# XcalableMPによるFiberミニアプリ集の実装と評価


平成28年11月7日
一般財団法人 高度情報科学技術研究機構
原山 卓也　井上 孝洋　手島 正吾

# 目的

- XcalableMPのローカルビューモデルであるXMPのCoarray機能を用いて、Fiberミニアプリ集への実装と評価を行う

  → PGAS(Pertitioned Global Address Space)言語であるCoarrayのベンチマークとして整備することも考慮している
- Coarrayによる並列化に関する知見を得る

# FIBERミニアプリとは

- FIBERミニアプリとは、アプリとシステムのコデザインのために理研AICSにて整備・開発されたツールであり、さまざまな分野の実アプリ（フルアプリ）から重要な特徴を抽出して作られたミニアプリと呼ばれるアプリである

| | |
|---|---|
| CCS QCD | A QCD miniapp originally developed by Kenichi Ishikawa (Hiroshima University), et al. |
| FFVC-MINI | A Navier-Stokes solver for 3D unsteady thermal flow of incompressible fluid. Derived from the FFVC simulation program developed by Kenji Ono (RIKEN AICS), et al. |
| NICAM-DC-MINI | A miniapp based on NICAM-DC that is derived from NICAM (Nonhydrostatic ICosahedral Atmospheric Model). |
| mVMC-MINI | A suite of mVMC program and test data. mVMC analyzes the physical characteristics of the strongly correlated electron systems. |
| NGS Analyzer-MINI | A miniapp for genome analysis. NGS Analyzer performs human genome analysis to identify genetic differences among persons or cancer cell's mutations. |
| MODYLAS-MINI | A miniapp based on a general-purpose molecular dynamics simulation program MODYLAS. |
| NTChem-MINI | An ab-initio quantum chemistry miniapp for the molecular electronic structure calculation. |
| FFB-MINI | A miniapp based on a general-purpose thermal flow FEM program, FrontFlow/blue. |

http://fiber-miniapp.github.io

# XMPの実装方針

- 既存の MPI 通信部分を coarray の片側通信(Putベース)と集団通信(co_*XXXX* ルーチン)に置換える
  - MPI_Send 関数または MPI_Isend 関数 → coarray 代入文(Put)
  - MPI_Recv 関数または MPI_Irecv 関数 → 削除
  - MPI_Wait 関数または MPI_Waitall 関数 → sync all
  - 集団通信関数 → co_broadcast, co_sum など
  - サブコミュニケータに対する通信処理 → 後述

MPIの記述

```
integer a,b,c

if (myrank == 0) then
  call MPI_Isend(a, 1, ..., 1, ..., ierr)
else if (myrank == 1) then
  call MPI_Irecv(b, 1, ..., 0, ..., ierr)
end if
  ⋮
call MPI_Wait(irec, istat, ierr)

call MPI_Bcast(c, 1, ..., 0, ..., ierr)
```

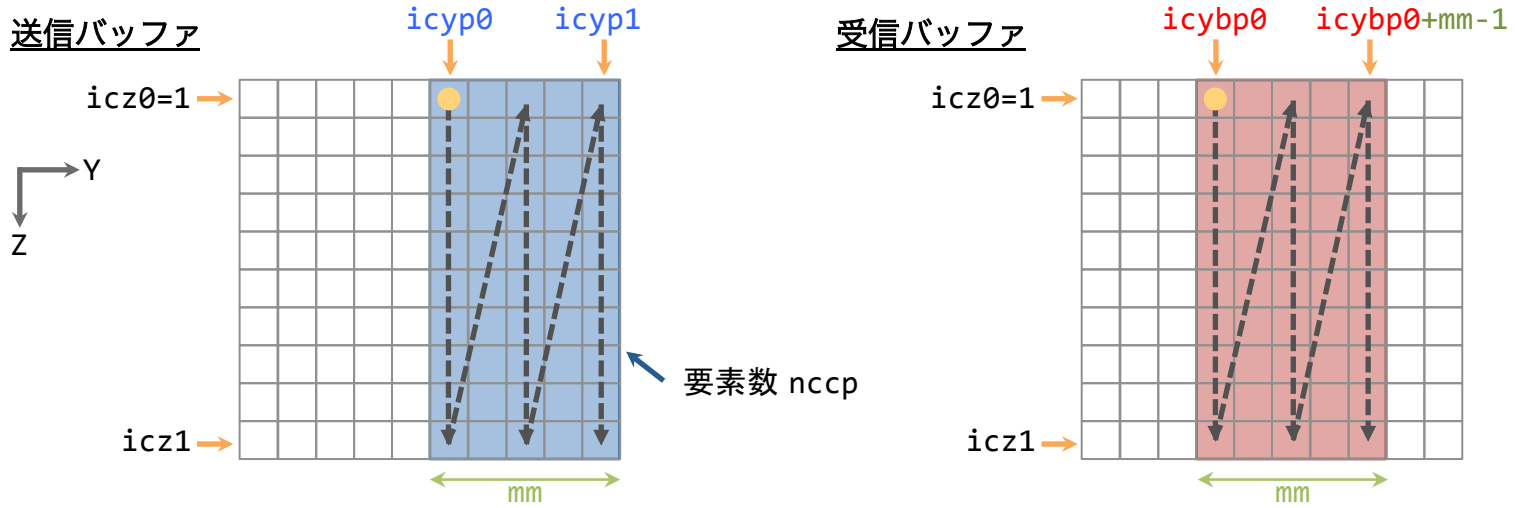
coarrayの記述

```
integer a, b[*], c

if (this_image() == 1) then
  b[2] = a
else if (this_image() == 2) then
  continue
end if
  ⋮
sync all

call co_broadcast(c,source_image=1)
sync all
```

# Coarray による実装例(1)

## MPI_Sendrecv 関数

```
integer(4),allocatable :: na_per_cell(:,:,:)

allocate(na_per_cell(lzdiv+4,lydiv+4,lxdiv+4))
nccp = (icz1-icz0+1) * (icyp1-icyp0+1)
call mpi_sendrecv(na_per_cell(icz0,icyp0,icx),  nccp, MPI_INTEGER, ipy_pdest, myrank,   &
                  na_per_cell(icz0,icybp0,icx), nccp, MPI_INTEGER, ipy_psrc,  ipy_psrc, &
                  MPI_COMM_WORLD, istatus, ierr)
```




## Coarary 記法

```
integer(4),allocatable :: na_per_cell(:,:,:)[:]

allocate(na_per_cell(lzdi+4,lydiv+4,lxdiv+4)[*])
mm = icyp1-icyp0+1
na_per_cell(icz0:icz1,icyp0:icyp0+mm-1,icx)[ipy_pdest+1] &
  = na_per_cell(icz0:icz1,icybp0:icybp0+mm-1,icx)
sync all
```

# Coarray による実装例(2)

- MPI_Isend/Irecv/Wait 関数

```
REAL(8), pointer :: SendBuf(:), RecvBuf(:)

DO IbBat_proc = 1, NOccBat_per_Pro
  RecvBuf => RIInt3c3a(:,Ib_Send:)
    ⋮
  DO Jarank_diff = 0, NProcs_half
      ⋮
    DO IaBat_proc = 1, IaBat_Proc_End
       ⋮
      if( commSizeEach(commPhase) > 0 ) then
        CALL MPI_ISend(SendBuf(1,commIndexEach(commPhase)), commSizeEach(commPhase), &
             MPI_DOUBLE_PRECISION, Jranksend_1, commPhase, MPI_COMM_MO, ireq(1), IErr)
        CALL MPI_IRecv(RecvBuf(1,commIndexEach(commPhase)), commSizeEach(commPhase), &
             MPI_DOUBLE_PRECISION, Jrankrecv_1, commPhase, MPI_COMM_MO, ireq(2), IErr)
      end if
        ⋮
      DO LNumber_base = 1, LCount + (NUM_STREAM-1)
         ⋮
        if ( LNumber >=1 .and. LNumber <= LCount ) then
           ⋮
          if ( commPhase <= commCount .and. commSizeEach(commPhase) > 0 ) then
            CALL MPI_Wait(ireq(1), istat1, IErr)
            CALL MPI_Wait(ireq(2), istat2, IErr)
          end if
            ⋮
          if ( commPhase <= commCount .and. commSizeEach(commPhase) > 0 ) then
            CALL MPI_ISend(SendBuf(1,commIndexEach(commPhase)), commSizeEach(commPhase), &
                           MPI_DOUBLE_PRECISION, Jranksend_1, commPhase, MPI_COMM_MO, ireq(1), IErr)
            CALL MPI_IRecv(RecvBuf(1,commIndexEach(commPhase)), commSizeEach(commPhase), &
                           MPI_DOUBLE_PRECISION, Jrankrecv_1, commPhase, MPI_COMM_MO, ireq(2), IErr)
          end if
           ⋮
        end if
      END DO
       ⋮
    END DO
  END DO
END DO
```

送受信バッファにポインタ配列を用いた非同期通信

## Coarary 記法

```
REAL(8), pointer :: SendBuf(:), RecvBuf(:)
①REAL(8), allocatable :: sbuf(:)[:] , rbuf(:)[:]
integer :: bufsize
integer, save :: jsta


DO IbBat_proc = 1, NOccBat_per_Pro
  RecvBuf => RIInt3c3a(:,Ib_Send:)
    ⋮
   DO Jarank_diff = 0, NProcs_half
       ⋮
     DO IaBat_proc = 1, IaBat_Proc_End
        ⋮
       if( commSizeEach(commPhase) > 0 ) then
         bufsize = commSizeEach(commPhase)
         allocate(sbuf(bufsize)[*])            処理①
         allocate(rbuf(bufsize)[*])
         jsta = commIndexEach(commPhase)
       ②sbuf(1:bufsize) = SendBuf(1:bufsize,jsta)
         rbuf(1:bufsize)[Jranksend_1+1] = sbuf(1:bufsize)
       end if
         ⋮
       DO LNumber_base = 1, LCount + (NUM_STREAM-1)
          ⋮
         if ( LNumber >=1 .and. LNumber <= LCount ) then
            ⋮
           if ( commPhase <= commCount .and. commSizeEach(commPhase) > 0 ) then
             sync all
           ③RecvBuf(1:bufsize,jsta) = rbuf(1:bufsize)
             if (allocated(sbuf)) deallocate(sbuf)
             if (allocated(rbuf)) deallocate(rbuf)
           end if
             ⋮
           if ( commPhase <= commCount .and. commSizeEach(commPhase) > 0 ) then
                        処理①
           end if
            ⋮
```

- ポインタ配列は coarray 配列に置換えることができない

ここでの非同期通信の置換えは…

① ローカルに coarray 配列を準備
② Coarray の Put 通信を実行
③ 通信終了後は元の配列にデータを戻す

# Coarray による実装例(3)

- MPI_Bcast 関数

```
call MPI_Bcast(arg, 1, MPI_INTEGER, ids, MPI_COMM_WORLD, ierr)
```

- Coarray 記法

```
call co_broadcast(arg, ids+1)
sync all
```

- MPI_Allreduce 関数（MPI演算: MPI_SUM）

```
call MPI_Allreduce(r8, r8tmp, 1, MPI_REAL8, MPI_SUM, MPI_COMM_WORLD, ierr)
r8 = r8tmp
```

- Coarray 記法

```
r8tmp = r8
call co_sum(r8tmp, r8)
sync all
```

- MPI_Allreduce 関数（MPI演算: MPI_MAX）

```
call MPI_Allreduce(nGrp, nGrpMax, 1, MPI_INTEGER, MPI_MAX, MPI_COMM_WORLD, Ierr)
```

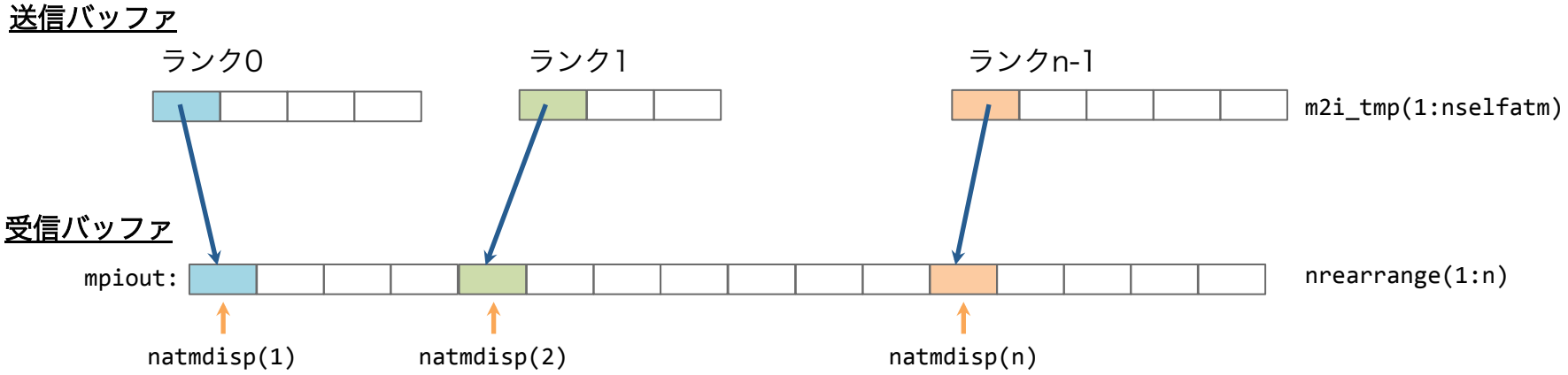
- Coarray 記法

```
call co_max(nGrp, nGrpMax)
sync all
```

# Coarray による実装例(4)

- MPI_Gatherv 関数

```
integer(4),allocatable :: nrearrange(:)
integer(4) :: m2i_tmp(na1cell*lxdiv*lydiv*lzdiv)

allocate(nrearrange(n))
call MPI_Gatherv(m2i_tmp, nselfatm, MPI_INTEGER, nrearrange, natmlist, natmdisp, MPI_INTEGER, &
                 mpiout, MPI_COMM_WORLD, ierr)
```



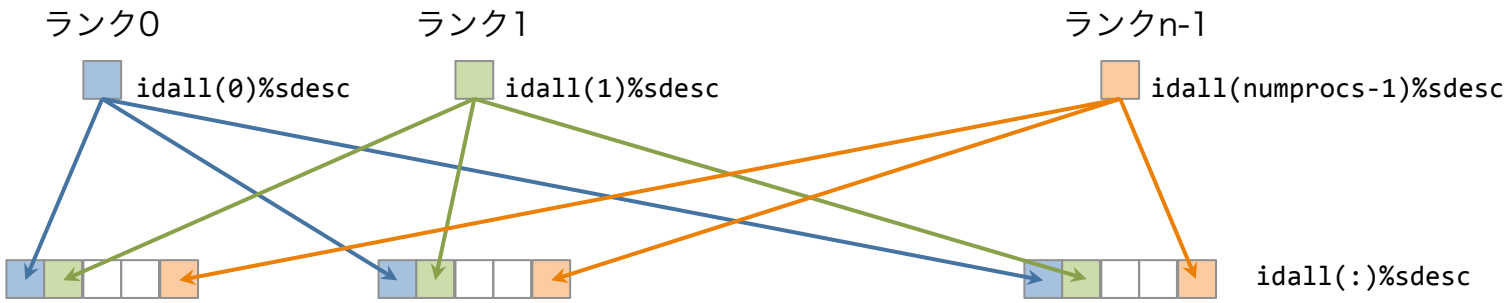

- Coarary 記法

```
integer(4),allocatable :: nrearrange(:)[:]

allocate(nrearrange(n)[*])
me = this_image()
ms = natmdisp(me)
nrearrange(ms:ms+nselfatm-1)[mpiout+1] = m2i_tmp(1:nselfatm)
sync all
```

# Coarray による実装例(5)

- MPI_Bcast 関数

```
do i=0, numprocs-1
  call MPI_Bcast(idall(i)%sdesc, 1, MPI_INTEGER, i, MPI_COMM_WORLD, ierr)
end do
```




- Coarary 記法

```
integer buf

do i=1, numprocs
  buf = idall(i)%sdesc
  call co_broadcast(buf,i)
  sync all
  idall(i-1)%sdesc = buf
end do
```

# Coarray による実装例(6)

- MPI_Allgather 関数 + reduction 演算

```
sendbuf(1) = localsum
call MPI_Allgather( sendbuf,                &
                    1,                      &
                    MPI_DOUBLE_PRECISION, &
                    recvbuf,                &
                    1,                      &
                    MPI_DOUBLE_PRECISION, &
                    ADM_COMM_RUN_WORLD,   &
                    ierr)

globalsum = sum( recvbuf(:) )
```

コミュニケータ ADM_COMM_RUN_WORLD は MPI_COMM_WORLDと同等

通信パターンが MPI_Allreduce 関数（MPI_SUM）に相当するため、co_sum ルーチンで置換えが可能

- Coarary 記法

```
localsumc = localsum
call co_sum(localsumc, globalsum)
sync all
```

# サブコミュニケータの利用

- コミュニケータ MPI_COMM_WORLD の分割により新たに作られたサブコミュニケータへの Coarray Fortran の対応は未対応（Fortran2008）
- オリジナルの FIBER ミニアプリでは、NICAM-DC-MINI と NTChem-MINI でサブコミュニケータを使用した通信を行っている

  → FIBER ミニアプリにおけるサブコミュニケータは MPI_COMM_WORLD と等価であるものと1プロセスだけを含むサブコミュニケータのみ

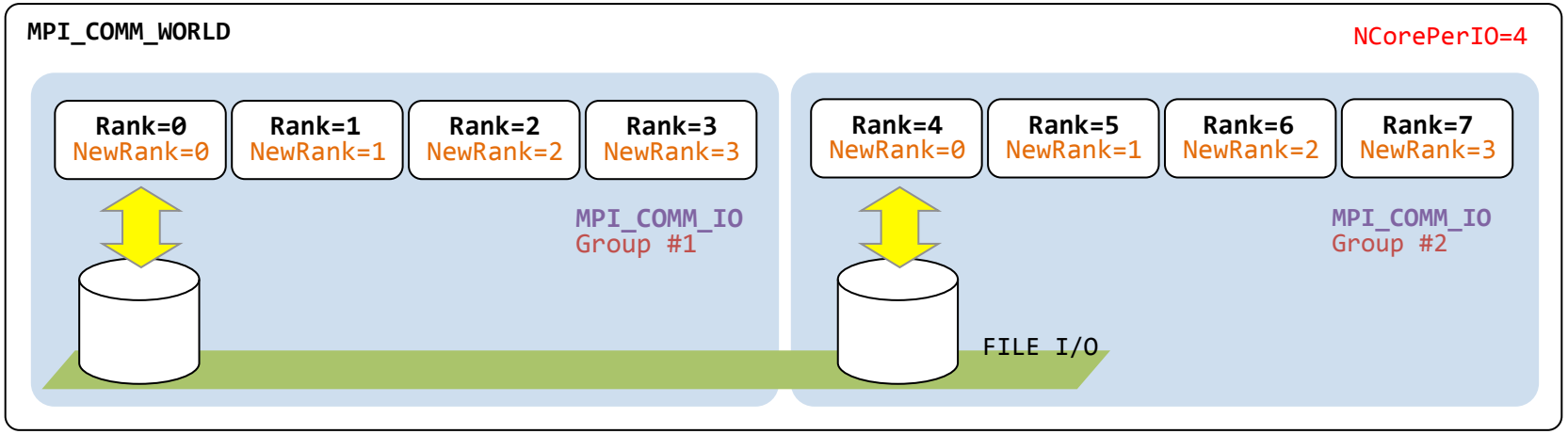
  今回の実装では
  - MPI_COMM_WORLD と等価
    MPI_COMM_WOLRD として扱う
  - 1プロセスだけを含むサブコミュニケータ
    集団通信はコメントアウトまたは代入文に変更

※ XMPでは、task 構文を使用することでサブコミュニケータへの対応が可能

# XMP task 構文を使用したサブコミュニケータへの対応例

## MPI_COMM_SPLIT 関数

```
MyColor = MyRank / NCorePerIO
MyKey   = MOD(MyRank, NCorePerIO)
CALL MPI_COMM_SPLIT(MPI_COMM_WORLD, MyColor, MyKey, MPI_COMM_IO, IErr)
```




## XMP task 構文

```
  integer,parameter :: iounit_size = 4
  integer,parameter :: n_iounit    = 2
  integer,parameter :: io_node_id  = 1
!$xmp nodes allnodes(iounit_size, n_iouni)
!$xmp nodes iounit(iounit_size) = allnodes(:,*)
!$xmp nodes ionodes(n_iounit)   = allnodes(io_node_id,:)
  ⋮
!$xmp task on iounit
  if (this_images() .eq. 1) write(ounit) buf
!$xmp end task
```

データマッピング

ワークマッピング

# XMP Coarray の実装状況と動作確認状況

◯： 並列実行評価
◎：スケーラビリティ評価（強スケール）

| FIBER MiniApp | ベース言語 | XMP 実装状況 | 動作確認状況 |
|---|---|---|---|
| CCS QCD | Fortran | ◯ | ◯ |
| FFVC-MINI | Fortran/C++ | 対象外※ | - |
| NICAM-DC-MINI | Fortran | ◯ | ◎ |
| mVMC-MINI | C | ◯ | 未 |
| NGS Analyzer-MINI | C | ◯ | ◎ |
| MODYLAS-MINI | Fortran | ◯ | ◯ |
| NTChem-MINI | Fortran | ◯ | ◯ |
| FFB-MINI | Fortran | ◯ | ◯ |

※ FFVC-MINI は、通信および領域分割を行うCPMlibライブラリ（C++）と演算部分（Fortran）から構成される。CPMlibライブラリの処理はXMPと等価である。また、XMPコンパイラが現時点でベース言語C++に対応していないため、XMPの実装は対象外とした。

# 評価の実施環境

- 評価機器の主な構成

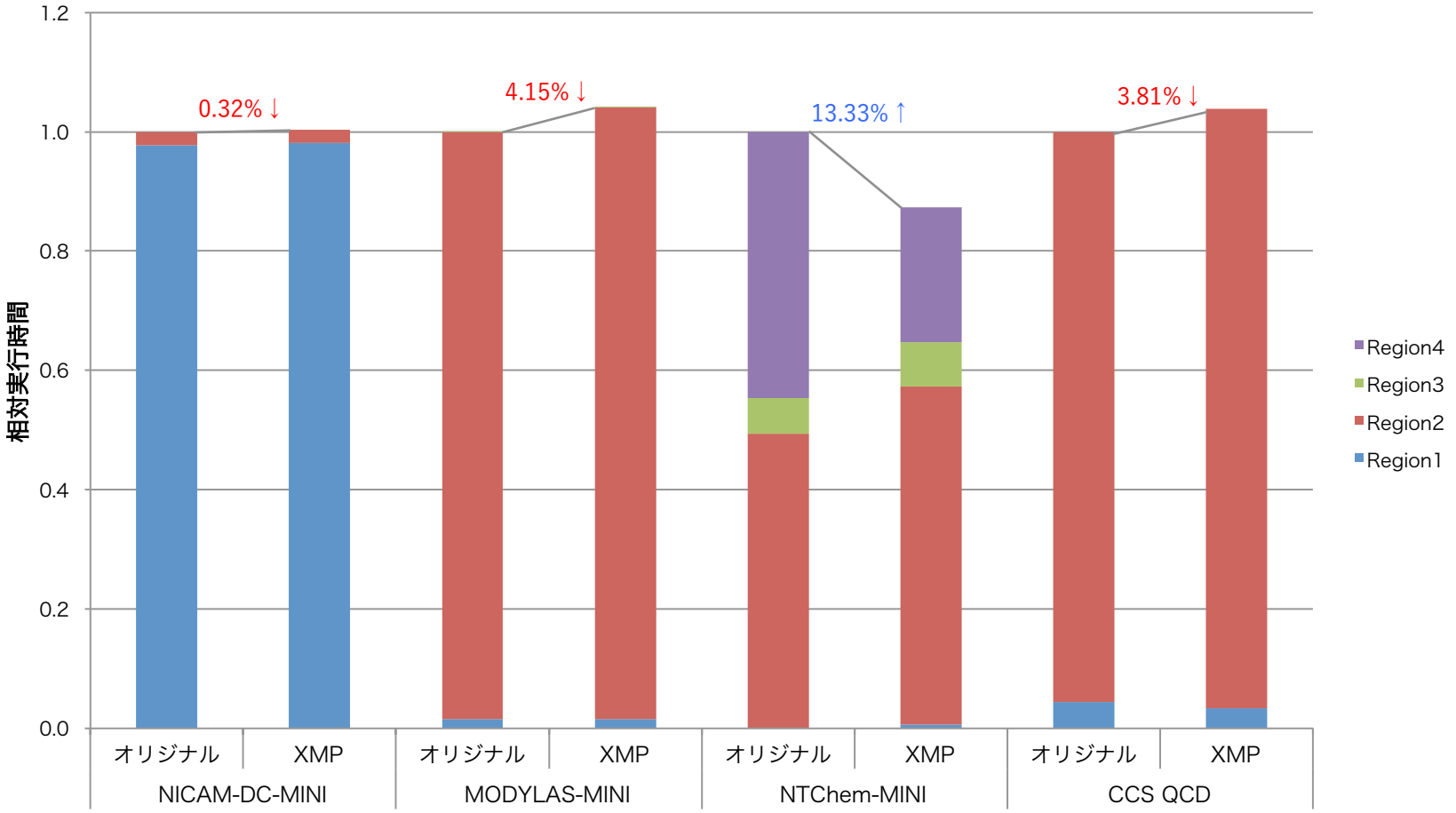
| | |
|---|---|
| CPU | Intel(R) Xeon(R) E7-4820 v3@1.90GHz (10コア) x 4ソケット |
| Memory | 256GB |
| Network | Intel QPI (6.4GT/s QPI) |
| OS | Linux Kernel 4.4.0-22-generic x86_64 |
| その他 | GCC 5.3.1, MPICH3.2, BLAS 3.6.0 |

- FIBERミニアプリとXMPコンパイラのバージョン

| FIBER MiniApp | Omni compiler |
|---|---|
| MODYLAS-MINI | 0.9.3-release |
| NGS Analyzer-MINI | |
| NICAM-DC-MINI | 0.9.3-20160224 |
| CCS QCD | 1.0.3-20160902 |
| NTChem-MINI | |

# 評価(1)

■ オリジナル版とXMP版の実行時間の比較




データ：gl06rl01z80
並列数：10

データ：wat222
並列数：8

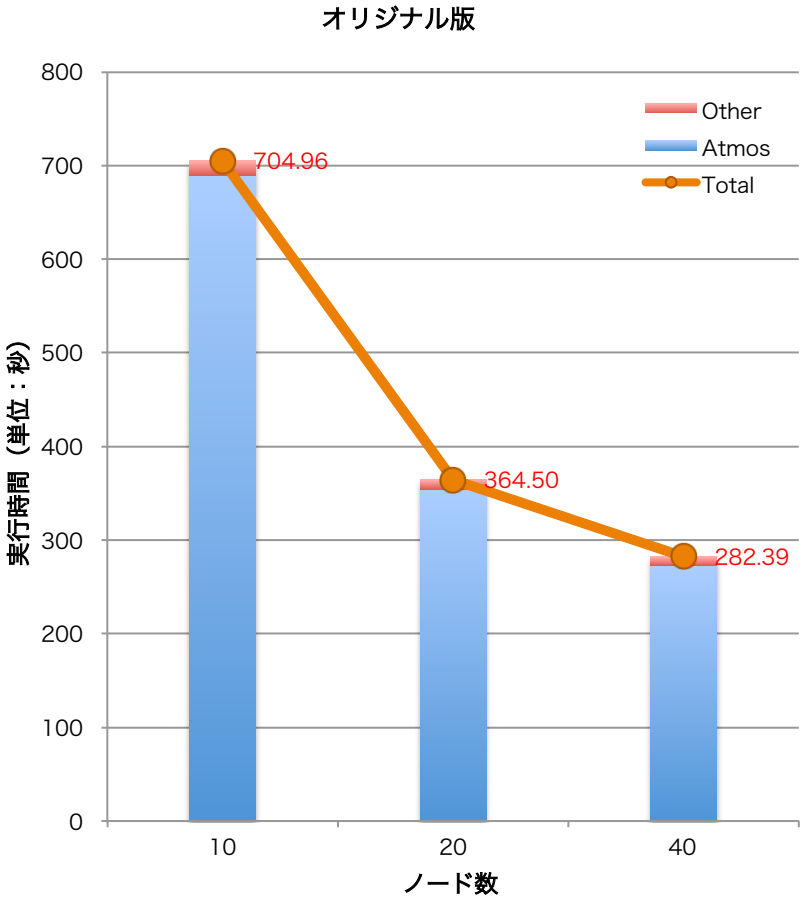
データ：c6h6
並列数：2

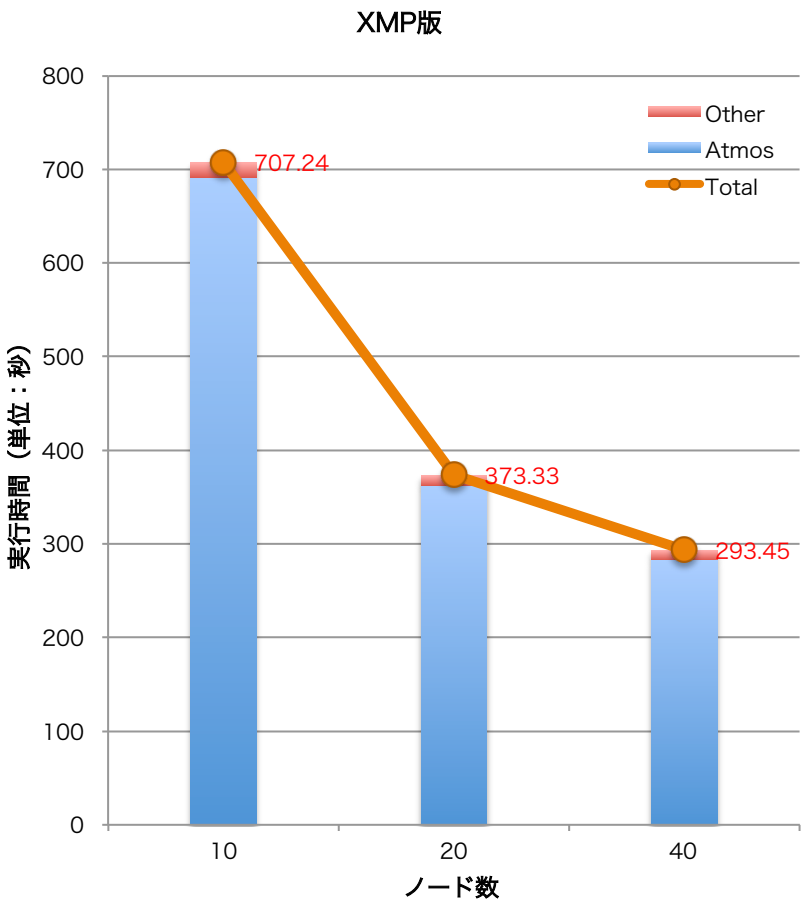
データ：Class2
並列数：8

# 評価(2)

- NICAM-DC-MINI
  データ：gl06rl01z80

Strong scale



# まとめ

- FIBER ミニアプリ集の5本を coarray を用いて実装し、XMP コンパイラを用いて評価を行った
- 実装について
  - MPI通信はcoarrayの片側通信と集団通信で置換えることが可能である
    ただし、片側通信Putの場合は送受信バッファのサイズに注意する必要がある
  - サブコミュニケータを使用しているMPIプログラムへの対応は、XMPの task 構文を使用することで対応が可能となる
- 評価について
  - オリジナル版とXMP版と実行時間の比較を行い、XMP版はオリジナル版とほぼ同等の結果（最大で4%程度の性能低下）となった