アクセラレータクラスタ向け PGAS言語XcalableACCの実装状況報告

田渕晶大[†1]、中尾昌広[†2]、村井均[†2]、朴泰祐[†1,3]、佐藤三久[†1,2]

[†1] 筑波大学大学院システム情報工学研究科
[†2] 国立研究開発法人理化学研究所計算科学研究機構
[†3] 筑波大学計算科学研究センター

概要

- 背景・目的
- XcalableACC (XACC)
- Omni XACC compiler
- Omni XACC compiler の実装状況
- 評価
  - 姫野ベンチマーク
  - NPB-CG
- まとめ・今後の予定

背景と目的

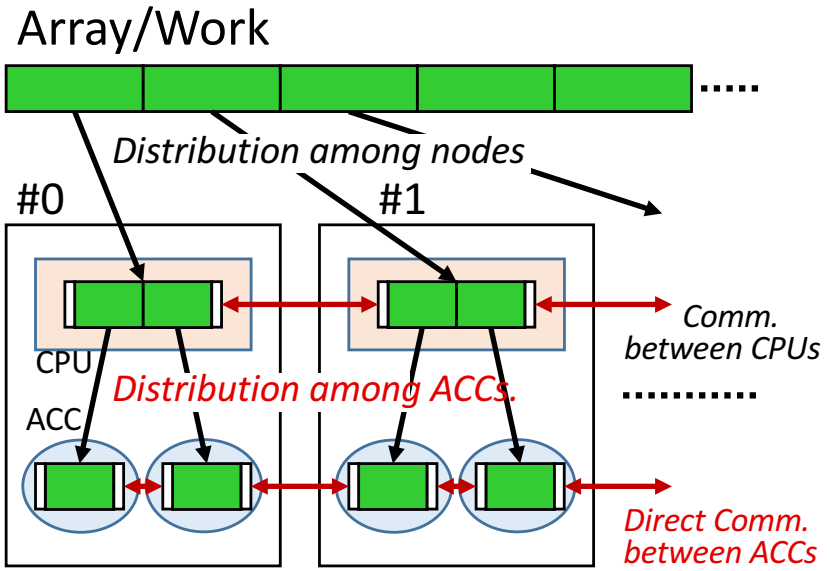
- アクセラレータ (GPU, MIC など) をもつクラスタが増加
- アクセラレータのプログラミング
  - CUDA、OpenCL
  - OpenACC、OpenMP 4.0 (指示文ベース)
- ノード間通信・同期のプログラミング
  - MPI
  - XcalableMP (XMP) (指示文ベース)

- ハイブリッドプログラミング
  - MPI + CUDA ・・・ MPI, CUDAともに記述が煩雑
  - XMP + OpenACC ・・・ 指示文で簡易に記述できるが、アクセラレータ間の通信指示文がない

アクセラレータクラスタ向けに性能・生産性の高いプログラミングモデルが必要

XcalableACC (XACC)

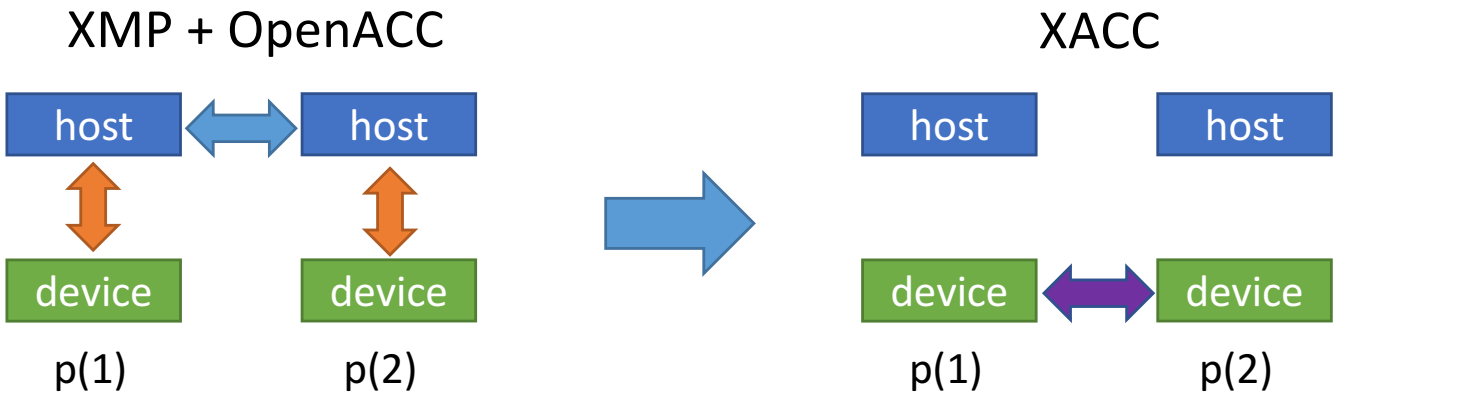
- XMP と OpenACC を統合した PGAS 言語
  - 理研・筑波大で開発中
  - XMP・・・分散メモリプログラミング
  - OpenACC・・・アクセラレータプログラミング
  - 単に XMP + OpenACC ではない
- デバイスメモリ間の通信
  - ホストを介さない通信が可能
- 複数デバイスサポート
  - OpenACCでは書きにくい1プロセスから複数デバイスの利用が簡単に可能



XACC の機能 - デバイス間通信

- XMPの通信の拡張
  - #pragma xmp [reflect|gmove|reduction|bcast] ... *acc*

- 記述が簡易
- 実装が適切な通信を発行可能 (e.g. GPUDirect for RDMA)




```
#pragma acc update host(sum)
#pragma xmp reduction(+: sum)
#pragma acc update device(sum)
```

```
#pragma xmp reduction(+: sum) acc
```

XACC の機能 - デバイス間通信

- Coarray にも対応
  - OpenACC の仕様にはまだ Fortran の Coarray に関する記述はない

Fortran

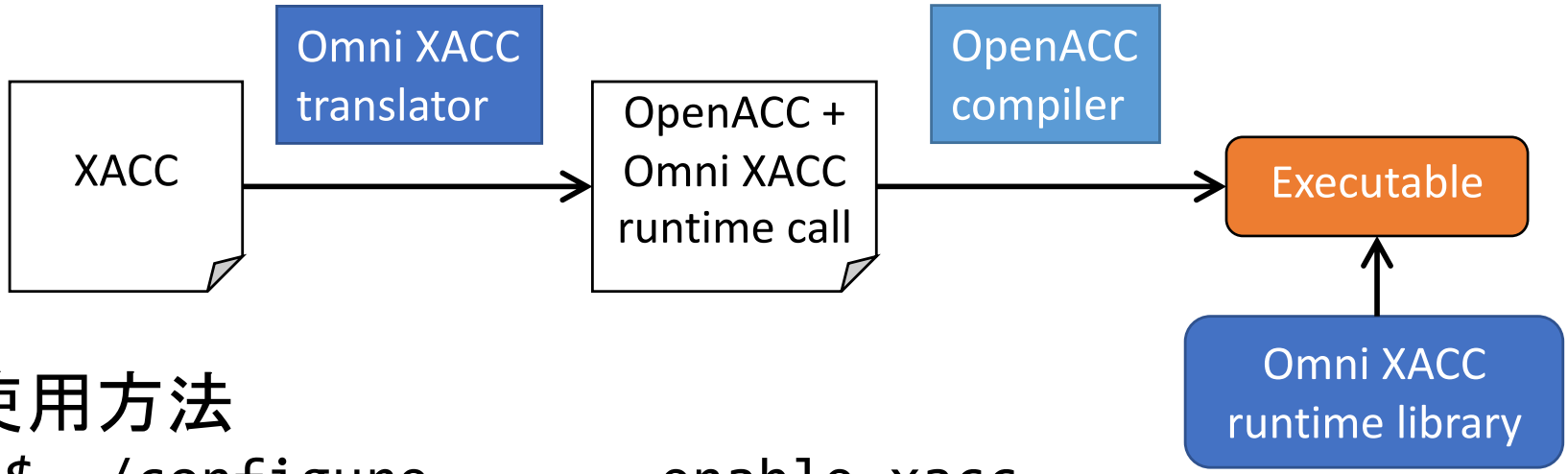
```
real a(N)[*]
!$acc declare create(a)
...
if(this_image() == 1) then
   !$acc host_data use_device(a)
   a(:)[2] = a(:)
end if
```

C

```
double a[N]:[*];
#pragma acc declare create(a)
...
if( xmp_node_num() == 1){
   #pragma acc host_data use_device(a)
   a[:]:[2] = a[:];
}
```

Omni XACC compiler

- Omni compiler で実装した XACC コンパイラ
  - Omni XMP compiler の拡張
  - ソースtoソースコンパイラ：XACC → OpenACC
  - 一般的な OpenACC コンパイラを利用可能



- 使用方法

```
$ ./configure ... --enable-xacc
$ xmpcc ... -xacc
```

Omni XACC compiler の実装状況

- コード変換 (translator)
  - 指示文の変換は実装済み
  - Coarray の変換は C のみ対応
- デバイス間通信 (runtime library)
  - 現在は NVIDIA GPU のみ対象
  - MPI 版
    - CUDA aware MPI を想定
    - おおよそ実装済み (次のスライドに詳細)
  - TCA 版
    - reflect のみ
  - MPI + TCA 版
    - reflect, reduction のみ

MPI版デバイス間通信

- 実装済み機能

通信	C	Fortran
reflect	○	○
reduction	○	○
bcast	○	×
gmove	△	×
Coarray	○	×

性能評価

- ベンチマーク
  - Himeno Benchmark
  - NAS Parallel Benchmarks CG (NPB-CG)
- 3種類のコードを比較
  - MPI + OpenACC (MPI+ACC)
  - XACC global-view (XACC-G)
  - XACC local-view (XACC-L)
- HA-PACS/TCAで評価
  - 1GPU / MPIプロセス

HA-PACS/TCA の構成

CPU	Intel Xeon-E5 2680v2 2.8GHz x 2
Memory	DDR3 1866MHz, 128GB
GPU	Tesla K20X x 4
Interconnect	InfiniBand Mellanox Connect-X3 Dual-port QDR
Compiler	GCC 4.4.7, CUDA 7.5, MVAPICH2-GDR 2.2rc1 Omni Compiler 1.0.3 相当

Himeno Benchmark

- 非圧縮流体解析コードのベンチマーク

- サイズ:
  - Large (256 x 256 x 512)
- 分割
  - i, j次元の2次元分割
- 主な処理
  - 演算:19点ステンシル
  - 通信:袖通信

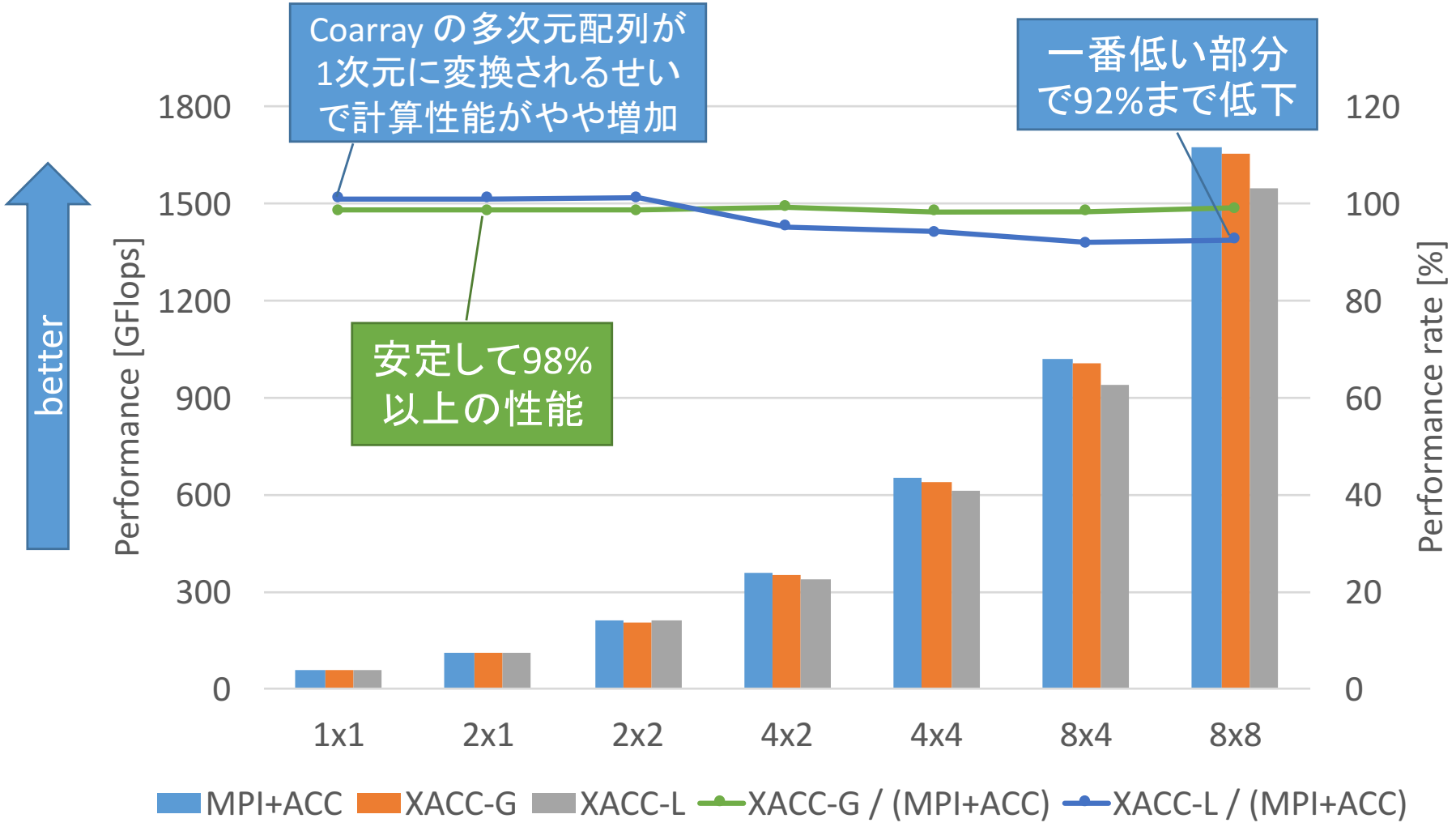
- 反復回数は1000回

```
#pragma xmp template t(0:MKMAX-1, 0:MJMAX-1,
                                    0:MIMAX-1)
#pragma xmp nodes n(1, NDY, NDX)
#pragma xmp distribute t(block, block, block) onto n
#pragma xmp align p[k][j][i] with t(i, j, k)
#pragma xmp shadow p[1:2][1:2][0:1]
...
#pragma acc data copy(p,…)
{
    ...
#pragma xmp loop (k,j,i) on t(k,j,i)
#pragma acc parallel loop reduction(+:gosa) collapse(2)
gang
  for(i=1 ; i<imax-1 ; ++i)
    for(j=1 ; j<jmax-1 ; ++j)
#pragma acc loop vector reduction(+:gosa)
      for(k=1 ; k<kmax-1 ; ++k){
        s0 = a[0][i][j][k]*p[i+1][j][k]+a[1][i][j][k] + …;
      }
  …
#pragma xmp reflect(p) width(1,1,0) acc
  …
} //end of acc data …
```

ステンシル計算

袖通信

Himeno Benchmark の性能



XACC-L (Coarray)の性能低下

- 両側プロセスとの袖通信が同時にされていない
  - 例 (j次元方向の通信)

```
xmp_sync_images(num_npy, npy, NULL);                          // 両側プロセスと同期
lo_recvbuf[0:len] = lo_sendbuf[0:len]:[mez][mey-1][mex];  // get
hi_recvbuf[0:len] = hi_sendbuf[0:len]:[mez][mey+1][mex];  // get
xmp_sync_images(num_npy, npy, NULL);
```

MPI_Get(..., win);
MPI_Win_flush_local(..., win);

```
xmp_sync_images(num_npy, npy, NULL);                          // そのまま
_XMP_coarray_shortcut_get( .., _DESC_lo_recvbuf, ..);     // 関数callに置き換え
_XMP_coarray_shortcut_get( .., _DESC_hi_recvbuf, ..);     // 関数callに置き換え
xmp_sync_images(num_npy, npy, NULL);                          // そのまま
```

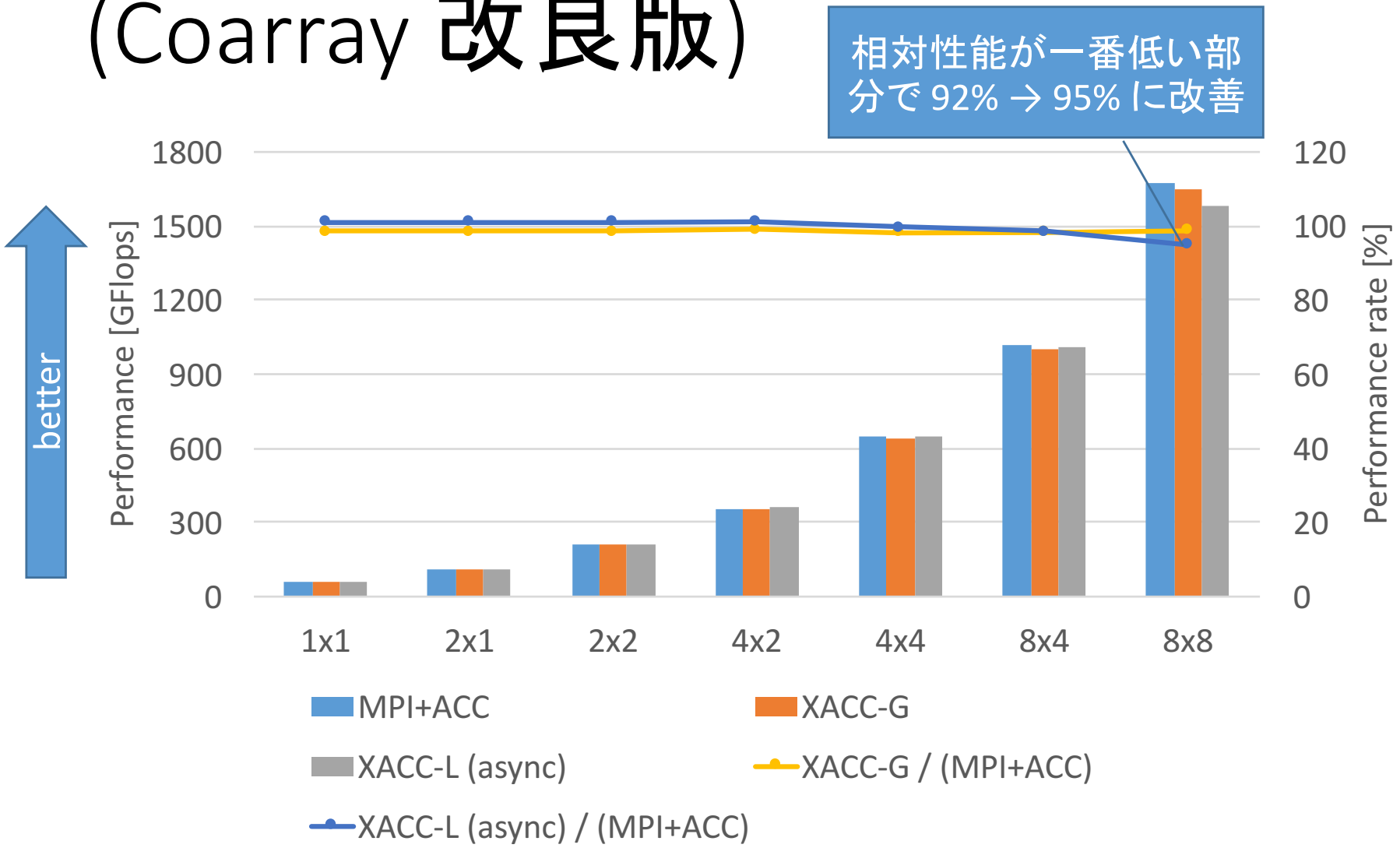
- 本来はコンパイラが2つの通信に依存がないことを解析して同時に発行するべき
- Omni では単純に置き換えるため、関数内では put/get の直後にそのローカルでの完了を待つようにしている
  - 通信に用いたローカル変数を読み書きする可能性があるため

XACC-L (Coarray)の改善案

- Coarray 通信のローカルでの完了を待たないようにする指示文があるとよい
  - 例: Cray compiler の独自機能 #pragma pgas defer_sync

- 試験的に環境変数 XMP_COARRAY_ASYNC=1 の時は通信発行後にローカルの完了待たないようにした
  - もちろん sync_image で通信の完了は保証される

Himeno Benchmark の性能 (Coarray 改良版)



NPB-CG

- 疎行列の最小固有値を共役勾配法で求めるベンチマーク
- 問題サイズ
  - Class D (1,500,000 x 1,500,000)
- 行と列の2次元分割
- 主な処理
  - 演算：SpMV
  - 通信：配列リダクション、行分割配列→列分割配列

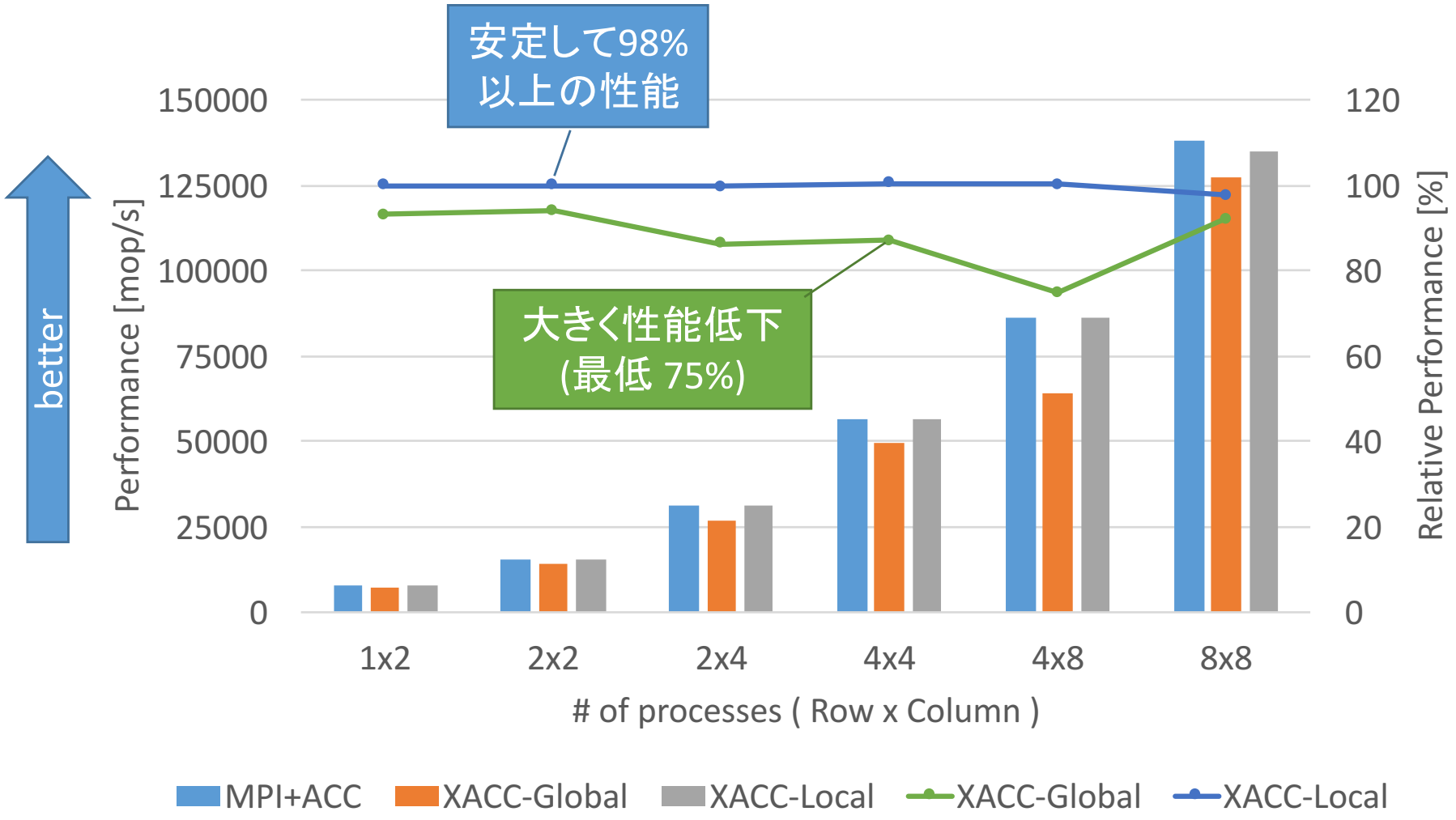
```
#pragma xmp nodes p(NUM_COLS, NUM_ROWS)
#pragma xmp template t(0:NA-1,0:NA-1)
#pragma xmp distribute t(block, block) onto p
#pragma xmp align w[i] with t(*,i)
#pragma xmp align q[i] with t(i,*)
double a[NZ];
int rowstr[NA+1], colidx[NZ];
…
#pragma acc data copy(p,q,r,w, a[0:NZ], ...)
{
  …
#pragma xmp loop on t(*,j)
#pragma acc parallel loop gang
  for(j=0; j < NA; j++){
    double sum = 0.0;
#pragma acc loop vector reduction(+:sum)
    for (k = rowstr[j]; k < rowstr[j+1]; k++)
      sum = sum + a[k]*p[colidx[k]];
    w[j] = sum;
  }
#pragma xmp reduction(+:w) on p(:,*) acc
#pragma xmp gmove acc
  q[:] = w[:];
} //end acc data
```

SpMV

リダクション

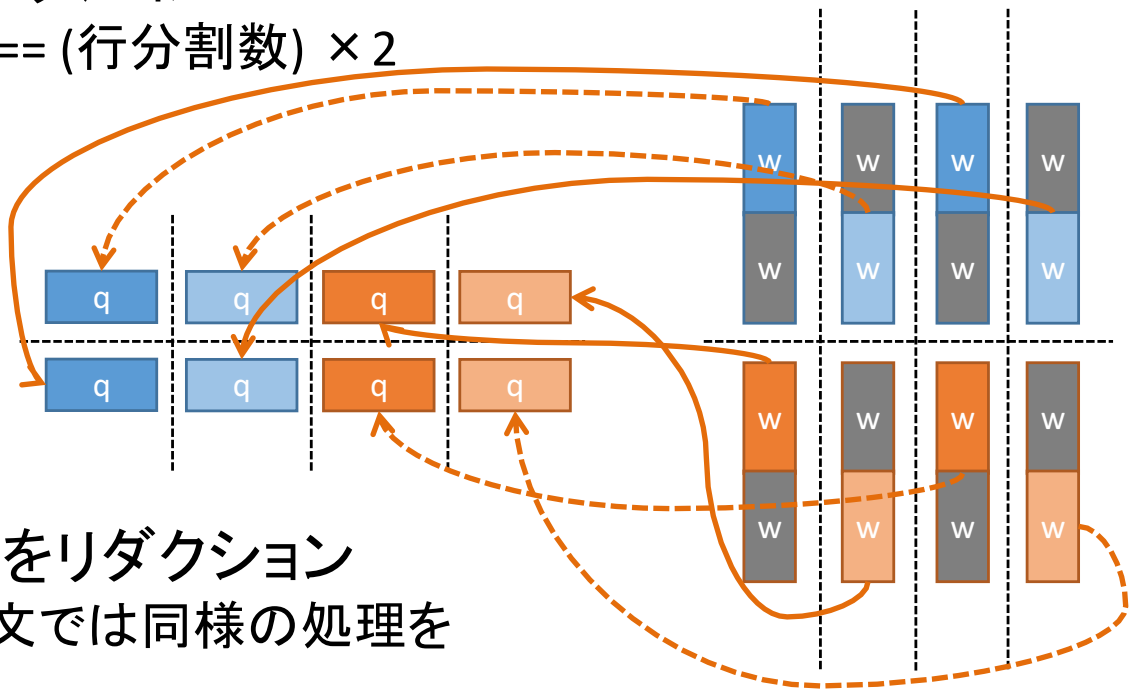
列分割←行分割

NPB-CG の性能


安定して98%
以上の性能
大きく性能低下
(最低 75%)
better
Performance [mop/s]
Relative Performance [%]
150000
125000
100000
75000
50000
25000
0
120
100
80
60
40
20
0
1x2
2x2
2x4
4x4
4x8
8x8
# of processes (Row x Column)
MPI+ACC
XACC-Global
XACC-Local
XACC-Global
XACC-Local

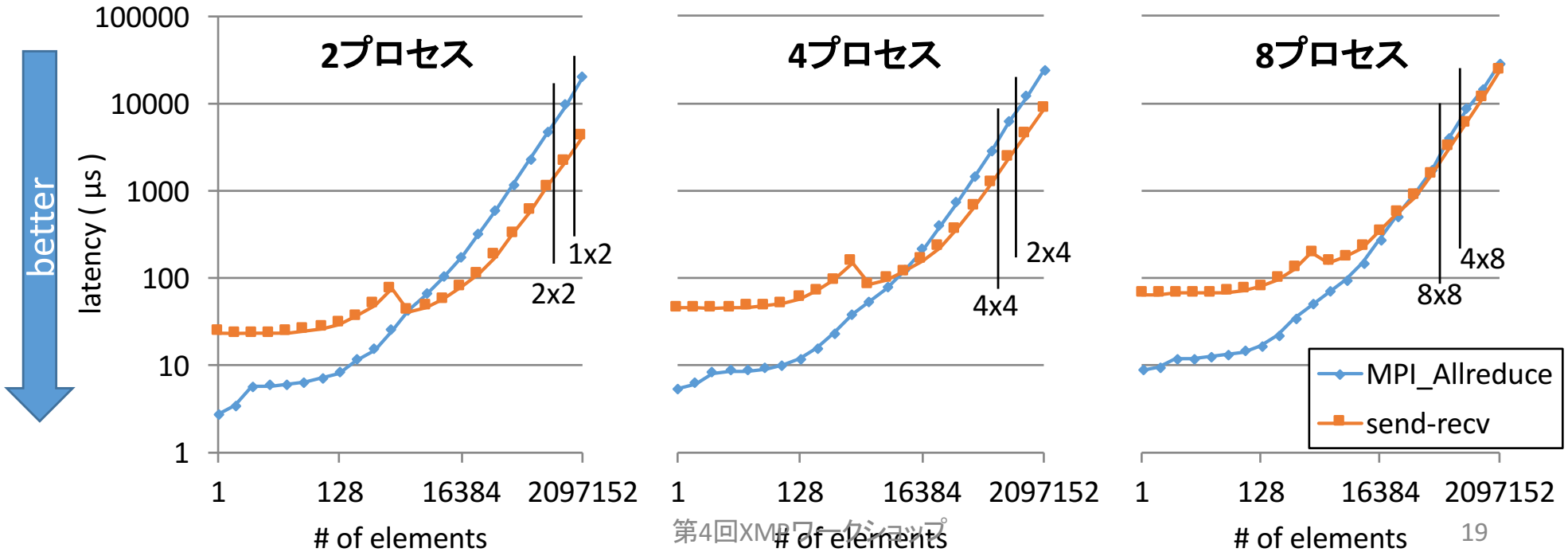
配列リダクションの性能低下 (1)

- 処理する配列サイズの違い
  - MPI + OpenACC 版では直後の行分割→列分割の通信に必要な部分のみリダクション
    - 例 : (列分割数) == (行分割数) ×2
  - XACC-G では全体をリダクション
    - reduction 指示文では同様の処理を記述できない



配列リダクションの性能低下 (2)

- リダクションの方法の違い
  - MPI + OpenACC は MPI_Isend/Recv と加算ループにより GPU 上で Recursive-Doubling 法
  - XACC-G は MPI_Allreduce
    - MV2 ではホストにコピーしてリダクション
  - CG の Class D のサイズだと MPI_Allreduce の方が遅い



まとめ

- XACC
  - XMP と OpenACC の統合
  - デバイス間通信をサポート
- Omni XACC compiler
  - [C]は指示文・Coarray ともにおおよそ実装済
  - [F]は一部の指示文のみ対応
- 性能
  - Himeno Bench, NPB-CG では 適切な記述を用いれば MPI + OpenACC と同等の性能
  - Coarray は複数通信時の改善が必要

今後の予定

- デバイス間通信(GPU)の実装を継続
  - MPI版はgmove ,coarrayの実装
  - TCA版は未定。GASNet/TCA の利用？
  - アプリの実装を通してさらに性能を改善する

- 各種アクセラレータへの対応
  - 現在開発中の Omni OpenACC for PEZY-SC を用いて PEZY-SC でも XACC を利用可能にする