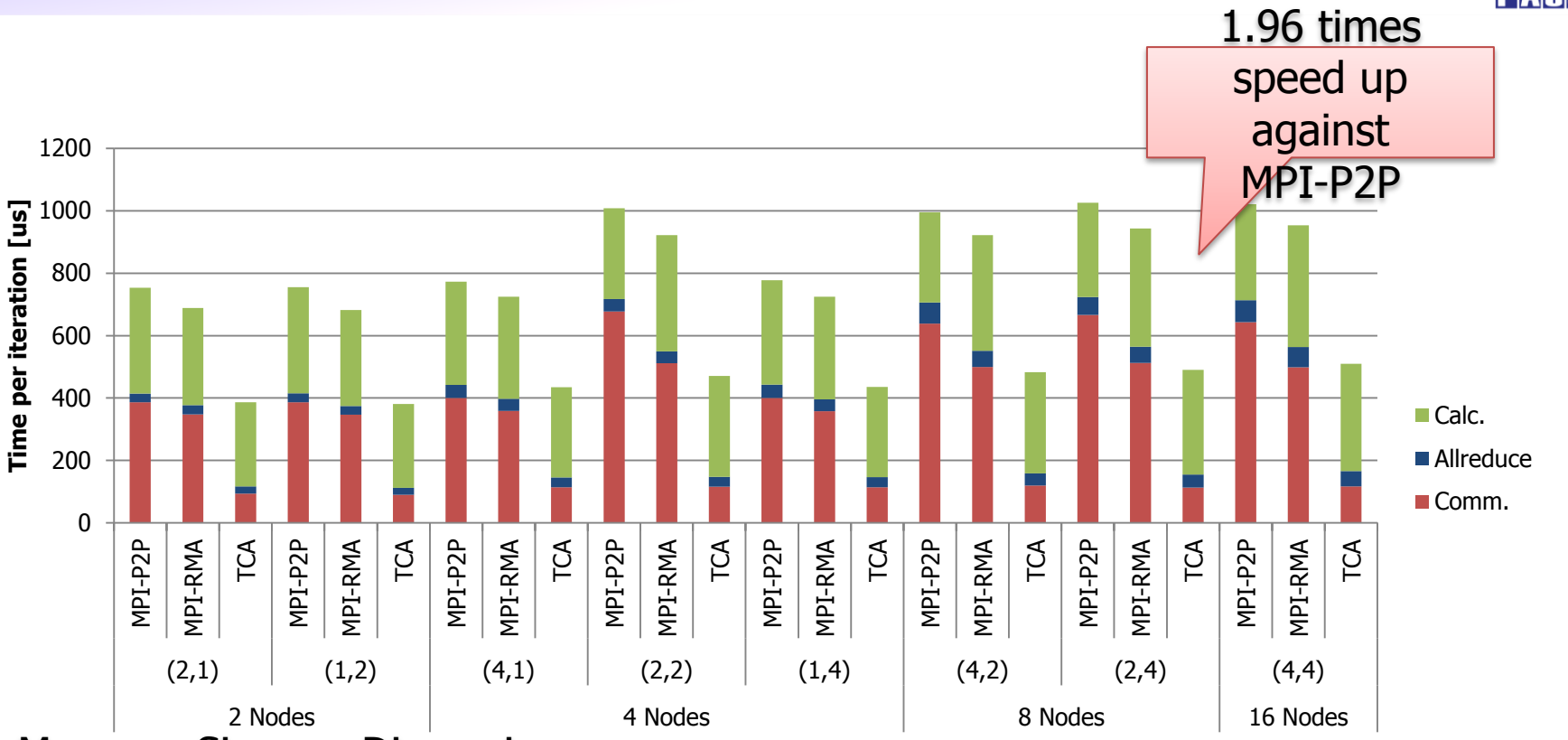
QUDA results: Large Model (16^4)



(x,y) nodes

Message Size per Dimension = 2 × (192KB / # of nodes in each dim.)

QUDA Results: Small Model (8^4)



Message Size per Dimension = 2 × (24KB / # of nodes in each dim.)

(x,y) nodes

Summary

- TCA: Tightly Coupled Accelerators
  - TCA enables direct communication among accelerators as an element technology becomes a basic technology for next gen's accelerated computing in exa-scale era.
- PEACH2 board: Implementation for realizing TCA using PCIe technology
  - Bandwidth: max. 3.5 Gbyte/sec between CPUs (over 95% of theoretical peak), 2.8 Gbyte/sec between GPUs
    Min. Latency: 0.8 us (PIO), 1.8 us (DMA between CPUs), 2.0 us (DMA between GPUs)
  - GPU-GPU communication over the nodes can be utilized with 16 node sub-cluster.
- Ping-pong program: PEACH2 can achieve lower latency than MPI in small data size.
- Collective communications on TCA
  - Allreduce: much faster than 2x of MPI
  - Allgather: slightly faster than MPI
- QUDA: TCA has a good performance on short messages
  - Small Model: All configurations
    - But, speedup was not shown…
  - Large Model: 8 and 16 nodes configurations
- FFTE: Small & Medium size is good for TCA

Future Work

- Offload functions in PEACH2
  - Reduction, etc.
- Prototype of PEACH3 is under development with PCIe Gen3 x8.
  - Altera Stratix V GX
  - Max bandwidth between CPUs is approx. 7GB/s with Gen3 x8, double of PEACH2 [CANDAR2014]

XcalableACC
a parallel programming language for accelerated parallel systems

Taisuke Boku
Center for Computational Sciences
University of Tsukuba

Complexity of parallel GPU programming

- Multiple orthogonal paradigms
  - MPI – array must be distributed and communicated (two-side or one-side)
  - CUDA, OpenCL, OpenACC – memory allocation, data movement (to/from host), computation
  - controlling multiple devices if there are – CUDA 4.0 or with OpenMP multithreading
- Issues
  - how to combine array distribution, internal-communication, external-communication, …
  - simple and easy-to-understand programming model is required for high productivity

XcalableACC (XACC)

- PGAS language (C & Fortran) with directive base parallel programming for massively parallel accelerated computing
- Based on our traditional PGAS language XcalableMP (XMP)
- OpenACC is used for control on accelerating devices
- Developed in AICS, RIKEN under JST-CREST joint project
- We implement the compiler and run-time system both for general MPI-base system and TCA architecture

Outline of base language XcalableMP

- Execution model: SPMD (=MPI)
- Two programming model on data view
  - Global View (PGAS): based on data parallel concept, directives similar to OpenMP is used for data and task distribution (easy programming)
  - Local View: based on local data and explicit communication (easy performance tuning)
- OpenMP-like directives
  - Incremental parallelization from original sequential code
  - Low cost for parallelization -> high productivity
- Not "fully automatic parallelization", but user must do:
  - Each node processes the local data on that node
  - User can clearly imagine the data distribution and parallelization for easiness of tuning
  - Communication target of variables (arrays) and partitions can be simply specified
  - Communication point is specified by user, in easy manner

Data Distibution Using Template

- **Template**
  - virtual array representing data(index) space
  - array distribution, work-sharing must be done using template

Example)

#pragma xmp nodes p(4) declare node set

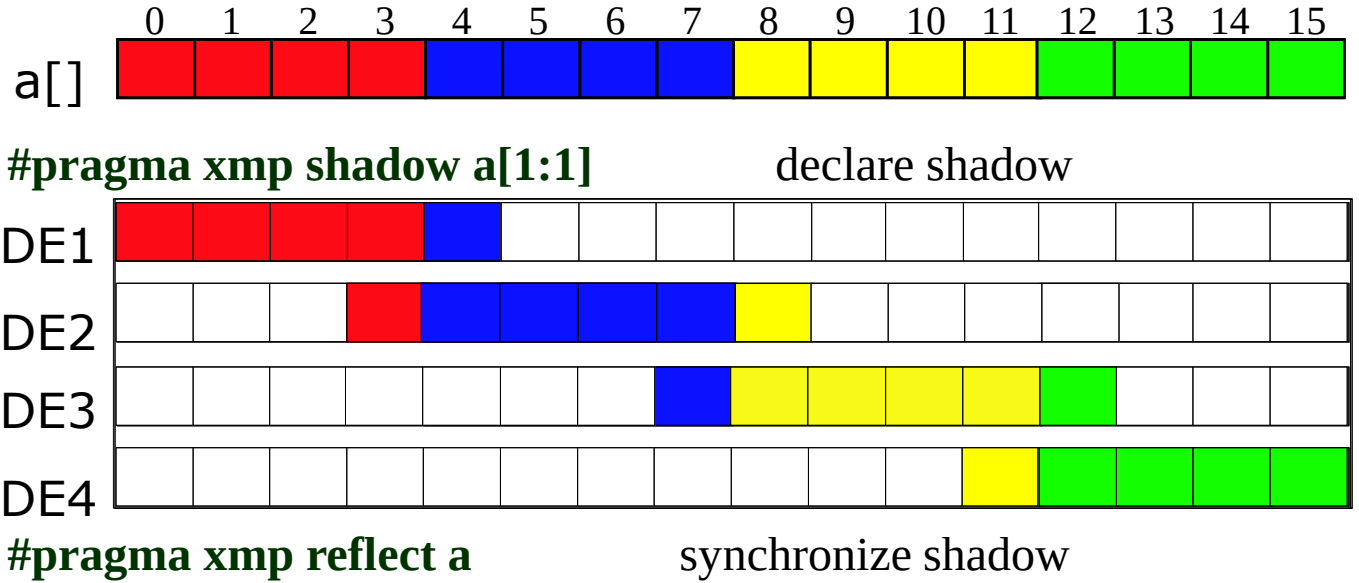
#pragma xmp template t(0:99) declare template

#pragma xmp distribute t(BLOCK) on p distribute template

#pragma align array[i] with t(i) distribute array : owner of t(i) has a[i]

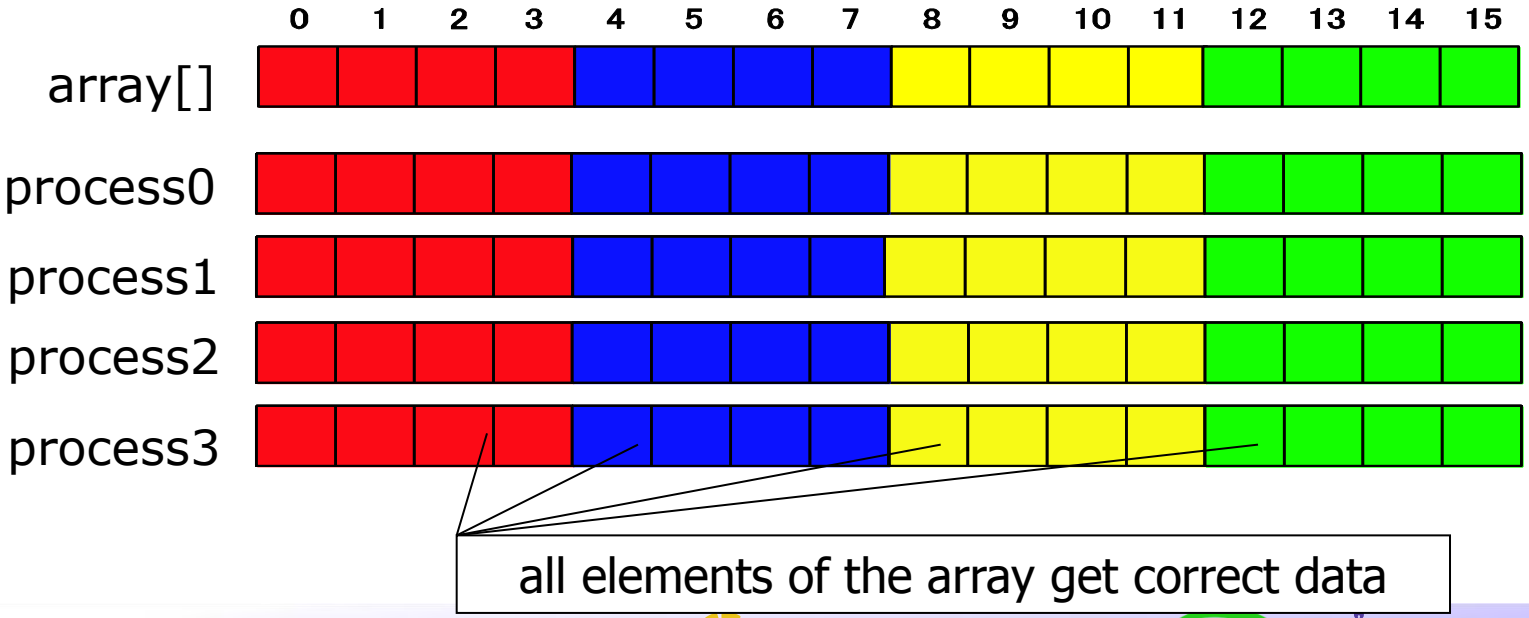
Data Synchronization of Array(shadow)

- **Shadow Region**
  - in XMP, memory access is always local
  - duplicated overlapped data distributed onto other nodes
  - data synchronization: **reflect directive**



Data Synchronization of Array(gather)

- gather array data (collect entire elements)
- #pragma xmp gather(var=*list*)



Internode Communication

- **broadcast**

#pragma xmp bcast *var* on *node* from *node*

- **barrier synchronization**

#pragma xmp barrier

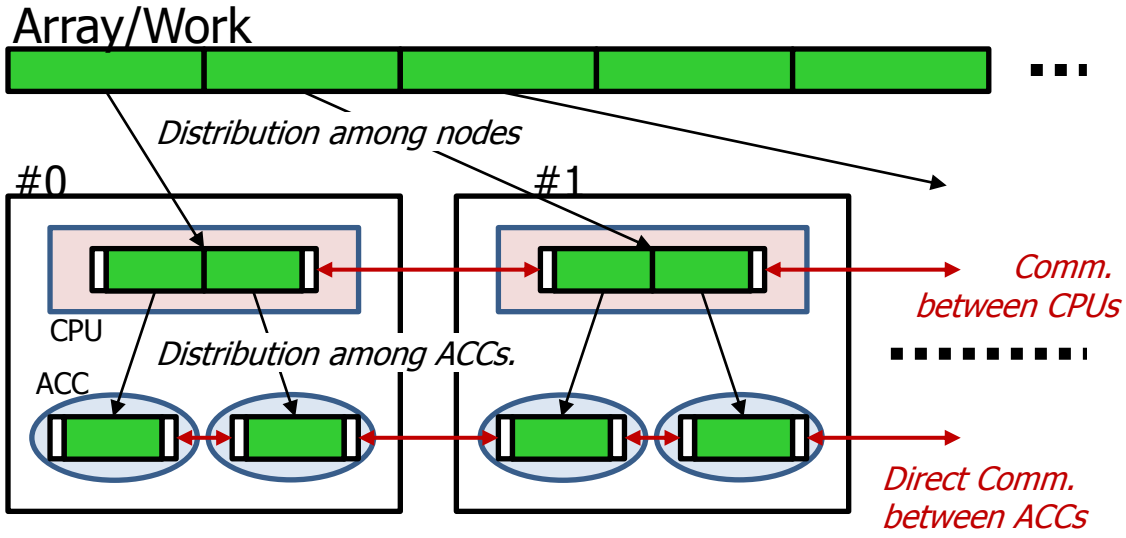
- **reduce operation**

#pragma xmp reduction (*var*:*op*)

- **data movement in global view**

#pragma xmp gmove

Processing model of XACC




```
#pragma acc device d = nvidia(0:3)

#pragma xmp reflect_init (a) device

#pragma xmp loop (i) on t(i)
for (int i = 0; i < 100; i++){
  #pragma acc kernels loop on_device(d)
  for (int j = 0; j < 100; j++){
    a[i][j] = ...
  }
}

#pragma xmp reflect_do (a)
```

Two implementations of XACC

- based on traditional communication library
  - for MPI
  - directive-base communication on distributed arrays are automatically performed with OpenACC data I/O and MPI communication
- based on TCA
  - using TCA for direct GPU-memory copy

Example of XcalableACC program

```
double u[XSIZE][YSIZE], uu[XSIZE][YSIZE];




…

{
  for(k=0; k<MAX_ITER; k++){


   for(x=1; x<XSIZE-1; x++)
      for(y=1; y<YSIZE-1; y++)
        uu[x][y] = u[x][y];




    for(x=1; x<XSIZE-1; x++)
      for(y=1; y<YSIZE-1; y++)
        u[x][y] = (uu[x-1][y]+uu[x+1][y]+
                   uu[x][y-1]+uu[x][y+1])/4.0;
  } // end k
```

2-D Laplace Eq.

Example of XcalableACC program

```
double u[XSIZE][YSIZE], uu[XSIZE][YSIZE];
#pragma xmp nodes p(x, y)
#pragma xmp template t(0:YSIZE-1, 0:XSIZE-1)
#pragma xmp distribute t(block, block) onto p
#pragma xmp align [j][i] with t(i,j) :: u, uu
#pragma xmp shadow uu[1:1][1:1]
…


  for(k=0; k<MAX_ITER; k++){
#pragma xmp loop (y,x) on t(y,x)

   for(x=1; x<XSIZE-1; x++)
      for(y=1; y<YSIZE-1; y++)
        uu[x][y] = u[x][y];

#pragma xmp reflect (uu)

#pragma xmp loop (y,x) on t(y,x)

    for(x=1; x<XSIZE-1; x++)
      for(y=1; y<YSIZE-1; y++)
        u[x][y] = (uu[x-1][y]+uu[x+1][y]+
                   uu[x][y-1]+uu[x][y+1])/4.0;
  } // end k
```

2-D Laplace Eq.

array distribution and "sleeve" declaration

exchange sleeves on array "uu"

Example of XcalableACC program

```
double u[XSIZE][YSIZE], uu[XSIZE][YSIZE];
#pragma xmp nodes p(x, y)
#pragma xmp template t(0:YSIZE-1, 0:XSIZE-1)
#pragma xmp distribute t(block, block) onto p
#pragma xmp align [j][i] with t(i,j) :: u, uu
#pragma xmp shadow uu[1:1][1:1]
…
#pragma acc data copy(u) copyin(uu)
{
  for(k=0; k<MAX_ITER; k++){
#pragma xmp loop (y,x) on t(y,x)
#pragma acc parallel loop collapse(2)
   for(x=1; x<XSIZE-1; x++)
     for(y=1; y<YSIZE-1; y++)
       uu[x][y] = u[x][y];

#pragma xmp reflect (uu) acc

#pragma xmp loop (y,x) on t(y,x)
#pragma acc parallel loop collapse(2)
    for(x=1; x<XSIZE-1; x++)
      for(y=1; y<YSIZE-1; y++)
        u[x][y] = (uu[x-1][y]+uu[x+1][y]+
                   uu[x][y-1]+uu[x][y+1])/4.0;
  } // end k
} // end data
```

2-D Laplace Eq.

array distribution and "sleeve" declaration

copy partial (distributed) array to device memory

distributed array by XMP is processed according to OpenACC directive

exchange sleeves on array "uu"

"acc" clause indicates to target the array on device memory

Example of XcalableACC program

```
double u[XSIZE][YSIZE], uu[XSIZE][YSIZE];

…
#pragma acc data copy(u) copyin(uu)
{
  for(k=0; k<MAX_ITER; k++){

#pragma acc parallel loop collapse(2)
   for(x=1; x<XSIZE-1; x++)
     for(y=1; y<YSIZE-1; y++)
       uu[x][y] = u[x][y];



#pragma acc parallel loop collapse(2)
    for(x=1; x<XSIZE-1; x++)
      for(y=1; y<YSIZE-1; y++)
        u[x][y] = (uu[x-1][y]+uu[x+1][y]+
                   uu[x][y-1]+uu[x][y+1])/4.0;
  } // end k
} // end data
```

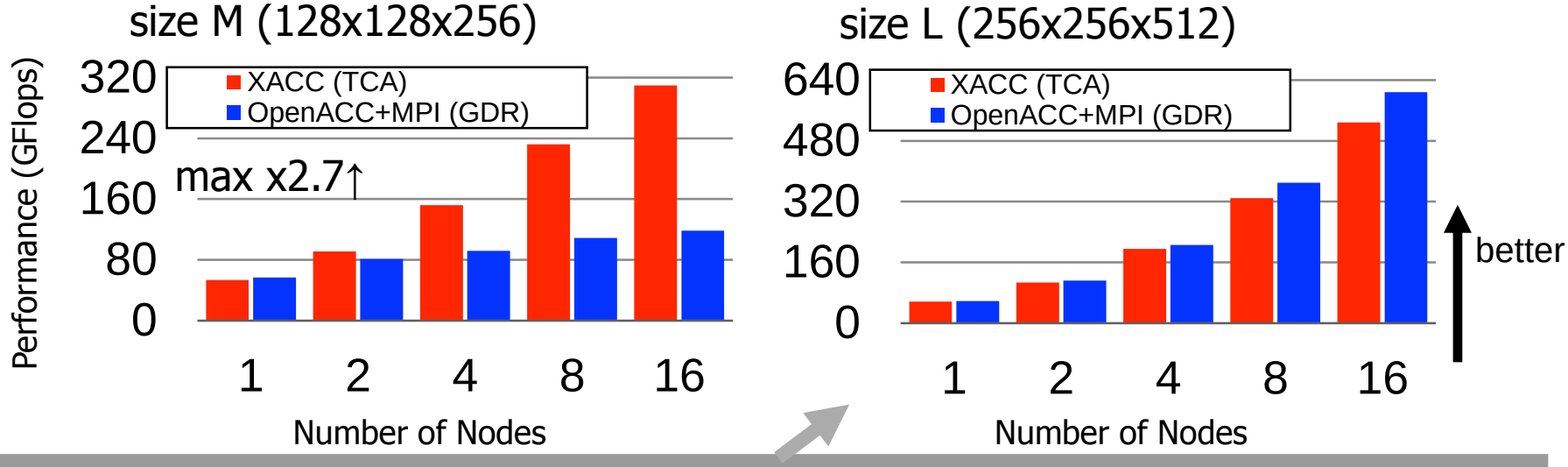
2-D Laplace Eq.

copy partial (distributed) array to device memory

distributed array by XMP is processed according to OpenACC directive

Performance on Himeno Benchmark by XcalableACC

2-D stencil computing for fluid dynamics



For size L, size of sleeve area is approximately 520KB, so TCA's advantage is small compared to MVAPICH2-GDR.

Additionally, TCA requires a barrier synch. after DMA transfer to cause additional overhead

Summary

- TCA is a basic research on the possibility on direct network between accelerators (GPUs) on current available technology
- Toward strong-scaling on post-peta to exascale HPC research, such a direct network for accelerators is essential
- Language/Programming is also very important issue for high productivity over multiple programming paradigms
- XcalableACC + TCA is a solution
- Awarded in HPC Challenge Class2 Best Performance Award at SC14

References

- [AsHES2015] Kazuya Matsumoto, Toshihiro Hanawa, Yuetsu Kodama, Hisafumi Fujii, Taisuke Boku, ” Implementation of CG Method on GPU Cluster with Proprietary Interconnect TCA for GPU Direct Communication,” The Internatonal Workshop on Accelerators and Hybrid Exascale Systems (AsHES2015) , May 2015 (To appear)
- [CANDAR2014] Takuya Kuhara, Takahiro Kaneda, **Toshihiro Hanawa**, Yuetsu Kodama, Taisuke Boku, and Hideharu Amano, “A preliminarily evaluation of PEACH3: a switching hub for tightly coupled accelerators,” 2nd International Workshop on Computer Systems and Architectures (CSA‘14), in conjunction with the 2nd International Symposium on Computing and Networking (CANDAR 2014), pp. 377 - 381, Dec. 2014.
- [WACCPD2014] Masahiro Nakao, Hitoshi Murai, Takenori Shimosaka, Akihiro Tabuchi, **Toshihiro Hanawa**, Yuetsu Kodama, Taisuke Boku, Mitsuhisa Sato, "XcalableACC: Extension of XcalableMP PGAS Language using OpenACC for Accelerator Clusters," Workshop on accelerator programming using directives (WACCPD 2014), in conjunction with SC14, pp. 27-36, Nov. 2014
- [HeteroPar2014] Norihisa Fujita, Hisafumi Fujii, **Toshihiro Hanawa**, Yuetsu Kodama, Taisuke Boku, Yoshinobu Kuramashi, and Mike Clark, "QCD Library for GPU Cluster with Proprietary Interconnect for GPU Direct Communication," 12th International Workshop Algorithms, Models and Tools for Parallel Computing on Heterogeneous Platforms (HeteroPar2014), LNCS 8805, pp. 251-262, Aug. 2014.
- [HEART2014] Yuetsu Kodama, **Toshihiro Hanawa**, Taisuke Boku and Mitsuhisa Sato, "PEACH2: FPGA based PCIe network device for Tightly Coupled Accelerators," Fifth International Symposium on Highly-Efficient Accelerators and Reconfigurable Technologies (HEART2014), pp. 3-8, Jun. 2014
- **[HOTI2013] Toshihiro Hanawa**, Yuetsu Kodama, Taisuke Boku, and Mitsuhisa Sato, "Interconnect for Tightly Coupled Accelerators Architecture," IEEE 21st Annual Sympsium on High-Performance Interconnects (HOT Interconnects 21), short paper, pp. 79-82, Aug. 2013
- **[AsHES2013] Toshihiro Hanawa**, Yuetsu Kodama, Taisuke Boku, and Mitsuhisa Sato, "Tightly Coupled Accelerators Architecture for Minimizing Communication Latency among Accelerators," The Third International Workshop on Accelerators and Hybrid Exascale Systems (AsHES2013), pp. 1030-1039, May 2013.

- Contact to:
  - Toshihiro Hanawa
    hanawa@cc.u-tokyo.ac.jp
  - Taisuke Boku
    taisuke@cs.tsukuba.ac.jp