XcalableACCによる格子QCDコードの性能評価

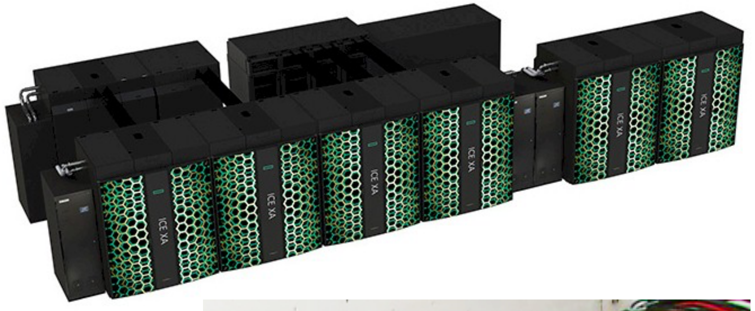
Masahiro Nakao,[†] Hitoshi Murai,[†] Hidetoshi Iwashita,[†]
Akihiro Tabuchi,[‡] Taisuke Boku,[‡$] Mitsuhisa Sato[†]

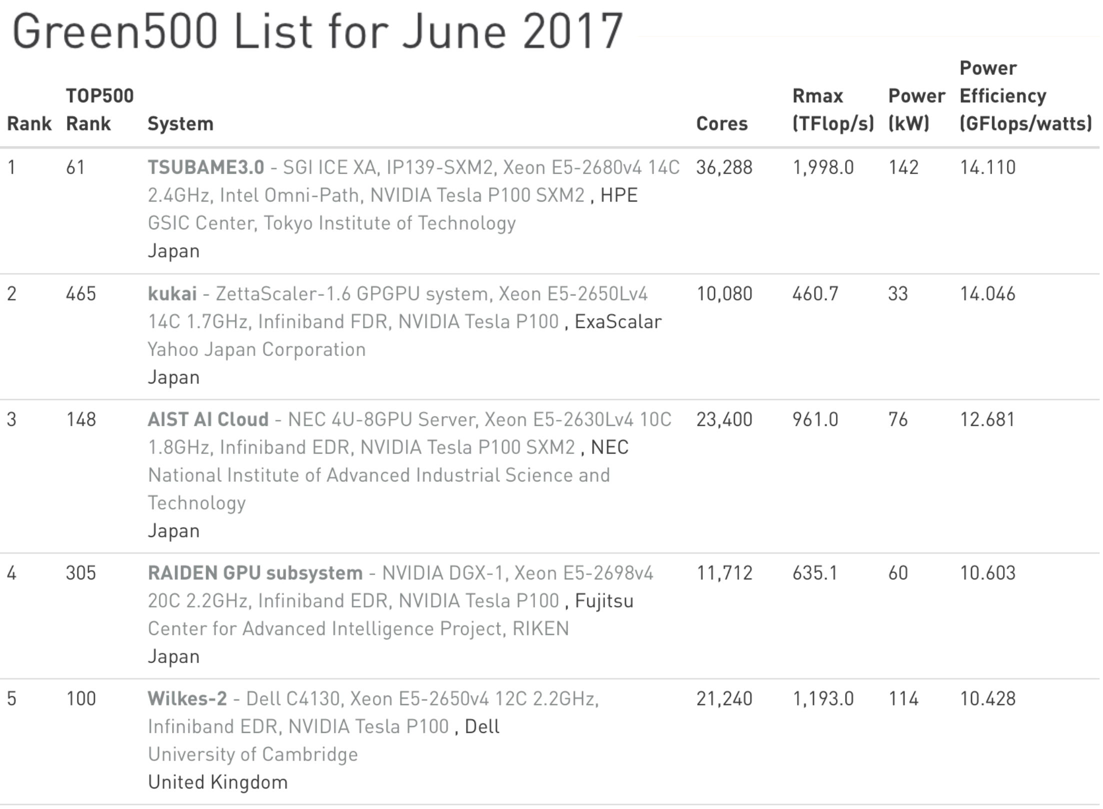
†RIKEN Advanced Institute for Computational Science, Japan
‡Graduate School of Systems and Information Engineering, University of Tsukuba, Japan
$Center for Computational Sciences, University of Tsukuba, Japan

5th XMP Workshop, Oct. 31th , 2017

Background

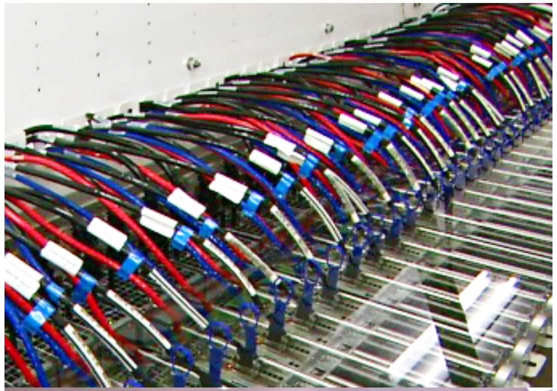
- Accelerated cluster (cluster equipped with accelerators)
  - High computing performance
  - High energy efficiency





Green500 List for June 2017

Rank	TOP500 Rank	System	Cores	Rmax (TFlop/s)	Power (kW)	Power Efficiency (GFlops/watts)
1	61	**TSUBAME3.0** - SGI ICE XA, IP139-SXM2, Xeon E5-2680v4 14C 2.4GHz, Intel Omni-Path, NVIDIA Tesla P100 SXM2 , HPE GSIC Center, Tokyo Institute of Technology Japan	36,288	1,998.0	142	14.110
2	465	**kukai** - ZettaScaler-1.6 GPGPU system, Xeon E5-2650Lv4 14C 1.7GHz, Infiniband FDR, NVIDIA Tesla P100 , ExaScalar Yahoo Japan Corporation Japan	10,080	460.7	33	14.046
3	148	**AIST AI Cloud** - NEC 4U-8GPU Server, Xeon E5-2630Lv4 10C 1.8GHz, Infiniband EDR, NVIDIA Tesla P100 SXM2 , NEC National Institute of Advanced Industrial Science and Technology Japan	23,400	961.0	76	12.681
4	305	**RAIDEN GPU subsystem** - NVIDIA DGX-1, Xeon E5-2698v4 20C 2.2GHz, Infiniband EDR, NVIDIA Tesla P100 , Fujitsu Center for Advanced Intelligence Project, RIKEN Japan	11,712	635.1	60	10.603
5	100	**Wilkes-2** - Dell C4130, Xeon E5-2650v4 12C 2.2GHz, Infiniband EDR, NVIDIA Tesla P100 , Dell University of Cambridge United Kingdom	21,240	1,193.0	114	10.428



Motivation

- Complex programming on accelerated cluster
  - MPI+CUDA is often used, can fully exploit computational performance
  - Hard to distribute data/work in MPI, and offload them in CUDA

```
__global__ void kernel(int a[MAX], int llimit, int ulimit)
{ ... }
 :
int main(int argc, char *argv[]){
 MPI_Int(&argc, &argc);
 MPI_Comm_rank(MPI_COMM_WORLD, &rank);
 MPI_Comm_size(MPI_COMM_WORLD, &size);
 dx = MAX/size;
 llimit = rank * dx;
 ulimit = (rank != size-1)? ulimit = llimit + dx : MAX;
 kernel <<< N_GRID, N_BLOCK >>> (a, llimit, ulimit);

 MPI_Send(a, ... , MPI_COMM_WORLD);
 MPI_Recv(a, ... , MPI_COMM_WORLD, &status);
```

Create a new kernel for GPU

Divide data and calculations

Send and receive local data by using primitive MPI functions

MPI+OpenACC has emerged as an alternative to MPI+CUDA.
But, its approach still faces a programming issue due to MPI.

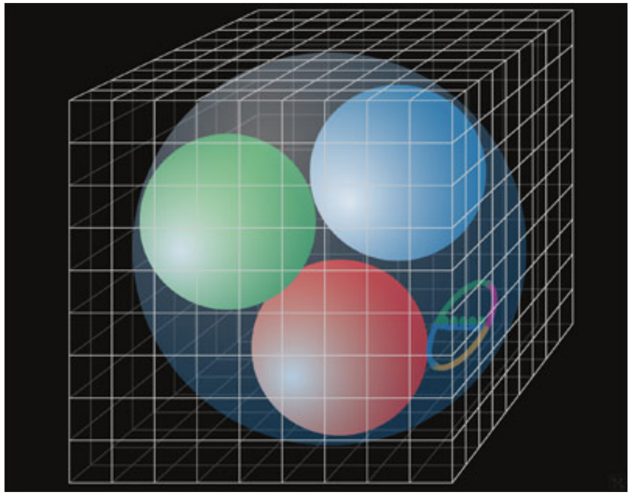
XcalableACC (XACC)

- **A new programming model for accelerated clusters, called XcalableACC (XACC)**
  - To reduce programming cost
  - Directive-based language extension based on C and Fortran (C++ on the table)
  - Multi-level parallelism (inter-node/intra-node/accelerators)
- **Orthogonal integration** of XcalableMP (XMP) and OpenACC
  - XACC = XMP + OpenACC + XACC extensions

XMP	Programming model for distributed memory system instead of MPI
OpenACC	Use of accelerators instead of CUDA
XACC extensions	Communication among accelerators, and so on

Objectives

- Compare the productivity and performance of XACC with those of MPI+CUDA and MPI+OpenACC
  - In previous study[1], we used some micro-benchmarks
  - In this study, target application is the Lattice QCD code on accelerated cluster (64 compute nodes, 256 GPUs)



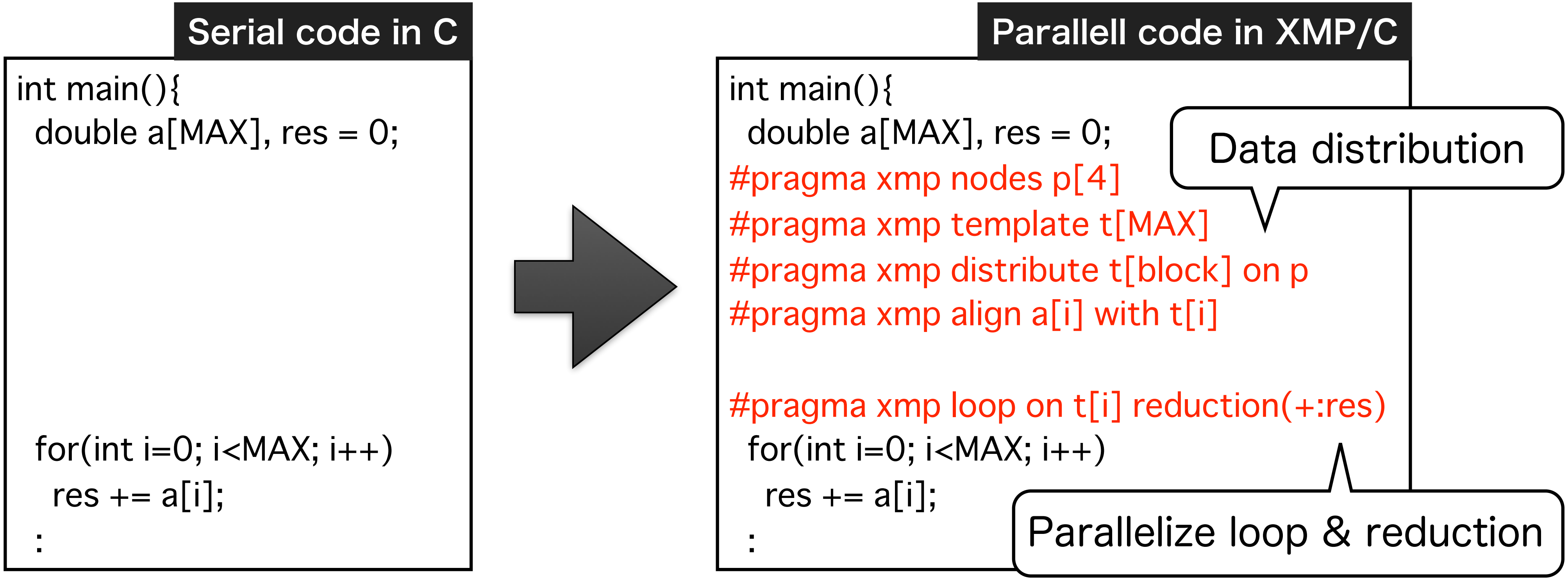
[1] Masahiro Nakao et al, “XcalableACC: Extension of XcalableMP PGAS Language Using OpenACC for Accelerator Clusters,” in Proceedings of WACCPD, 2014, pp. 27–36.

Agenda

1. Background
2. Overview of XMP, OpenACC, XACC
3. Lattice QCD code using XACC
4. Productivity and performance of XACC, MPI+CUDA, and MPI+OpenACC
5. Summary

XcalableMP (XMP) http://xcalablemp.org

- Directive-based language extension based on C and Fortran for distribute memory system (C++ on the table)
  - Add XMP directives to a serial code
  - The specification is designed by PC cluster consortium



Serial code in C

```
int main(){
  double a[MAX], res = 0;

  for(int i=0; i<MAX; i++)
    res += a[i];
  :
```

Parallell code in XMP/C

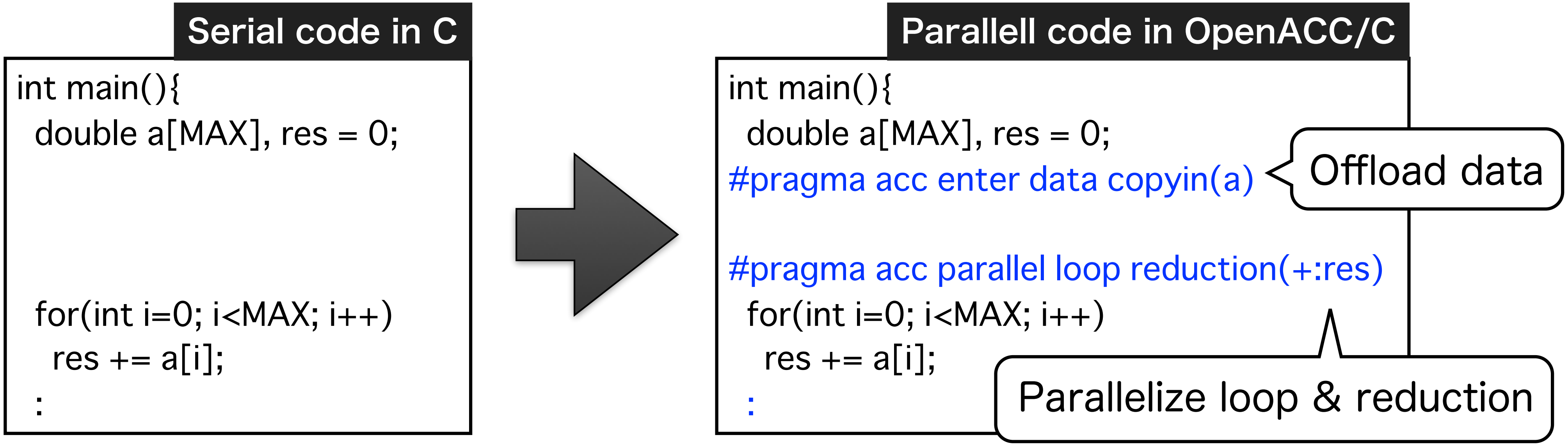
```
int main(){
  double a[MAX], res = 0;
#pragma xmp nodes p[4]
#pragma xmp template t[MAX]
#pragma xmp distribute t[block] on p
#pragma xmp align a[i] with t[i]

#pragma xmp loop on t[i] reduction(+:res)
  for(int i=0; i<MAX; i++)
    res += a[i];
  :
```

Data distribution

Parallelize loop & reduction

OpenACC https://www.openacc.org

- Directive-based language extension based on C, C++, and Fortran
  - Loops and regions of code to be offloaded from a host CPU to an attached accelerator (GPU, MIC, and so on)



Serial code in C

```
int main(){
  double a[MAX], res = 0;



  for(int i=0; i<MAX; i++)
    res += a[i];
  :
```

Parallell code in OpenACC/C

```
int main(){
  double a[MAX], res = 0;
#pragma acc enter data copyin(a)

#pragma acc parallel loop reduction(+:res)
  for(int i=0; i<MAX; i++)
    res += a[i];
  :
```

Offload data

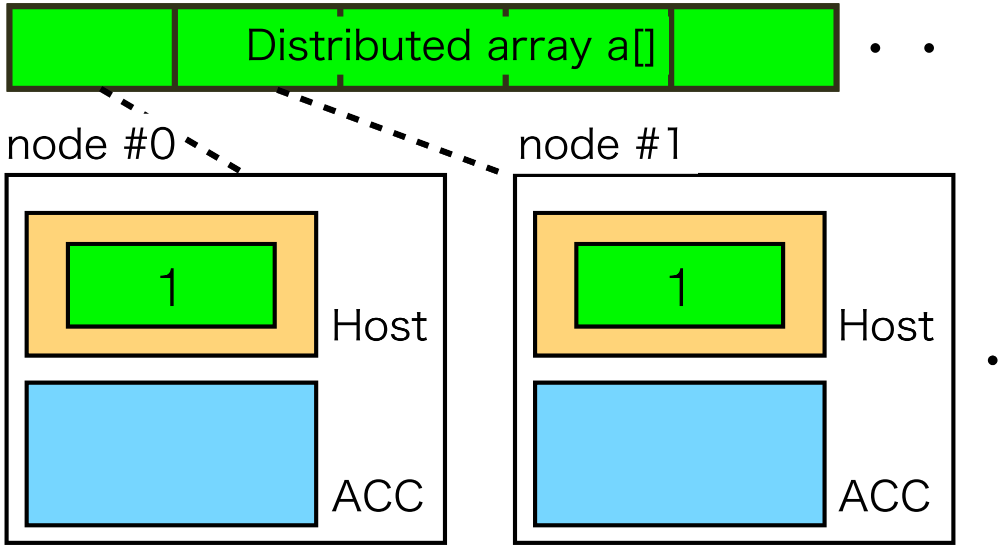
Parallelize loop & reduction

XcalableACC (XACC)

- XACC facilitates cooperation between XMP and OpenACC
  - XMP is used to distribute data and work on nodes
  - OpenACC is used to deal with an accelerator on each node

Parallell code in XACC/C

```
int main(){
  double a[MAX], res = 0;
#pragma xmp nodes p[4]
#pragma xmp template t[MAX]
#pragma xmp distribute t[block] on p                    1
#pragma xmp align a[i] with t[i]
#pragma acc enter data copyin(a)

#pragma xmp loop on t[i] reduction(+:res) acc
#pragma acc parallel loop reduction(+:res)
  for(int i=0; i<MAX; i++)
    res += a[i];
  :
```

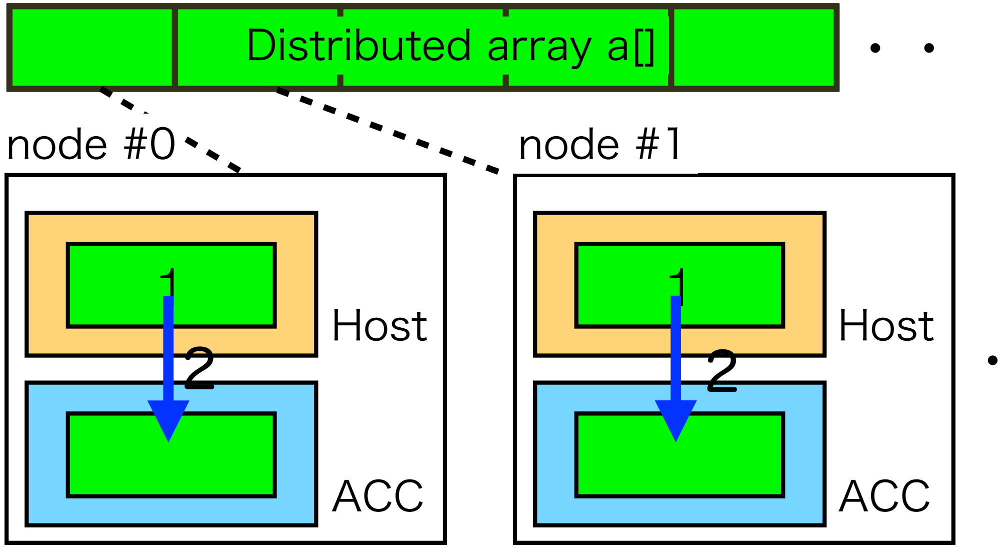


XcalableACC (XACC)

- XACC facilitates cooperation between XMP and OpenACC
  - XMP is used to distribute data/work on nodes
  - OpenACC is used to deal with an accelerator on each node

Parallell code in XACC/C

```
int main(){
  double a[MAX], res = 0;
#pragma xmp nodes p[4]
#pragma xmp template t[MAX]
#pragma xmp distribute t[block] on p                1
#pragma xmp align a[i] with t[i]
#pragma acc enter data copyin(a)                    2

#pragma xmp loop on t[i] reduction(+:res) acc
#pragma acc parallel loop reduction(+:res)
  for(int i=0; i<MAX; i++)
    res += a[i];
  :
```

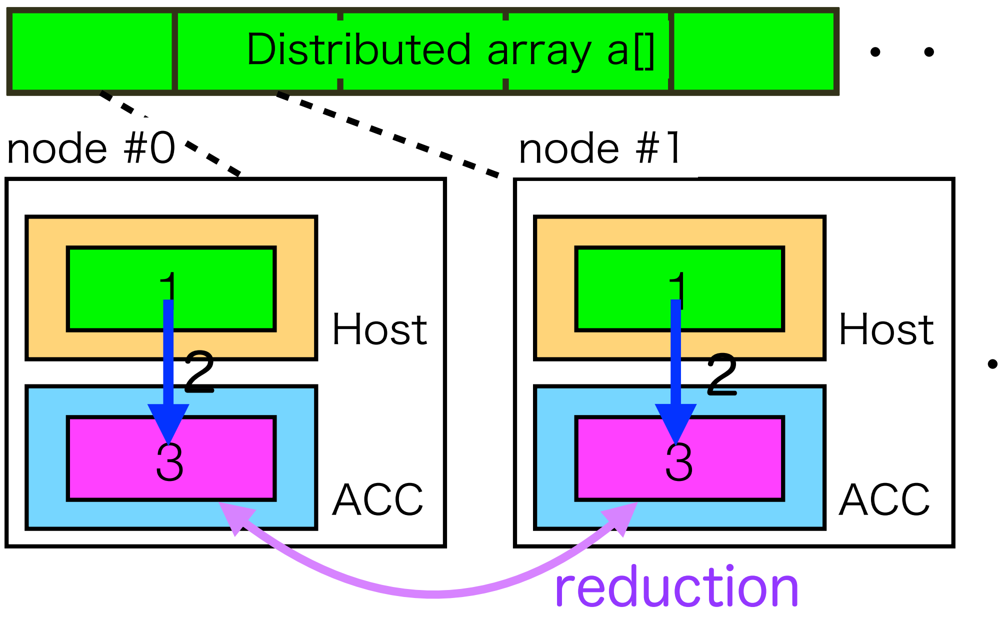


XcalableACC (XACC)

- XACC facilitates cooperation between XMP and OpenACC
  - XMP is used to distribute data and work on nodes
  - OpenACC is used to deal with an accelerator on each node

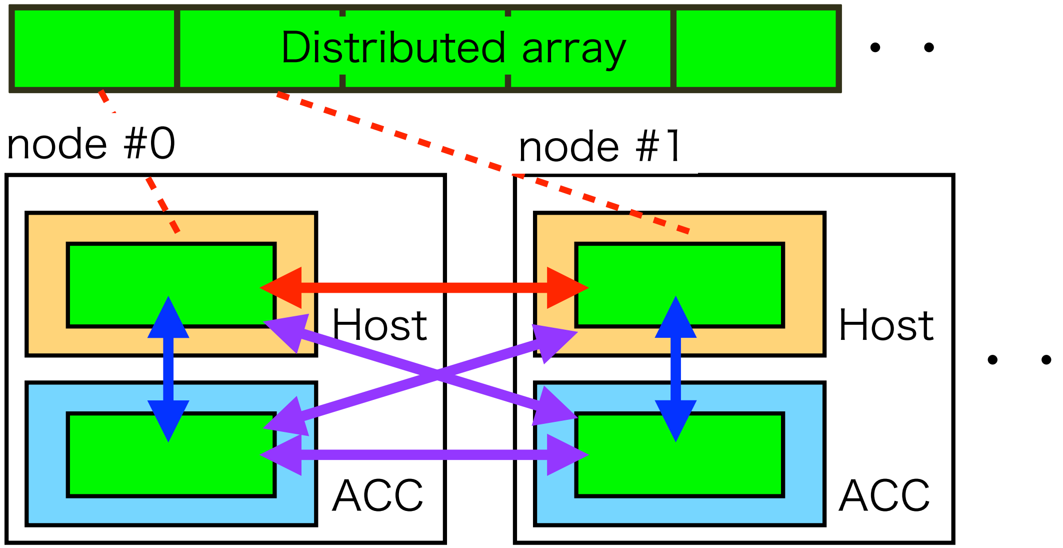
Parallell code in XACC/C

```
int main(){
  double a[MAX], res = 0;
#pragma xmp nodes p[4]
#pragma xmp template t[MAX]
#pragma xmp distribute t[block] on p                1
#pragma xmp align a[i] with t[i]
#pragma acc enter data copyin(a)                    2

#pragma xmp loop on t[i] reduction(+:res) acc
#pragma acc parallel loop reduction(+:res)          3
  for(int i=0; i<MAX; i++)
    res += a[i];
  :
```



The order of loop directives does not matter

Memory model in XACC



XMP and **XACC** support reflect (neighborhood comm.), reduction, bcast, global-copy, put, get

⟷ XMP

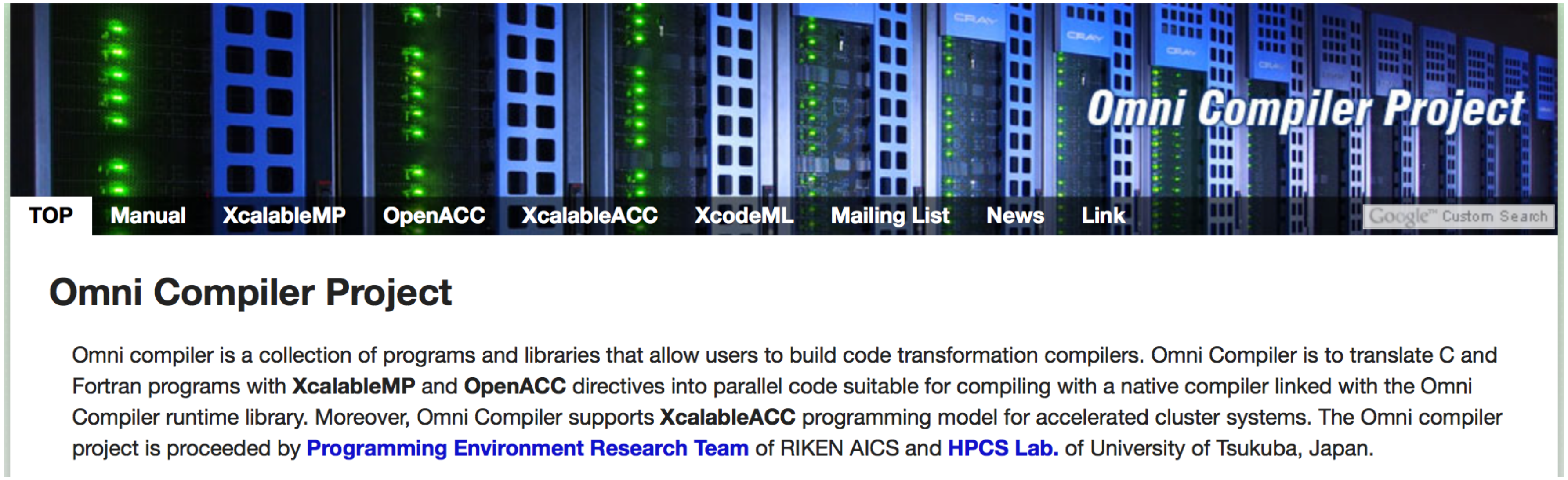
⟷ OpenACC

⟷ XACC extensions

XACC = XMP + OpenACC + XACC extensions

- **XMP** transfers data among host memories on different nodes
- **OpenACC** transfers data between host and accelerator memories on the same node
- **XACC extension** transfers data among accelerator memories and between host and accelerator memories on different nodes

Omni compiler



Omni Compiler Project

Omni compiler is a collection of programs and libraries that allow users to build code transformation compilers. Omni Compiler is to translate C and Fortran programs with **XcalableMP** and **OpenACC** directives into parallel code suitable for compiling with a native compiler linked with the Omni Compiler runtime library. Moreover, Omni Compiler supports **XcalableACC** programming model for accelerated cluster systems. The Omni compiler project is proceeded by **Programming Environment Research Team** of RIKEN AICS and **HPCS Lab.** of University of Tsukuba, Japan.

- Source-to-source compiler for directive-based languages
  - Support XMP, XACC, OpenACC directives
  - Any OpenACC compiler can be used as a backend compiler
- Open source software

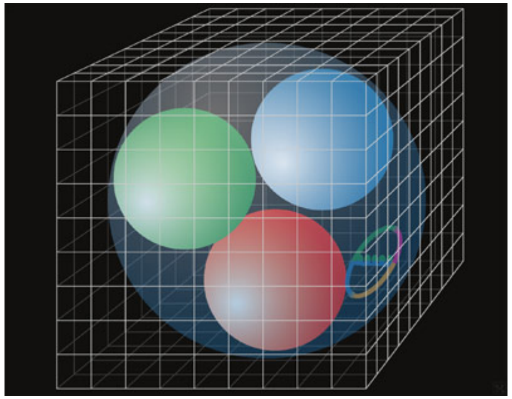
https://omni-compiler.org

Agenda

1. Background
2. Overview of XMP, OpenACC, XACC
3. Lattice QCD code using XACC
4. Productivity and performance of XACC, MPI+CUDA, and MPI+OpenACC
5. Summary

Overview of Lattice QCD

- One of the most important applications in high-performance computing
- Lattice QCD is a discrete formulation of QCD (Quantum Chromodynamics)
  - Describe the strong interaction among “quarks” and “gluons”
  - Quark is a species of elementary particles
  - Gluon is a particle that works between quarks
- Lattice QCD is formulated on a four-dimensional lattice (Time and XYZ axes)
  - Periodic boundary condition in all the direction



Lattice QCD mini-application

- Lattice QCD codes in XACC, MPI+CUDA, and MPI+OpenACC based on an existing Lattice QCD mini-application (http://research.kek.jp/people/matufuru/Research/Programs/index.html)
  - By High Energy Accelerator Research Organization, Japan
  - Written in C language, SLOC (Source Lines of Code) is 842
  - Implemented by extracting the main kernel of the Bridge++ (SLOC > 75,000), which is a real-world application for lattice gauge theories including QCD (http://bridge.kek.jp/Lattice-code/index_e.html)

Overview of algorithm

To solve many times a linear equation for a sparse matrix that represents the interaction between the quark and gluon fields.

Conjugate Gradient method

```
do while(not converged ?)
    T = WD(U,P)
    V = WD(U,T)
    pap = dot(V,P)
    cr = rr/pap
    X = axpy(X, cr, P)
    R = axpy(R, -cr, V)
    rr = norm(R)
    bk = rr/rrp
    P = scal(bk, P)
    P = axpy(P, 1.0, R)
    rrp = rr
enddo
```

WD() is the Wilson-Dirac operator

$$D_{x,y} = \delta_{x,y} - \kappa \sum_{\mu=1}^{4} \{(1-\gamma_\mu)U_\mu(x)\delta_{x+\hat{\mu},y} + (1+\gamma_\mu)U^\dagger_\mu(x-\hat{\mu})\delta_{x-\hat{\mu},y})\}$$

- Main kernel (most costly)
- Stencil calculation
- Calculates how the quarks interact with each other in the gluon filed

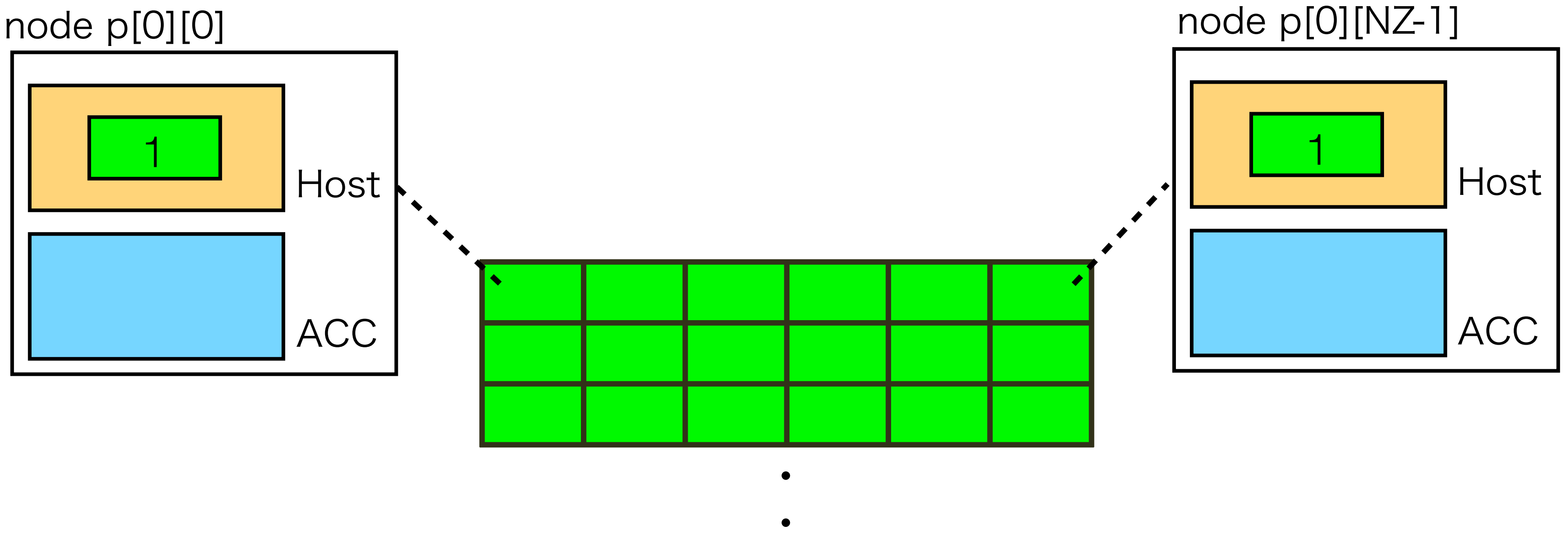
I will explain how to parallelize **WD()**.
Please refer to our paper about other functions.

U is a gluon.
Other uppercase characters are quarks

Declare distributed array

```
Quark_t v[NT][NZ][NY][NX];
#pragma xmp template t[NT][NZ]
#pragma xmp nodes p[PT][PZ]
#pragma xmp distribute t[block][block] onto p
#pragma xmp align v[i][j][*][*] with t[i][j]
#pragma xmp shadow v[1][1][0][0]
#pragma acc enter data copyin(v)
```

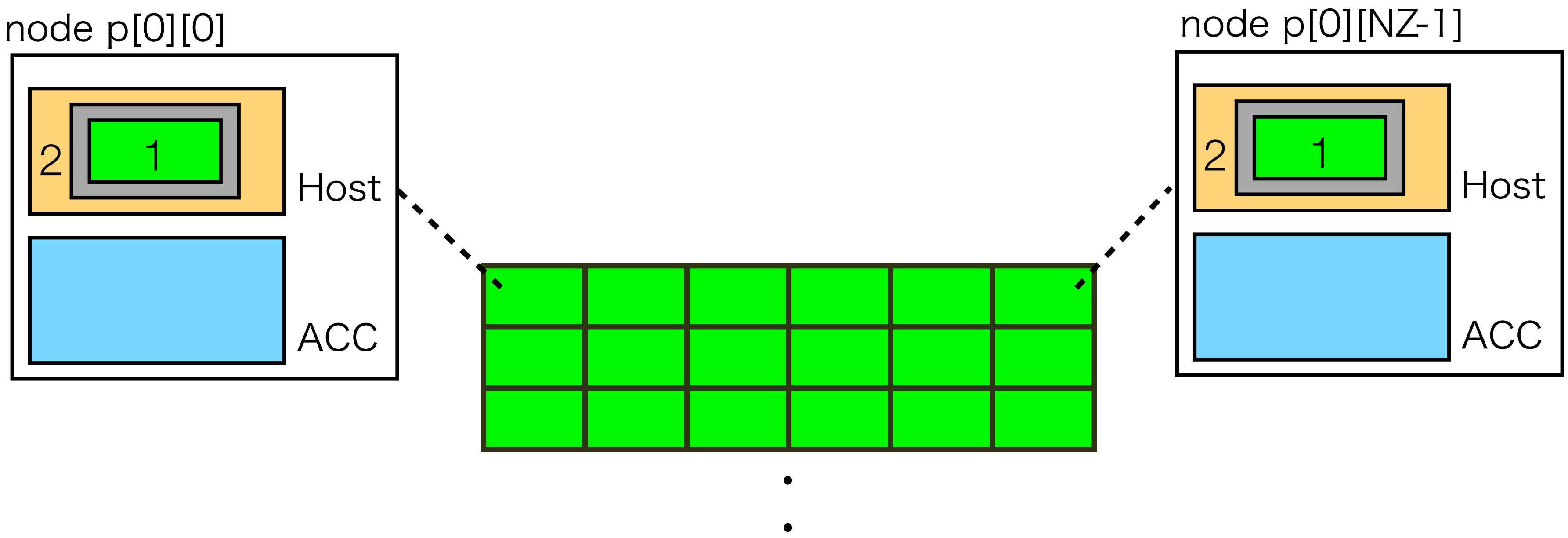
1. Define 2-dim. process grid.
   Dimension T and Z of arrays are distributed.
2. Add shadow area to the distributed array for stencil calculation
3. Offload the distributed array with shadow to accelerator memory



Declare distributed array

```
Quark_t v[NT][NZ][NY][NX];
#pragma xmp template t[NT][NZ]
#pragma xmp nodes p[PT][PZ]
#pragma xmp distribute t[block][block] onto p
#pragma xmp align v[i][j][*][*] with t[i][j]
#pragma xmp shadow v[1][1][0][0]
#pragma acc enter data copyin(v)
```

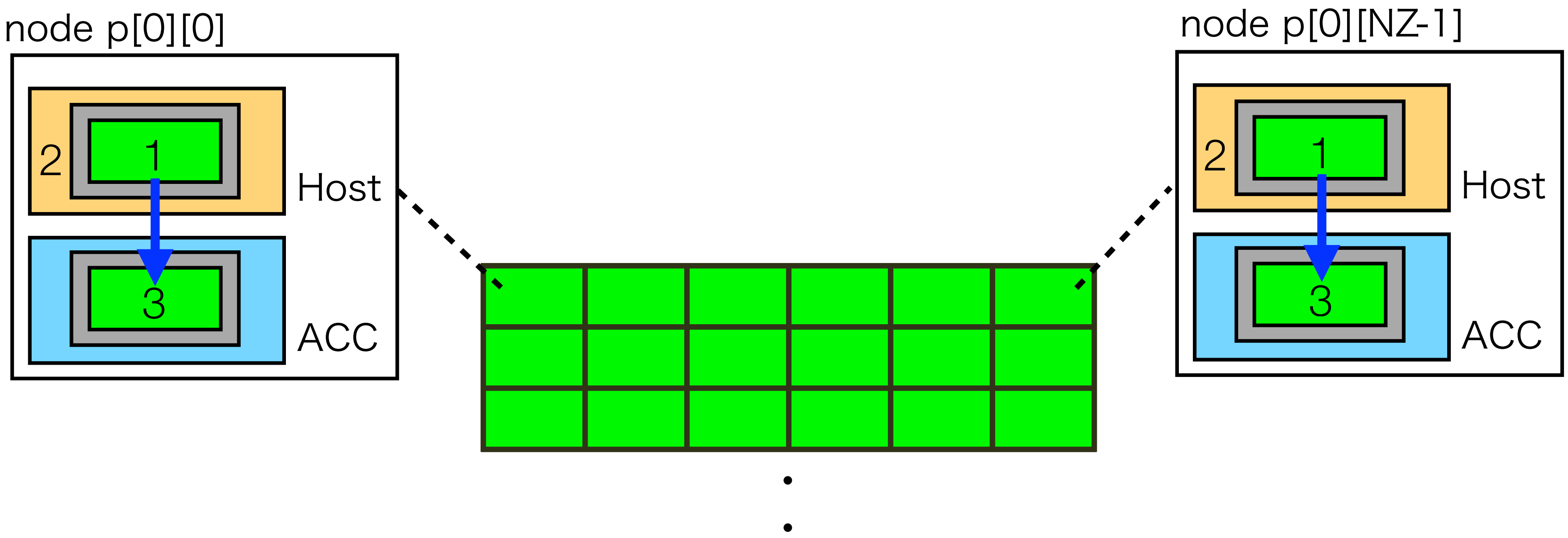
1. **Define 2-dim. process grid.**
   Dimension T and Z of arrays are distributed.
2. **Add shadow area to the distributed array for stencil calculation**
3. **Offload the distributed array with shadow to accelerator memory**



Declare distributed array

```
Quark_t v[NT][NZ][NY][NX];
#pragma xmp template t[NT][NZ]
#pragma xmp nodes p[PT][PZ]
#pragma xmp distribute t[block][block] onto p
#pragma xmp align v[i][j][*][*] with t[i][j]
#pragma xmp shadow v[1][1][0][0]
#pragma acc enter data copyin(v)
```

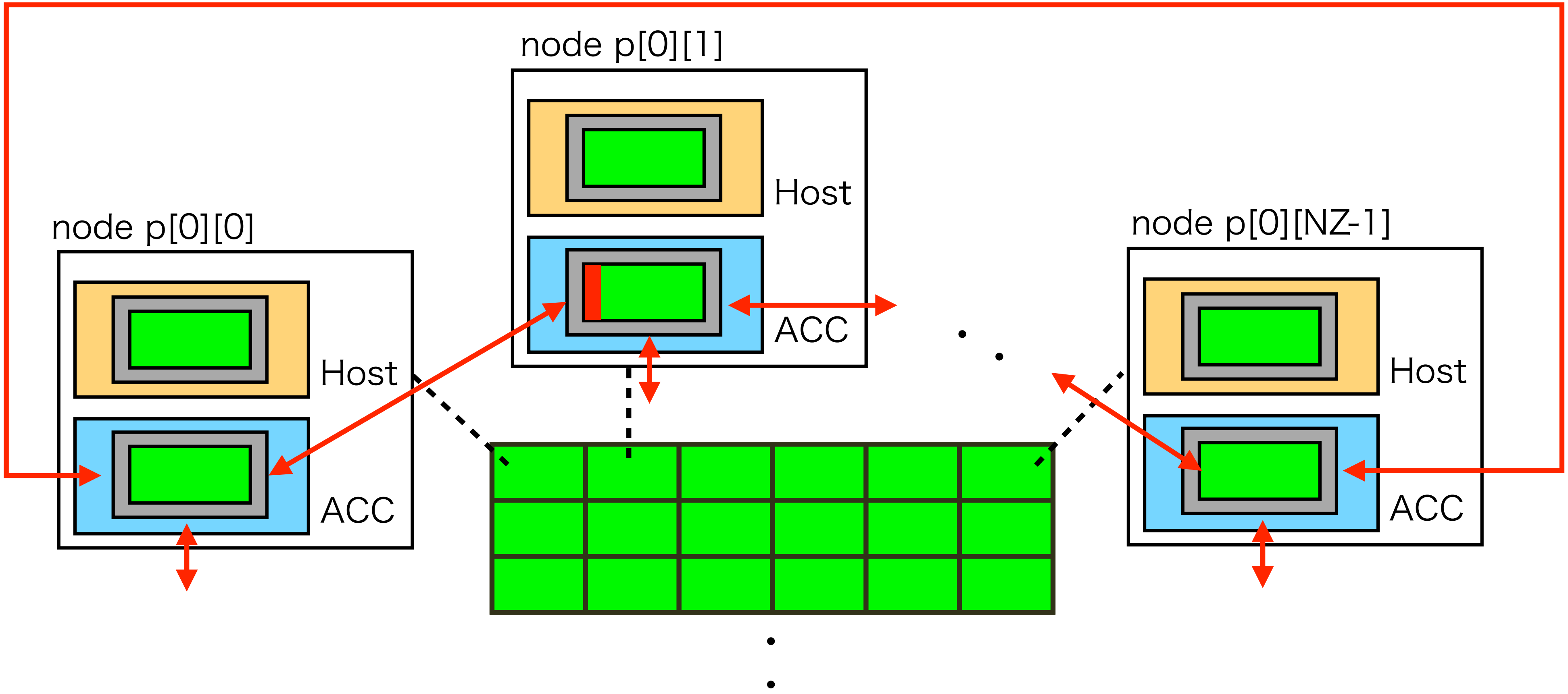
1. **Define 2-dim. process grid. Dimension T and Z of arrays are distributed.**
2. **Add shadow area to the distributed array for stencil calculation**
3. **Offload the distributed array with shadow to accelerator memory**



Neighborhood communication

Before calling WD(), XMP reflect directive updates shadow area on accelerator

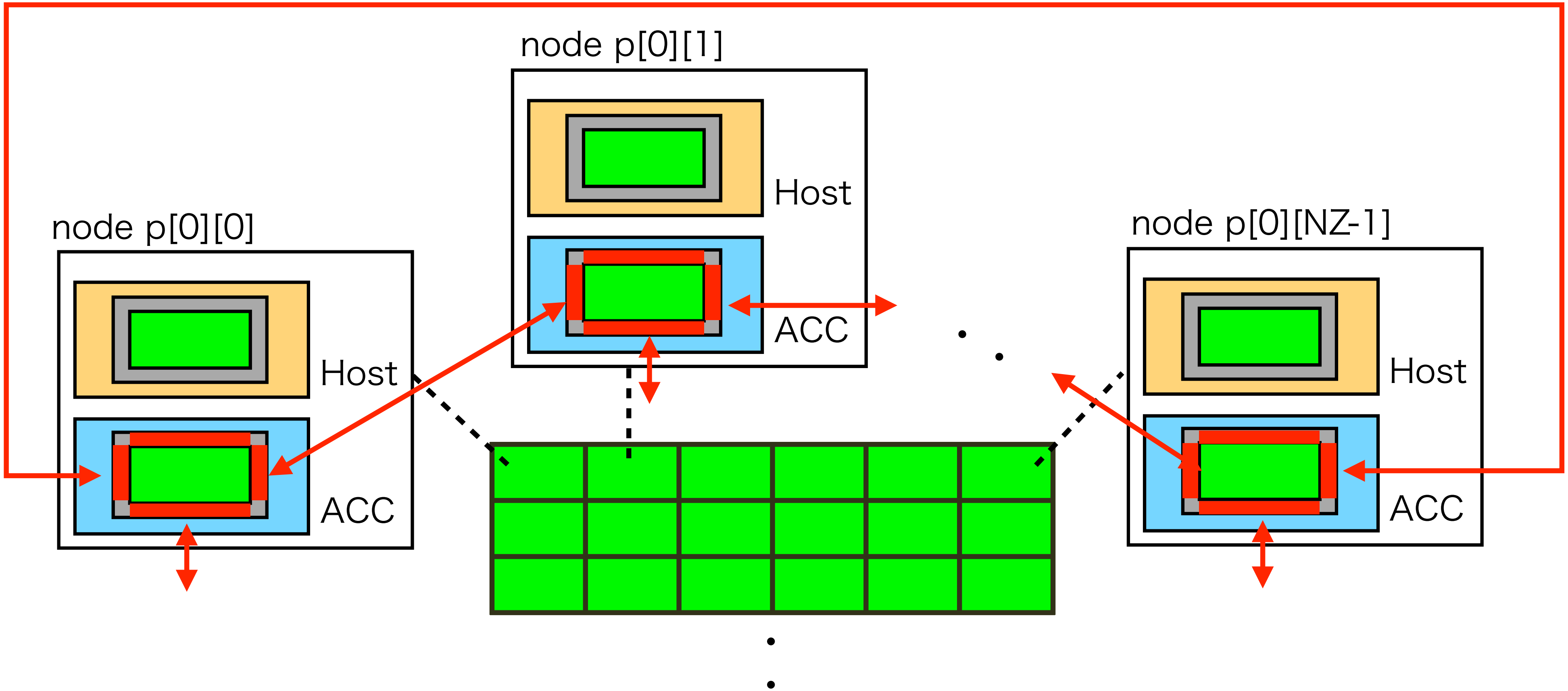
```
#pragma xmp reflect(v) width(/periodic/1:1,/periodic/1:1,0,0) orthogonal acc
WD(..., v); // Stencil calculation
```



Neighborhood communication

Before calling WD(), XMP reflect directive updates shadow area on accelerator

```
#pragma xmp reflect(v) width(/periodic/1:1,/periodic/1:1,0,0) orthogonal acc
WD(..., v); // Stencil calculation
```

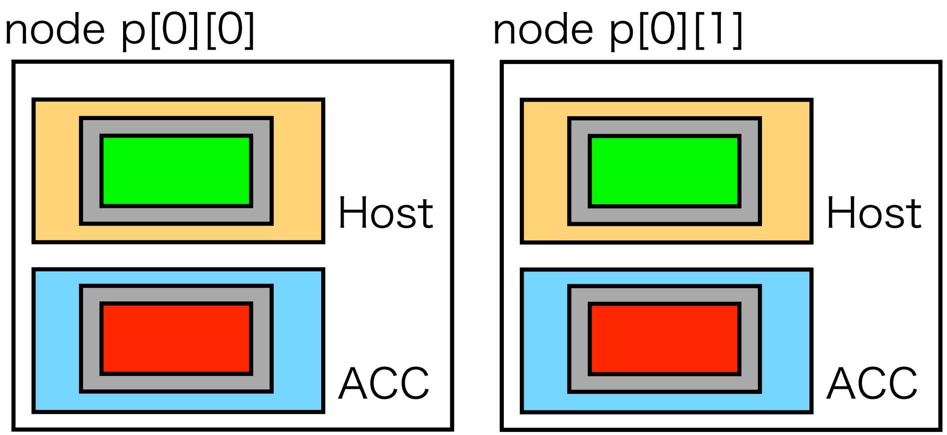


Parallelize Wilson-Dirac operator

Before calling **WD()**, XMP reflect directive updates own halo region on acc.

```
#pragma xmp reflect(v) width(/periodic/1:1,/periodic/1:1,0,0) orthogonal acc
WD(..., v);
```

```
void WD(..., Quark_t v[NT][NZ][NY][NX])
{
  …
#pragma xmp loop (iz,it) on t[it][iz]
#pragma acc parallel loop collapse(4) …
  for(it = 0; it < NT; it++){
    for(iz = 0; iz < NZ; iz++){
      for(iy = 0; iy < NY; iy++){
        for(ix = 0; ix < NX; ix++){
          …
```



1. XMP loop directive parallelizes the following loop statement on each node
2. OpenACC loop directive also parallelizes the following loop statement on each accelerator

Agenda

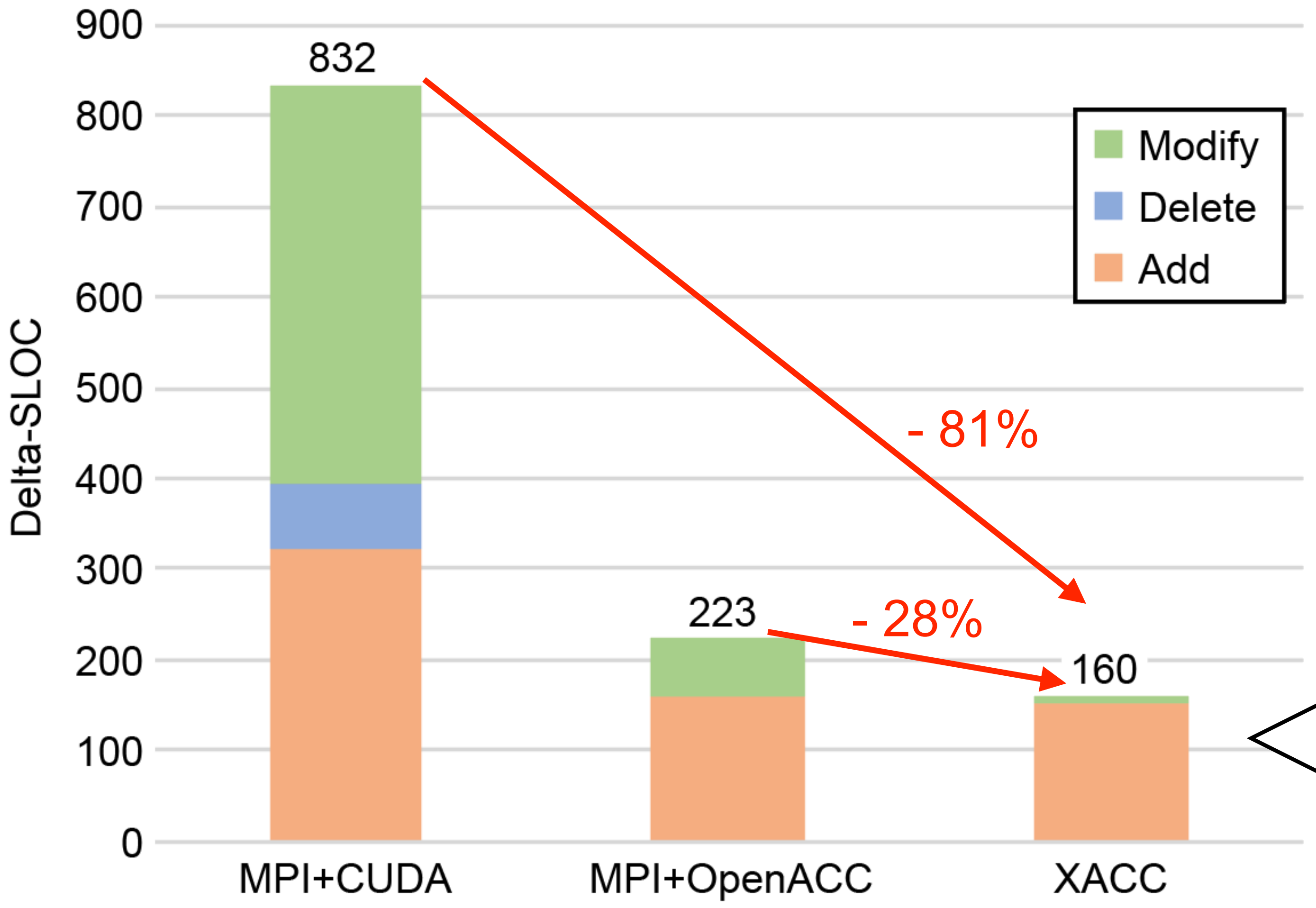
1. Background
2. Overview of XMP, OpenACC, XACC
3. Lattice QCD code using XACC
4. Productivity and performance of XACC, MPI+CUDA, and MPI+OpenACC
5. Summary

How do we evaluate productivity ?

- A parallel code is often based on an existing serial code
  - This code changes are likely to trigger program bugs
- Delta Source Lines of Codes (Delta-SLOC) metric [2]
  - Indicates how many lines the code changed from a serial code to a parallel code
  - Sum of three components: how many lines are added, deleted and modified
  - When the Delta-SLOC is small, the possibility of program bugs will be small

[2] Andrew I. Stone et al, “Evaluating coarray fortran with the cgpop miniapp,” in Proceedings of the Fifth Conference on Partitioned Global Address Space Programming Models (PGAS), October 2011.

Productivity results

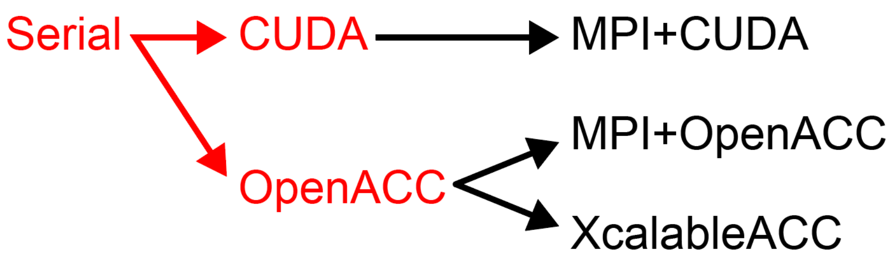


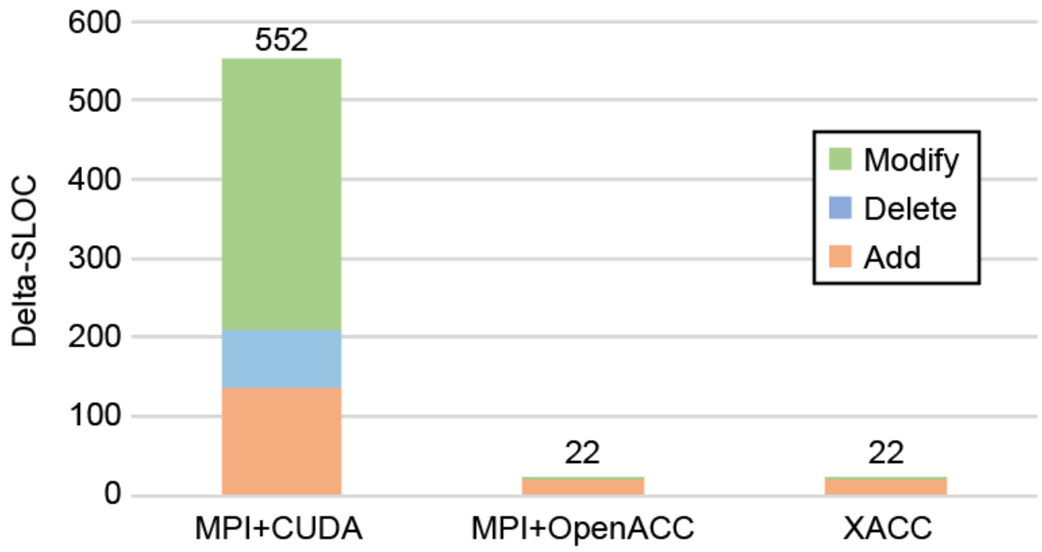
By using XACC, a programmer develops a parallel code by adding and modifying 160 lines to the serial code

SLOC of the serial code is 842

Breakdown of the results(1/2)

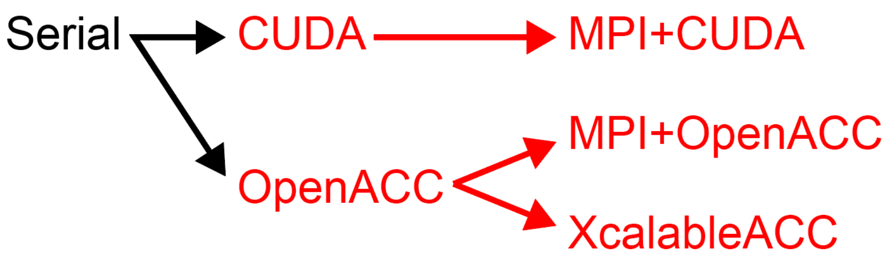
- **For accelerator**
  - CUDA requires large code changes
    - Add new kernel code for GPU
    - Modify and delete existing code
  - **OpenACC and XACC can reuse most of existing codes**
    - Basically, only add its directives

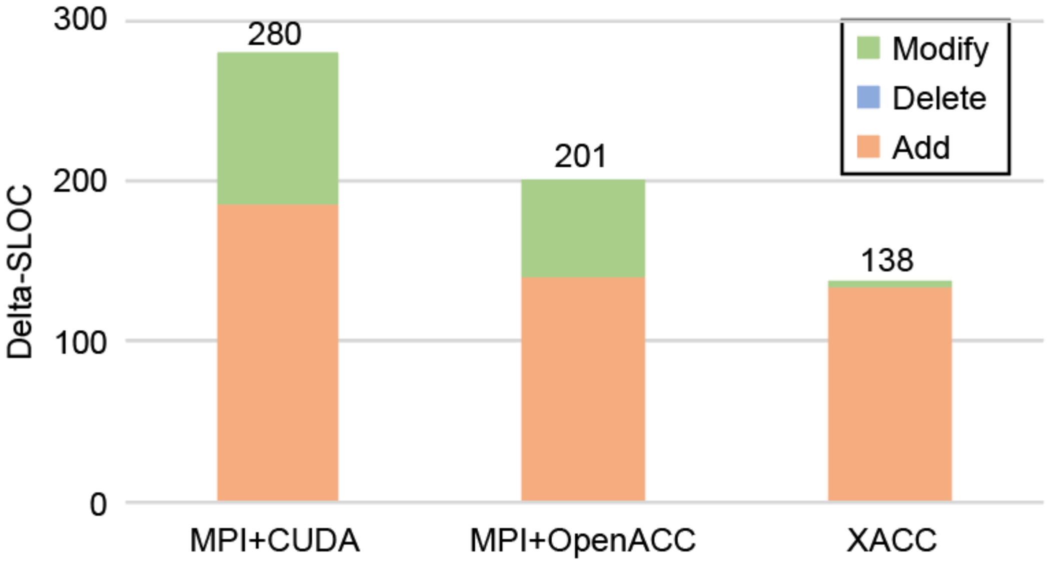




Breakdown of the results(2/2)

- **For distributed memory system**
  - MPI requires a lot of modifications which change to convert global indices into local indices
  - MPI also requires to add shadow area and communicate among neighborhood processes to update the shadow area
  - **XACC can easily perform them by adding its directives**





Performance evaluation environment

HA-PACS/TCA cluster system located at University of Tsukuba

CPU/Memory	Intel Xeon-E5 2680v2 2.8 GHz / DDR3 SDRAM 128GB 59.7GB/s x 2
GPU/Memory	NVIDIA Tesla K20X / GDDR5 6GB 250GB/s x 4
Network	InfiniBand Mellanox Connect-X3 4xQDR x 2rails 8GB/s

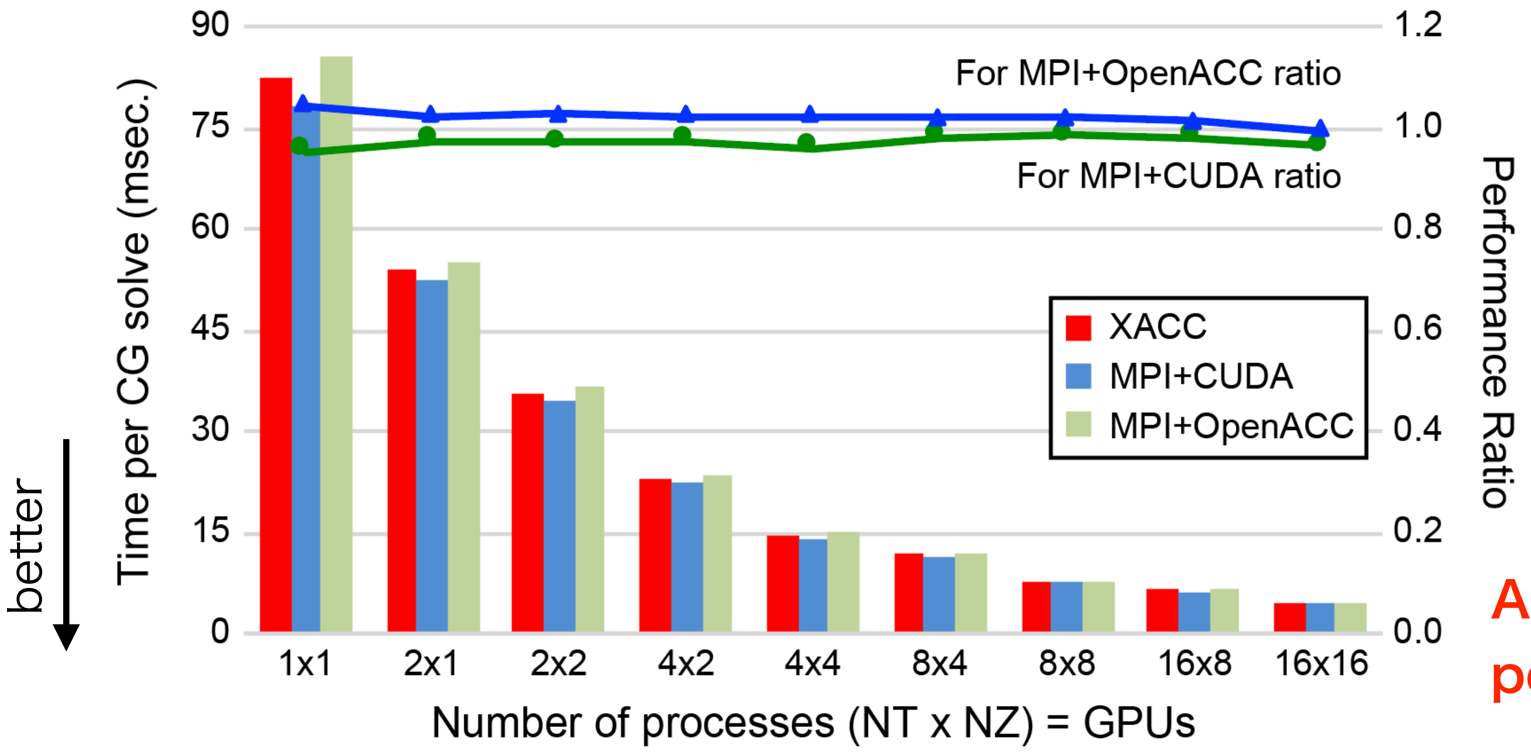
64 compute nodes, 256 GPUs

- Omni XACC compiler 1.1
- Omni OpenACC compiler 1.1
- Intel 16.0.2
- CUDA 7.5.18
- MVAPICH2 2.1

Performance result

Data size is 32x32x32x32 (T x Z x Y x X axes) with strong scaling

Each process deals with a single GPU, 4 processes run on a single compute node



Almost the same performance !!

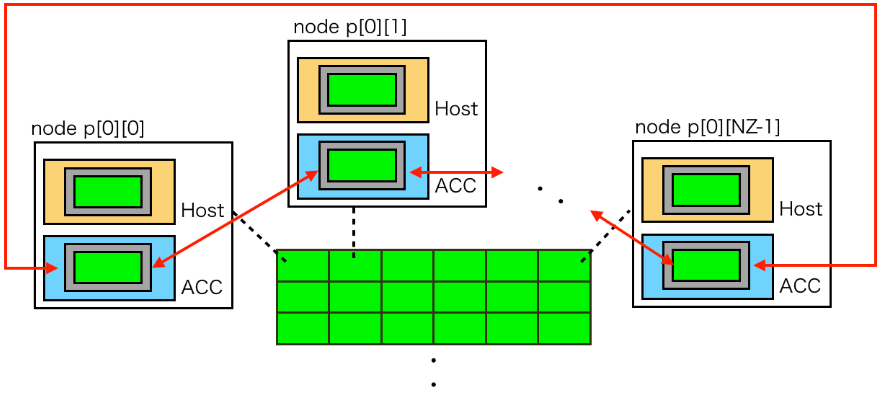
The performance of XACC is 100 - 104% of that of MPI+OpenACC, and 95 - 99% of that of MPI+CUDA

Discussion for performance result (1/3)

- Why is the performance of XACC a little **better** than that of MPI+OpenACC ?
    - The reason is due to how to update shadow area
    - The shadow updating process requires pack/unpack operations for non-contiguous regions.
        - The operations of XACC implementation are executed by XACC runtime
        - While XACC runtime is written in CUDA, MPI+OpenACC implementation uses OpenACC to perform the pack/unpack operations

another question !

Why was the performance difference caused between CUDA and OpenACC ?



Discussion for performance result (2/3)

- Why is the performance of XACC a little **worse** than that of MPI+CUDA ? (Why is OpenACC worse than CUDA ?)
  - The reason is due to how to use GPU threads in loop statements
  - In the MPI+CUDA implementation, loop iterations are assigned to GPU threads explicitly in a cyclic manner by the programmer.
  - In the XACC and OpenACC implementations, an implementation-dependent manner by an OpenACC compiler
    - In the Omni OpenACC compiler, loop iterations are assigned to GPU threads by a threadblock in a block manner, and then are also assigned to them by a vector in a cyclic manner

CUDA

```
int i = IDXV(threadIdx.x, blockIdx.x, blockDim.x);

while(ivx < NX*NY*NZ*NT){
    ivx += blockDim.x * gridDim.x;
      :
}
```

XACC or OpenACC

```
#pragma acc parallel loop ...
for(int t=0;t<NT;t++)
  for(int z=0;z<NZ;z++)
    for(int y=0;y<NY;y++)
      for(int x=0;x<NX;x++){
```

Discussion for performance result (3/3)

- With the gang clause with the static argument proposed in the OpenACC specification version 2.0, programmers can determine how to use GPU threads to some extent,
  - Omni OpenACC compiler does not yet support it.
    - This is a future work
  - In PGI compiler, performance decreases

```
#pragma acc parallel loop gang(static:1)
for(int t=0;t<NT;t++)
  for(int z=0;z<NZ;z++)
    for(int y=0;y<NY;y++)
      for(int x=0;x<NX;x++){
```

Summary

- **Objectives**
  - To develop applications on accelerated cluster in ease, we have designed the XACC language
  - To evaluate performance and productivity of XACC, we implement the Lattice QCD code
- **Results**
  - Delta-SLOC of XACC is **quite smaller** than those of MPI+CUDA and MPI+OpenACC
  - The performance of XACC is **almost the same** as those of MPI+CUDA and MPI+OpenACC on the accelerated cluster (64nodes, 256GPUs)