圧縮性流体シミュレーションコード CloverLeaf の XcalableACC による実装と評価

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

背景

- XcalableACC (XACC) の提案・処理系実装を行ってきた
  - XMP と OpenACC を統合したアクセラレータクラスタ向けPGAS言語
  - アクセラレータメモリ上のデータに対する通信に対応
    !$xmp {reflect | reduction |…} + acc

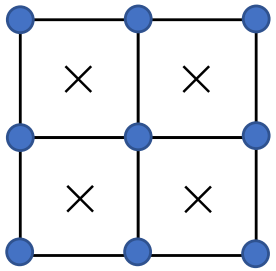
  - ベンチマーク (Himeno benchmark, など) における評価では、MPI+OpenACC とほぼ同等の性能を簡易に記述できた

➢XACCをアプリケーションで評価する必要がある

- 一部の流体シミュレーションではスタッガード格子を用いる
  - 変数によって配置を変えることで計算が安定する

➢XMP/XACC による実装例はまだない

スタッガード格子における変数配置



× セル中心
● セル頂点

目的

- XACC による実アプリケーションの性能・生産性を確かめる
  - 流体力学ミニアプリCloverLeafを用いる
- XMP/XACC によるスタッガード格子を用いたプログラムの実装方法を示す

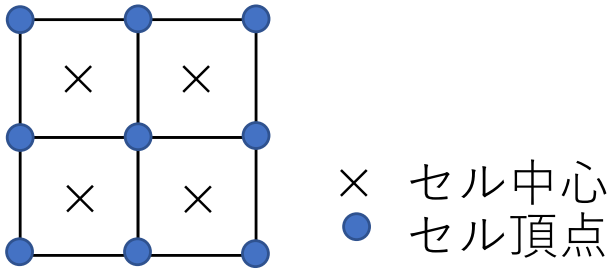
CloverLeaf

- 圧縮性流体シミュレーションのミニアプリ
  - UK Mini-App Consortium (UKMAC) が公開
  - Mantevo project という様々な分野のミニアプリ集にも含まれる

- 2次元の圧縮性オイラー方程式を解く
  - 有限体積法で2次精度

- 時間発展処理
  1. Lagrangian ステップ。予測子修正子法を用いてタイムステップを進める
  2. 移流ステップ。１つ目のステップで流速により移動した物理量をセルに再配置 (remap) する

- 物理量はスタッガード格子 (Staggered Grid) 上に配置されている
  - セル中心 (pressure, etc.)
  - セル頂点 (velocity, etc.)



CloverLeaf の既存実装

- メインコードは Fortran
- カーネル (計算部分) は Fortran or C
  - 今回は Fortran を使用

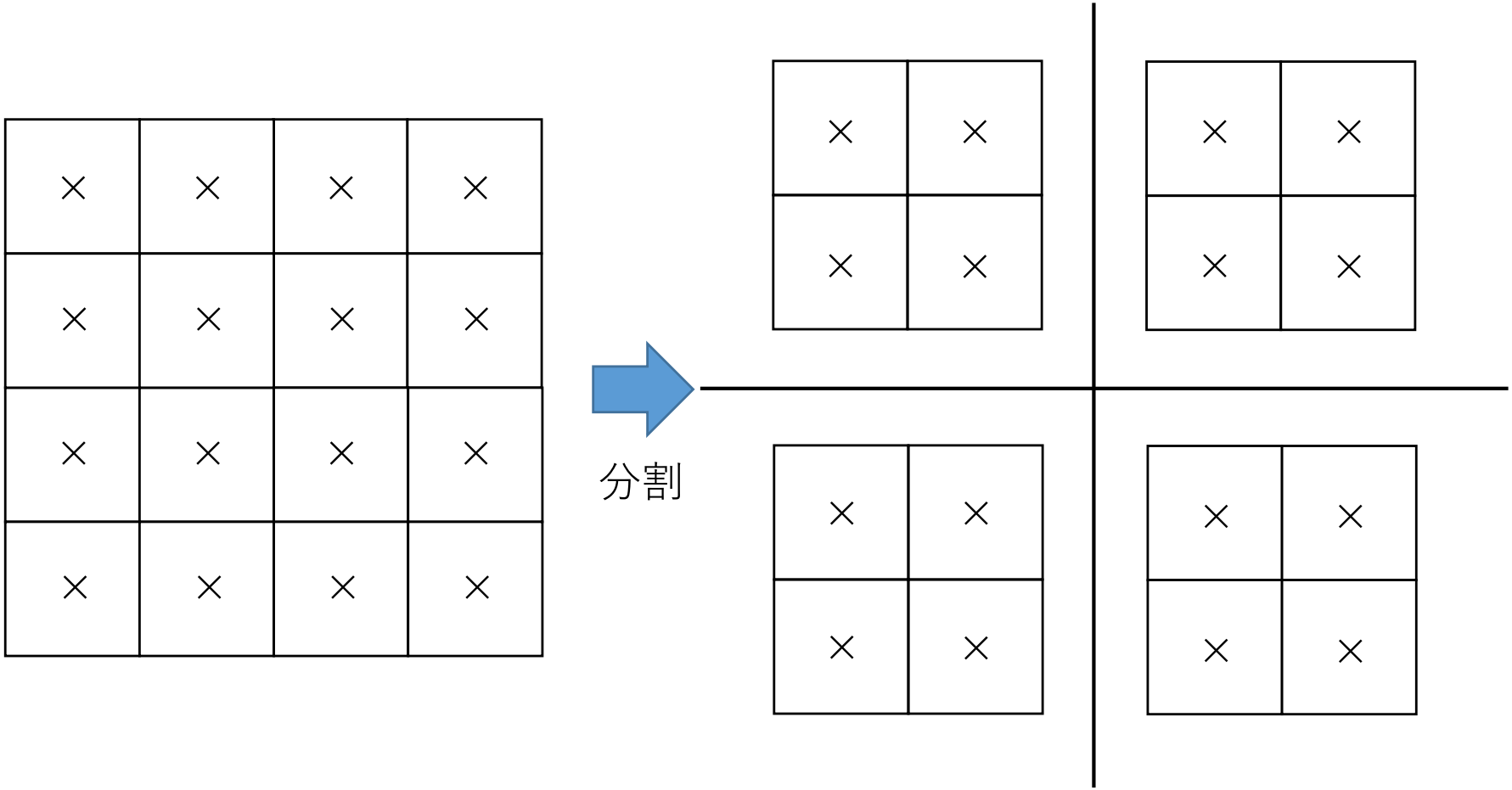
- さまざまな種類のコード開発されている
  - Serial
  - MPI
  - MPI+OpenMP
  - MPI+CUDA
  - MPI+OpenACC
  - etc.

- GitHub で公開されている
  - https://github.com/UK-MAC/CloverLeaf

XACC による CloverLeaf の実装概要

- 使用するのは global-view モデル
  - 基本的に指示文の追加
- 2次元セルを2次元ブロック分割
- 主にアクセラレータ上で実行
  - 計算
  - 袖交換

データ分散 (セル中心)

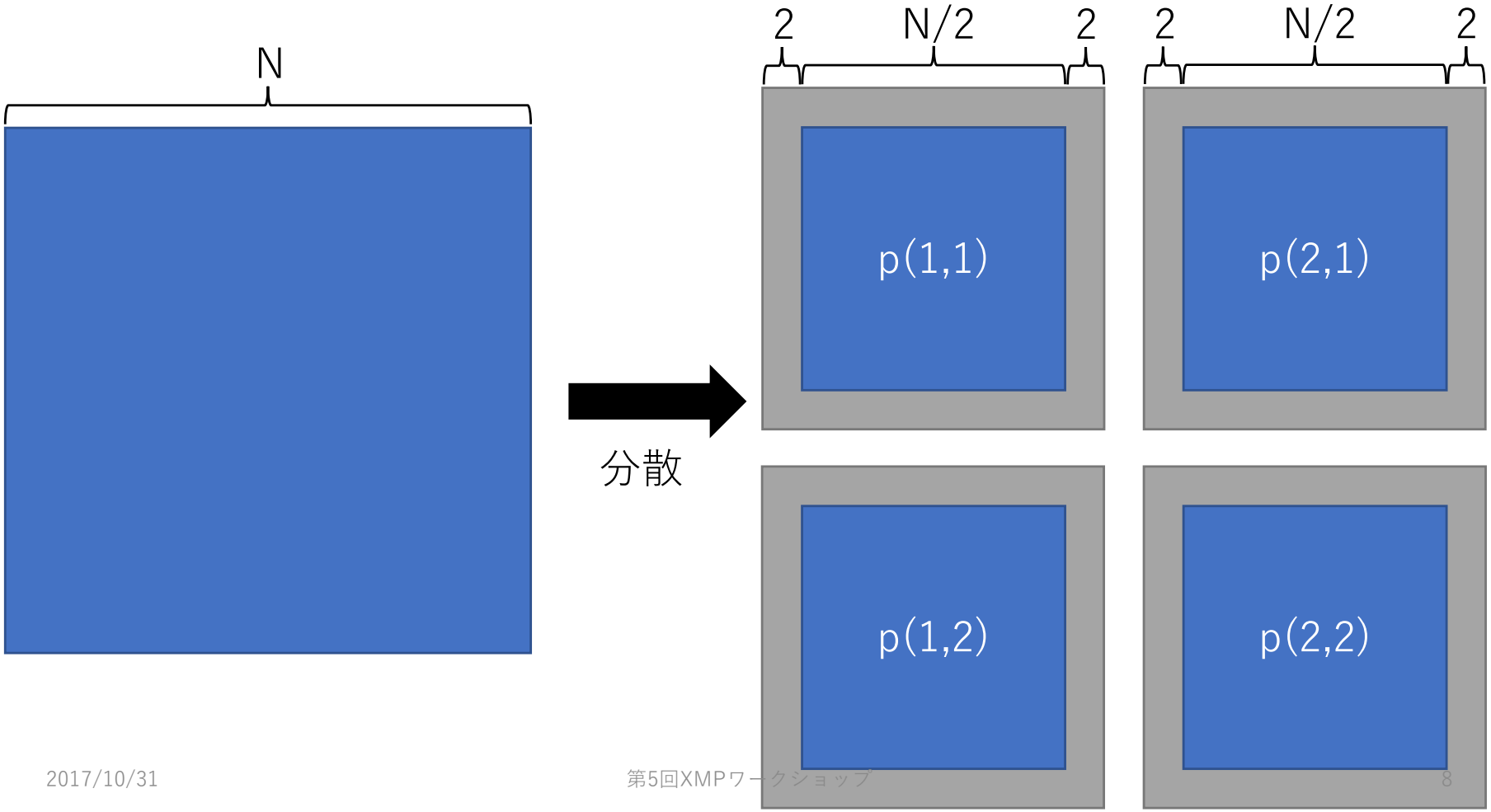


セルの中心はセルの数と同じなので
均等に分割できる

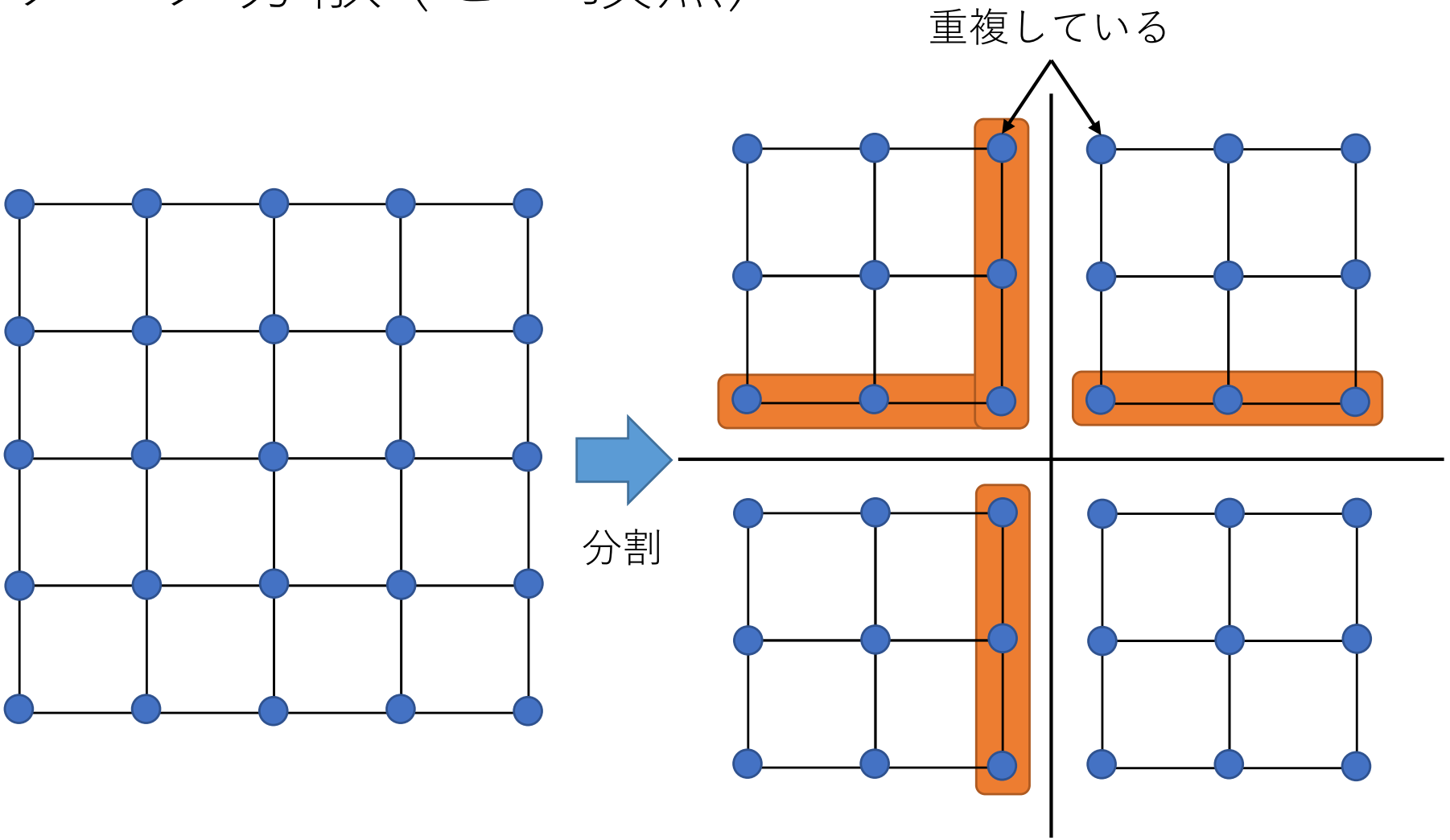
データ分散 (セル中心)

```
REAL(KIND=8) pressure(x_min-2:x_max+2, y_min-2:y_max+2)
!$xmp align (i,j) with t(i,j) :: pressure
!$xmp shadow (2:2,2:2) :: pressure
```

袖領域を
shadow 指示文で追加



データ分散 (セル頂点)

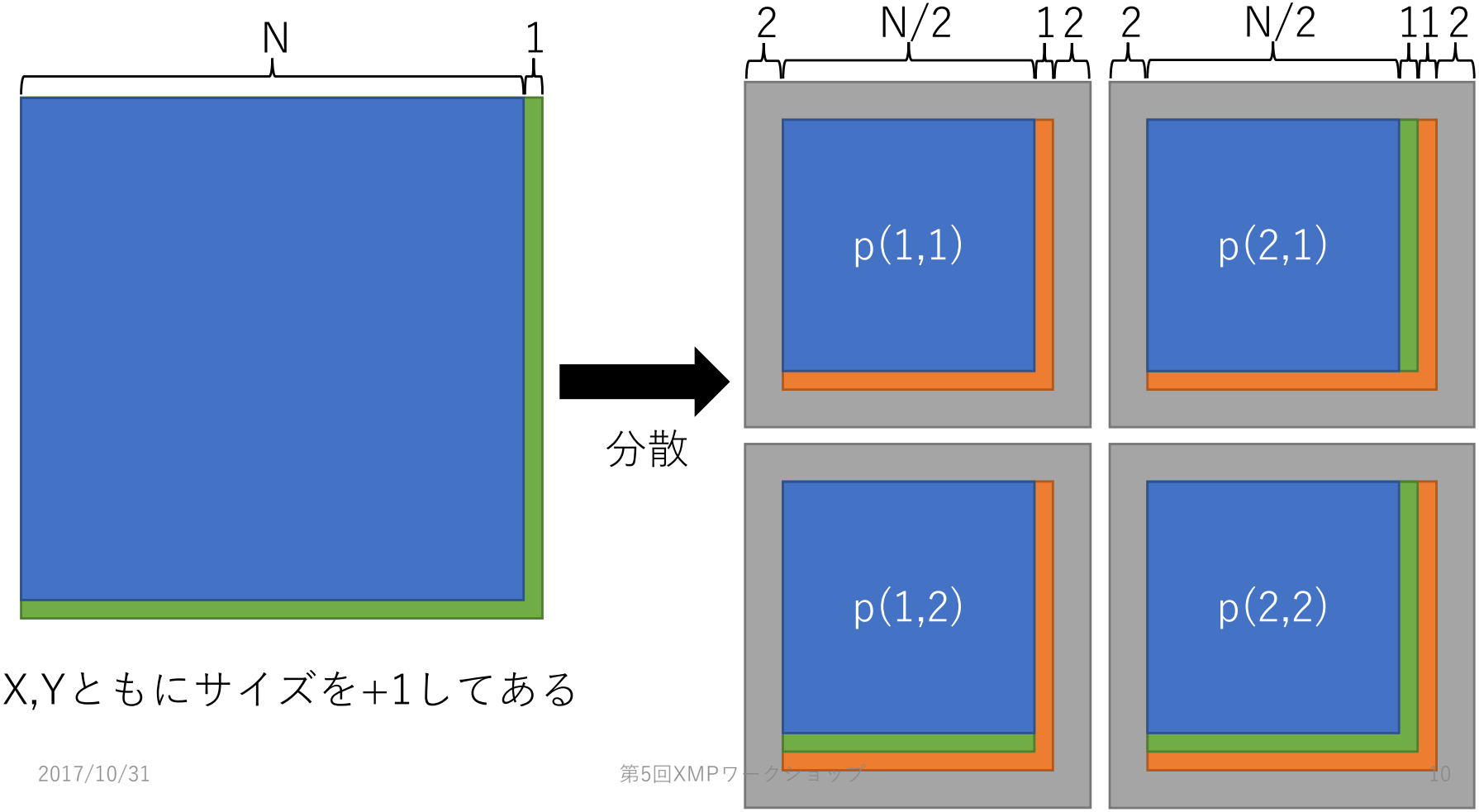


セルを分割すると 部分の頂点を重複して
保持する必要がある → shadow を利用

データ分散 (セル頂点)

```
REAL(KIND=8) xvel0(x_min-2:x_max+3, y_min-2:y_max+3)
!$xmp align (i,j) with t(i,j) :: xvel0
!$xmp shadow (2:3,2:3) :: xvel0
```

重複する頂点を保持するために
shadow 幅を+1



X,Yともにサイズを+1してある

初期化部分のコード

```
!$xmp nodes p(num_nodes_x,num_nodes_y)
!$xmp template t(:,:)
!$xmp distribute t(gblock(*),gblock(*)) onto p

REAL(KIND=8),ALLOCATABLE,DIMENSION(:,:) :: pressure, xvel0, ...
!$xmp align (i,j) with t(i,j) :: pressure, xvel0, ...
!$xmp shadow (2:2,2:2) :: pressure, ...
!$xmp shadow (2:3,2:3) :: xvel0, ...

! Read input parameters (x_min, x_max, y_min, y_max, etc.)
! Set blocksizes (bsize_x and bsize_y)

!$xmp template_fix (gblock(bsize_x), gblock(bsize_y))&
!$xmp                t(x_min-2:x_max+3, y_min-2:y_max+3)

ALLOCATE(pressure(x_min-2:x_max+2,y_min-2:y_max+2))
ALLOCATE(xvel0     (x_min-2:x_max+3,y_min-2:y_max+3))...

!$acc enter data copyin(pressure,xvel0,...)
```

テンプレートサイズは後で指定

ノード毎にブロックサイズが異なるので、gblock分散を指定

テンプレートサイズとブロックサイズの指定

分散配列をアクセラレータメモリに確保

計算

- 大半は2重ループで構成されている

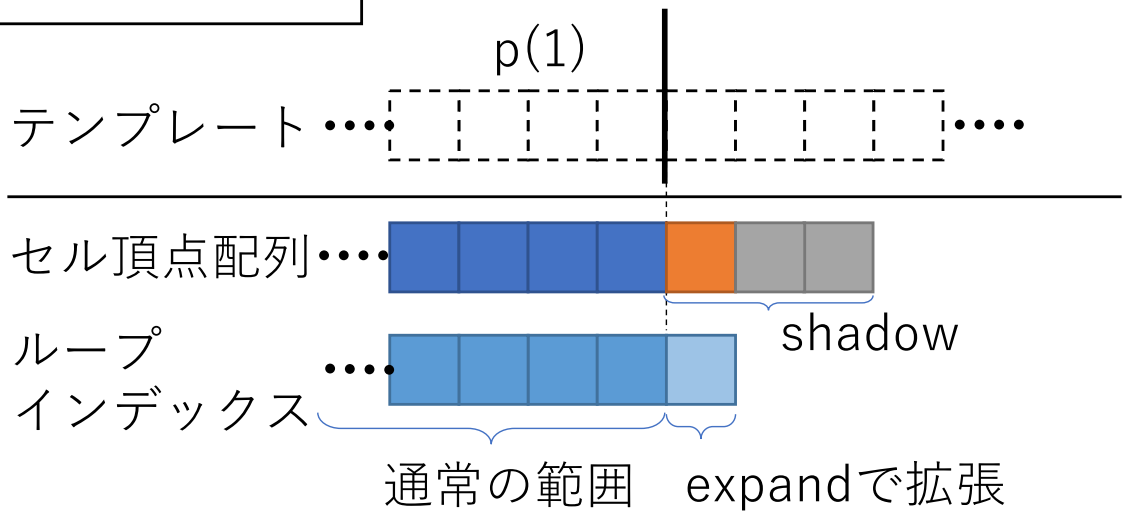
```
!$xmp loop (j,k) on t(j,k) expand(0:1,0:1)
!$acc kernels loop independent
DO k=y_min,y_max+1
   !$acc loop independent
   DO j=x_min,x_max+1
     xvel1(j,k) = xvel0(j,k) - ...
   ENDDO
ENDDO
```

ループの分散

オフロード・
反復間に依存無し

反復間に依存無し

セル頂点配列を処理する際に
XMP 1.3 からの機能 expand 節
を使ってローカルのループ範囲
を拡大



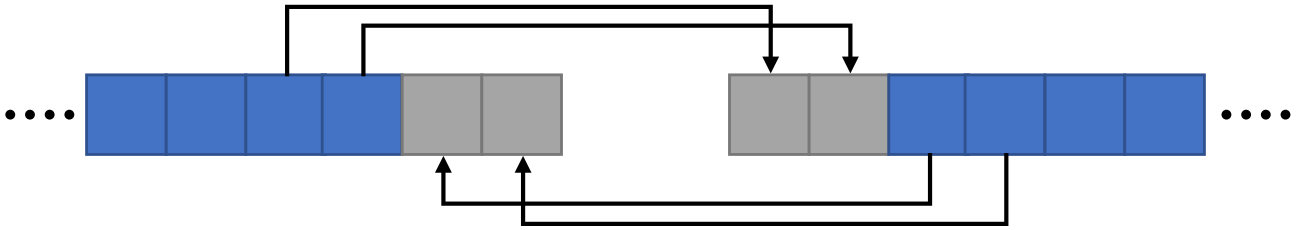
袖交換

袖交換の幅 depth = 1 or 2

アクセラレータメモリ上の
分散配列の通信

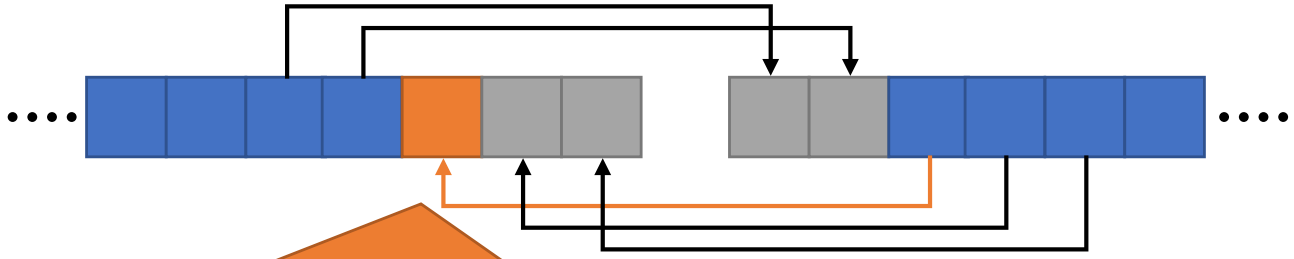
- セル中心

```
!$xmp reflect width(depth:depth, depth:depth) acc
```



- セル頂点

```
!$xmp reflect width(depth:1+depth, depth:1+depth) acc
```



内側の shadow 1要素は更新不要だが、現状記述できない
値はどちらも同じものが入っているので結果は変わらない

評価

- 3種類のコードを比較
  - MPI+CUDA
  - MPI+OpenACC
  - XACC

※ MPI+CUDA, MPI+OpenACC は GitHub 版に少し変更を加えてある

- 性能評価
  - 問題サイズは 3840×3840 セル
  - タイムステップは1000回に固定
  - 強スケーリングと弱スケーリング時の実行時間

- 生産性評価
  - SLOC (コード行数)
  - DSLOC (逐次コードからの行数差分)

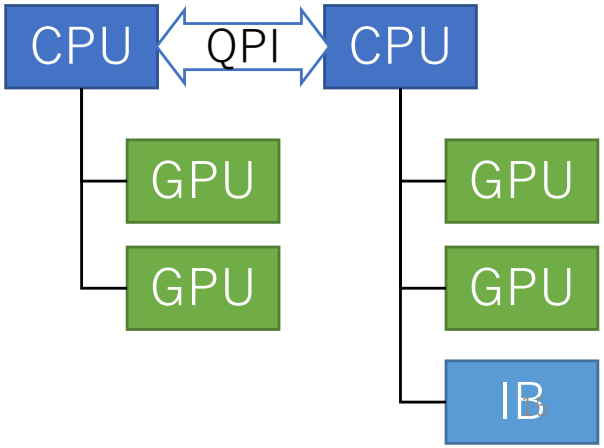
性能評価環境

- 筑波大学計算科学研究センターの HA-PACS/TCA

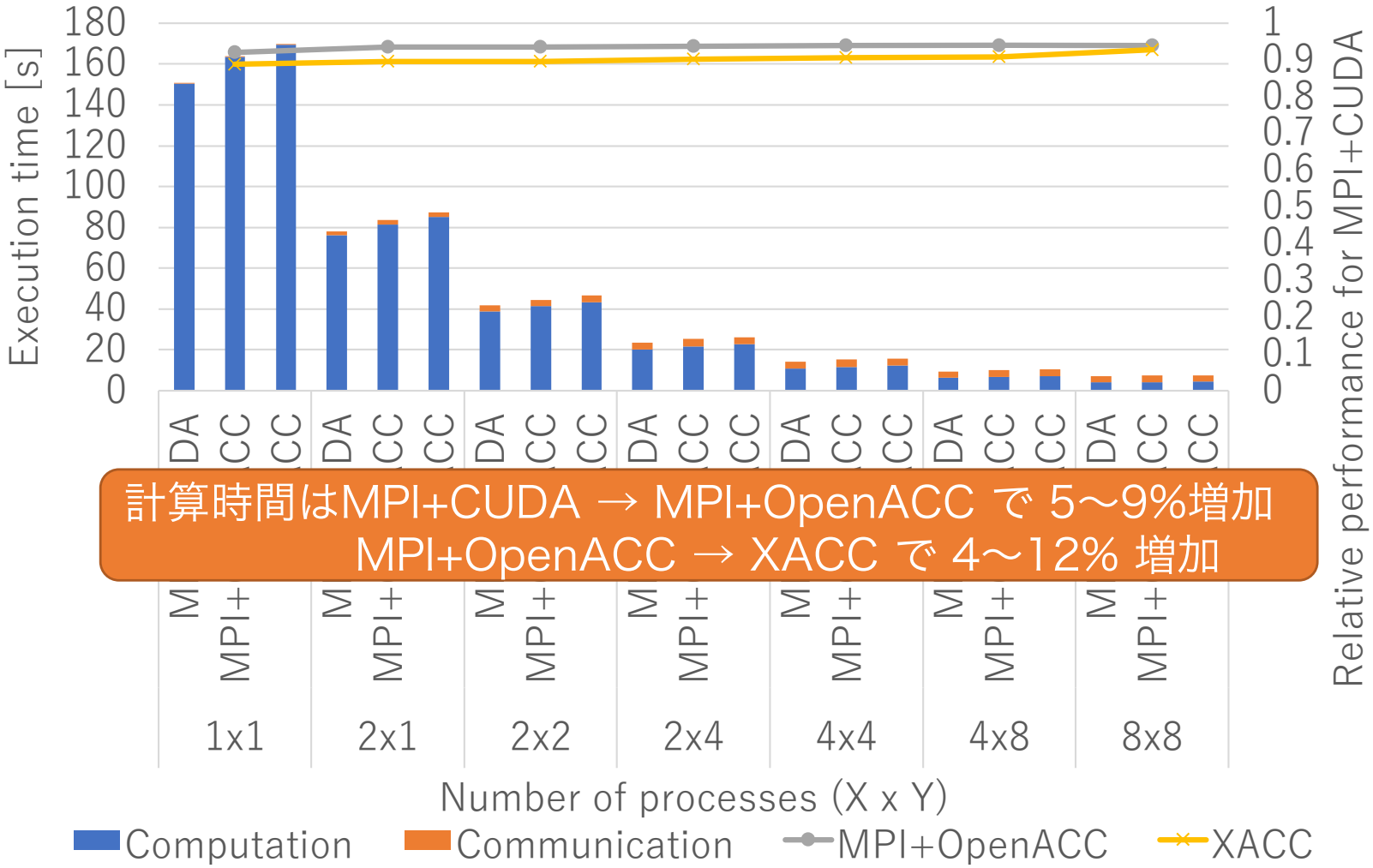
ノード構成とソフトウェア

CPU	Intel Xeon-E5 2680v2 2.8GHz ×2
Memory	DDR3 1866MHz, 128GB
GPU	Tesla K20X ×4
Interconnect	InfiniBand Mellanox Connect-X3 FDR
Software	PGI 16.10, CUDA 8.0, MVAPICH2 2.2 Omni Compiler 1.2.1 + extension

- 1GPU/プロセス, 4プロセス/ノード
- 最大16ノード上で64プロセスを実行



3840^2セル, 強スケーリング, 全体時間



計算時間はMPI+CUDA → MPI+OpenACC で 5〜9%増加
MPI+OpenACC → XACC で 4〜12% 増加

MPI+CUDAと比べて89-93%, MPI+OpenACCと比べて97-99%

XACC版の計算性能の低下

- XACCコンパイラのコード変換が原因

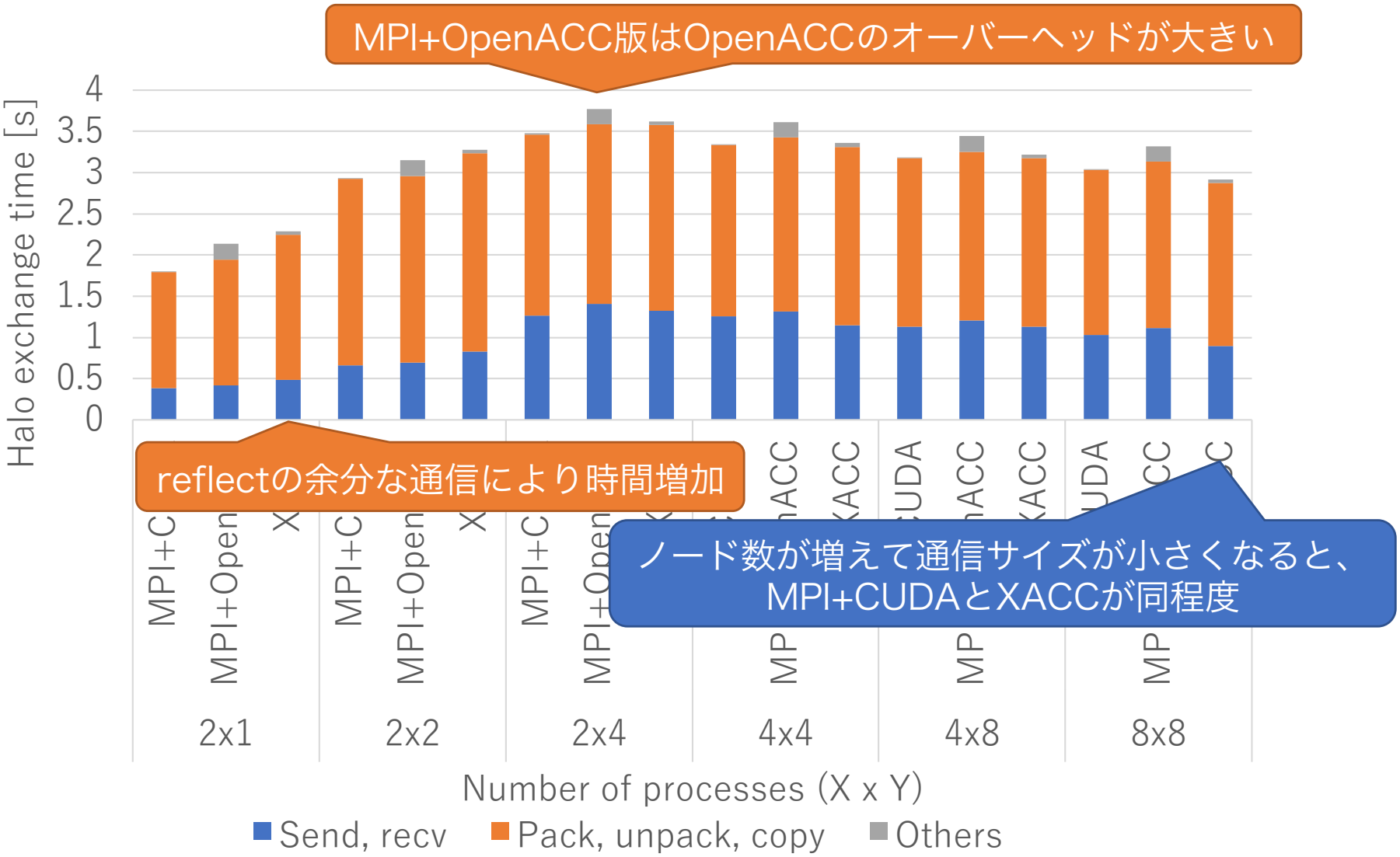
```
REAL(KIND=8) :: pressure(x_min-2:x_max+2, y_min-2:y_max+2)
```

変換

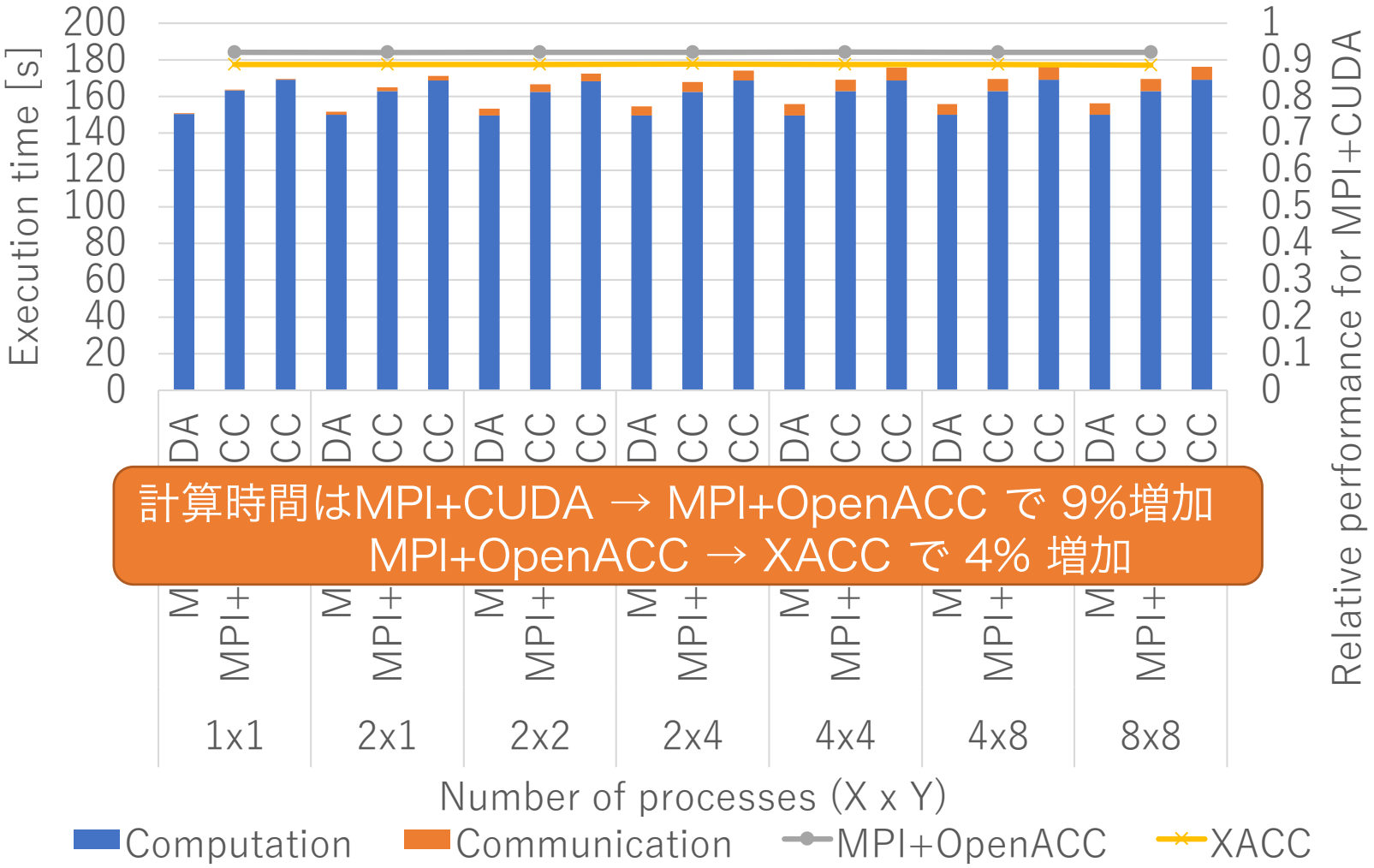
```
REAL(KIND=8) :: XMP__pressure(0: , 0:)
```

- 配列サイズが動的に決まるため、配列アクセス時に必要な変数が配列数に比例して増加
- GPUカーネルのレジスタ使用数の増加により同時実行スレッド数が減少

3840^2セル, 強スケーリング, 袖交換時間

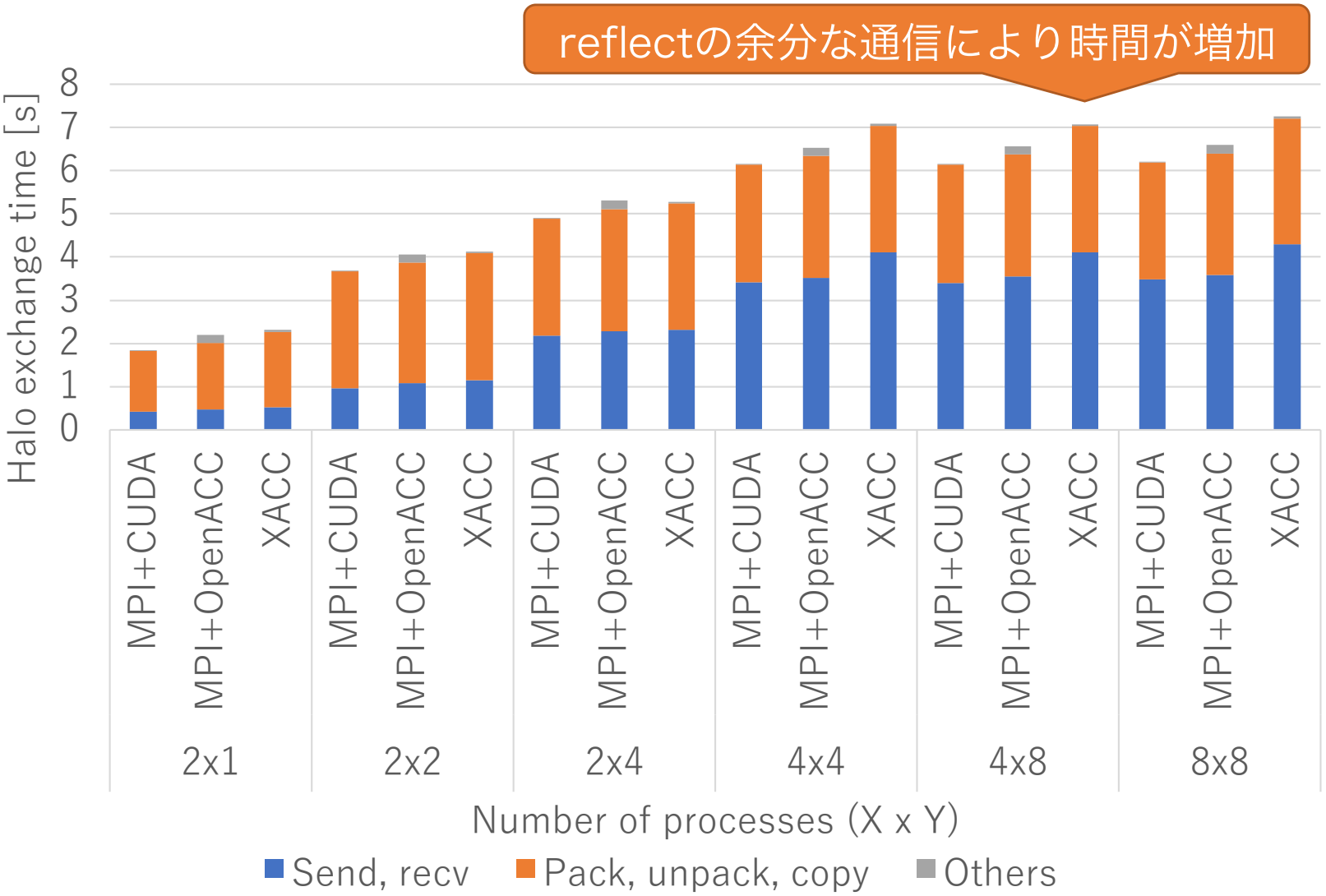


3840^2セル, 弱スケーリング, 実行時間



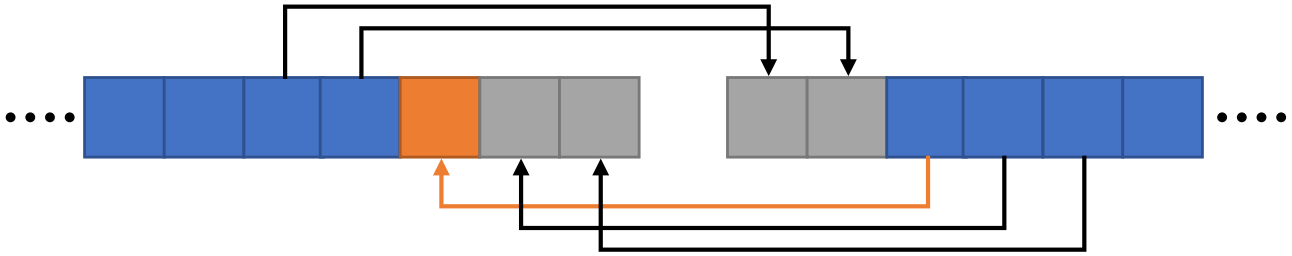
MPI+CUDAと比べて89%, MPI+OpenACCと比べて96-97%

3840^2セル, 弱スケーリング, 袖交換時間

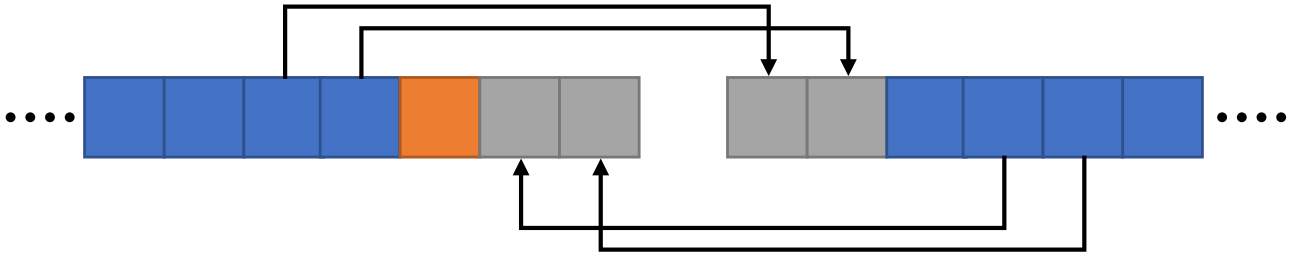


reflect 指示文の拡張の提案

- 現在のreflectは更新する幅の設定しかできない
  - !$xmp reflect (array) width(2:3)



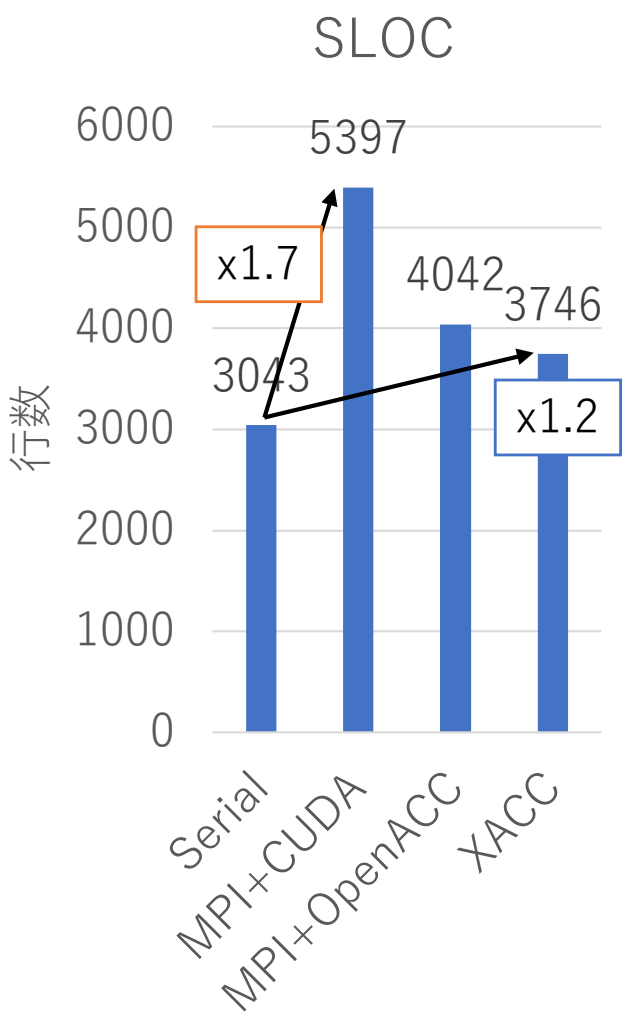
- 内側の袖を飛ばす offset 節を提案
  - !$xmp reflect (array) width(2:2) offset(0:1)

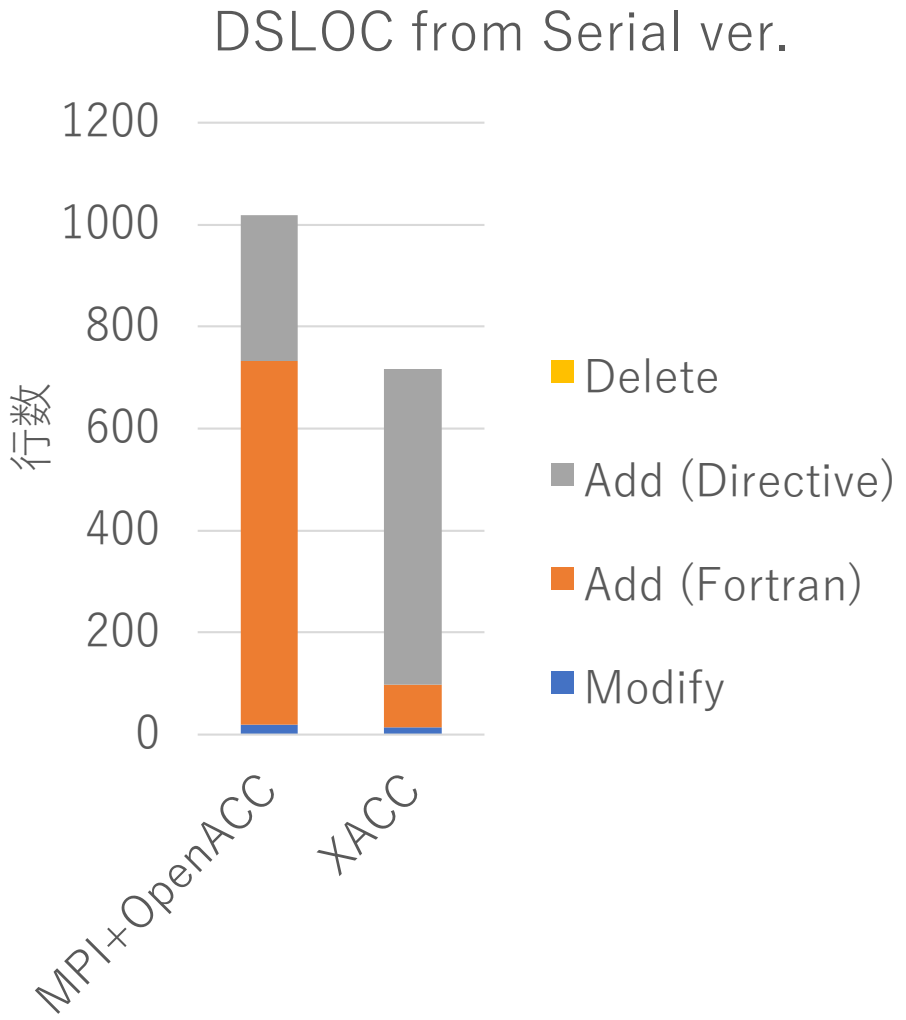


[F] !$xmp reflect ... *[* offset (*reflect-offset [*, *reflect-offset]*...)*]*
[C] #pragma xmp reflect ... *[* offset (*reflect-offset [*, *reflect-offset]*...)*]*

reflect-offset is int-expr *[*:int-expr*]*

行数の比較





XACCは最も少ない行数で記述できた

MPI+OpenACC は Fortran コードの追加が多いが、XACC は指示文の追加がほとんど

まとめ

- 圧縮性流体力学コードCloverLeafをXACCで実装
  - 変数で配置の異なるスタッガード格子

- 十分に良い性能を達成
  - MPI+CUDA と比べて89%以上の性能
  - MPI+OpenACC と比べて96%以上の性能

- 生産性も高い
  - 指示文により逐次コードをベースに少ない変更で記述できた

今後の課題

- 計算性能の改善のためにXACCコンパイラのコード変換の改良を検討

- reflect 指示文の offset 節などの拡張を実装して効果を確認