Performance Evaluation for a Hydrodynamics Application in XcalableACC PGAS Language

Akihiro Tabuchi[†1], Masahiro Nakao[†2], Hitoshi Murai[†2],
Taisuke Boku[†1,3], and Mitsuhisa Sato[†1,2]

†1 Graduate School of Systems and Information Engineering, University of Tsukuba
†2 RIKEN Advanced Institute for Computational Science
†3 Center for Computational Sciences, University of Tsukuba

Outline

- Background
- Objective
- XcalableMP and XcalableACC
- CloverLeaf in XACC
- Evaluation
- Conclusion and future work

Background (1/3)

- Clusters equipped with accelerators are increasing
  - Tianhe-2 (KNC), Piz Daint (P100), Gyoukou (PEZY-SC2), etc.
  - we refer to these clusters as “accelerated clusters”

- Programming for accelerated clusters is difficult
  - MPI for communication and CUDA / OpenCL for offloading
    - distributing data and work manually
    - communicating data using MPI function
    - managing accelerator memory using CUDA/OpenCL functions
    - describing parallel codes for accelerator

Background (2/3)

- There are some directive-based extensions to make the programming simpler

- XcalableMP (XMP) for distributed-memory systems
  - PGAS language that extends C and Fortran
  - two programming models
    - Global-view model
      directive-based easy description
    - Local-view model
      detailed communication using coarray

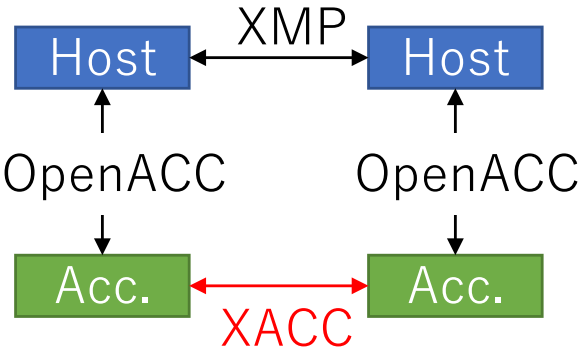
http://www.xcalablemp.org/

- OpenACC for accelerators
  - standard specification defined by NVIDIA, et al.

https://www.openacc.org/

Background (3/3)

- New PGAS language XcalableACC (XACC)
  - an integration of XMP and OpenACC
  - supporting communication between accelerators
  - global-view and local-view models



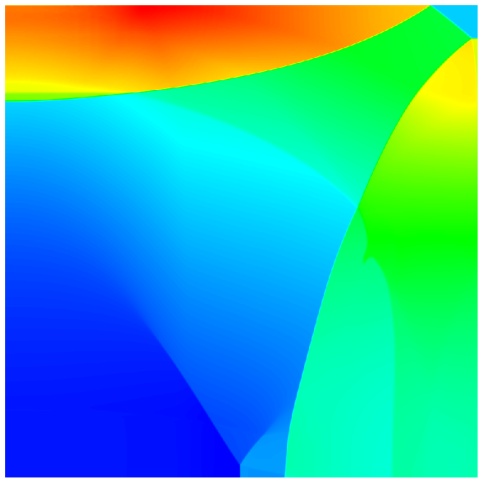
- XACC has good performance and high productivity for some benchmark programs [1,2]
  - For himeno benchmark, XACC version achieves over 97% performance of MPI+OpenACC version on GPU cluster
- To assess XACC, we need evaluations in practical applications

[1] M. Nakao, et al. "XcalableACC: Extension of XcalableMP PGAS Language using OpenACC for Accelerator Clusters", Workshop on accelerator programming using directives (WACCPD), pp.27-36, New Orleans, LA, USA, Nov. 2014

[2] A. Tabuchi, et al. "Implementation and Evaluation of One-sided PGAS Communication in XcalableACC for Accelerated Clusters", The 17th IEEE/ACM International Symposium on Cluster, Cloud and Grid Computing, Spain, May 2017.

Objective

- To evaluate performance and productivity for XACC application program
  - Target application : a hydrodynamics mini-application CloverLeaf
  - We implement CloverLeaf using XACC global-view model
  - We evaluate the performance and productivity and compare them with MPI+CUDA and MPI+OpenACC versions



http://uk-mac.github.io/CloverLeaf/

XcalableMP (XMP)

- PGAS language for distributed-memory system
  - defined by XMP Spec. W.G. of PC Cluster Consortium in Japan
  - base languages are C and Fortran
  - providing two programming models

- Global-view model
  - directive-based programming model influenced by High Performance Fortran
  - providing directives for data distribution, work mapping, communication, and synchronization

  ➢We can describe action of all nodes by adding directives to serial codes

- Local-view model
  - using coarray for communication
  - not used in this work

Example of global-view model

```
integer a(N)
!$xmp nodes p(*)
!$xmp template t(N)
!$xmp distribute t(block) onto p
!$xmp align a(i) with t(i)
!$xmp shadow a(1:1)
...


!$xmp loop (i) on t(i)
    do i = 1, N
        a(i) = func(i)
    end do

!$xmp reflect (a)
```

dist.

work mapping

comm.

XcalableACC (XACC)

- PGAS language for accelerated clusters
  - an integration of XMP and OpenACC
- communication between accelerators
  - Global-view model
    - `acc` clause for communication directives
  - Local-view model
    - coarray declaration and communication on accelerator

Example of global-view model

```
integer a(N)
!$xmp nodes p(*)
!$xmp template t(N)
!$xmp distribute t(block) onto p
!$xmp align a(i) with t(i)
!$xmp shadow a(1:1)
...
!$acc data create(a)
!$xmp loop (i) on t(i)
!$acc parallel loop
    do i = 1, N
        a(i) = func(i);
    end do

!$xmp reflect (a) acc
```

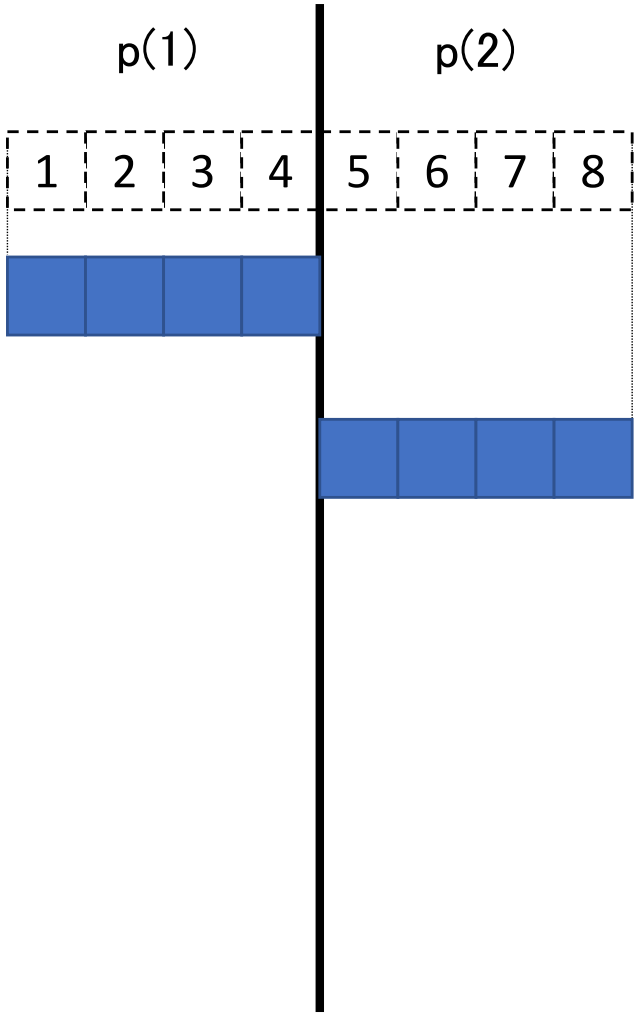
data offload

work offload

comm. on acc.

Example of XACC global-view model

```
integer a(8)
!$xmp nodes p(2)
!$xmp template t(8)
!$xmp distribute t(block) onto p
!$xmp align a(i) with t(i)
```

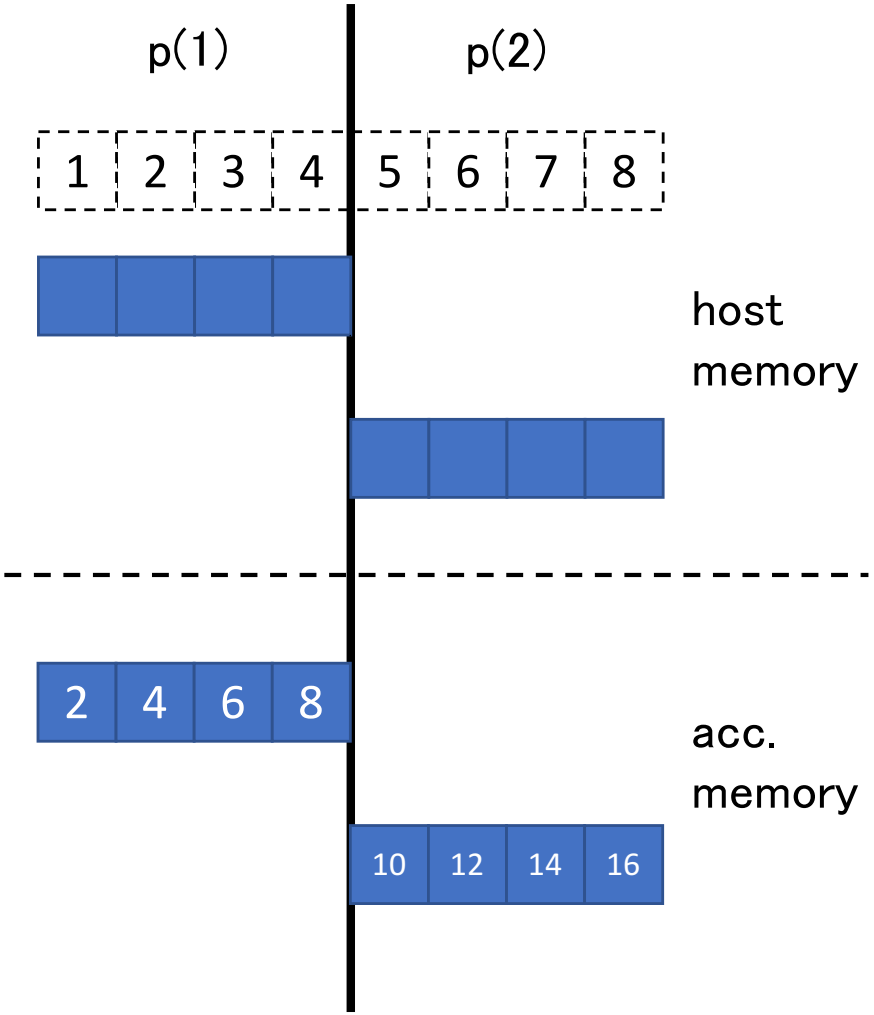


- `nodes` directive defines node set
- `template` directive defines template
  - virtual index array
  - used for data and work mapping
- `distribute` directive distributes template on nodes
- `align` directive distributes array as well as template

Example of XACC global-view model

```
!$acc data copy (a)
!$xmp loop (i) on t(i)
!$acc kernels loop
  do i = 1, 8
    a(i) = i * 2
  end do
```

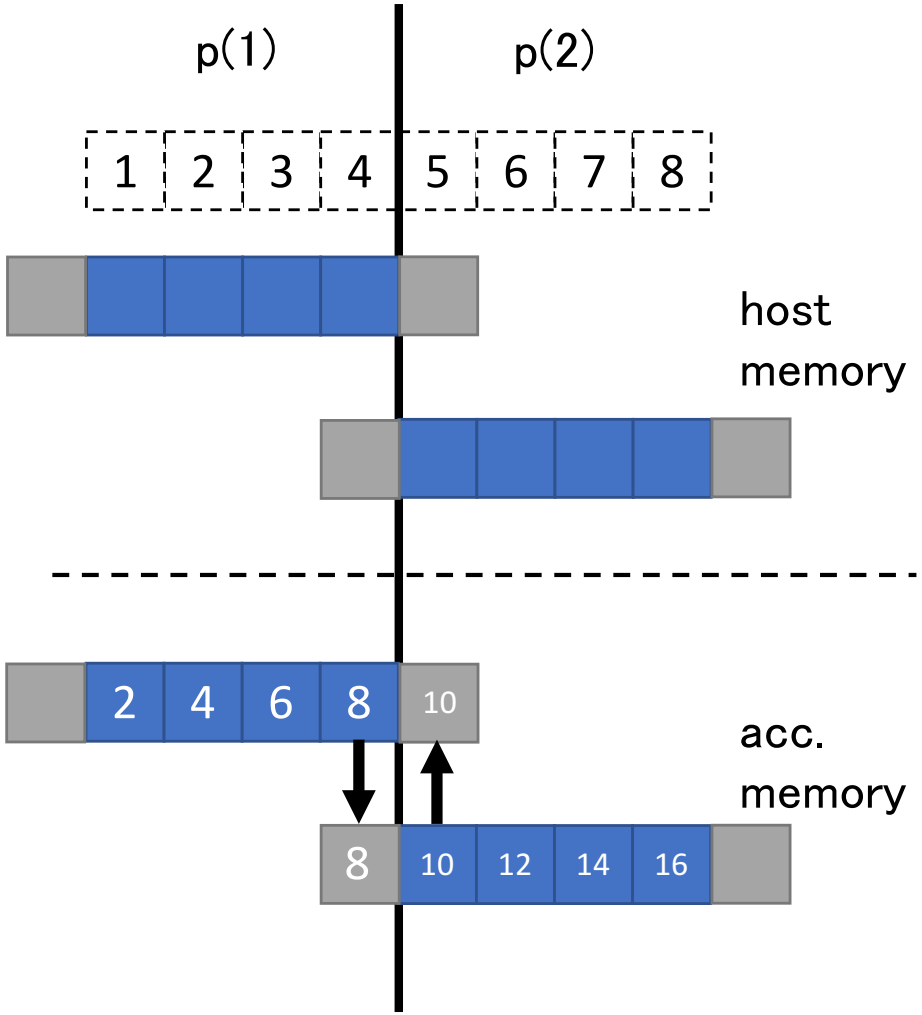
- `data` directive allocate and copy data to accelerator memory
  - for distributed array, only assigned parts are allocated
- `xmp loop` directive distributes loop iterations as well as the template
- `acc kernels loop` directive offloads loop to accelerator



Example of XACC global-view model

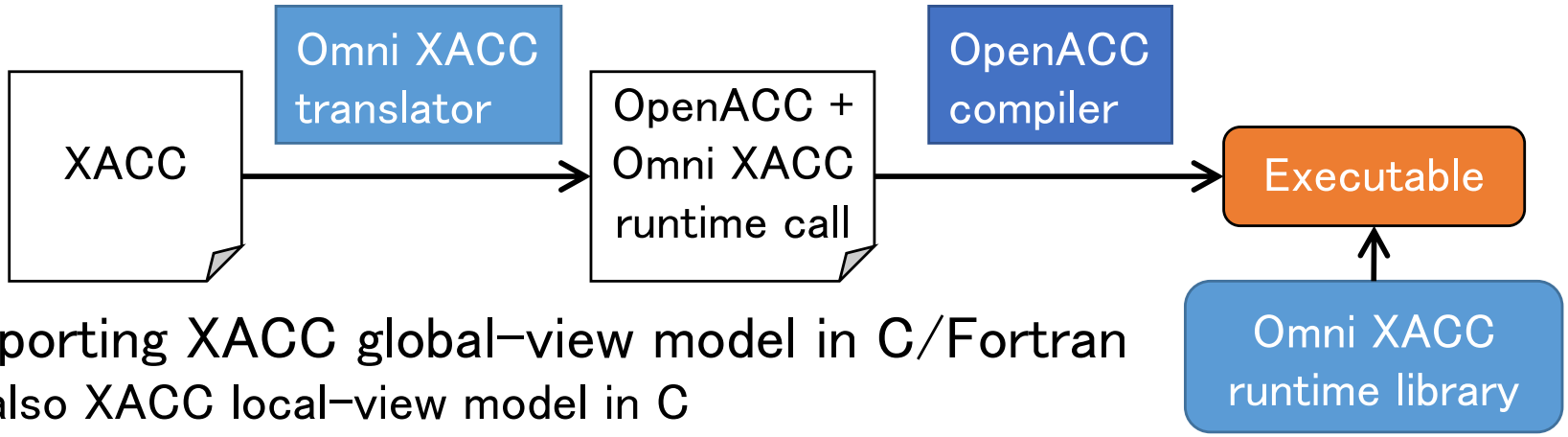
```
!$xmp align a(i) with t(i)
!$xmp shadow a(1:1)
...

!$xmp reflect (a) acc
```

- `shadow` directive adds shadow region where can be used as halo region
- `reflect` directive updates shadow region with the value of actual region
  - `acc` clause specifies update shadow on accelerator memory



Omni XACC compiler

- A source-to-source XACC compiler
  - extension of Omni XMP compiler
- XACC → OpenACC + Omni XACC runtime call
  - XMP based directives are translated to runtime calls
  - enabling to use general OpenACC compilers (PGI, Cray, Omni, etc.)
- Runtime library is implemented by MPI and CUDA for GPU clusters
  - CUDA is used for pack/unpack kernels

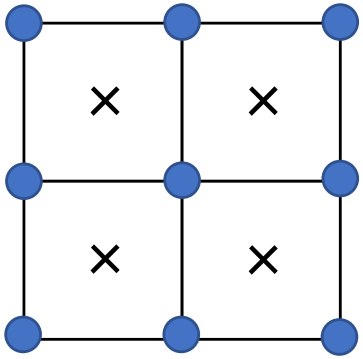


- supporting XACC global-view model in C/Fortran
  - also XACC local-view model in C

CloverLeaf

- A hydrodynamics mini-application
  - developed by UK Atomic Weapons Establishment and University of Warwick
  - published on GitHub (https://github.com/UK-MAC/CloverLeaf)

- Solves compressible Euler equations on a 2D Cartesian grid
  - finite volume method with second-order accuracy
- Physical quantities are arranged on a staggered grid
  - often used for flow simulation on structured grid

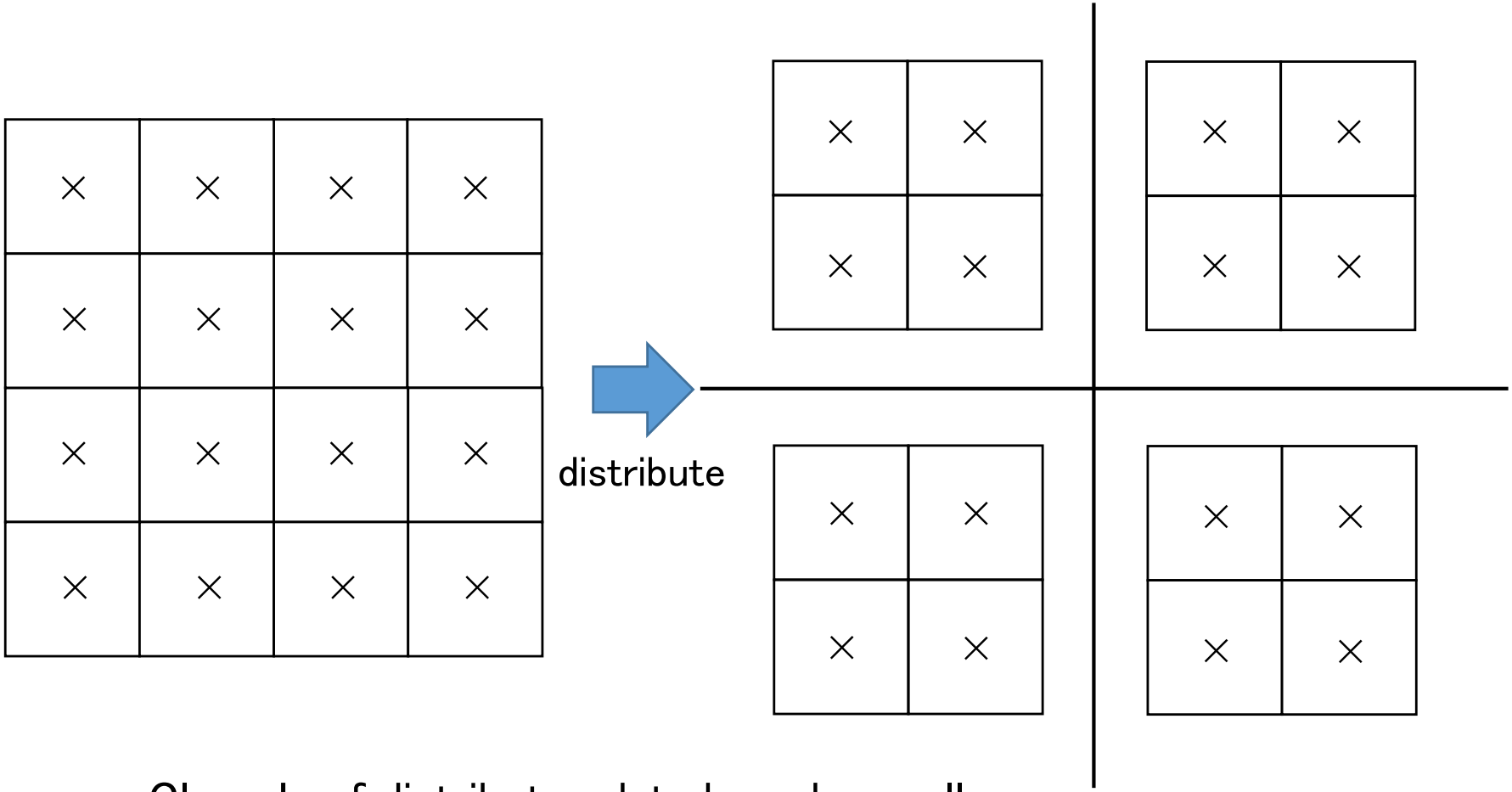


Cells

× Quantities at the centers of cells (e.g. pressure)

● Quantities at the corners of cells (e.g. velocity)

Data distribution (quantities at centers of cells)



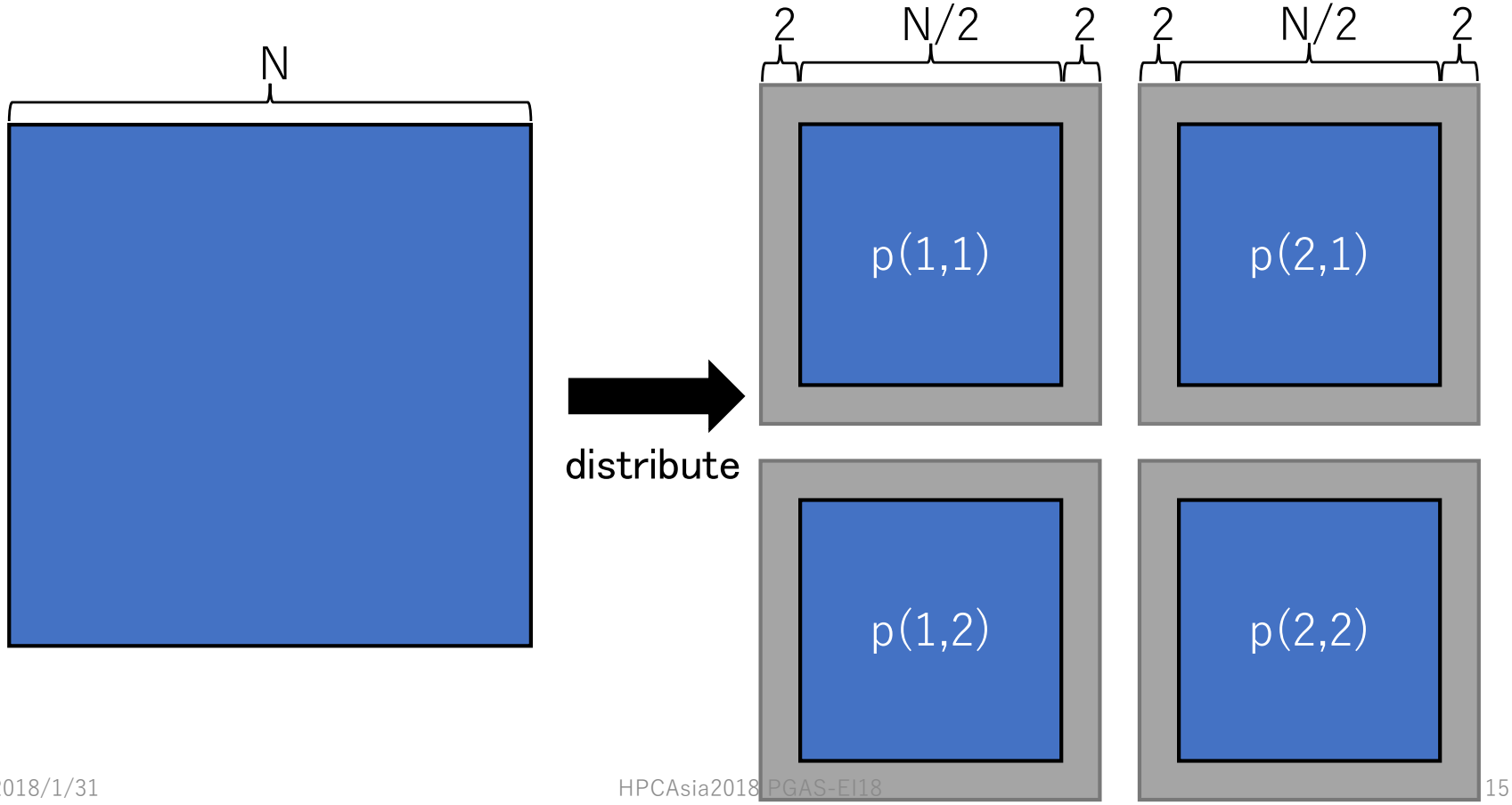
- CloverLeaf distributes data based on cells
- We can distribute quantities just like cells because the number of centers of cell is equal to the number of cells

Distributed array in XACC global-view model (quantities at centers of cells)

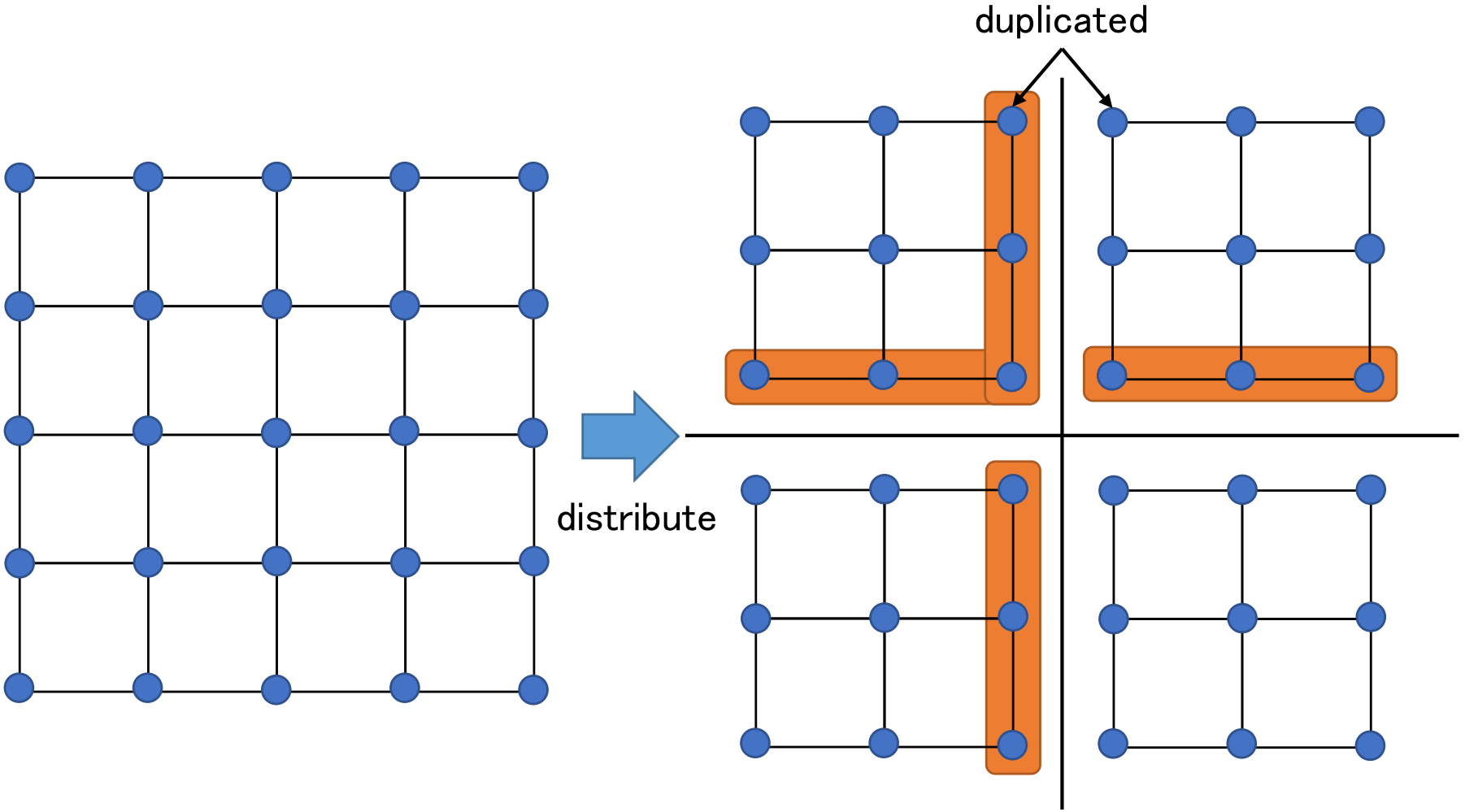
```
REAL(KIND=8) pressure(x_min-2:x_max+2, y_min-2:y_max+2)
!$xmp align (i,j) with t(i,j) :: pressure
!$xmp shadow (2:2,2:2) :: pressure
```

distributing array according to template

adding shadow region for halo region



Data distribution (quantities at corners of cells)



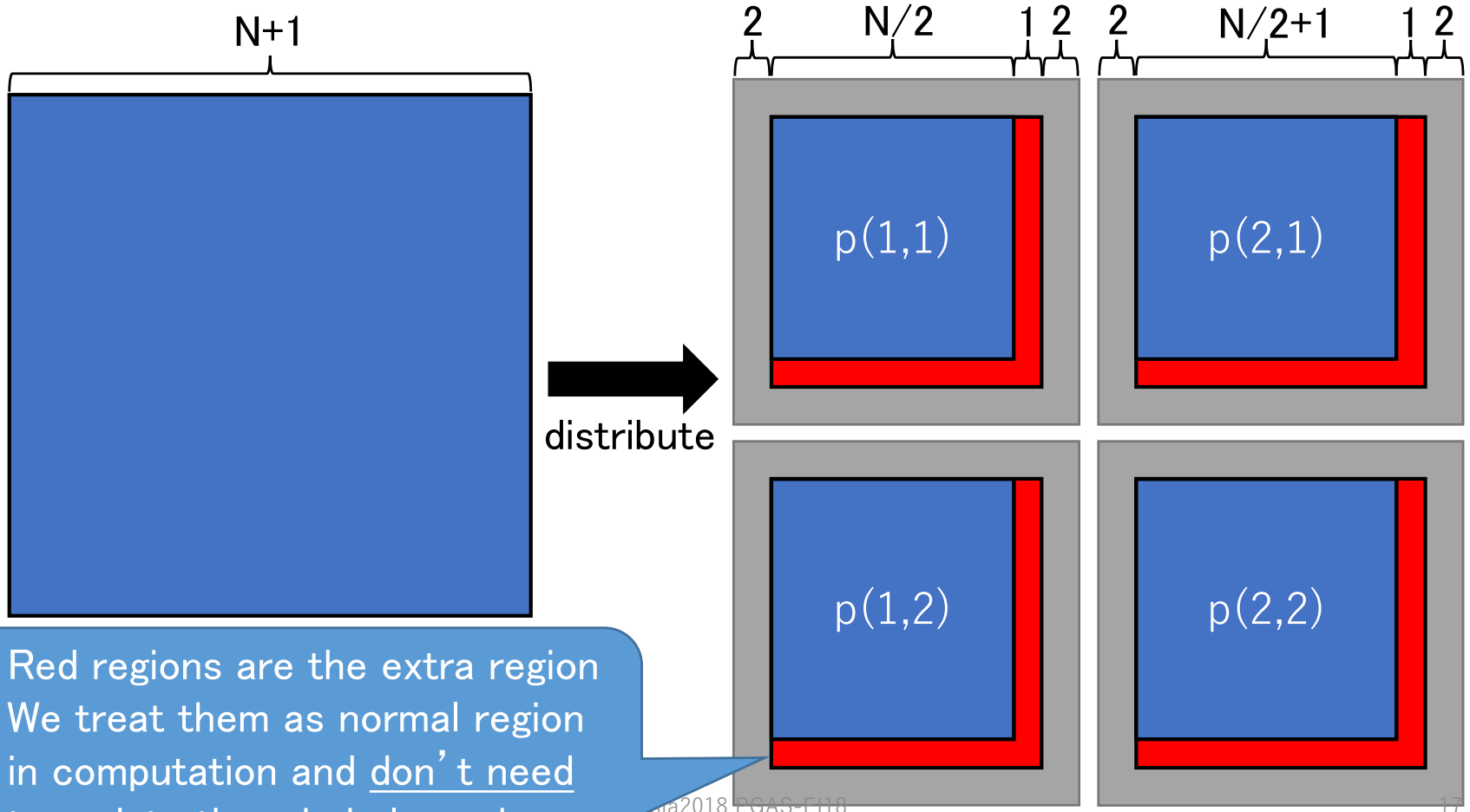
By cell based distribution, corners of cells are duplicated on divided face,
but highlighted elements do not exist in original serial version.
→ We refer the duplicated parts as “extra region”

To allocate the extra region, we utilize `shadow` directive.

Distributed array in XACC global-view model (quantities at corners of cells)

```
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
```

To add the extra region,
incrementing upper width of shadow region by one



- Red regions are the extra region
- We treat them as normal region in computation and don't need to update them in halo exchange

Computation

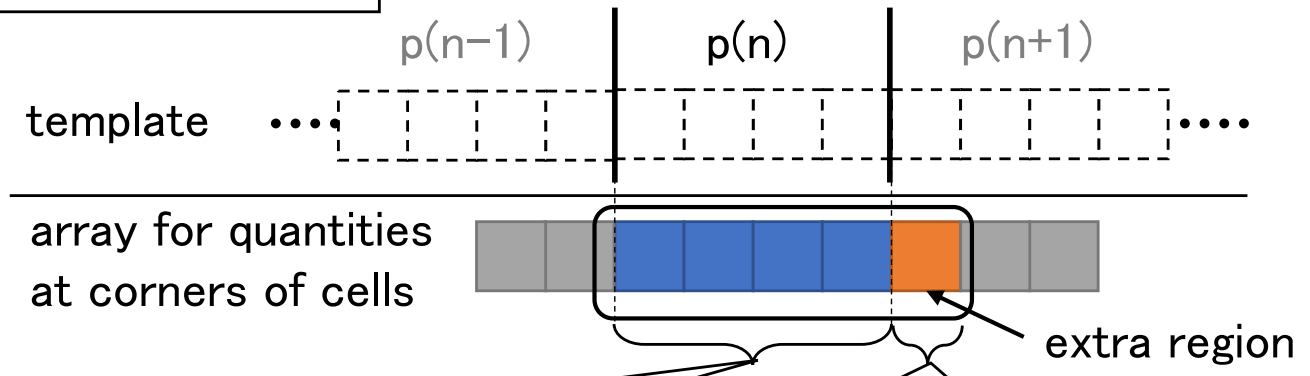
- double nested loops construct main computation

```
!$xmp loop (j,k) on t(j,k) expand(0:1,0:1)
!$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
```

distributing nested loops according to template same as quantity arrays

offloading nested loops and specifying iterations are data-independent

- important feature is `expand` clause
- `expand` clause expands local iteration range



Normal local iteration range is same to local template range
→ extra region is not processed

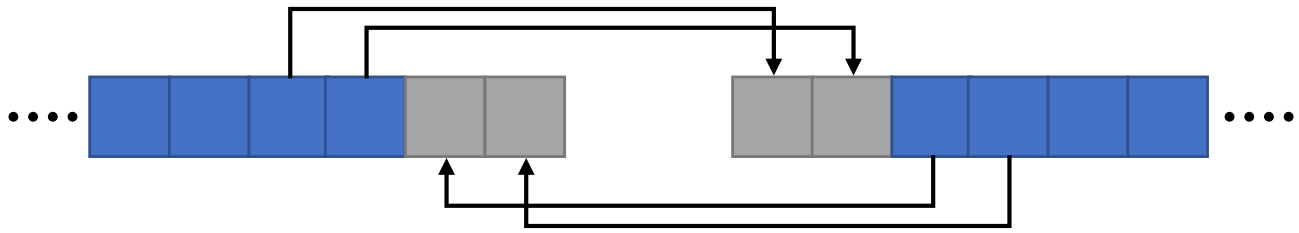
To process extra region, we add `expand` clause

Communication

`width` clause specifies update widths from inside
width (*lower-width* : *upper-width*, ...)

- Main communication is halo exchange
  - Quantities at centers of cells

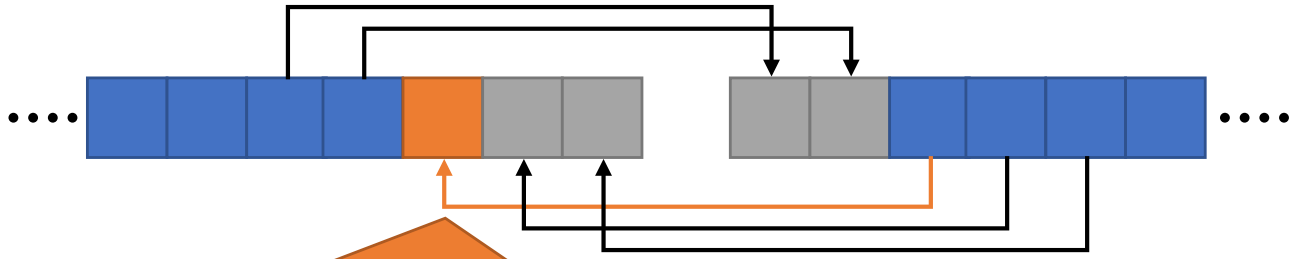
```
!$xmp reflect width(2:2, 2:2) acc
```



communication on accelerator

  - Quantities at corners of cells

```
!$xmp reflect width(2:3, 2:3) acc
```



upper widths are three

- We don't need to update innermost shadow element, which is extra region
- However, we update shadow region including extra region because there is no clause to exclude it

Evaluation

- Three implementations
  - MPI+CUDA
  - MPI+OpenACC
  - XACC

※ We got MPI+CUDA and MPI+OpenACC versions from GitHub and use them with modifications.

- Performance evaluation
  - problem sizes are 960 × 960 and 3840 × 3840 cells
  - the number of time steps is 1000
  - execution time with strong and weak scaling

- Productivity evaluation
  - Source lines of codes (SLOC)
  - Delta SLOC (DSLOC, difference from serial version)

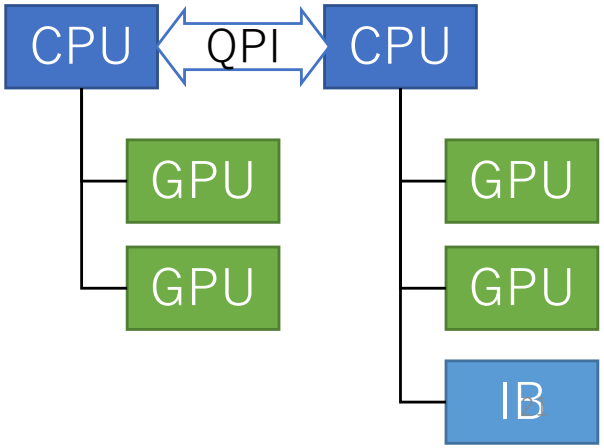
Evaluation environment

- HA-PACS/TCA at Center for Computational Sciences, University of Tsukuba

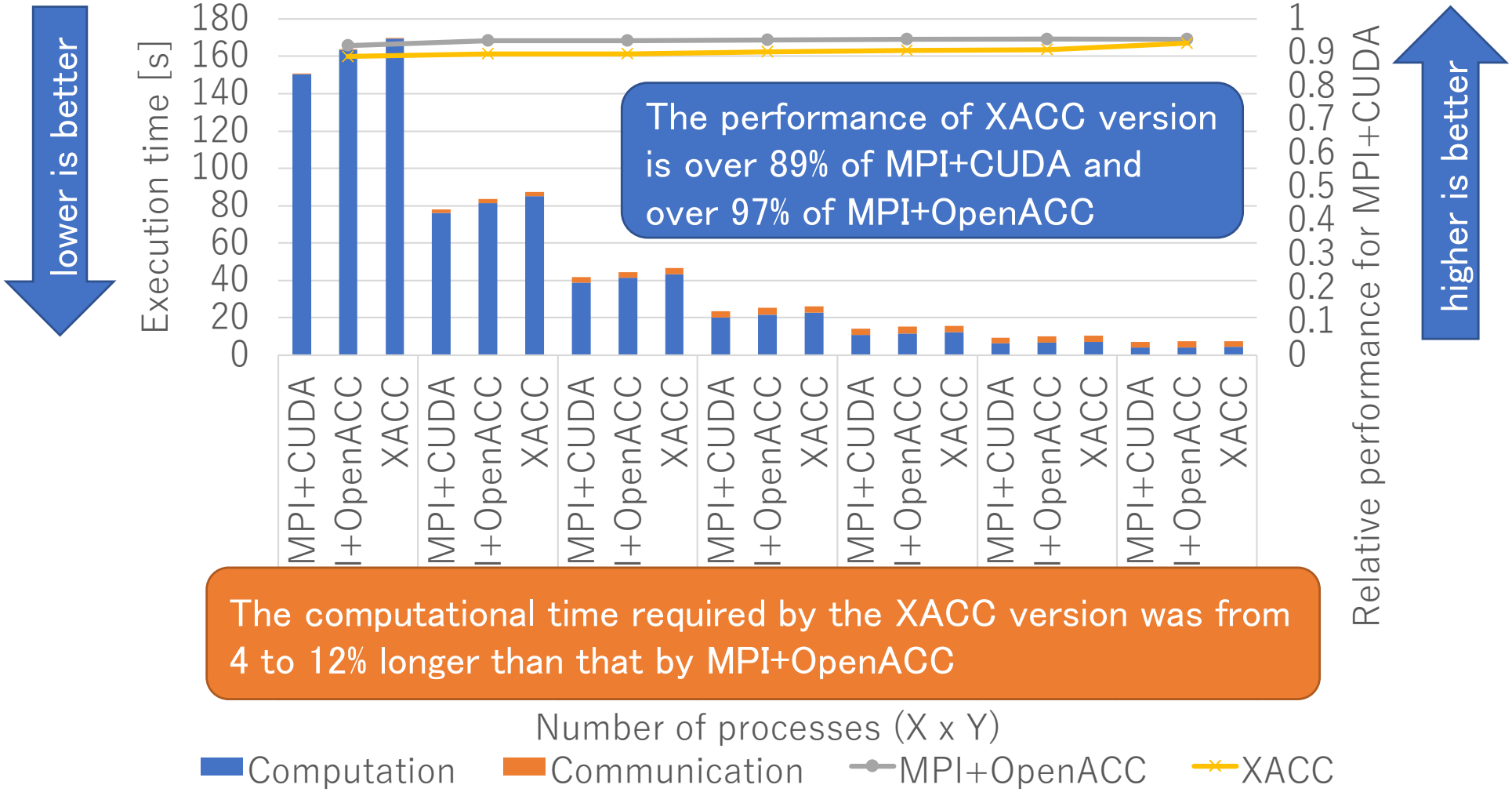
Node configuration and software

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

- 4 processes/node
- 1 GPU/process
- up to 64 processes on 16 nodes



Execution time (3840^2 cells, strong scaling)



Computation performance degradation of XACC version

- The reason is code translation by XACC compiler
  - XACC code : Array is declared as explicit-shape array

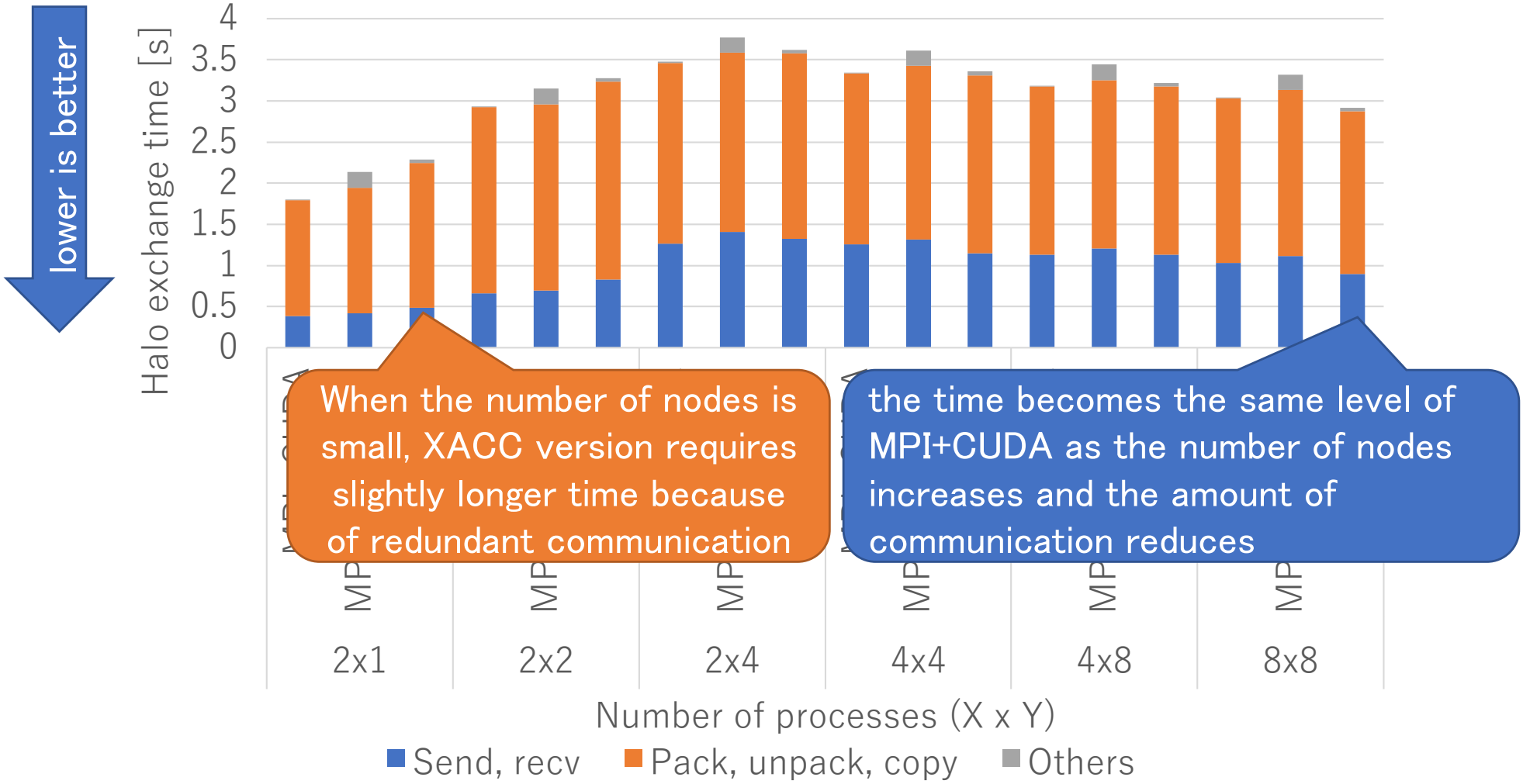
```
REAL(KIND=8) :: pressure(x_min-2:x_max+2, y_min-2:y_max+2)
!$xmp align (i,j) with t(i,j) :: pressure
```

  - Translated code : Array is declared as assumed-shape or deferred-shape array and XACC runtime determines the size dynamically

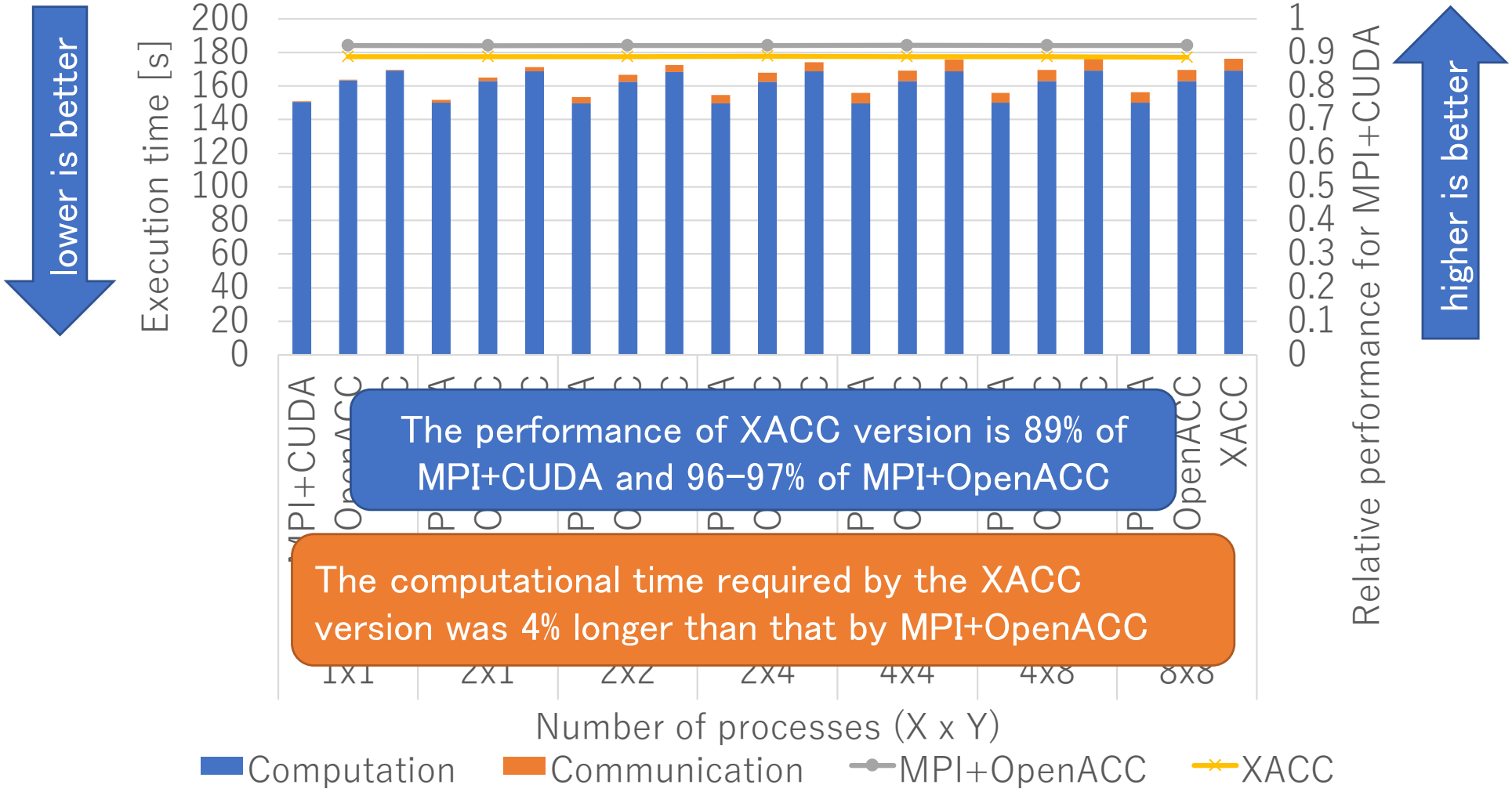
```
REAL(KIND=8) :: XMP__pressure(0: , 0:)
```

- array sizes are undefined at the compile time and the compiler cannot optimize the offset calculations, which increases the register utilization by GPU kernels
- it decreases the number of concurrent execution threads

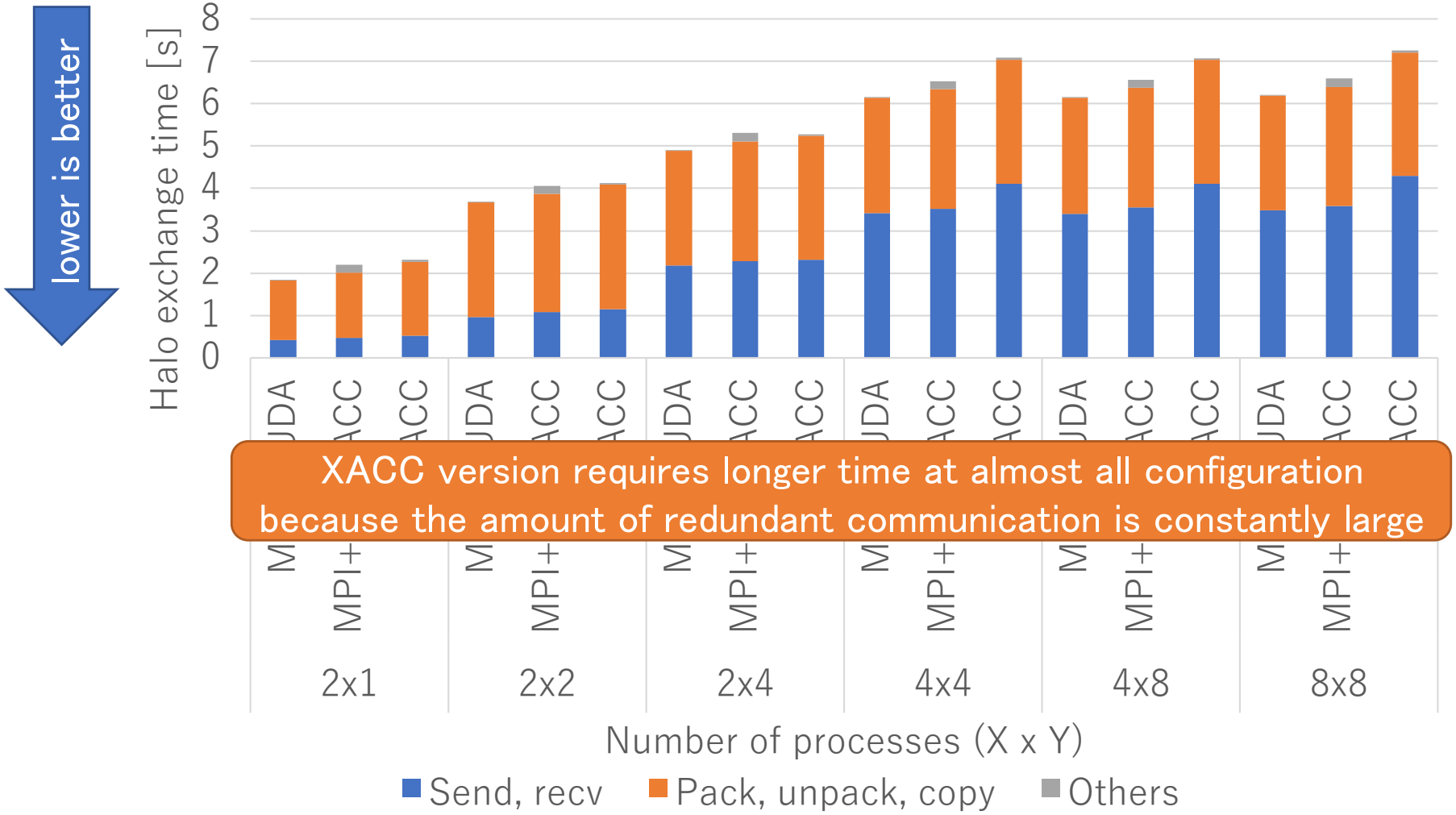
Halo exchange time (3840^2 cells, strong scaling)



Execution time (3840^2 cells, weak scaling)

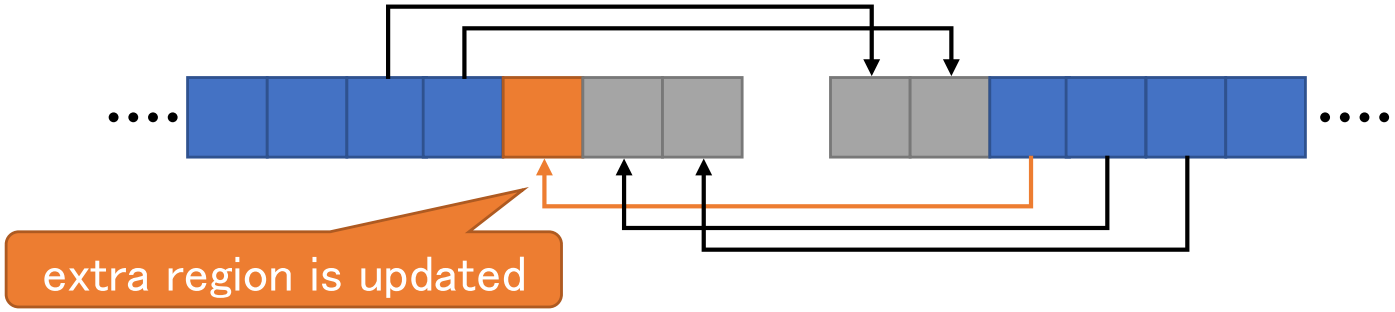


Halo exchange time (3840^2 cells, weak scaling)

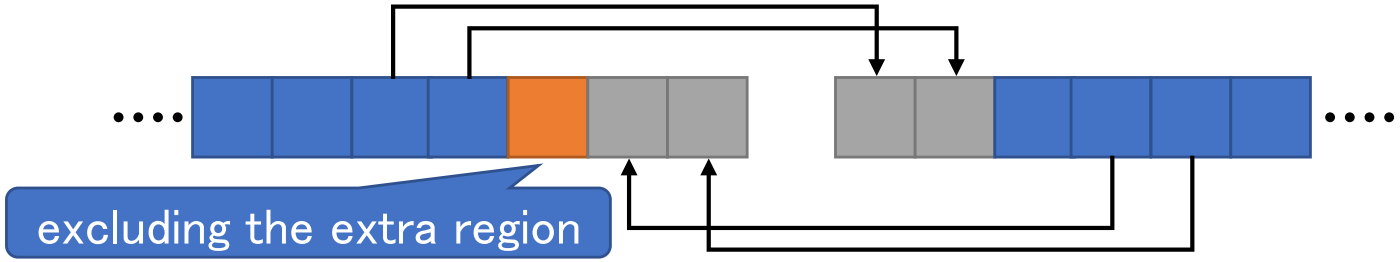


Proposal: extension for reflect directive

- We can specify only `widths` for partial shadow update
  - !$xmp reflect (array) width(2:3)



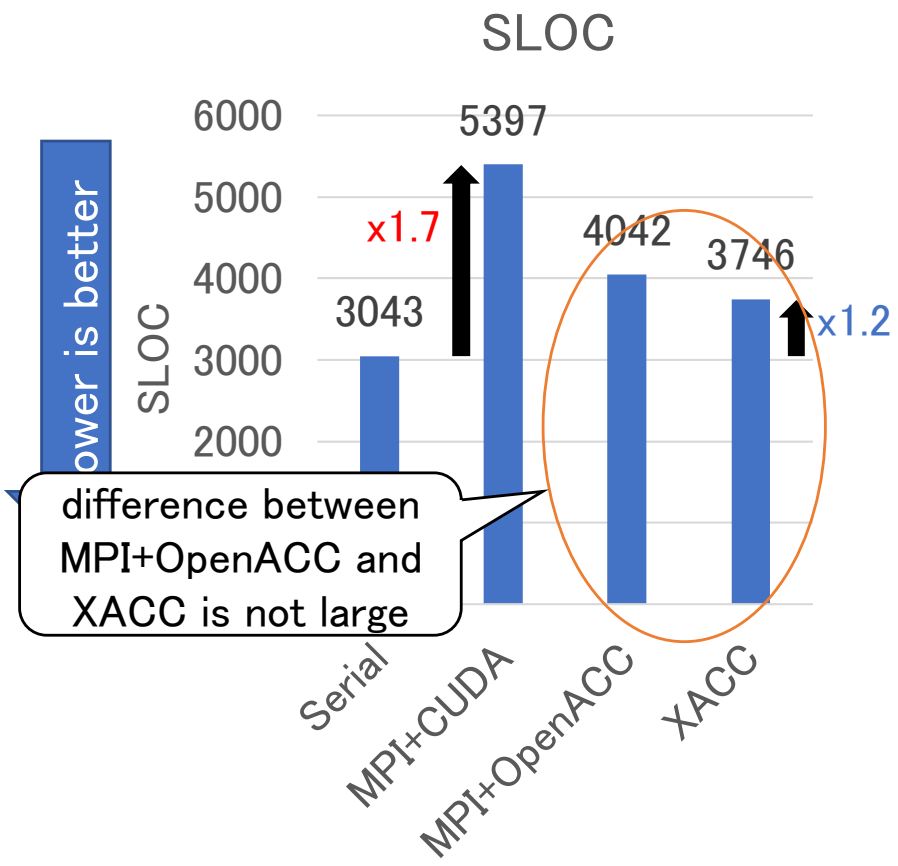
- We propose `offset` clause to skip inner shadow element
  - !$xmp reflect (array) width(2:2) offset(0:1)

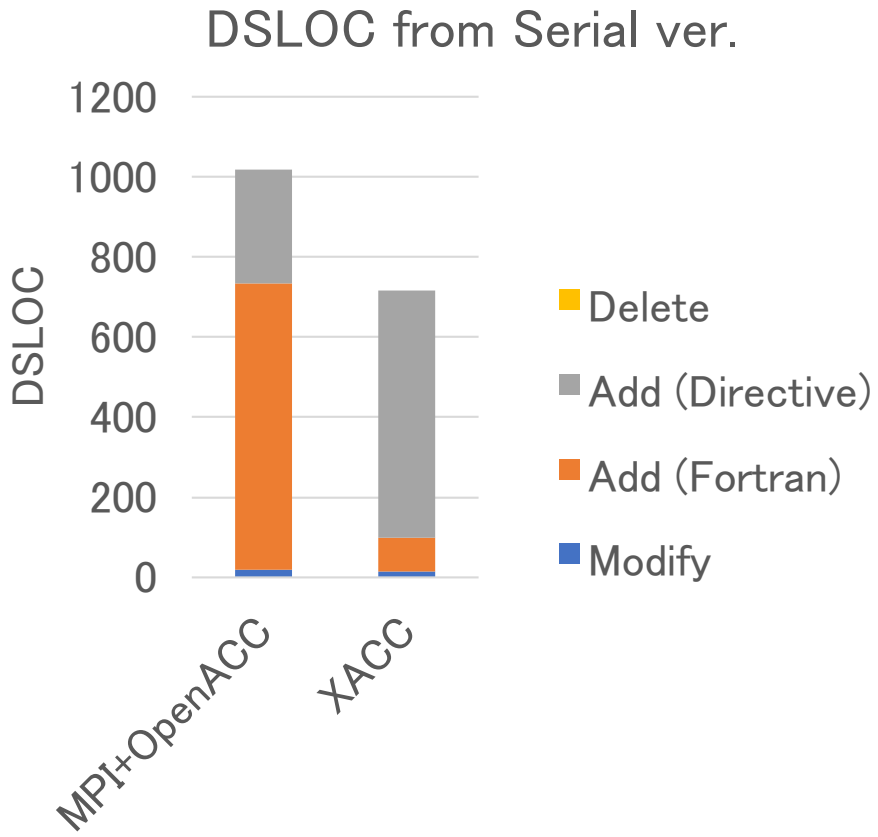


[F] !$xmp reflect ... *[* offset (*reflect-offset [*, *reflect-offset]*...)*]*
[C] #pragma xmp reflect ... *[* offset (*reflect-offset [*, *reflect-offset]*...)*]*

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

Comparison of SLOC and DSLOC





- MPI+CUDA version requires 1.7 times more SLOC than serial version
- XACC version requires only 1.2 times more SLOC than serial version

- major change in MPI+OpenACC version is non-directive code, while that of XACC version is directive addition.

→ XACC version retains a better image of the serial version than MPI+OpenACC version

Related work

- CloverLeaf is implemented for GPU using OpenACC, OpenCL, and CUDA [3]
  - OpenACC achieves comparable performance and high productivity comparing with OpenCL and CUDA
    - XACC also uses OpenACC for accelerator programming

- CloverLeaf is implemented using PGAS model OpenSHMEM and coarray [4]
  - OpenSHMEM and coarray are kind of local-view model
    - we use global-view model of XACC
  - the implementations target normal clusters
    - our work targets accelerated clusters

[3] J.A.Herdman, et al. 2012. Accelerating Hydrocodes with OpenACC, OpenCL and CUDA. In 2012 SC Companion: High Performance Computing, Networking Storage and Analysis. 465-471.

[4] A.C.Mallinson, et al. 2014. Experiences at Scale with PGAS Versions of a Hydrodynamics Application. In Proceedings of the 8th International Conference on Partitioned Global Address Space Programming Models (PGAS'14). ACM, NY, USA, Article 9, 11 pages.

Conclusion

- To evaluate performance and productivity for XACC application program, we implemented a hydrodynamics mini-application CloverLeaf in XACC
  - We showed how staggered grid arrangement is implemented
- XACC has sufficient performance
  - the performance of XACC was over 89% of MPI+CUDA
- XACC has good productivity
  - XACC global-view model allows us to describe applications by adding directives to the serial version of codes

- Future work
  - implementing proposed `offset` clause
  - considering to improve XACC code translation