Preliminary Performance Evaluation of Coarray-based Implementation of Fiber Miniapp Suite using XcalableMP PGAS Language

RIKEN

Hitoshi Murai, Masahiro Nakao,

Hidetoshi Iwashita, Mitsuhisa Sato

Introduction (1)

- For higher performance and productivity, we are developing the *Omni XcalableMP* compiler for a PGAS language *XcalableMP (XMP).*
- XMP supports coarrays.
- We implement the *Fiber* miniapp suite using the coarray feature of XMP.
  - with minimal efforts to rewrite it from the original MPI-based impl.

Goals

- Evaluating the performance and productivity of the coarray feature of XMP and Omni XMP;
- Developing a new miniapp suite of coarray;
- Accumulating know-how on coarray-based parallel programming.

What's XcalableMP ?

www.xcalablemp.org

- A directive-based PGAS language
  - extension for C/Fortran
  - defined by XMP WG of the PC Cluster Consortium.
- Two parallelization models:
  - Global view (HPF-like data/work mapping directives)
  - Local view (coarray)

Data Mapping

```
!$xmp nodes p(2,2)
!$xmp template t(n,n)
!$xmp distribute t(block,block) onto p
      real a(n,n)
!$xmp align a(i,j) with t(i,j)
!$xmp shadow a(1,1)

!$xmp reflect (a)

!$xmp loop (i,j) on t(i,j)
      do j = 2, n-1
      do i = 2, n-1
        w = a(i-1,j) + a(i+1,j) + ...
        ...
```

Work Mapping

Stencil Comm.

Coarray

- A PGAS feature adopted in Fortran 2008.
  - A set of *images*, corresponding to MPI processes, executes a coarray program.
  - The square bracket notation allows accesses to remote data (i.e. coarray).

```
real a(1024)[*], b(1024)

a(512:1024)[1] = b(1:512)

sync all
```

An array a is declared as a coarray.

A local array section `b(1:512)` is put to a remote array section `a(512:1024)` on image 1.

A memory fence and barrier synchronization is performed.

Coarray in XMP

NOTE: the subsection notation in XMP/C is `[base:length:stride]` whereas that in XMP/Fortran is `[base:ubound:stride]`.

- Coarray also available in XMP/C

```
float a[1024]:[*], b[1024];

a[512:512]:[0] = b[0:512];

xmp_sync_all(NULL);
```

cf. Coarrays in Fortran

```
real a(1024)[*], b(1024)

a(512:1024)[1] = b(1:512)

sync all
```

- Subset of images
  - is not allowed by Fortran 2008.
  - XMP allows coarrays to be allocated on a subset of nodes (i.e. images).

Omni XcalableMP Compiler

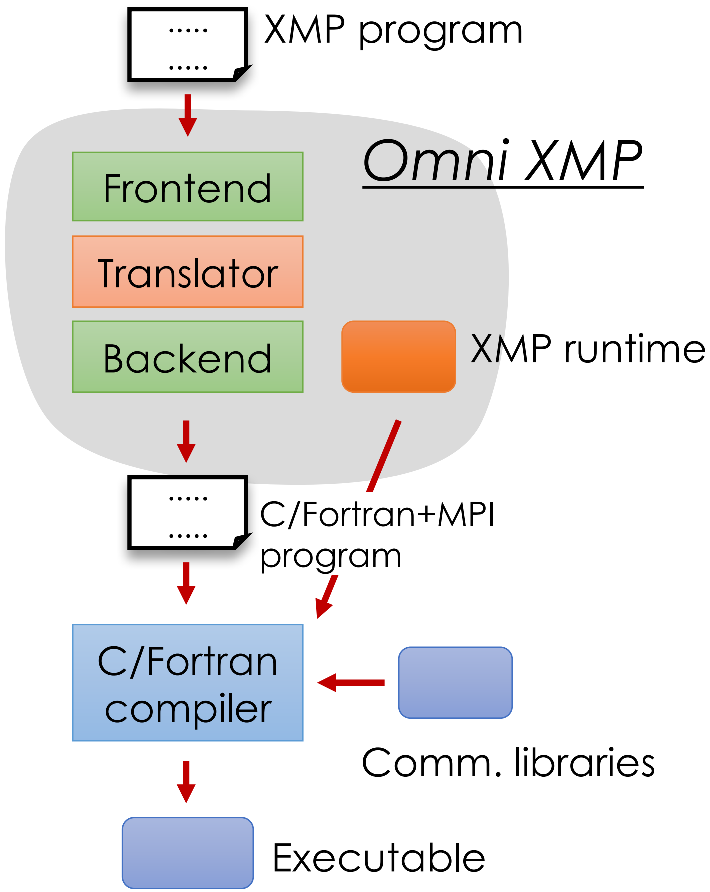
- A reference impl. being developed by RIKEN & U. Tsukuba.
- Latest Ver. 1.2.1 available at:

  `omni-compiler.org`

- Supported platforms include:

  K, Fujitsu FX100, NEC SX, IBM BlueGene, Hitachi SR, Cray, Linux clusters, etc.

- <u>Supports the coarray feature.</u>
  - Some new features of Fortran 2015 (e.g. `co_sum`) already supported in advance.
  - Based on MPI-3, GASNet, or Fujitsu's RDMA.



What's FIBER ?

fiber-miniapp.github.io

- A miniapp suite developed and maintained by RIKEN AICS.
  - The miniapps parallelized with MPI/OpenMP.

Miniapp	Area	Characteristics
CCS QCD	Quantum chromodynamics	Structured grid Monte Carlo
FFVC-MINI	Thermo-fluid analysis	3-dimensional cavity flow
NICAM-DC	Climate	Structured grid stencil
mVMC-MINI	Material science	Many variable variational Monte Carlo
NGS Analyzer-MINI	Genome sequence analysis	Multi task work flow
MODYLAS-MINI	Material science	Molecular dynamics
NTChem-MINI	Quantum chemistry	Molecular orbital method
FFB-MINI	Thermo-fluid analyses	Finite element method, unstructured grid

Basic Strategy for Coarray-based Impl.

1. Declare receive buffers as coarrays.
2. Replace MPI functions with coarray features:
   - `MPI_Send/Isend` → coarray assignment (i.e. put-based)
   - `MPI_Recv/Irecv` → to be deleted
   - collective comms. → intrinsic subroutines (e.g. `co_broadcast`)
   - `MPI_Wait` → the `sync all` statemetnt

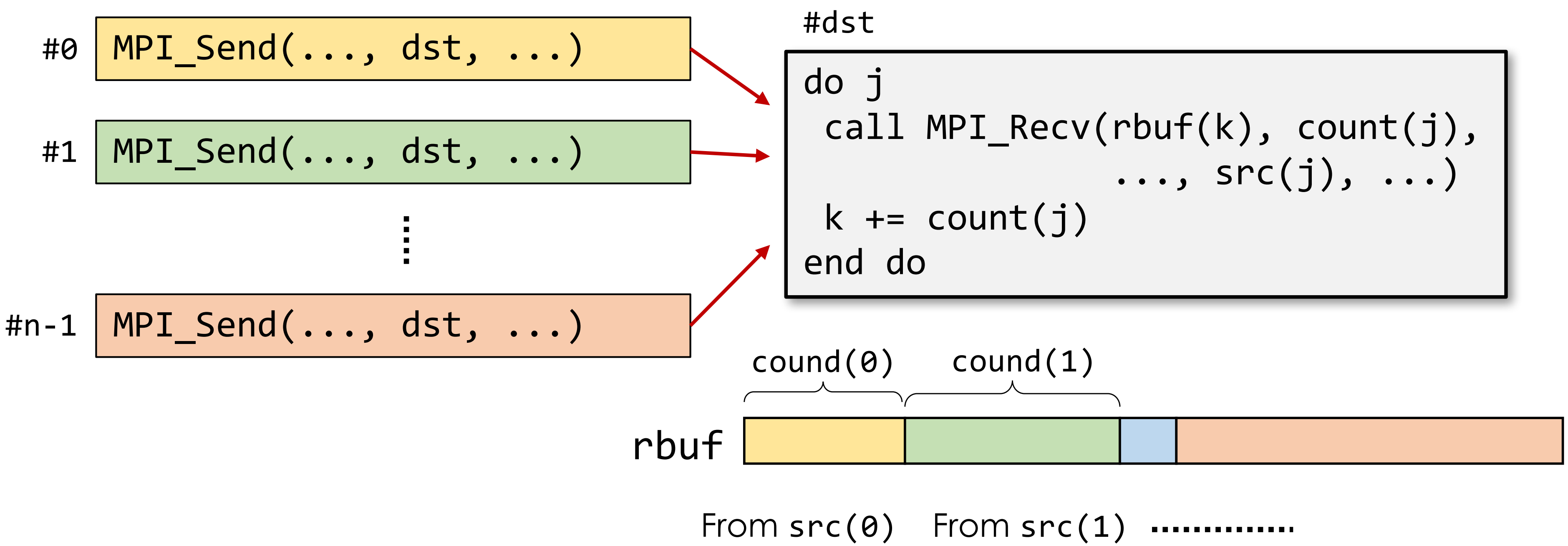
```
real a, b
if (myrank == 0) then
  call MPI_Isend(a, ..., 1, ...)
else if (myrank == 1) then
  call MPI_Irecv(b, ..., 0, ...)
end if

call MPI_Wait(...)
```

```
real a, b[*]
if (this_image() == 1) then
  b[1] = a
else if (this_image() == 2) then
  continue
end if

sync all
```

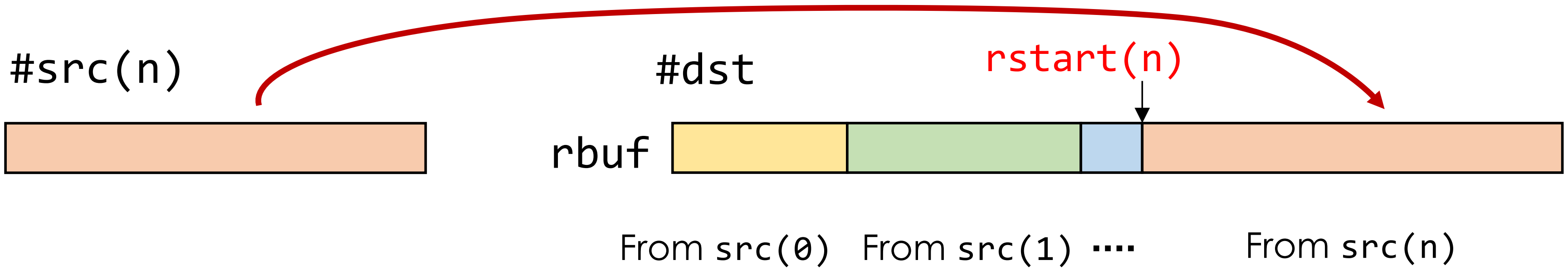
Send/Recv-based Impl. of irregular comms. in original FFB-MINI



Problem: the origin processes do not know which position in the buffer on the target they should put data to.

Coarray-based Impl. of FFB-MINI (1)

- *Images* should exchange the buffer information with each other in advance.



The target process **dst** tells in advance the origin process **src(n)** to put to the position **rstart(n)** in **rbuf**.

```
r_rstart(dst+1)[src(n)+1] = rstart(n)
```

Coarray-based Impl. of FFB-MINI (2)

```
real sbuf(MAXSBUF)
real rbuf(MAXRBUF)[*]
integer sstart(NDOM+1)
integer rstart(NDOM+1)

integer r_rstart(0:NRANK-1)[*]

! (1) exchange rstart (all-to-all)
rstart(1) = 1
do i = 1, NDOM
 r_rstart(myrank)[irank(i)+1] = rstart(i)
 rstart(i+1) = rstart(i) + rcount(i)
end do

sync all

! (2) put operation
sstart(1) = 1
do i = 1, NDOM
 rbuf(r_rstart(irank(i)):r_rstart(irank(i))+scount(i)-1)[irank(i)+1] &
  = sbuf(sstart(i):sstart(i)+scount(i)-1)
 sstart(i+1) = sstart(i) + scount(i)
end do

sync all
```

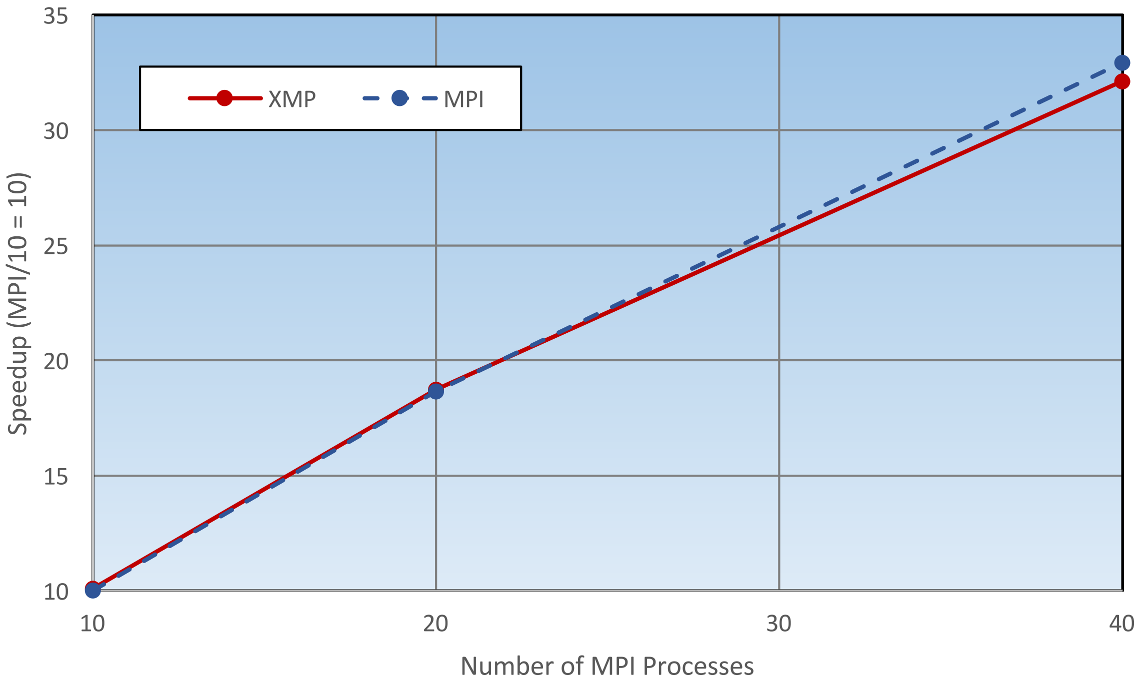
Evaluation

CPU	SPARC64 VIIIfx 2.0GHz, 8cores
Memory	DDR3 SDRAM 16GB, 64GB/s
Network	Tofu (6D mesh/torus), 5Gb/s x 2

- Environment
  - Target: K computer
  - Base language environment K-1.2.0-22
  - Omni 1.1.3-20170809
    - Coarrays based on the *extended RDMA interface* of Fujitsu's MPI.
    - An image is mapped to an MPI process at runtime; and
    - an MPI process is assigned to a compute node in hybrid parallelization or to a core of a compute node in flat parallelization.

NICAM-DC

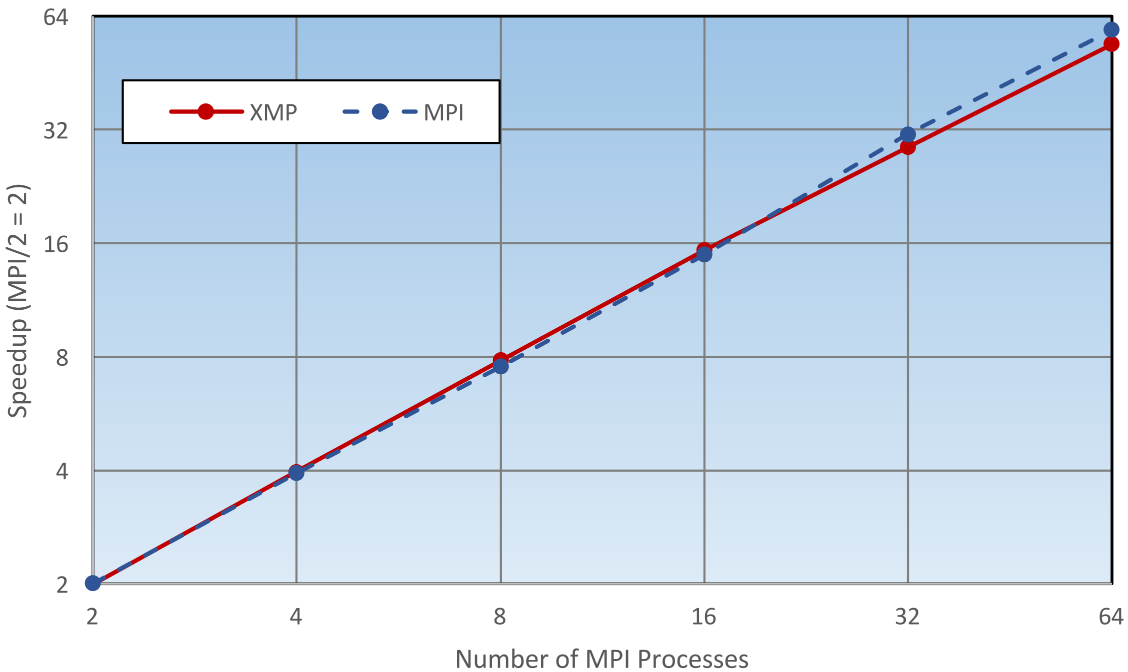
- Threading method: automatic parallelization provided by the compiler
- Target data: gl06rl01z80pe{10,20,40} (strong scaling)
- Compiler options: `-Kfast,parallel,auto,ocl,preex,array_private,noalias=s,mfunc=2 -Kparallel_iteration=8,instance=8,dynamic_iteration -Kprefetch_cache_level=all,prefetch_iteration_L2=50 -Ksimd -Ntl_notrt`
- Timing region: "Total" of the built-in timing feature



- The coarray-based impl. is almost comparable to the original MPI-based one.

NTChem-MINI

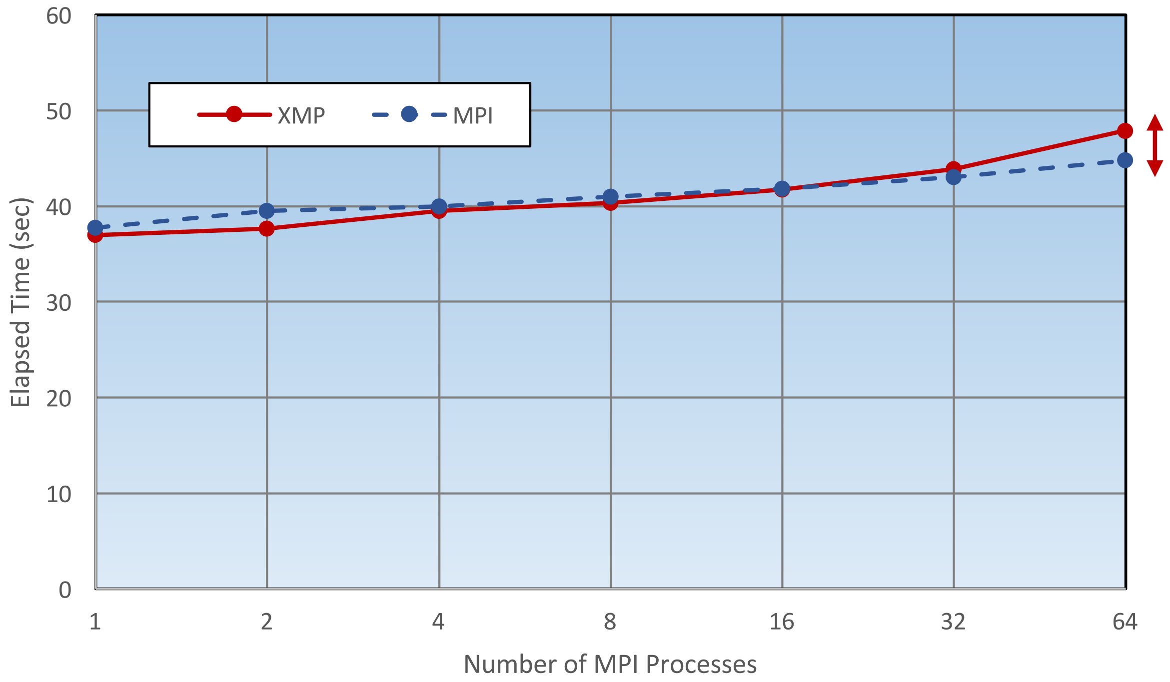
- Threading method: OpenMP only for BLAS
- Target data: taxol (strong scaling)
- Compiler options: `-Kfast,simd=2`
- Timing region: "RIMP2_Driver" of the built-in timing feature



- The coarray-based impl. is almost comparable to the original MPI-based one.

FFB-MINI

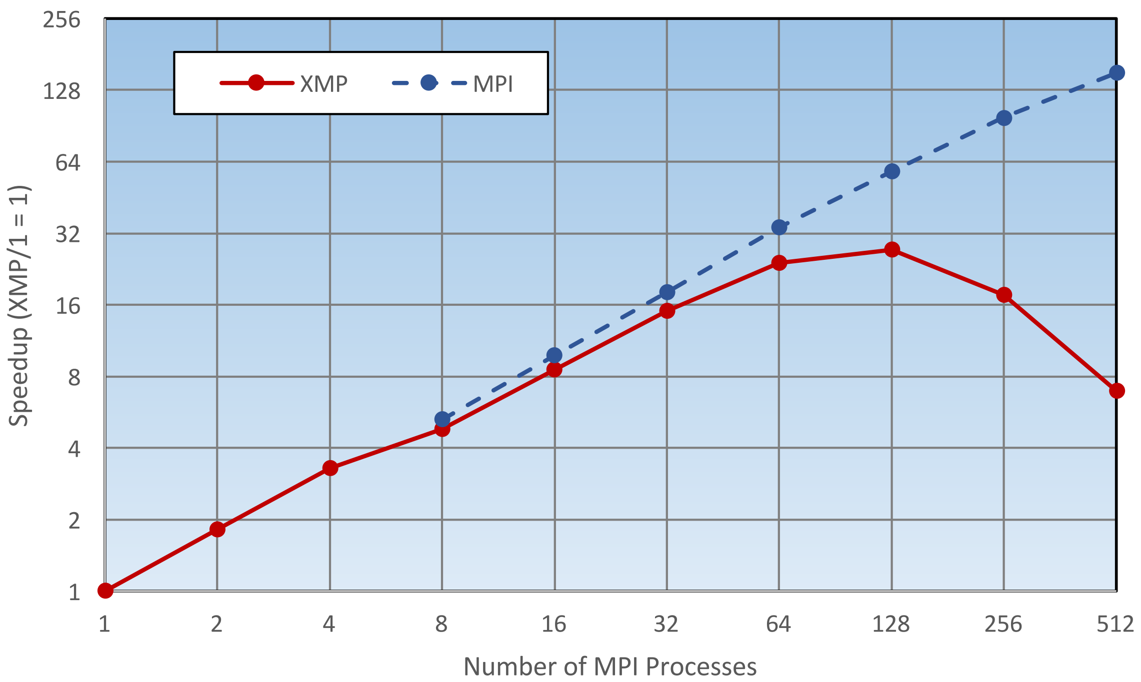
- Threading method: none (flat parallelization)
- Target data: #elements per domain = 463 (weak scaling)
- Compiler options: `-Kvisimpact,ocl`
- Timing region: "MAIN LOOP" of the built-in timing feature



- The coarray-based impl. is almost comparable to the original MPI-based one.
- A small performance degradation on 64 processes is due to the overhead of exchanging buffer information.

CCS QCD

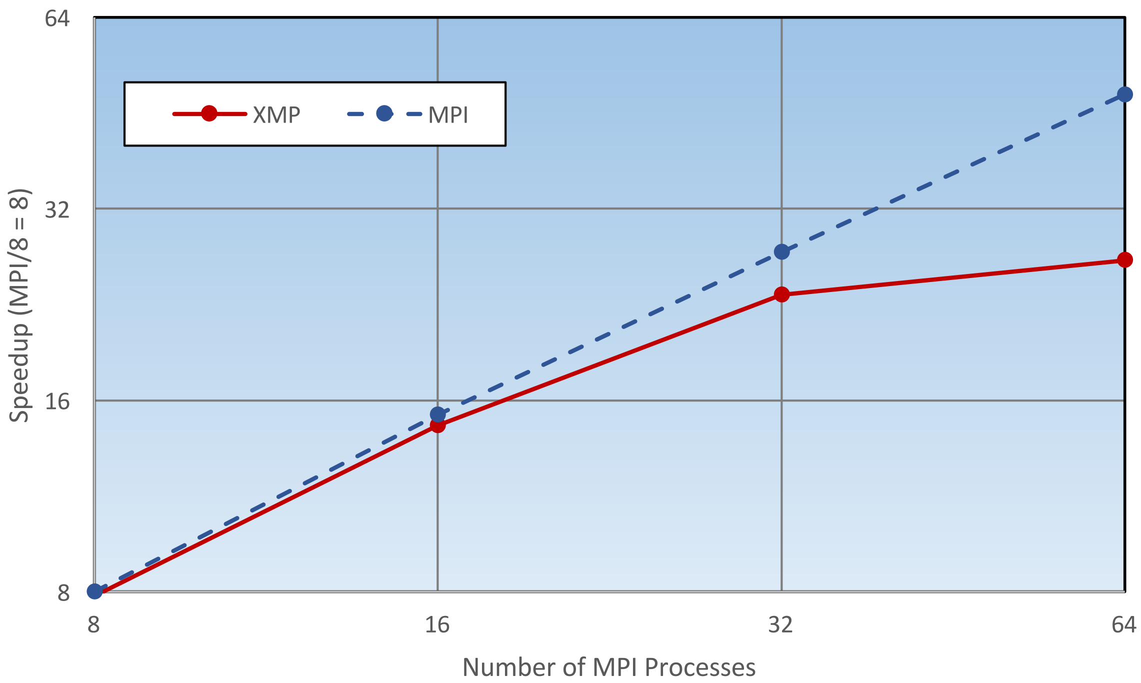
- Threading method: none (flat parallelization)
- Target data: Class 2 (32x32x32x32) (strong scaling)
- Compiler options: `-Kfast -KXFILL -Ksimd=2`
- Timing region: sum of "Clover + Clover_inv Performance" and "BiCGStab(CPU:double precision) Performance" of the built-in timing feature



- For the coarray-based impl., performance degrades in the >64-processes executions.

MP
MODYLAS-MINI

- Threading method: OpenMP
- Target data: wat111 (strong scaling)
- Compiler options: `-Kfast,openmp,parallel,array_private,auto,ilfunc,ocl,preex,NOFLTLD,simd=2,mfunc=2`
- Timing region: “Main Loop” of the built-in timing feature or its equivalent.



- For the coarray-based impl., performance degrades in the >32-processes executions.

Discussion on the Slowdown in CCS QCD and MODYLAS-MINI (1)

- Coarrays in CCS QCD and MODYLAS-MINI are declared as "allocatable."
  - The receive buffer is a dummy argument.
  - The size of the receive buffer is not identical on every image or not a constant.
- The overhead for allocation of a coarray is very large.
  - ∵ Its address must be exchanged among all images.

```
subroutine sub

real a(n), b(n)
...
MPI_Isend(a, ...)
MPI_Irecv(b, ...)
```

```
subroutine sub

real a(n), b(n)
real, allocatable :: buf(:)[*]
allocate (buf(n)[*])
buf = b
...
buf[p] = a
b = buf
...
```

Discussion on the Slowdown in CCS QCD and MODYLAS-MINI (2)

- Possible solutions:
  - [*For users*] Make the allocations less frequent.
    - Declare the coarrays with the maximum size and the SAVE attribute; or
    - at the root (or as high level as possible) of the call tree.
  - [*For compiler developers*] Delete or reduce the overhead of allocatable coarrays.

Summary

- We implemented five of the Fiber miniapp suite using coarrays;
  - For regular ones, it was possible in a straightforward way;
  - More complicated rewrites were needed for FFB-MINI.
- evaluated their performance using Omni XMP on the K computer.
  - Three of them could achieve good performance;
  - The other two suffered performance degradation due to the large overhead of allocatable coarrays.

Future Works

- Evaluating our impl. for other platforms (including accelerators) and/or compilers;
- Releasing it as a miniapp suite of coarrays;
- Exploring even better coarray-based impl.;
- Improving allocatable coarrays in Omni XMP;
- Trying impl. based on the global-view model.