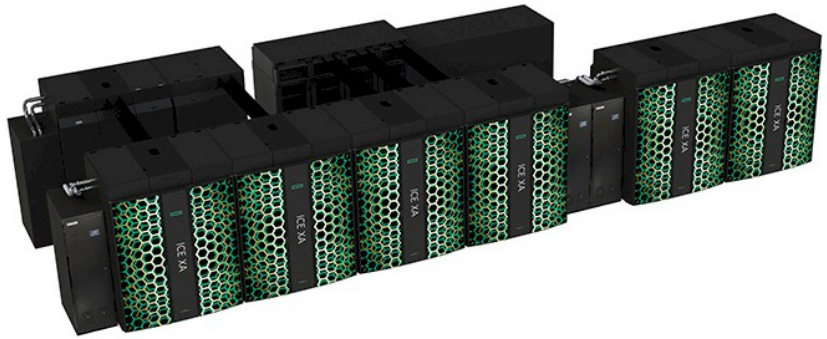
XcalableACCの概要

中尾 昌広（理化学研究所 計算科学研究機構）

背景（1/2）

- アクセラレータを搭載したクラスタシステム
  - High computing performance
  - High energy efficiency
- Top500やGreen500の上位のほとんどは
  アクセラレータクラスタ



バックエンドサブシステムB：3.0ペタフロップス
2基のCPUと4基のGPUを搭載したノード 128台で構成

背景（2/2）

- MPI+CUDAによるプログラミングが主流
  - しかし，プログラミングが難しい
  - MPIでデータ転送と分割を行い，CUDAで計算は記述（逐次コードと異なる）

```
__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

OpenACCがCUDAの代わりに利用するケースもあるが，MPIのプログラミングの難しさは変わらない

XcalableACC (XACC)

- **アクセラレータ用並列言語XcalableACC (XACC)**
  - XcalableMP（XMP）のアクセラレータ拡張
  - CとFortranに対応
  - XMPプログラム中にOpenACC指示文の記述を許す
  - XMPによって各ノードに分散された配列や演算を対象に，OpenACCによるデータ移動・演算オフローディングを行う
  - XMP指示文とOpenACC指示文は基本的に直交関係だが，単純なXMPとOpenACCの組合せではなく、協調動作できるように工夫
    - XACC = XMP + OpenACC + XACC extensions

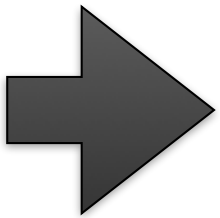
XMP	クラスタ用並列言語（MPIの代わりに利用)
OpenACC	アクセラレータ用並列言語（CUDAの代わりに利用）
XACC extensions	アクセラレータ間の通信など

XcalableMP (XMP) http://xcalablemp.org

- クラスタシステムのための指示文ベースの並列言語
  - C, Fortran, (C++)
  - 指示文を逐次コードに追記．片側通信記法もサポート
  - PCクラスタコンソーシアムが仕様を策定

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

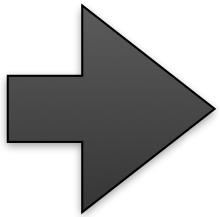
Parallelize loop & reduction

OpenACC https://www.openacc.org

- アクセラレータ用の指示文ベースの並列言語
  - C, Fortran, C++
  - 演算をアクセラレータにオフロード
  - GPU以外のアクセラレータにも対応可能な汎用的な仕様

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];
  :
```

Copy data

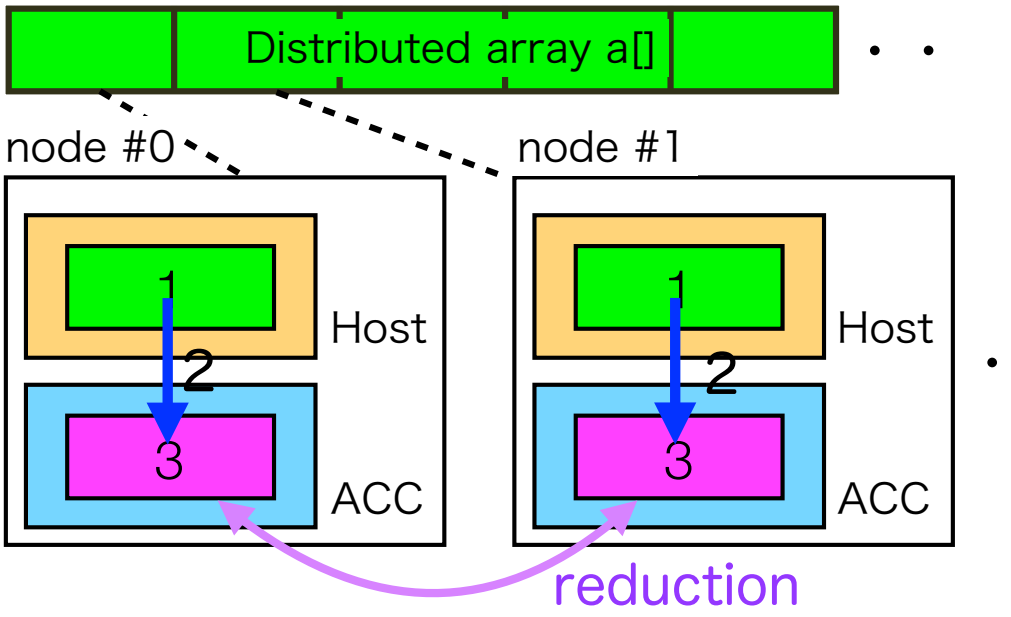
Parallelize loop & reduction

XcalableACC (XACC)

- XACCはOpenACCを利用したXMPのアクセラレータ拡張
  - XMP：データの分散やワークマッピングなど
  - OpenACC：アクセラレータの利用

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];
  :
```

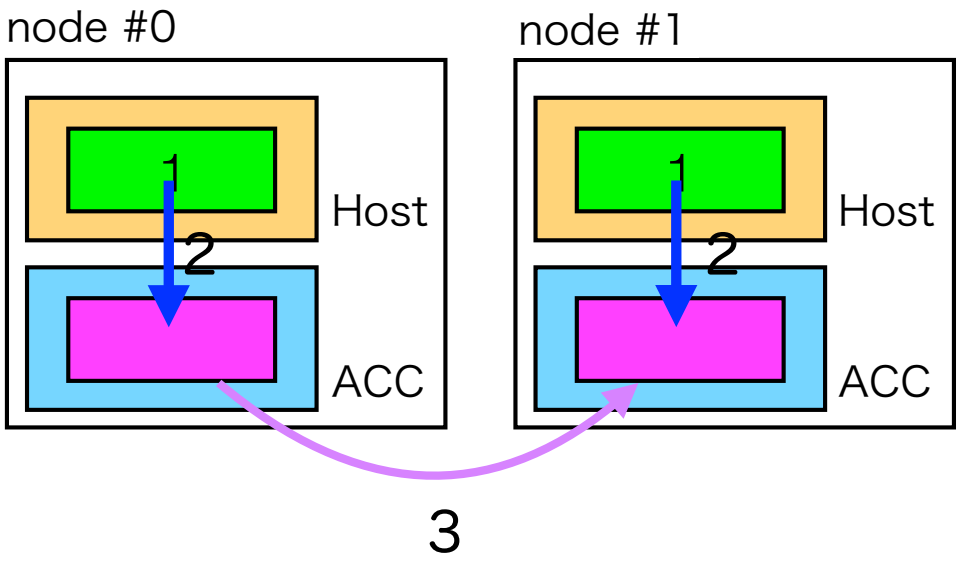


XcalableACC (XACC)

- XACCはOpenACCを利用したXMPのアクセラレータ拡張
  - XMP：データの分散やワークマッピングなど
  - OpenACC：アクセラレータの利用

Parallell code in XACC/C

```
int a[N]:[*];   // Declare coarray                    1
int b[N];
#pragma acc declare create(a, b)                      2

if(xmpc_this_image() == 0){
#pragma acc host_data use_device(a, b)
  a[:]:[1] = b[:];                                    3
}
```

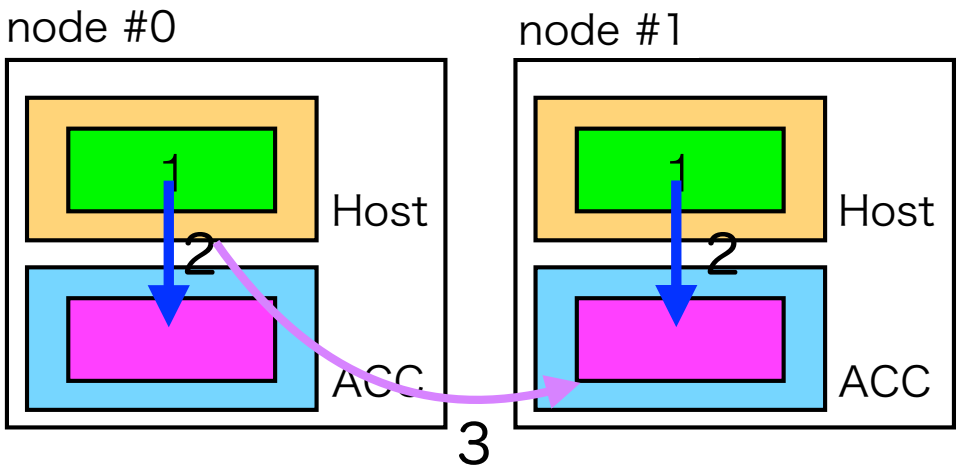


XcalableACC (XACC)

- XACCはOpenACCを利用したXMPのアクセラレータ拡張
  - XMP：データの分散やワークマッピングなど
  - OpenACC：アクセラレータの利用

Parallell code in XACC/C

```
int a[N]:[*];   // Declare coarray                    1
int b[N];
#pragma acc declare create(a)                         2

if(xmpc_this_image() == 0){
#pragma acc host_data use_device(a)
  a[:]:[1] = b[:];                                    3
}
```

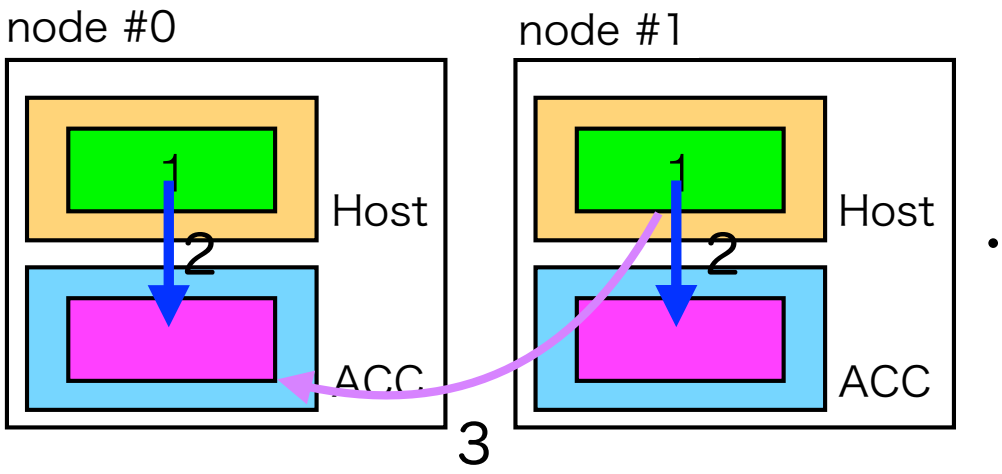


XcalableACC (XACC)

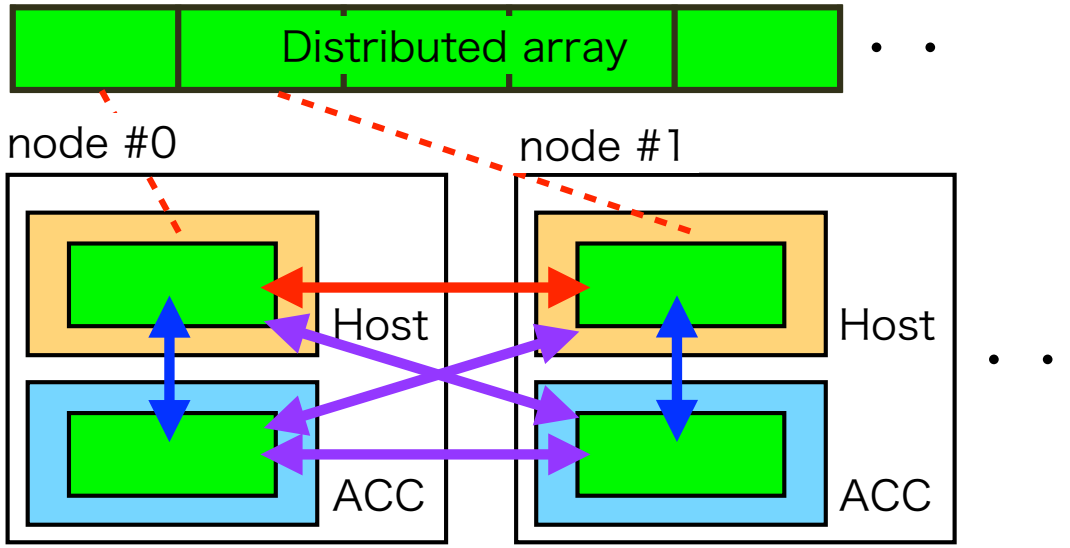
- XACCはOpenACCを利用したXMPのアクセラレータ拡張
  - XMP：データの分散やワークマッピングなど
  - OpenACC：アクセラレータの利用

Parallell code in XACC/C

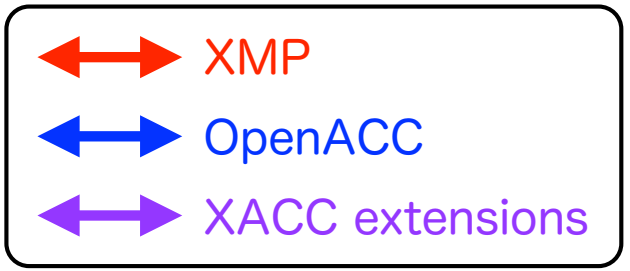
```
int a[N]:[*];   // Declare coarray                    1
int b[N];
#pragma acc declare create(a)                         2

if(xmpc_this_image() == 0){
#pragma acc host_data use_device(a)
  a[:] = b[:]:[1];                                    3
}
```



XACCのメモリモデル



XMP and **XACC** は放送通信、集約通信，隣接通信，片側通信など多様な通信を提供



XACC = XMP + OpenACC + XACC extensions

- **XMP**：異なるノード間のホストメモリ間の通信
- **OpenACC**：同じノード内のホストとアクセラレータ間の通信
- **XACC extensions**：異なるノード間のアクセラレータ間およびホストとアクセラレータ間の通信

XcalableACCのプログラミング例

```
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次元ラプラス方程式

XcalableACCのプログラミング例

```
double u[XSIZE][YSIZE], uu[XSIZE][YSIZE];
#pragma xmp nodes p[NY][NX]
#pragma xmp template t[YSIZE][XSIZE]
#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]
…


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

   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[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次元ラプラス方程式

2次元分散配列と袖の定義

配列uuの袖交換

XcalableACCのプログラミング例

```
double u[XSIZE][YSIZE], uu[XSIZE][YSIZE];
#pragma xmp nodes p[NY][NX]
#pragma xmp template t[YSIZE][XSIZE]
#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]
…
#pragma acc data copy(u) copyin(uu)
{
  for(k=0; k<MAX_ITER; k++){
#pragma xmp loop (y,x) on t[x][y]
#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[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次元ラプラス方程式

2次元分散配列と袖の定義

分散配列をアクセラレータのメモリに転送

XMP指示文で分散したループをOpenACC指示文が分散して処理

配列uuの袖交換

acc節を指定すると，アクセラレータ上のデータが送信される

XcalableACCのプログラミング例

```
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
```

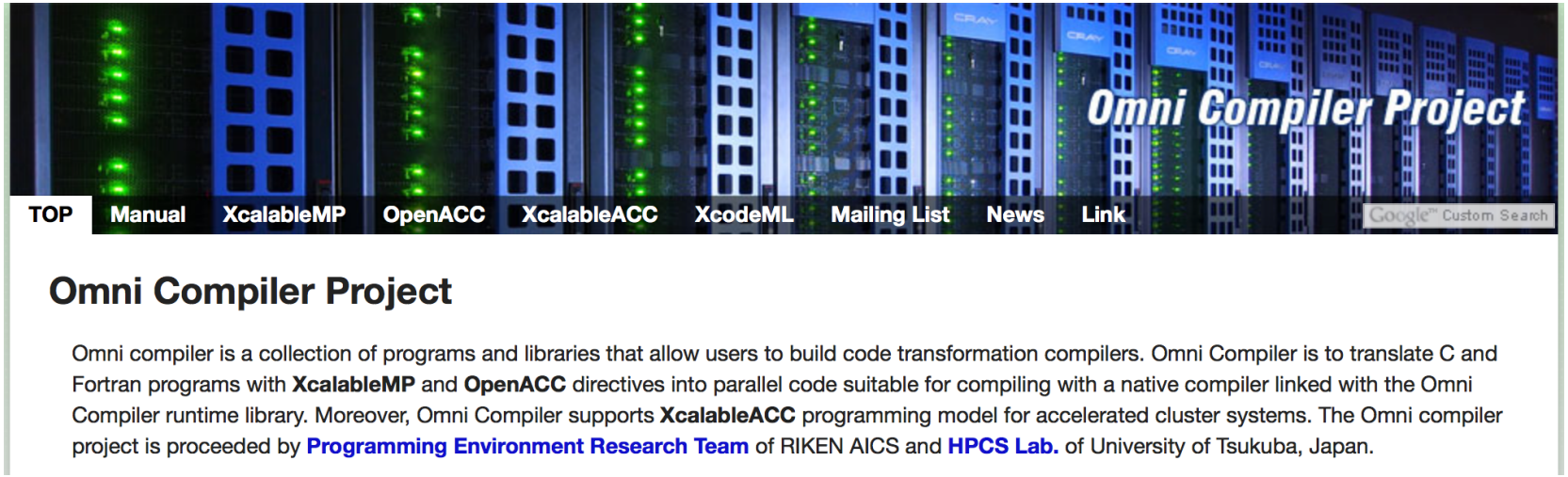
2次元ラプラス方程式

分散配列をアクセラレータのメモリに転送

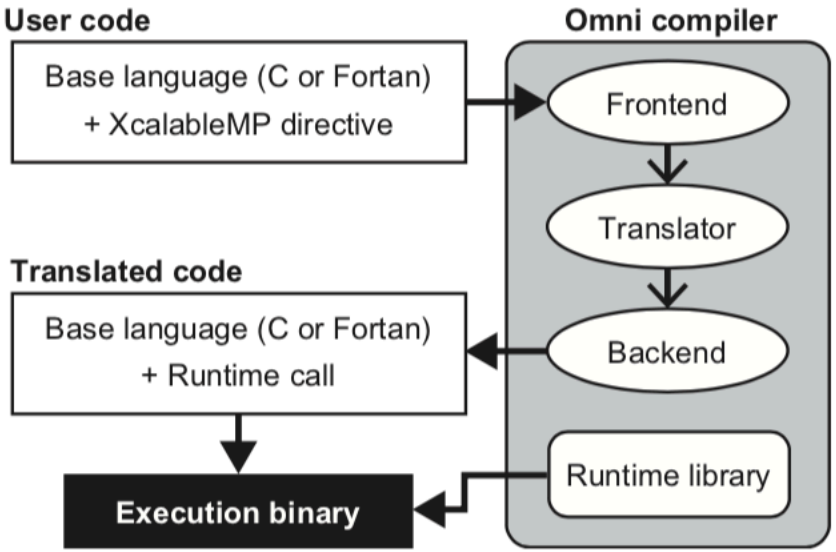
XMP指示文で分散したループをOpenACC指示文が分散して処理

Omni compiler

https://omni-compiler.org

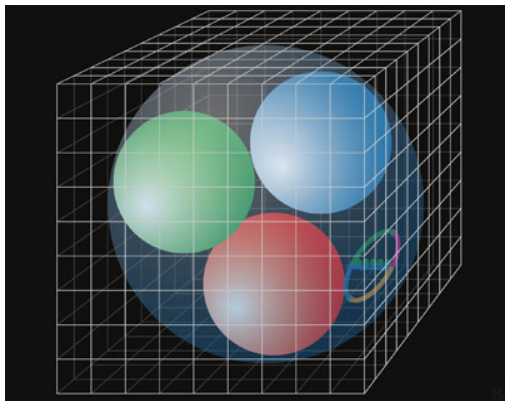


- Source-to-source compiler
  - XMP，XACC，OpenACC, OpenMPに対応
  - Omni XACC compilerを使う際, OpenACC compilerは任意のものを利用可能（PGI，Crayなど）
- オープンソースソフトウェア



Lattice QCDアプリケーション

- HPC分野で重要なアプリケーションの1つ
- QCD（Quantum Chromo-Dynamics：量子色力学）は物質の最小単位であるクォークと，クォーク間における相互作用を結ぶグルーオン（糊粒子）を表す基本方程式

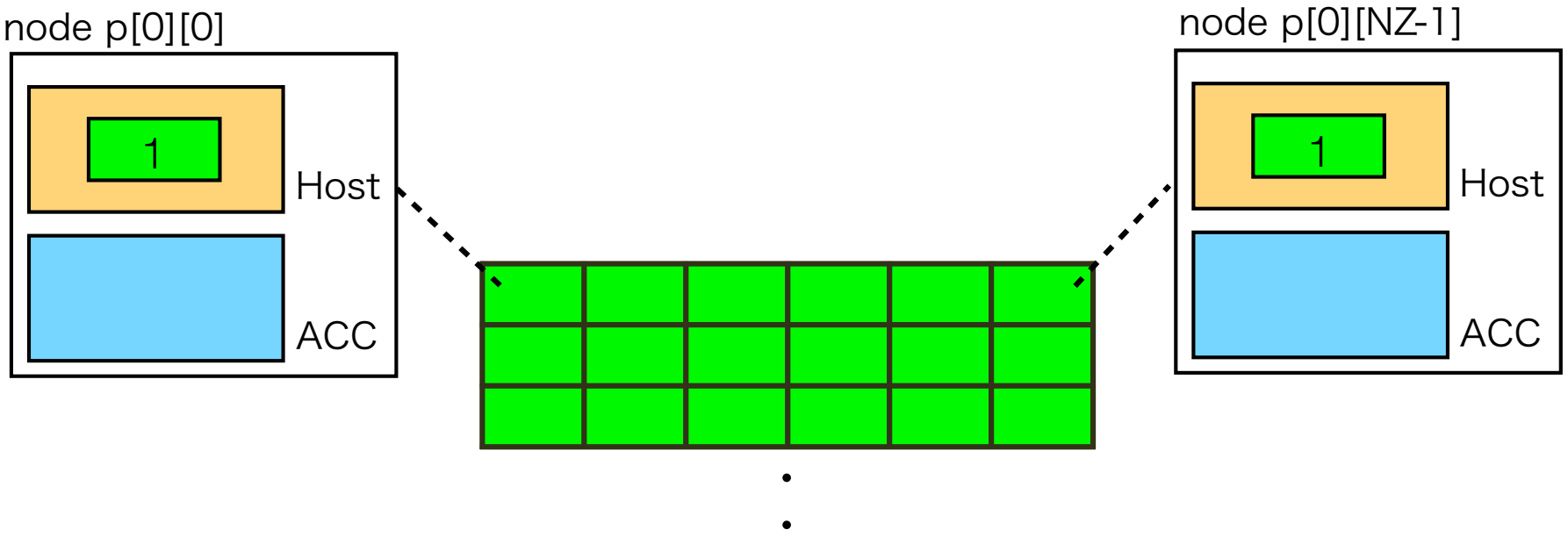


- 格子QCDは4次元（時間+XYZ軸）の格子上で行うQCDのシミュレーション
- 格子QCDミニアプリケーション
  - C言語，逐次コード，840行程度
  - 実アプリケーションBridge++のカーネル部分を抽出
  - http://research.kek.jp/people/matufuru/Research/Programs/index.html

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)
```

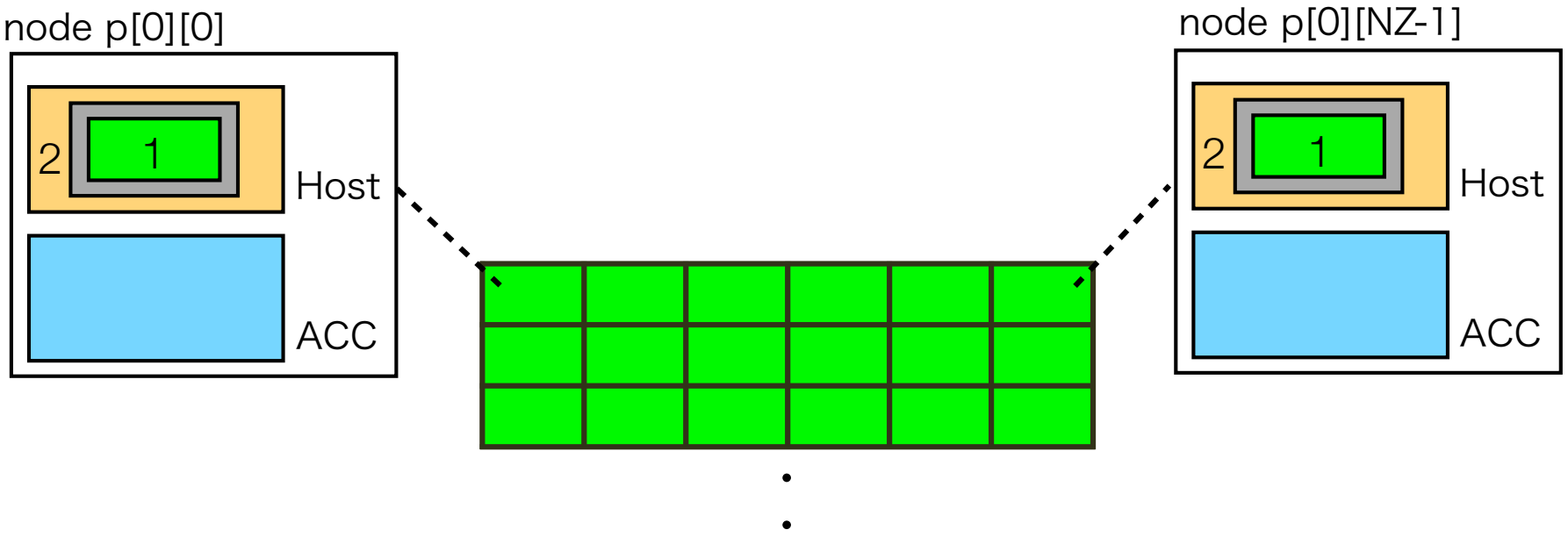
1. 2次元プロセスグリッドを作成し，クォークの4次元配列を各プロセスに分散させる
2. ステンシル計算のために，袖領域を分散配列に確保
3. 分散配列をアクセラレータに転送



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)
```

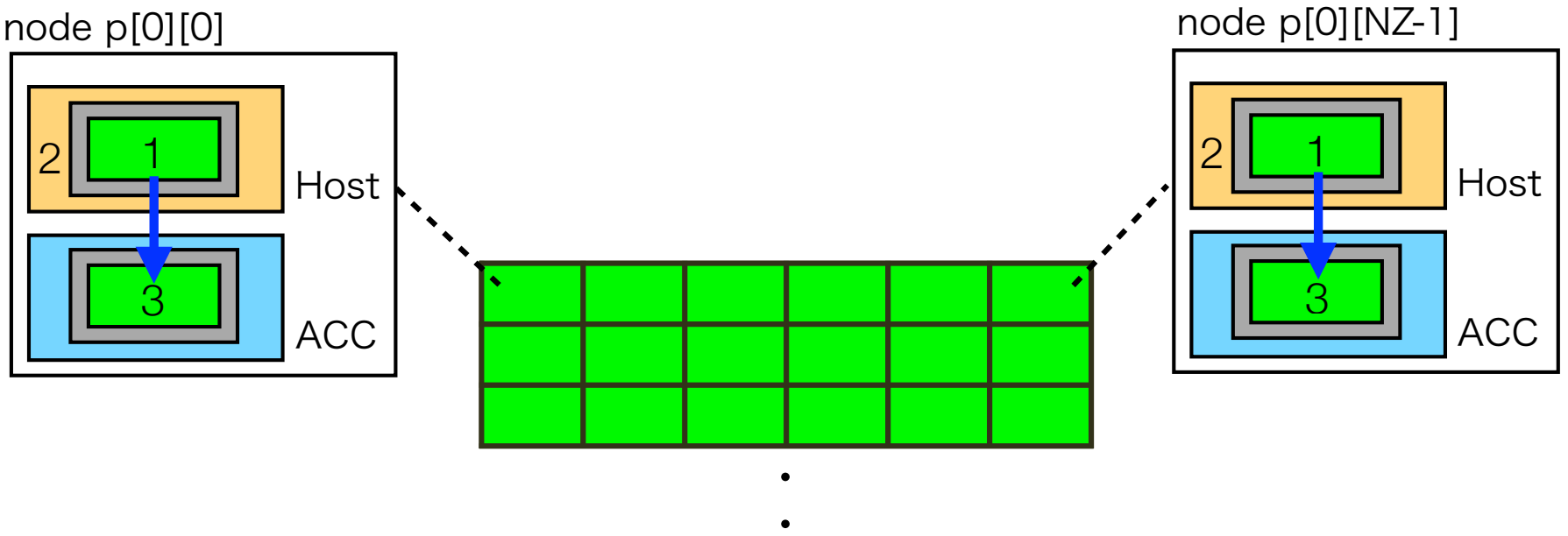
1. 2次元プロセスグリッドを作成し，クォークの4次元配列を各プロセスに分散させる
2. ステンシル計算のために，袖領域を分散配列に確保
3. 分散配列をアクセラレータに転送



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 shadow v[1][1][0][0]
#pragma acc enter data copyin(v)
```

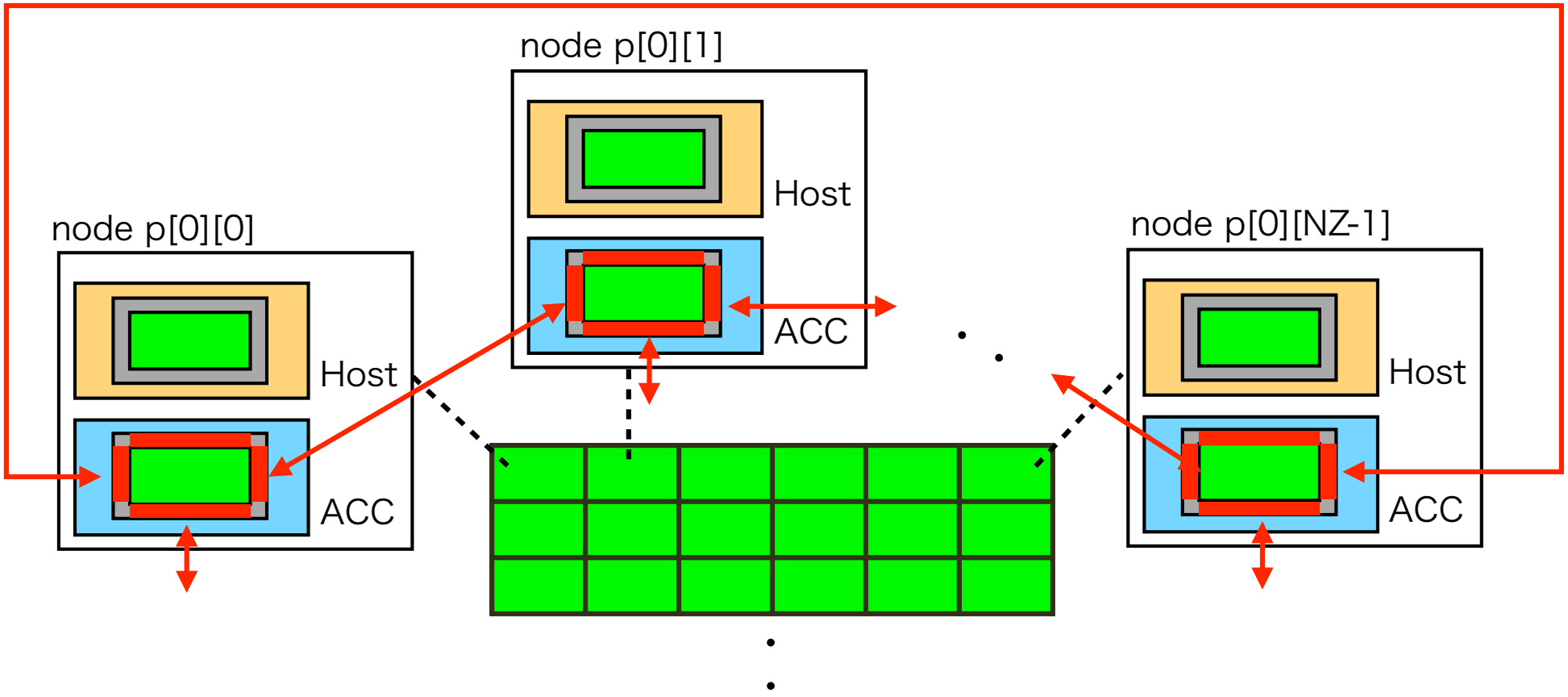
1. 2次元プロセスグリッドを作成し，クォークの4次元配列を各プロセスに分散させる
2. ステンシル計算のために，袖領域を分散配列に確保
3. 分散配列をアクセラレータに転送



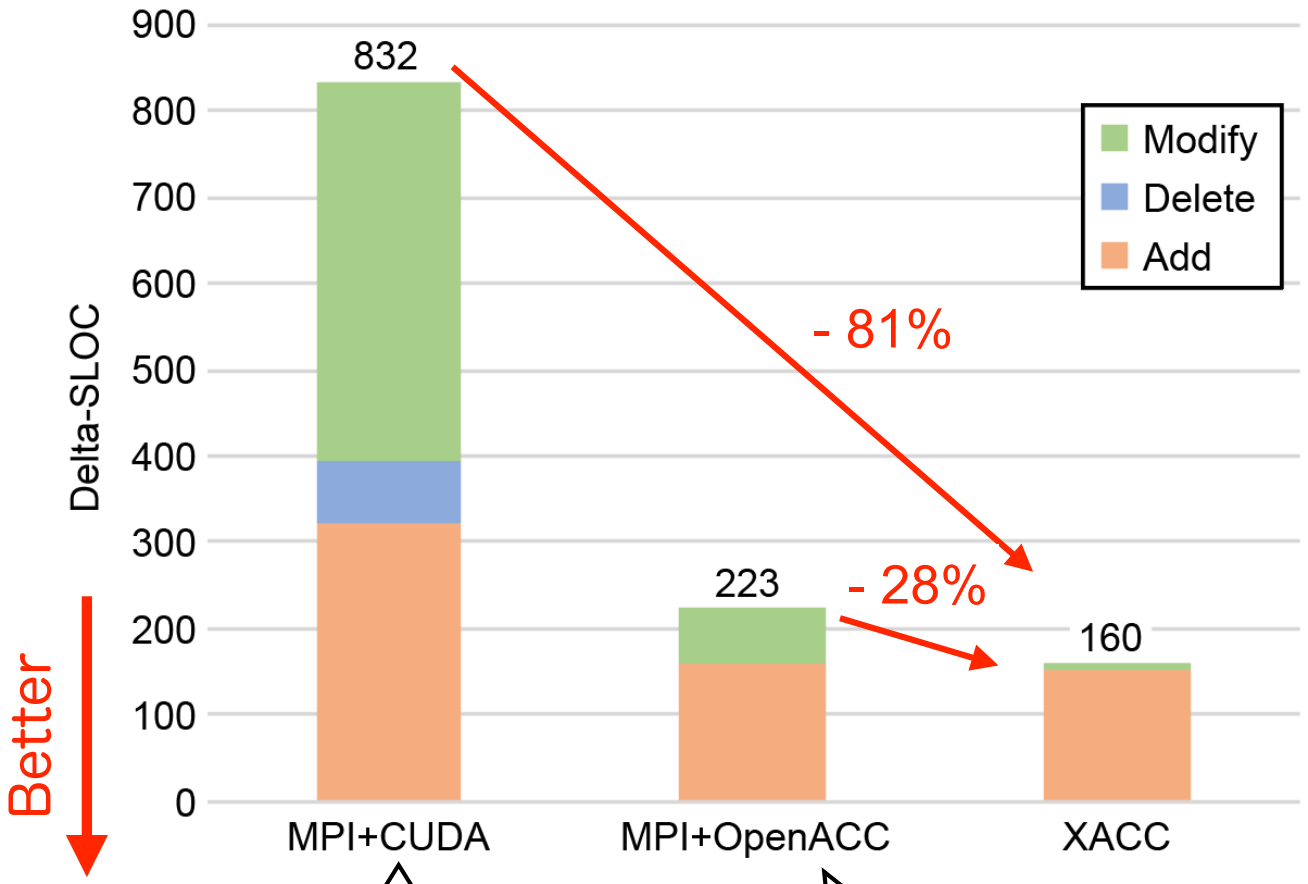
Neighborhood communication

XMP reflect指示文により，アクセラレータ上の袖領域の更新を行う

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



Productivity results



- 袖領域の更新
- インデックスの変換
- カーネル関数の作成

- 袖領域の更新
- インデックスの変換
- 指示文の追加

- 指示文の追加

Delta-SLOC: 逐次コードからの変更した行をカウント

Delta-SLOCが小さいほど、バグが入る可能性やプログラミングコストは低いと言える

定量的評価

定性的評価

Performance evaluation environment

HA-PACS/TCAクラスタシステム@筑波大学

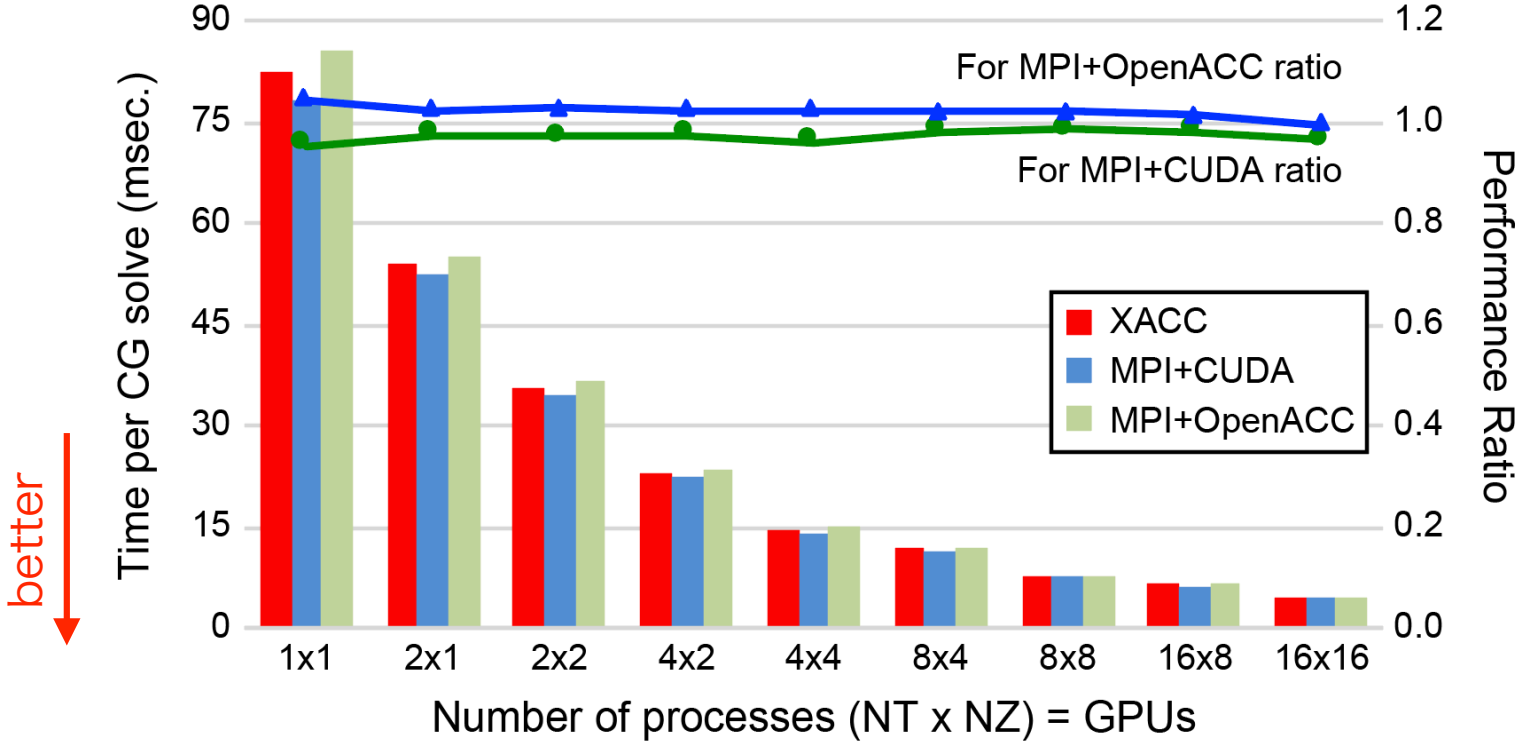
CPU/Memory	Intel Xeon-E5 2680v2 2.8 GHz / DDR3 SDRAM 128GB 59.7GB/s x 2
GPU/Memory	NVIDIA Tesla K20X / GDDR5 6GB 250GB/s x 4
Network	InfiniBand Mellanox Connect-X3 4xQDR x 2rails 8GB/s

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

データサイズは32x32x32x32 (T x Z x Y x X axes)．強スケーリング．
1計算ノードにつき4プロセスをアサイン．各プロセスは1つのGPUを操作．

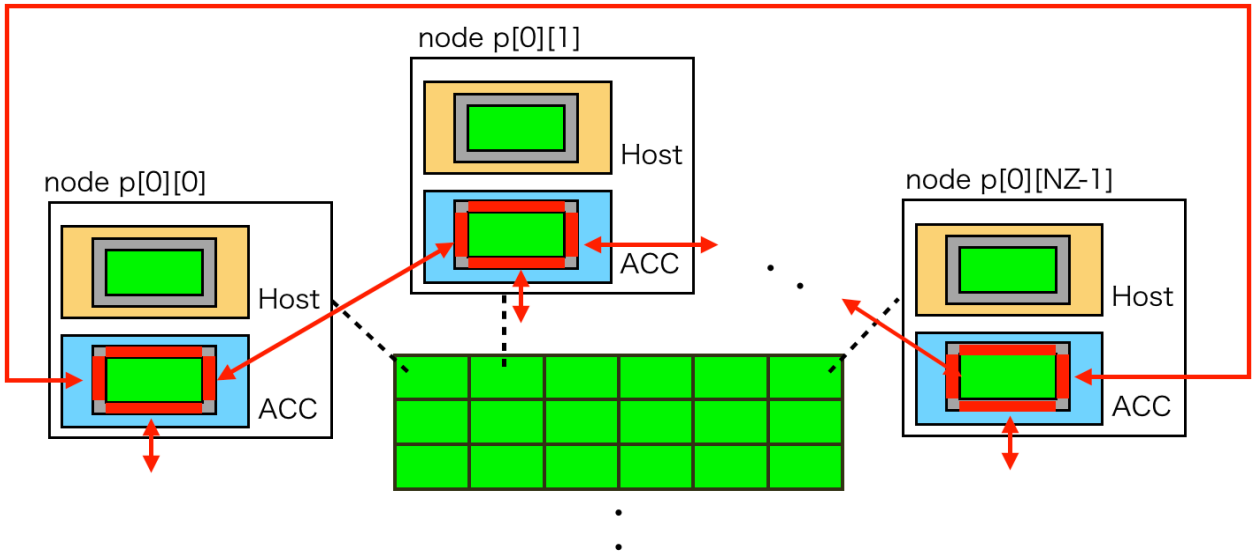


XACCの性能はMPI+OpenACCの100 - 104%の性能であり，MPI+CUDAの95 - 99%の性能（MPI+OpenACC <= XACCの性能 < MPI+CUDA）

Discussion for performance result

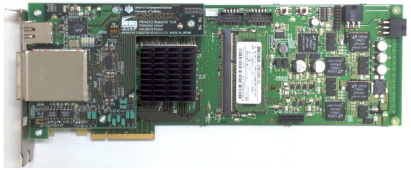
- XACCの性能がMPI+OpenACCの性能よりも良かった理由は？
  - 袖領域の更新には非連続領域に対するpack/unpackが必要
  - XACCは，その計算はOmni compiler内のXACCランタイムが行う
    - relfect指示文が自動的にそのランタイムを呼び出す
  - XACCランタイムはCUDAで作成されているのに対し，MPI+OpenACCのLattice QCDのpack/unpackの箇所はOpenACCで作成している

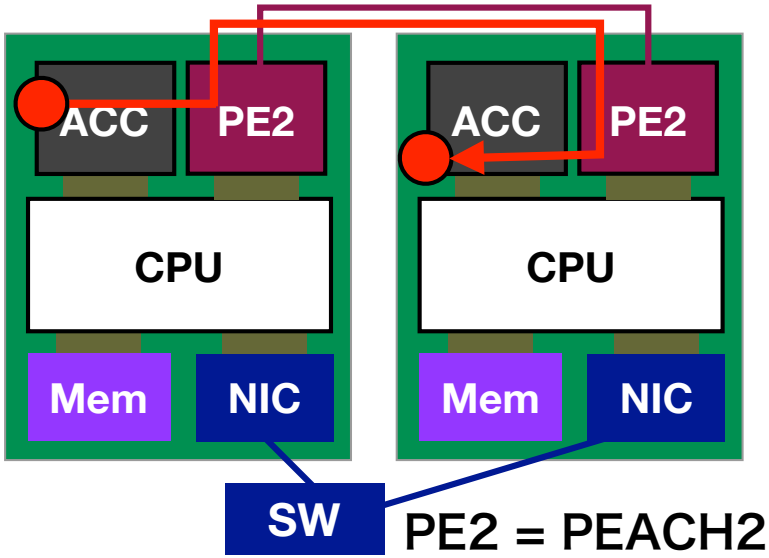
```
#pragma xmp reflect(v) width(/periodic/1:1,/periodic/1:1,0,0) orthogonal acc
Wilson_Dirac(..., v); // Stencil calculation
```



Tightly Coupled Accelerators (TCA)

- Communication architecture based on PCIe [1] technology
  - Developed by HA-PACS Project in Univ, of Tsukuba, Japan
  - Nodes are connected using PCIe external cable through **PEACH2,** which is a TCA interface Board
  - Direct, low latency data transfers among accelerator memories
    - No host memory copies
    - No MPI software stack
    - No protocol conversions

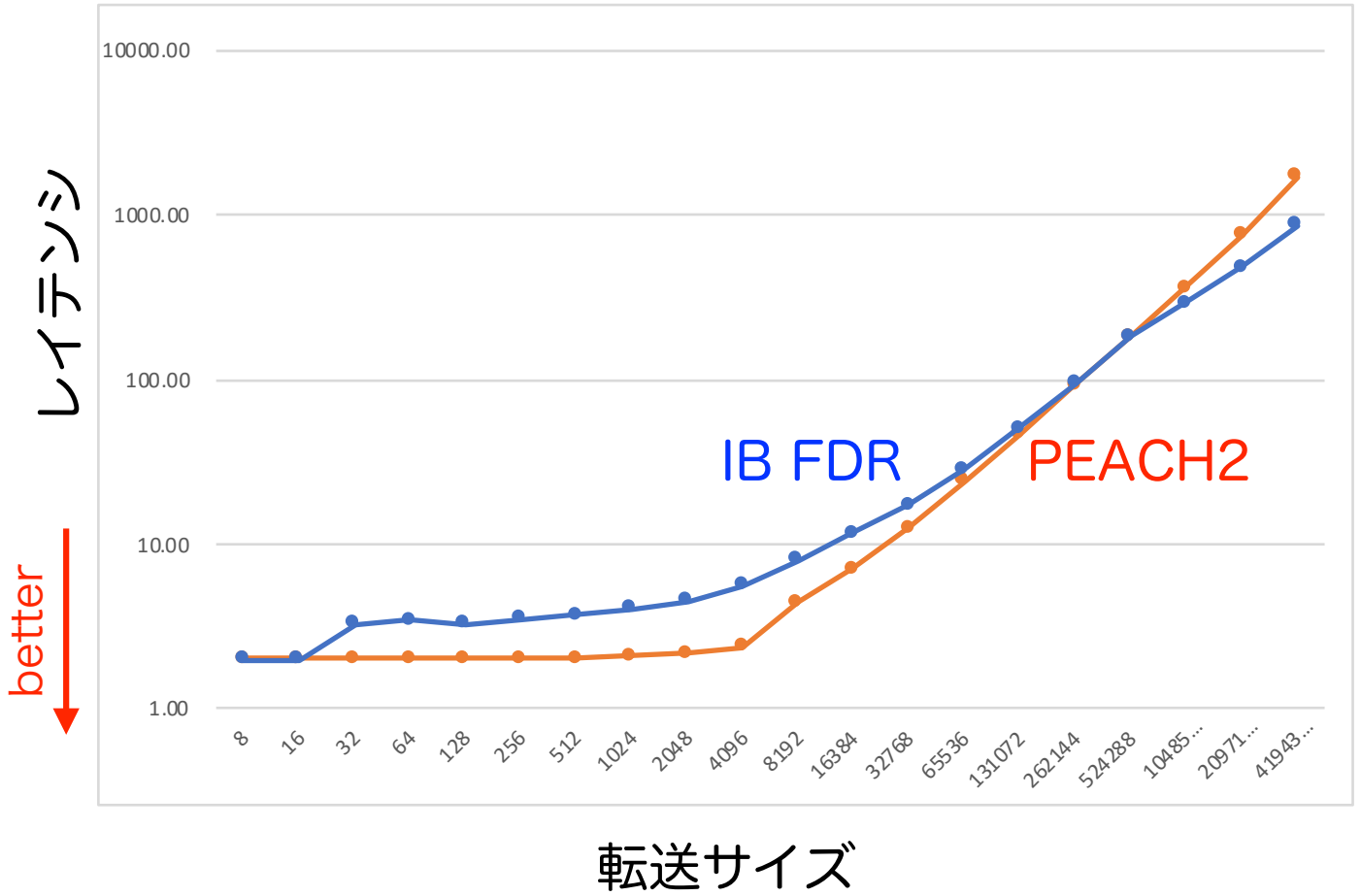




[1] Toshihiro Hanawa et al. "Tightly Coupled Accelerators Architecture for Minimizing Communication Latency among Accelerators," in IPDPSW '13 Proceedings of the 2013

Tightly Coupled Accelerators (TCA)

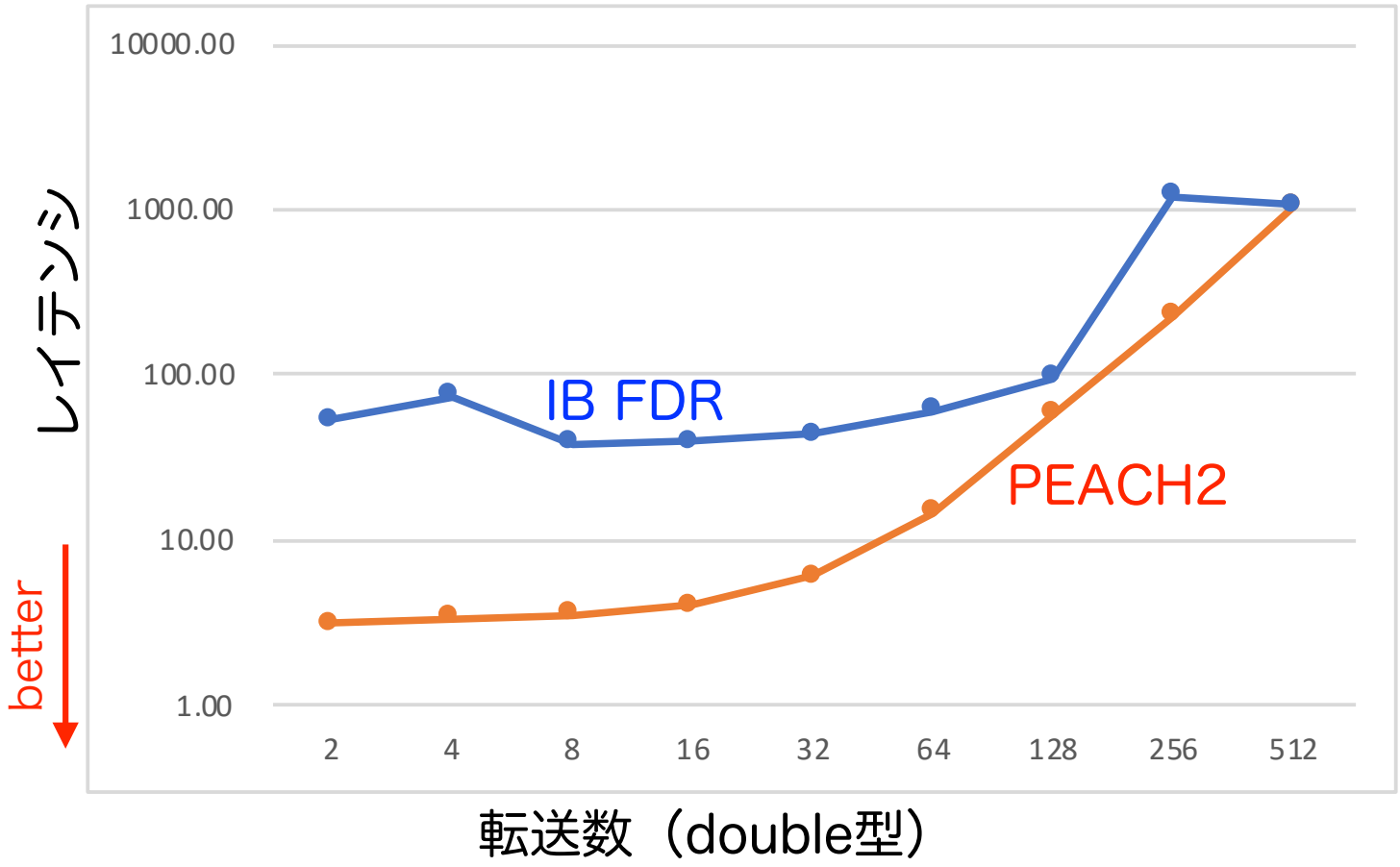
- Pingpong (PEACH2 v.s. InfiniBand FDR)



32 ～64K Byteの小・中サイズの転送はPEACH2が有利

Tightly Coupled Accelerators (TCA)

- ブロックストライド通信 (PEACH2 v.s. InfiniBand FDR)



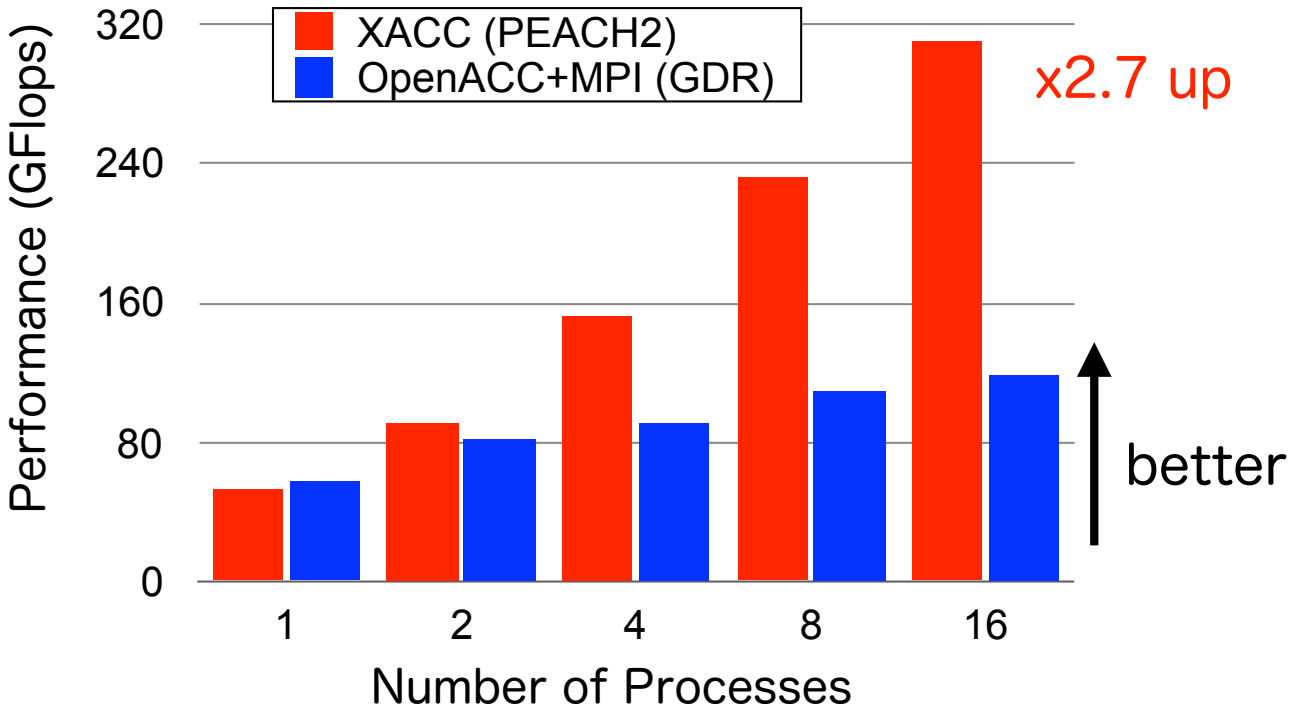
常にPEACH2が有利

姫野ベンチマーク

HA-PACS, Univ. of Tsukuba

NVIDIA K20X
InfiniBand 4xQDR x 2rails
PCIe Gen2 x8 for PEACH2
MVAPICH-GDR2.0b
gcc-4.7, CUDA6.0, Omni OpenACC Compiler 0.9b

Array size 128x128x256 : Strong scaling, 1GPU/procs



- In XACC, all communication uses the **PEACH2**
- **PEACH2** vs. **GPUDirect RDMA over InfiniBand**

XACC利用のメリット

- XMPコードがベースなものについて，XMPからのincrementalなアクセラレータ向けコード開発が楽である
- OpenACCコードがベースなものについて，分散メモリ並列によるGPUクラスタへの適用が楽である
- GPUオフローディングとノード間通信を1つの言語処理系が扱うため，通信を含めた最適化やTCAのような専用ハードウェア通信機構への対応が可能である
- 基本的に上位概念に近い形で書いたコードの方が各世代のハードウェア特性や制約に対応した最適化が可能（多次元配列がどのように各ノードに分散されているかはコンパイル時に解析できる）

まとめ

- **目的**
  - アクセラレータクラスタにおける生産性向上
  - XACCの性能と生産性を評価するために，Lattice QCDコードの作成
  - 通信ハードウェアであるPEACH2を利用した評価

- **結果**
  - XACCの生産性は，既存のプログラミングモデルであるMPI+CUDAやMPI+OpenACCよりも良い
  - XACCの性能を評価した結果，XACCの性能はMPI+OpenACCの100 - 104%の性能であり，MPI+CUDAの95 - 99%の性能であった (64nodes, 256GPUs)
  - PEACH2を用いた姫野ベンチマークでは，XACCの性能はMPI+OpenACCの2.7倍であった

Thank you for listening

Q & A