高性能並列プログラミング言語XcalableMPの紹介
Introduction of a PGAS parallel programming language XcalableMP

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
国立研究開発法人理化学研究所計算科学研究機構
村井　　均

現在の京を中核とするHPCIにおけるスーパーコンピュータやPCクラスタでは、CPUに多くの計算コアを搭載している。今後もこの計算コア数は増大する傾向にあり、ポスト「京」と呼ばれる次世代スーパーコンピュータにおいては、その数はこれまでと比較して膨大なものになることが予想される。このような大規模並列計算機システムで、プログラムの並列化にMPIを用いると、並列化は複雑化し、それに伴いプログラミングコストが高くなり、最終的にはプログラムの生産性や性能が低下するといった問題が発生する。

そこでこのような問題を解決するために、高性能で且つ扱いやすさを目指した新たな並列プログラミング言語として、PGAS言語XcalableMP（XMP）の研究・開発が進められている。本稿では、XMPの研究・開発の現状について紹介する。

1．はじめに

現在、計算機科学・計算科学分野やHPC（High Performance Computing）分野では、CPUとしてメニーコアまたはマルチコアと言った複数の計算コアを持つ複数のノードをネットワークにより結合したPCクラスタや大型並列計算機が広く利用されている。このような並列計算機でプログラムを効率よく利用するため、プログラムを並列化する手法としては、メニーコアCPUに対するスレッド並列、ノード間に対するプロセス並列があり、近年ではこれらを組合せたハイブリッド並列が多く利用されている。プログラミング言語としては、スレッド並列ではOpenMPが用いられ、プロセス並列ではMessage Passing Interface（MPI）が一般的に用いられる。OpenMPは共有メモリ型マシンで並列プログラミングを可能にするAPIで、指示文（ディレクティブ）を挿入するだけで並列化が行える。一方、MPIは分散メモリ型マシンで並列プログラミングを可能にするAPIで、ノード間のデータの送受信やブロードキャストといった集団通信が数多く用意されているが、その利用にはデータの分散や並列処理を明示しなければならず、OpenMPによる並列化と比べて手間がかかるのが実情である。XMPは、MPIのプログラミングコストの削減とOpenMPと同等の生産性の実現を目的とした新たな並列プログラミング言語であり、今後の大規模並列計算機システムにおける並列プログラム開発の生産性の向上に大きく寄与することが十分に期待される。本稿では、研究･開発が進められているこのXMPについて、その現状を紹介する。

2．XcalableMPについて

XcalableMP（XMP）は、分散メモリ環境を抽象化し、論理的に単一アドレス空間として扱うといったPGAS（Partitioned Global Address Space）機能に基づいて開発された

新しい並列プログラミング言語である。分散メモリ環境において、OpenMPのようにMPIとは異なるシンプルな言語構文と指示文による並列化を目指している。ベース言語はFortran言語とC言語となっており、2011年6月に発足したPCクラスタコンソーシアムXcalableMP規格部会が仕様の策定を行っている。

XMPの普及促進のため、言語仕様をはじめ、マニュアルや活動内容といった情報がXcalableMPのWebサイト（http://xcalablemp.org）を通じて公開されている。

プログラミングモデルには、グローバルビュー、ローカルビューと呼ばれる2つの並列化モデルを提供している。グローバルビューモデルではHigh Performance Fortran（HPF）のようなデータ／ワークマッピング指示文による並列化をサポートしており、元の逐次コードに指示文を挿入するだけの手軽さで並列化を実現している。また、ローカルビューモデルでは、Fortran2008から導入されたcoarray機能を取り入れ、MPIの場合と遜色ない並列化とプログラミングコストの削減を実現している。

3．実行モデルとメモリモデルについて

実行モデル

XMPの実行モデルは、MPIと同じSingle Program Multiple Data（SPMD）である。そのため、プログラムは各ノードで同じ処理が行われる。しかし、ノード間のデータの送受信は、MPIと異なり指示文により行われる。

メモリモデル

各ノードは、自身のローカルメモリ上のデータのみにアクセスできる。他のノード上のデータ（リモートデータ）にアクセスする場合は、XMP指示文またはcoarrayを用いて明示的にノード間の通信を記述する必要がある。

4．プログラミングモデルについて

グローバルビューモデル

グローバルビューモデルは、分散メモリ環境で、OpenMPのように既存のコードに指示文を挿入することで並列プログラミングを可能にするものである。

このグローバルビューモデルは、逐次プログラムのイメージから出発して、データを各ノードに分散し、それに応じた並列化を考えていくプログラミングスタイルに適している。

グローバルビューモデルにおける指示文の記述は、次のリスト1に示す形式で始まる。

リスト1．グローバルビューの記述

#pragma xmp	*C* 言語版
!$xmp	*Fortran* 言語版（自由形式）
C$xmp	*Fortran* 言語版（固定形式）

グローバルビューモデルにおける指示文には、宣言指示文と実行指示文が用意されており、実行指示文には単独指示文（stand-alone directive）と指示構文（directive construct）を構成する指示文がある。単独指示文は同期指示行などのその行単独で一つの機能を有し、指示構文は次のリスト2に示す形式となり、関連付けられた言語との組合せで一つの機能を有する。

リスト2．グローバルビューの指示構文

Fortran 言語の場合

!$xmp *directive–name clause …*
　statement
　…
!$xmp end *directive–name*
※ Loop 指示構文の場合は，end を省略できる

C 言語の場合

#pragma xmp *directive–name clause …*
　statement

指示文の種類はリスト 3 に示すように、データの分散に関するデータマッピング、実行処理を割り当てるワークマッピング、通信・同期の 3 種類があり、それぞれの処理に応じた指示文が提供されている。

リスト 3．指示文の種類

```
データマッピング
  ・node 指示文
  ・template 指示文
  ・distribute 指示文
  ・align 指示文
ワークマッピング
  ・loop 指示文
  ・task 指示文
通信・同期
  ・shadow 指示文
  ・reflect 指示文
  ・gmove 指示文
  ・bcast 指示文
  ・barrier 指示文
```

リスト 4．グローバルビューの指示文の記述例

```
program main
!$xmp nodes p(4,*)              ← データマッピング
!$xmp template t(n,n)
!$xmp distribute t(m,m) onto p
!$xmp align u(i,j) with t(i,j)
!$xmp align uu(i,j) with t(i,j)
!$xmp shadow uu(1:1,1:1)
 ...
!$xmp loop (i,j) on t(i,j)      ← ワークマッピング
  do j=2,n-1
    do i=2,n-1
      uu(i,j) = u(i,j)
    end do
  end do
 ...
!$xmp reflect (uu)              ← 同期・通信
 ...
!$xmp loop (i,j) on t(i,j) reduction(+:value)
  do j=2,n-1
    do i=2,n-1
      value = value + dabs(uu(i,j)-u(i,j))
    end do
  end do
 ...
end program main
```

グローバルビューモデルにおいて指示文による並列化を記述した例はリスト 4 のようになる。

ローカルビューモデル

ローカルビューモデルは、各ノードが行う処理を記述するプログラミングモデルである。MPIではデータの分散やその通信の記述をユーザが記述するが、XMPでも同様のプログラミングのスタイルを採用している。これはデータの局所性やノード間の通信を十分に把握した上でのプログラミングとなるため、その自由度は高くなり、高い性能を引き出すことを目的としているためである。XMPではFortran 2008のcoarray機能をベースとした言語拡張を提供しており、変数の宣言や代入文を拡張とした構文により、ノード間の通信をMPIと比べて簡単に記述することを可能としている。XMP/Fortranにおいてはは Fortran 2008との互換性が保たれている。XMP/Cにおいてはcoarray指示文と言語拡張で実装がされている。ノード間のデータの送受信は、片側通信と集団通信を使用することができる。Fortran言語とC言語におけるcoarrayの配列宣言と片側通信の記述例をリスト 5 に示す。

リスト 5．グローバルビューの指示文の記述例

Fortran 言語の場合

```
integer a(N)[*]
a(1:N)[2] = a(1:N)  ! Put 通信
a(1:N) = a(1:N)[2]  ! Get 通信
```

C 言語の場合

```
int a[N]:[*]
a[0:N-1]:[2] = a[0:N-1] ! Put 通信
a[0:N-1] = a[0:N-1]:[2] ! Get 通信
```

一方、集団通信には分散データの総和・最大値・最小値を求めるco_sum、co_max、co_minといった組込み関数があり、また

sync all文といった同期機能が提供されている。また、あるデータをブロードキャストするco_broadcast関数も提供されている。この集団通信co_broadcast関数のFortran言語における記述例をリスト 6 に示す。

リスト6. 集団通信co_broadcast関数の記述例

```
integer me, buf
me = this_image()
if (me .eq. 1) buf = me

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

XMPのcoarray機能に触れる前に、ベースとなるCoarray Fortran言語に触れてみたいという人は、Fortran 2008規格をサポートしているベンダーが提供するコンパイラを使用するか、またはオープンソースで提供されているOpenCoarrays（http://opencoarrays.org）を参照して欲しい。

プログラミングモデルの切替え

プログラミングモデルはユーザが使用するアプリケーションの性質によって使い分けることが望ましいが、他の言語モデルにはないXMPの特徴として、同じ変数に対してグローバルビューモデルとローカルビューモデルの切替えを行うことができることがあげられる。

ローカルビューの変数は、宣言された形状のデータが全てのノードに割り付けられるため、データを参照するには、変数名と各次元の添字に加えて、ノード番号を指定しなければならない（ノード番号を省略した場合には、自ノードのデータであると見なされる）。一方、グローバルビューの変数は、データは仮想的な共有空間に置かれているように見えるため、原則として配列要素の特定にノード番号を必要としない。ただし、XMPでは他ノードのデータの参照は明示的な記述を必要とするため、グローバルビューであっても、ユーザがそのデータが自ノードに配置されているかを把握していなければならない。

XMPでは、1 つのデータに対してグローバルビューとローカルビューの 2 つの名前を付け、使い分けるための指示文も提供されている[*1]。詳細は仕様書を参照されたい。

5．Omni XcalableMPについて

Omni XcalableMP（XMP コンパイラ）は、筑波大学HPCS研究室と理化学研究所計算科学研究機構（理研AICS）プログラミング環境研究チームにより、オープンソースプロジェクトとして開発されているXMPコンパイラのリファレンス実装である。

Omni XcalableMPは、XcalableMPをはじめ、XcalableACC、OpenACCといった指示文を含むコードを対象としたコンパイラとなっており、Omni Compiler ProjectのWebサイト（http：//omni-compiler.org）よりダウンロードができる。

残念ながら現時点ではC++言語には対応していないが、近年ではC++言語による科学技術計算の開発コードも多く見られるため、今後対応する予定である。

XMPコンパイラは、次の条件を満たす一般的なLinuxクラスタや京コンピュータや地球シミュレータの他、MPIが動作する任意のプラットフォームに対応している。各プラットフォームに対応したバイナリは用意されておらず、ユーザがソースファイルからビルド・インストールをする必要がある。

＊1　ただし、現時点のOmni XcalableMPでは、グローバルデータの別名をcoarrayとして宣言する機能は利用できない。

リスト７．インストールに必要な環境

```
・Yacc
・Lex
・C Compiler (supports C99)
・Frotran Compiler (supports Fortran 90)
・C++ Compiler
・Java Compiler
・MPI Implementation (supports MPI−2 or over)
・libxml2
・make
```

インストールは、Linux利用者であればリスト８の手順に従い比較的簡単に行うことができる。

リスト８．インストール手順

```
$ tar jxf omnicompiler−1.0.3.tar.bz2
$ cd omnicompiler−1.0.3
$ ./configure
$ make
$ make install
```

インストールの際の注意点は、ローカルビューモデルの片側通信が行える環境にあるかである。

これはconfigureコマンド実行後のConfiguration Summaryの出力により確認ができる。Options項目のOnesidedがYesになっていれば良く、MPIライブラリ環境のバージョンが３以上であれば特に問題はない。また、オプションの指定により、数値計算ライブラリBLASを使用する場合は、configureコマンドの引数にオプション-with-libblasに静的ライブラリlibblas.aを指定することにより、その機能が有効となる。基本的にはここまでに述べた手順にてインストールは行えるはずであるが，不明な点がある場合は詳しい手順がガイドブックとして公開されている（https://omni-compiler.gitbooks.io/guidebook/content/ja/）ので参照して欲しい。

リスト９．Configuration Summaryの出力例

```
Options:
  BLAS Library : no
  OpenACC : no
  Onesided : yes
    Communication Library : MPI3
    Coarray : yes
    Post/Wait : yes
    Lock/Unlock : no
```

６．XMPの実装と性能評価について

分散メモリ環境を対象とした新しいプログラミングモデルであるXMPであるが、近年では更なる研究開発と普及促進のために、既存のプログラムへの実装を行い、その利便性や性能向上についての検証が盛んに行われている。

グローバルビューモデル

グローバルビューモデルの実装は、表１に示すアプリケーションに行われている。指示文による並列化が行われ、オリジナルのMPI版と実行時間を比較しても遜色ないことが過去に開催されたワークショップの資料として公開されている［１］。

表１．グローバルビューモデルを実装したアプリケーション

アプリ名	概要
IMPACT-３D	３次元Cartesian格子による流体コード（核融合研）
RTMコード	リバースタイムマイグレーション法による地中探査イメージング（仏Total社）
SCALE	次世代気象気候科学における基盤ライブラリ（理研AICS）
GTC-P	核融合シミュレーションコード（プリンストン大）

ローカルビューモデル

ローカルビューモデルの実装は表２に示すFIBER MiniAppの幾つかのアプリケーションに対して行われ、検証が行われている［４］。

表２．ローカルビューモデルを実装したアプリケーション

アプリ名	概要
CCS QCD	高エネルギー物理学で用いられる格子量子色力学プログラム
FFVC-MINI	三次元非定常非圧縮性熱流体シミュレーションプログラム（熱流体解析プログラム）
NICAM-DC-MINI	全球雲解像モデルNICAMの力学コアを計算するプログラム
mVMC-MINI	多変数変分モンテカルロ法
NGS Analyzer-MINI	ヒト個人間の遺伝的差異やがんゲノムの突然変異を固定するプログラム
MODYLAS-MINI	高並列汎用分子動力学シミュレーション
NTChem-MINI	分子科学計算ソフトウェア
FFB-MINI	非圧縮流体の非定常流動を高精度に予測可能なLarge Eddy Simulation（LES）に基づいた汎用流体解析コード

ここで、FIBER MiniAppとは、アプリケーション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から重要な特徴を抽出して作られたミニアプリである。FIBERのWebサイト（http://fiber-miniapp.github.io）にて公開されている。また、このFIBER MiniAppは国内のベンチマークプログラムとして性能評価を行う上で大変便利なアプリとなっている。

ローカルビューモデルの基本的な実装方針は、MPIの同期または非同期通信をcoarrayのPutベースの片側通信に置換え、MPIの集団通信についてはcoarrayの集団通信（co_broadcastなど）への置換えとしている。

実装のイメージは図１に示す通りである。

リスト10．XMPの実装の基本方針

- MPI_Send/Isend 関数 → coarray 代入文 (Put)
- MPI_Recv/Irecv 関数 → 削除
- MPI_Wait/Waitall → sync all
- 集団通信関数 MPI_Bcast など → co_broadcast など

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
```

図１．XMP 実装のイメージ

ここで、実際にMPIの通信部分をcoarrayに置換えた例を２つあげる。１つは同期通信MPI_Sendrecv関数の置換え（図２）であり、もう１つは集団通信MPI_Gatherv関数の置換え（図３）である。

まず、MPI_Sendrecv関数の置換えについてであるが、その前にcoarrayの片側通信ではMPIの場合とは異なり、通信相手のバッファに関する情報を明確に把握する必要がある。

MPI関数の呼び出しによる通信の指定は、送受信バッファの先頭アドレスと送受信データの個数、データ型、通信相手のランク番号などといった情報が必要となるが、coarray記法ではデータを送受信するバッファ代入文の指定となり、送受信バッファの他に通信相手のノード番号が必要となる。特に送受信バッファに配列を使用する場合は、データ転送を行う配列の領域範囲（上限・下限）の指定が必要となる。また、片側通信Putでは、代入文の左辺に通信相手のノード番号を［］で指定する。

coarray記法では、ノード番号の開始番号が１からと定められているため、ここではMPIの送信先ランク番号destに１を足している。

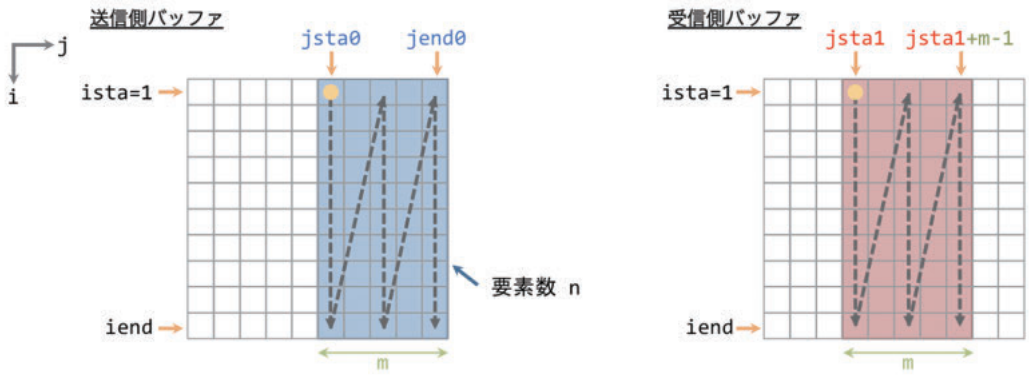
以上のことを踏まえるとMPI_Sendrecv関数のcoarrayへの置換えは図２に示す通りとなる。coarray記法はMPIの記述のように、データ型や送受信タグ、通信の状況といった引数の記述が不要となるため、通信の記述が簡素化される。

次に、集団通信MPI_Gatherv関数の置換えであるが、この通信処理は各ノードでサイズの異なるデータをあるノードの受信バッファに格納することである。先ほどのMPI_Sendrecv関数の置換えの例と同様の記述で、送受信バッファの領域範囲を正確に指定することでデータの送受信が実現される。

MPI_Sendrecv 関数

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

allocate(buf(nx,ny,nz))
n = (iend-ista+1) * (jend0-jsta0+1)
call mpi_sendrecv(buf(ista,jsta0,k), n, MPI_INTEGER, dest,   sendtag, &
                  buf(ista,jsta1,k), n, MPI_INTEGER, source, recvtag, &
                  MPI_COMM_WORLD, status, ierr)
```



coarray 記法による片側通信(Put)の記述

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

allocate(buf(nx,ny,nz)[*])
m = jend0-jsta0+1
buf(ista:iend,jsta1:jsta1+m-1,k)[dest+1] = buf(ista:iend,jsta0:jend0,k)
sync all
```

図２．MPI Sendrecv関数のcoarrayへの置換え例

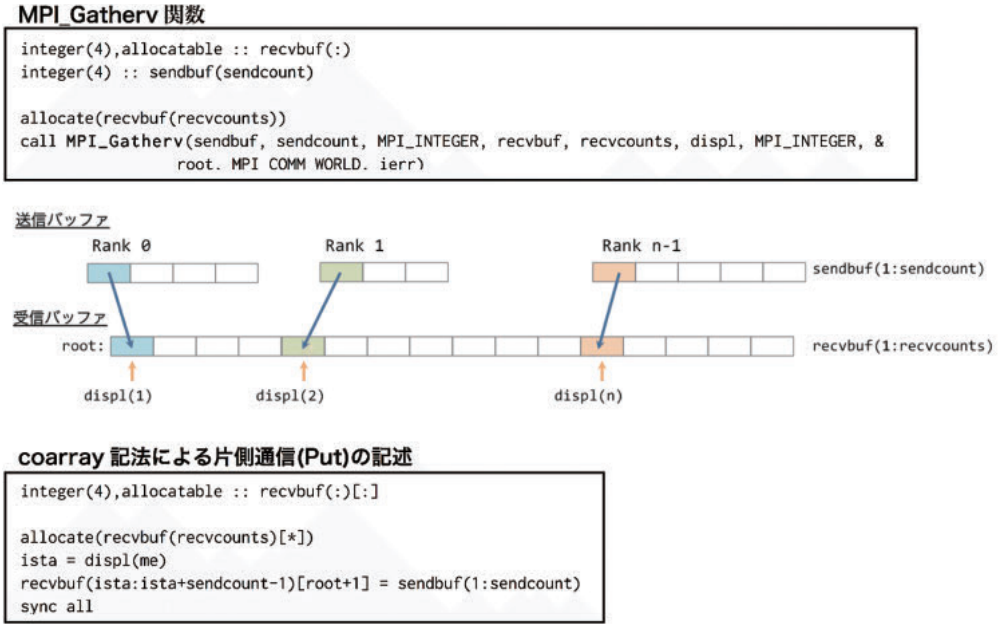


図 3．MPI Gatherv関数のcoarrayへの置換え例

性能評価

ローカルビューの実装を行ったアプリケーションの評価について述べる。

評価の対象となるアプリケーションは、NICAM-DC-MINI、MODYLAS-MINI、NTChem-MINI、CCS QCDであり、評価はオリジナルのMPIで記述されたアプリケーション（MPI版）とXMPを実装したアプリケーション（XMP版）とでそれぞれの実行時間を比較した。

評価を実施した機器の構成とその際に使用したXMP コンパイラのバージョンは、次の表 3、表 4 に示す通りである。

表 3．評価機器の主な構成

CPU	Intel（R） Xeon（R） E7-4820 v3 @1. 90GHz（10 コア）x4 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

表 4．評価に使用したXMPコンパイラのバージョン

アプリ名	XMP コンパイラのバージョン
NICAM-DC-MINI	0.9.3-20160224
MODYLAS-MINI	0.9.3-release
NTChem-MINI	1.0.3-20160902
CCS QCD	

表 4 のXMPコンパイラのバージョンにおいては、releaseと記述があるものはコンパイラの安定版を示しており、それ以外は改良が進められたNightly build版を示している。

評価対象のアプリケーションにおけるMPI版とXMP版のそれぞれの実行時間を相対時間により比較すると、相対時間比は図 4 の通りとなった（凡例は計測区分を示しているが、本稿ではプログラム全体の時間のみに注目する）。XMP版の実行時間はMPI版と比べて最大で 4 % の性能低下ではあるが、ほぼ同等の結果となった。

また、NICAM-DC-MINIについては、ストロング・スケールの計測が実施でき、その結果は図 5 に示す通りとなった。図 5 より、XMP版のスケーラビリティはMPI版と比べて遜色ない結果となった。

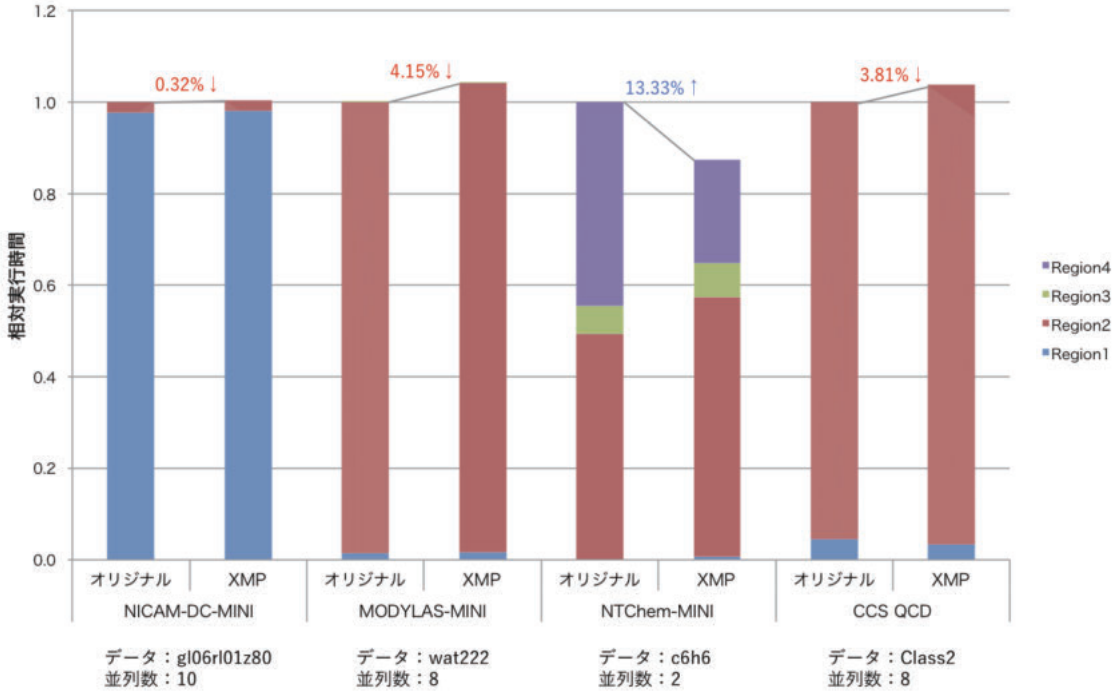


図４．オリジナル版とXMP版の相対実行時間

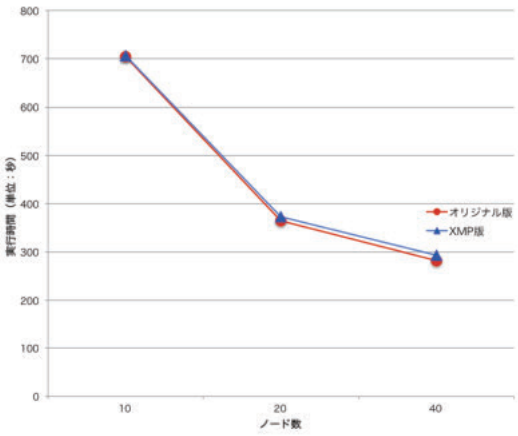


図５．NICAM-DC-MINI のスケーラビリティ

７．おわりに

並列計算機でプログラムを効率よく開発するためのプログラミング言語として、研究・開発が進められているPGAS言語のXcalableMPの現状を紹介した。現在では分散メモリ環境におけるプログラミング言語としてMPIが主に使用されていることは疑いの余地もないことである。しかし、ポスト「京」と呼ばれる次世代スーパーコンピュータにおいて、これまでに経験したことのないコア数が実装されるとなると、XcalableMPによる「グローバルビューモデルの指示文」や「ローカルビューモデルのcoarry記法」による並列化は、並列プログラム開発の生産性と性能向上に大きく貢献することが期待される。

参考文献

［１］ http://xcalablemp.org
［２］ http://omni-compiler.org
［３］ https://omni-compiler.gitbooks.io/guidebook/content/ja/
［４］ http://fiber-miniapp.github.io
［５］ 並列プログラミング言語XcalableMPにおけるステンシル通信の効率的な実装，情報処理学会研究報告 Vol2013-HPC-140 No.8, 2013/ 7 /31
［６］ 分散メモリ向け並列言語XcalableMPコンパイラの実装と性能評価，情報処理学会論文誌コンピューティングシステムVol.3 No.3 153-164（Sep. 2010）