Vol.2009-HPC-121 No.6
2009/8/4

分散メモリ向け並列言語 XcalableMP コンパイラの試作と評価

李　珍泌[†1]　朴　泰祐[†1,†2]　佐藤三久[†1,†2]

PC クラスタにおける標準的なプログラミングモデルである MPI は高いプログラミングコストが問題として指摘されている．分散メモリ型並列計算機におけるプログラミングをより簡単にするため，C と Fortran をベース言語として，指示文により拡張した並列プログラミングモデル XcalableMP が提案されている．XcalableMP では OpenMP-like な指示文を提供し，典型的なデータ並列化手法が有効なアプリケーションに対して逐次コードからのシームレスな並列化を可能にする．また，CAF-like な言語拡張を取り入れることにより，ノード内のメモリイメージとノード間通信を意識した効率的な並列化の記述が可能である．また，パフォーマンスチューニングのため，OpenMP や MPI を XcalableMP と併用することもできる．本稿では，XcalableMP について述べ，そのコンパイラの実装と予備性能評価について述べる．NAS Parallel Benchmakrs の CG を用いた性能評価の結果，XcalableMP が少ないプログラミングコストで MPI 版に近い性能を達成することが確認できた．

XcalableMP: A Prallel Programming Model for Distributed Memory System

Jinpil Lee ,[†1] Taisuke Boku [†1,†2]
and Mitsuhisa Sato [†1,†2]

Although MPI is a de-facto standard for parallel programming on distributed memory systems, writing MPI programs is often a time-consuming and complicated process. XcalableMP is a language extension of C and Frotran for parallel programming on distributed memory systems that helps users to reduce those programming efforts. XcalableMP provides two programming models. The first one is the global view model, which supports typical parallelization based on the data and task parallel paradigm, and enables parallelizing the original sequential code using minimal modification with simple, OpenMP-like directives. The other one is the local view model, which allows to use CAF-like expression to describe internode communications. Users can even use MPI and OpenMP explicitly in our language to optimize the performance explicitly. In this paper, we introduce XcalableMP, the implementation of the compiler, and the performance evaluation result by global view parallelization in XcalableMP. The experiemntal result shows that XcalableMP achieves close performance to MPI version on CG of the Nas Parallel Benchmarks given a small modification to the original sequential code.

1. はじめに

PC クラスタは比較的安価で容易に大規模並列システムを構築できるため，高性能コンピューティングの分野で広く使われている．そのプログラミングモデルとしてもっとも普及しているのが Message Passing Interface (MPI) である．MPI は通信ライブラリとして提供される．データの分散やプロセス間通信といった並列化作業はユーザが明示的に記述しなければならない．その結果，プログラミングコストが膨大になり PC クラスタの利用や普及を妨げる要因の一つとなっている．

近年その普及が目覚しいマルチコアプロセッサにおいては OpenMP による指示文ベースの並列プログラミングが可能である．逐次のソースコードに対し，並列化に関する情報を記述した指示文を挿入することで，コンパイラによる並列化が行われる．そのため，OpenMP は少ないプログラミングコストで逐次コードからのシームレスな並列化を可能にする．

分散メモリでの並列プログラミングを容易するために，OpenMP のような指示文による並列プログラミングモデルとして，XcalableMP [*1] が提案されている．XcalableMP は C と Fortran に OpenMP-like な指示文や明示的な通信のための言語拡張を行うもので，データの分散やプロセス間通信を記述するための指示文をユーザに提供する．XcalableMP はコンパイラやランタイムによる自動的な並列化を提供しない．並列化のための記述は全てユーザによって明示的に記述される．コンパイラによるコードの変換はユーザに対して明確であるため，チューニングが容易である．パフォーマンスチューニングのため，XcalableMP はプロセス間通信を記述できる言語拡張を提供する．また，XcalableMP のソースコードの

†1 筑波大学 大学院 システム情報工学研究科
Graduate School of Systems and Information Engineering, University of Tsukuba
†2 筑波大学 計算科学研究センター
Center for Computational Sciences, University of Tsukuba

⋆1 XcalableMP[1] の言語仕様は，文部科学省「e-サイエンス実現のためのシステム統合・連携ソフトウェアの研究開発」プロジェクトの次世代並列プログラミング言語検討委員会にて検討されている．本稿で述べられる言語仕様（指示文の名前や記号，記法など）は今後変更される可能性がある．

中に OpenMP や MPI が利用できるように設計されている．これらの要素を利用することで分散メモリ型並列計算機において効率のいい並列化を少ないプログラミングコストで行うことができる．

続く第 2 節では関連研究について述べ，既存の言語モデルの問題点を明確にする．第 3 節では XcalableMP の概要と設計におけるコンセプトを述べる．第 4 節では XcalableMP による並列プログラミングモデルについて説明を行う．第 5 節では試作したコンパイラの構成について述べ，第 6 節でベンチマークによる性能評価を行う．第 7 節で現状と今後の課題について述べ，本稿をまとめる．

2. 関 連 研 究

分散メモリ型並列計算機のための簡単なプログラミングのために様々な言語やライブラリが提案されて来た．その中で代表的なものとして，Unified Parallel C (UPC)[3], Co-Array Fortran (CAF)[5] のような Partitioned Global Address Space (PGAS) 言語や，High Performance Fortran (HPF)[9] が挙げられる．

PGAS 言語では共有メモリであるグローバルメモリが複数のスレッドで共有される．グローバルメモリの各領域はどれかのスレッドに割り当てられており，データの局所性を意識したプログラミングを行うことで高い性能を得ることができる．UPC は共有メモリに近いモデルをユーザに提供し，他のスレッドに割り当てられたメモリのアクセスが容易である．これはスレッド間通信がメモリアクセスの下で隠蔽されているからで，メモリアクセスが複雑なアプリケーションでは効率的な通信を行うことができず，性能が低下する恐れがある．CAF はイメージ（CAF におけるスレッド）間通信のため，Co-Array を用いる．Co-Array は通常の配列の次元をイメージ番号で拡張したものであり，イメージ番号を拡張した次元の参照 index として使うことでイメージ間通信を実現する．CAF のプログラミングモデルは MPI の片側通信を簡略に記述できるようにしたものである．ユーザの努力次第で高い性能を得ることができるが，MPI とほぼ同等のプログラミングコストが必要である．

HPF は指示文によるプログラミングモデルを提供する．プロセス間通信はコンパイラによって自動的に挿入される．UPC と同様にアプリケーションのメモリアクセスパターンが複雑な場合，最適な通信を生成することができず，性能が低下する．UPC も HPF も性能をチューニングするような手段が提供されないため，パフォーマンスを改善することは困難である．

既存の並列言語が持つ問題を解決するため，著者らは過去に指示文による並列言語，OpenMPD[12] を提案した．OpenMPD は OpenMP や HPF と同様，指示文による並列化の記述を行う．しかし，コンパイラによる自動的な通信の生成を行わず，ユーザによる明示的な通信の記述が必要である．OpenMPD はブロードキャストやリダクションなどの，データ並列化で行われる典型的な通信を指定する指示文をユーザに提供する．ループ文の並列化や通信が明示的に記述されるため，生成される並列コードのイメージが明確であり，MPI と併用することでパフォーマンスチューニングが可能である．

VPP/XPF Fortran[13] も OpenMPD と同様のアプローチを取っている並列言語である．並列化の記述は OpenMP-like な指示文で行われるが，ノード間通信は指示文の指定された場所でしか発生しない．

XcalableMP は OpenMPD や VPP/XPF Fortran と同様のコンセプトを持つ言語モデルである．他にも，これまでに挙げたきた並列言語モデルの影響を受けており，それらの長所を取り入れた言語モデルである．次節で XcalableMP の概要を述べ，これらの既存言語（モデル）との差異を明らかにしていく．

3. XcalableMP の概要

XcalableMP は PC クラスタのような分散メモリ型並列計算機を対象とした並列言語である．逐次の手続き型言語である C と Fortran をベースにして並列化のための独自の言語拡張を行っている．言語拡張の大半は OpenMP-like な指示文であり，構文を含む言語仕様の変更は最小限に留める．XcalableMP は少ないプログラミングコストと効率的な並列化を両立させるため, 一つの言語の中でグローバルビューとローカルビューという二つのプログラミングモデルを提供する．

XcalableMP の全体像を図 1 に示す．ユーザはアプリケーションの並列化のためにグローバルビューとローカルビューのプログラミングモデルを利用することができる．その他にも，XcalableMP が提供するインターフェイスを用いることで OpenMP や MPI と XcalableMP のハイブリッド並列プログラミングを行う．並列化に必要な情報（実行ノードの数，ノード番号など）は XcalableMP の組み込み関数を利用することで取得することができる．

3.1 グローバルビューモデル

グローバルビューモデルは OpenMP-like な指示文による並列プログラミングを可能にするものである．データの分散やループ文のワークシェアリング，バリアやリダクションなど集団通信のような典型的な並列化手法を指示文で記述することができる．指示文はコメントの一種であり，XcalableMP のコンパイラ以外では無視されるので逐次実行を保持しながら

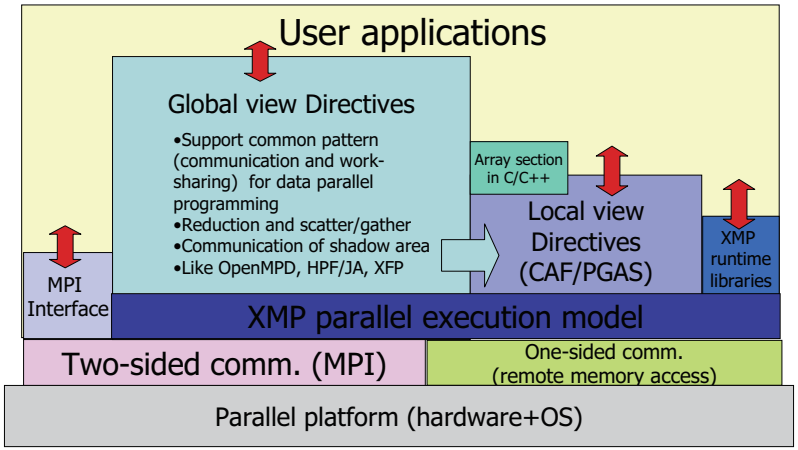

図 1 XcalableMP の概要

のインクリメンタルな並列化が可能である．XcalableMP の指示文は HPF をベースにしたものである．template，shadow などの概念は HPF から由来したものであるが，その意味や動作は XcalableMP のモデルに合わせて変更が加えられている．

XcalableMP はコンパイラによる自動並列化や自動的な通信の挿入を前提としない．コードの変換は指示文によって指定されたものに限られる．従って，プロセス間通信は通信のための指示文や後述するローカルビューモデルでの言語拡張を用いて明示的に記述しなければならない．指示文の記述による通信は変数の同期やバリア通信など，典型的で効率よく実装できるものに限られる．従って，グローバルビューモデルによるアプリケーションの並列化はその通信パターンが領域分割法等で現れる典型的かつ定型的なものに限られる．より複雑な通信パターンを持つアプリケーションの並列化やパフォーマンスチューニングのためにはより明示的な通信の記述手法が求められる．XcalableMP ではローカルビューのプログラミングモデルを提供することでそれを実現している．

3.2 ローカルビューモデル

従来の MPI ではデータの分散や通信の記述をユーザが行う．このようなモデルではデータの局所性やプロセス間通信を意識した並列化を行うことで高い性能を引き出すことができる．XcalableMP ではこのようなプログラミングモデルをローカルビューモデルとし，言語拡張による通信の記述手法を提供する．ローカルビューモデルを実現するため，XcalableMP では CAF をベースにした言語拡張を行っている．

3.3 ハイブリッド並列プログラミング

マルチコアのプロセッサを持つクラスタシステムでは OpenMP と XcalableMP（または MPI）のハイブリッドプログラミングが性能を出すために有効な場合がある．また，並列アルゴリズムの記述のために MPI が提供する機能を利用したい時があるかもしれない．XcalableMP は OpenMP や MPI と併用できるように設計されている．コードの変換はユーザが指定した場合のみに行われ，コンパイラの動作はユーザに対して明確であるため，他の並列プログラミングモデルとの併用が容易である．

4. プログラミングモデル

本節では XcalableMP のプログラミングモデルについて述べる．XcalableMP の二つのプログラミングモデル（グローバルビューモデルとローカルビューモデル）と，両方で共有される基本概念に関する説明を行う．

4.1 実行モデル

XcalableMP は分散メモリをターゲットとした並列言語である．実行単位であるプロセスを XcalableMP ではノードと定義する．XcalableMP の実行モデルは MPI と同様，Single Program Multiple Data (SPMD) である．つまり，通常実行時には各ノードで同じ処理が重複実行される．ソースコードで宣言されたデータは各ノードで重複して宣言される．通常のメモリアクセスはローカルメモリのデータに対する参照である．他のノードで宣言されたデータにアクセスするためには，XcalableMP が提供する通信記法を用いて明示的なノード間通信を行わなければならない．各ノードでのスレッドは実行開始時には一つである．XcalableMP は分散メモリを対象としているため，スレッド並列化のための機構を備えていない．しかし，ノード内並列化のために OpenMP やスレッドライブラリを XcalableMP と併用することは可能である．

4.2 グローバルビューモデル

XcalableMP のグローバルビューモデルは指示文の挿入による並列化を行うものである．図 2 にソースコードの例を示す．C 言語で記述された逐次コードに並列化を指定する指示文を追加している．

4.2.1 template を用いた index 空間の分割

nodes 指示文はプログラムを実行するノード集合の名前と形状（トポロジー）を宣言する．図 2 の例では四つのノードで構成される一次元のノード集合に p という名前をつけている．template とは index の集合を表現する仮想的な配列である．ここでの index とはデータ配

Vol.2009-HPC-121 No.6
2009/8/4

```
int array[YMAX][XMAX];

#pragma xmp nodes p(4)
#pragma xmp template t(0:YMAX-1)
#pragma xmp distribute t(BLOCK) on p
#pragma xmp align array[i][*] with t(i)

main() {
  int i,j,res = 0;
#pragma xmp loop on t(i)
  for(i = 0; i < YMAX; i++) {
    for(j = 0; j < XMAX; j++) {
      array[i][j] = func(i,j);
      res += array[i][j];
    }
  }
#pragma xmp reduction (+:res)
}
```

図 2 グローバルビューのプログラム例

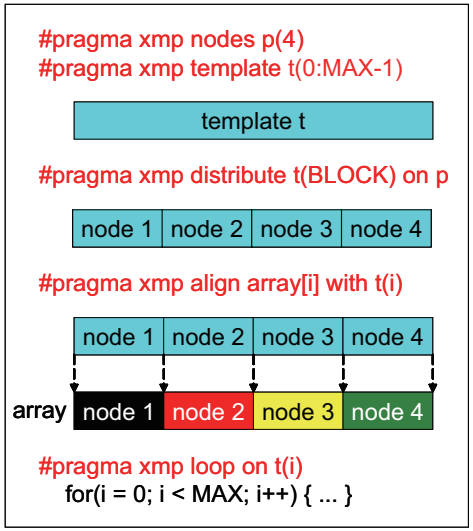

図 3 template による並列化

列やループ文の反復の添え字の総称である．template の属性として名前，各次元の上限値と下限値が与えられえる．

XcalableMP では配列の分散やループ文のワークシェアリングは template を用いて行われる．仮想的な index 空間である template を distribute 指示文で各ノード上に割り当て，align 指示文と loop 指示文を用いることで配列とループ文の並列化を行う．template による並列化のイメージを図 3 に示す．仮想的な index 空間である template を用いて配列の分散やループ文の並列処理を記述している．

distribute 指示文は template を各ノード上に分散配置することを宣言する．その属性として各次元の分割方式を指定することができる．ブロック分割の BLOCK とサイクリック分割の CYCLIC が利用できる．align 指示文は配列の分割の仕方を指定するものである．template 空間と配列の index の整合関係を宣言することで配列の割り当て方を指定する．図 2 の例ではブロック分割された一次元の template 空間と配列の Y 次元が整合されると宣言される．その結果，配列 array は Y 次元でブロック分割される．align によって分割された配列は各ノードで割り当てられた分だけが宣言される．コンパイラによる index 変換が行われるため，分散される配列の実体とその index を意識せず，配列の参照を行うことができる．

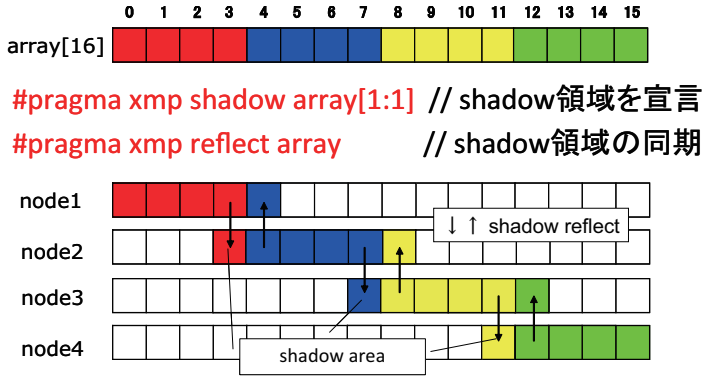

図 4 shadow 領域の宣言と同期

4.2.2 ループ文とタスクの並列実行

ループ文の並列化には loop 指示文を用いる．loop 指示文の直後に記述されたループ文は各ノードで処理を分担するように並列化される．ループ文の反復をどう分割するかは template を指定することで決定する．図 2 の例ではループ文が配列 array を処理している．array は align 指示文と template t によって分割されているため，ループ文の並列化も配列の分散と整合しなければならない．従って loop 指示文で template t を指定し，index の分割の仕方を指定している．distribute も loop も直接配列やループ文を並列化することはできない．これらの並列化は template を経由して行わなければならない．

loop 指示文は配列を処理するループ文を並列化するためのもので，データ並列化のための典型的な手法である．XcalableMP ではデータ並列化の他に task 指示文によるタスク並列化をサポートする．task 指示文は直後に記述されたブロック文を指示文の属性として指定したノード集合で実行するようにプログラムを変更する．独立して実行できる処理を異なるノード集合で実行させることでタスク並列化を行うことができる．

4.2.3 配列の重複宣言と通信の記述

XcalableMP ではループ文の並列化と配列の分散の整合はユーザ責任である．XcalableMP におけるメモリアクセスは常にローカルメモリに対するものであるため，他のノードに割り当てられた領域にアクセスした場合はエラーとなる．割り当てられていない（他のノードに割り当てられた）領域にアクセスする場合は領域を参照側と被参照側で重複宣言し，指示文により明示的な同期を行う必要がある．

配列の各要素に対する計算が隣の要素に依存するようなパターンは多くのアプリケーショ

ンで見られる（偏微分方程式の陽解法での領域分割等）．そのような場合は各ノードで割り当てられるデータ領域の境界要素を重複宣言し，必要に応じて通信を行わなければならない．XcalableMP はこのような並列化を指示文で記述するための指示文 shadow と reflect を提供する．指示文の記述と動作を図 4 に示す．shadow 指示文の属性として重複宣言する領域の次元と大きさが指定される．shadow 領域の下端と上端のサイズを triplet 形式（例，[1:2] は下端のサイズが 1 で，上端のサイズは 2）で記述する．図 4 の例では Y 次元方向の両端に要素 1 個分の領域が余分に宣言される（以下 shadow 領域）．shadow 領域の index は隣接するノードで宣言された境界要素に対応する．しかし，その実体はローカルに確保されたメモリ領域であるため，正しい値を得るためにはノード間通信による同期が必要である．shadow 領域の同期には reflect 指示文が用いられる．XcalableMP コンパイラは reflect 指示文を記述した場所に shadow 領域に対する通信を挿入する．その結果，reflect 指示文の直後は shadow 領域が隣接ノードの境界領域と同じ値を持つことが保証される．

必要なデータの index が不明な時は分散された全ての要素を集約することで配列の同期を行い，問題を解決することができる．shadow 領域のサイズを‘‘ : ’’と指定することで全ての領域の重複宣言を行うことができる（このような領域を full shadow と呼ぶ）．full shadow に対する reflect 指示文は MPI における MPI_Allgather() と同等である．従って full shadow の宣言と同期には通信コストに注意しなければならない．

shadow 領域の同期以外にも，典型的な通信の記述手法として以下のような指示文が提供される．

- **#pragma xmp bcast *var* on *node***
  指定されたノードからのデータのブロードキャストを行う
- **#pragma xmp barrier**
  バリア同期を行う
- **#pragma xmp reduction (*var*:*op*)**
  リダクション操作を行う
- **#pragma xmp gmove**
  直後に記述される代入文（または代入文で構成されるブロック文）がローカル領域ではなく，データが割り当てられたノードを参照するように通信を生成する

4.3 ローカルビューモデル

ローカルビューモデルでは各ノードでのローカルメモリとノード間通信を強く意識した並列プログラミングを行う．XcalableMP ではローカルビューの記述手法として CAF の仕様も用いる．CAF の実行モデルは XcalableMP と同じ（MPI と同様，SPMD 実行モデルを持つ）であるため，二つのモデルが違和感なく混在するように言語の設計を行うことができる．

CAF は Fortran 言語に Co-Array の概念を付加することで並列プログラミングを行うように拡張したものである．Co-Array 次元を持つ配列を Co-Array と呼び，言語拡張を用いることで他のノード（CAF ではイメージと呼ぶ）で宣言されたデータを参照することができる．以下に CAF の記述例を示す．

real dimension a(MAX)[*]
...
b(:) = a(:)[1]

各ノードで MAX 個の real 要素を持つ Co-array が宣言される．Co-array は Fortran の基本データ型の変数やその配列を要素として持つ配列である．Co-array で拡張された次元を Co-array 次元と呼ぶ．CAF や MPI のような SPMD 実行モデルの下では，各ノードのデータが異なる実体を持つ．Co-array は各ノードで宣言されたデータを一つの配列として表したものである．従って，Co-array 次元とはノードの集合を表すものである．CAF の言語仕様では Co-array 次元は常に一次であるため，実行ノードの集合も一次元の配列で表される．Co-array の次元をノード番号で参照することで他のノードの Co-array を参照することができる．上記の例はノード 1 が持つ Co-array a の値をローカル配列 b に代入することを CAF 仕様で記述したものである．

CAF は Fortran 言語の拡張である．XcalableMP は C と Fortran をベースにした言語であるため，C 言語に対しても同等の機能を提供しなければならない．Co-Array を宣言するため，coarray 指示文を用いる．coarray 指示文で指定された配列は Co-Array として扱うことができる．また，CAF でよく使われる Fortran 仕様の部分配列機能（array section）を C 言語に導入する．C 言語での Co-Array の記述方法は以下の通りである．

double a[MAX];
#pragma xmp coarray a(*)
...
b[:] = a[:]@(1);

言語仕様のため，C 言語ではローカル配列のアクセスと Co-array 次元のアクセスに用いる括弧の種類が Fortran と逆転する．Co-array 次元のアクセスのために‘‘ @ ’’記号の後にノード番号を記述する．部分配列機能によりノード 1 の a の全ての要素が b に代入される．

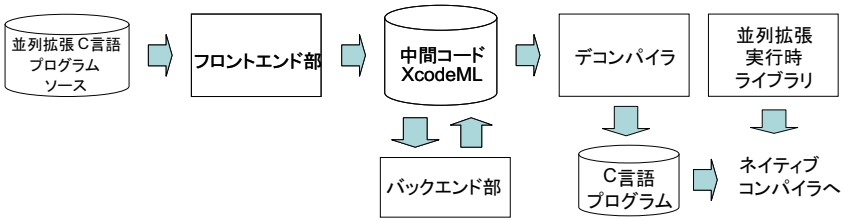

図 5 XcalableMP コンパイラの構造

CAF の言語拡張を用いることで MPI が提供する片側通信と同等のものを MPI に比べて少ないプログラミングコストで記述することができる．XcalableMP のローカルビューは MPI との親和性が高いため，MPI とローカルビューモデルを併用することも可能である．グローバルビューで記述しきれないアプリケーションはローカルビューでのより自由な記述手法を使うことで並列化することができる．

4.3.1 グローバルビューとローカルビューの連携

XcalableMP では，最初はグローバルビューで並列化を行い，ローカルビューを併用して最適化を行うことでアプリケーションのパフォーマンスチューニングをインクリメンタルに行うことができる．グローバルビューとローカルビューモデルを併用するためのインターフェイスとしてグローバルビューで記述された配列（正確にはグローバルビューモデルの指示文で分割されて各ノードに宣言されるローカル配列）にローカルビューとしてアクセスするための別の名前を用意する指示文を提供する．二つの名前を使い分けることで二つのプログラミングモデルが混在したプログラムを記述することができる．他にもモデル間の index 変換，配列の割り当て情報を取得するための組み込み関数がユーザに提供される．

5. コンパイラの試作

XcalableMP コンパイラは C または Fortran のベース言語と XcalableMP の言語拡張で記述されたソースコードを読み込み，並列コードに変換する．C 言語ベースの XcalableMP コンパイラの構造を図 5 に示す．フロントエンド部は XcalableMP の言語拡張を含む C 言語のソースコードを読み込み，計算機により近い形式である中間言語に変換する．中間言語に変換することでコードの変換が容易になるだけでなく，C と Fortran のコンパイラで以降の処理を共有することができる．XcalableMP の中間言語として，独自形式である XcodeML を用いる．XcodeML はベース言語と同等の情報を持ち，元の言語に復元可能である．XML によって記述されており，well-defined で human-readable なフォーマットである．バックエンド部はソースコードの変換を行う．指示文などの言語拡張で得られた並列化情報を元に逐次コードを並列コードに変換する．並列コードは通信や index 分割のためにランタイムライブラリの関数を利用している．並列コードはデコンパイラによって C 言語形式に戻される．最後に，コードはランライムライブラリと共にネイティブコンパイラ（gcc など）に渡されて並列プログラムのバイナリが生成される．

```
int **array;
__xmp_array_handle_t *__array_handle;                    // 配列の並列化情報を収納

main(int argc, char **argv) {
  int i,j,res = 0, __local_i_lower, __local_i_upper;     // ループ並列化に用いる変数
  __xmp_init(&argc, &argv);                              // ランタイムの初期化
  __array_handle = __xmp_distribute_array_2(&array, ...); // 配列の分散割り当て
  __local_i_lower = __xmp_get_lower(__array_handle, ...); // ループ反復の下限
  __local_i_upper = __xmp_get_upper(__array_handle, ...); // ループ反復の上限
  for(i = __local_i_lower; i < __local_i_upper; i++) {   // ワークシェアリング
    for(j = 0; j < XMAX; j++) {
      array[i][j] = func(i,j);
      res += array[i][j];
    }
  }
  __xmp_allreduce(&res, ...);                            // リダクション操作
  __xmp_finalize();                                      // ランタイム終了
}
```

図 6 コンパイラによって生成される並列コードの例

コンパイラによるコード変換の例として，図 2 から生成される並列コードを図 6 に示す．配列の分散割り当てや index 分割を行うため，ランタイムライブラリが呼び出される．ランタイムライブラリが生成した情報を収納するために内部変数が宣言される．loop や task 指示文が記述された場合，コンパイラはプログラムの構造を変更（ループ文の処理範囲や実行するノードを制限）することで並列コードを生成する．通信を記述する指示文（barrier, reduction など）はノード間通信を行うランタイムライブラリ関数呼び出しに置き換えられる．

現在，コンパイラを実装中であり，まだ完成には至っていない．指示文の構文解析を行うフロントエンド部がほぼ完成しており，ランタイムライブラリの機能を一部実装している．ランタイムライブラリの実装には通信ライブラリとして MPI を用いる．

6. 性能評価

ここでは XcalableMP を用いてベンチマークを並列化し，その性能評価を行う．評価対象は Nas Parallel Benchmarks の CG である（以降，NPB-CG）．コンパイラがまだ実装の途中であるため，性能評価には手動で並列化したソースコードを用いている．

6.1 評価環境

評価には一般的なタイプの PC クラスタを最大 16 ノード利用した．評価環境のノード構成を表 1 に示す．各ノードは Quad Core CPU を持つが，ノード内並列化は行わず，各々のノードで一つのコアのみを利用する．

6.2 NPB-CG の並列化

NPB-CG は正値対称な大規模疎行列の最小固有値を，共役勾配法を用いて求めるものである．行列・ベクトルの掛け算が実行時間の大半を占めるため，その並列化が性能に大きく影響する．XcalableMP で並列化された NPB-CG のコードを図 7 に示す．固有値を求める関数 conj_grad() が計算時間の大半を占めるため，この関数を並列化することでパフォーマンスを向上させることができる．今回の評価では効率が良いとされる 2 次元分割を用いて並列化を行う．

6.2.1 template を用いた index 空間の分割

まず，nodes 指示文で実行するノードの集合を宣言する．2 次元分割を行うため，2 次元のノード集合が宣言される．MPI 版と同様，NPCOL は NPROW と同じか，2 倍の値を持つと仮定する．

次に，2 次元の疎行列のデータ空間を表す template を宣言する．NPB-CG の疎行列は後述するように 1 次元の配列の実体を持つが，論理的には 2 次元のデータ空間を持つ．疎行列の index 空間を表すため，2 次元の template を宣言する．template を宣言したら distribute 指示文で index 空間の分割を行う．各ノードで処理を均等に分散させるため，2 次元のブロック分割を行う．

疎行列のデータを収納する配列 a はメモリの利用効率を考慮し，CRS 形式の 1 次元配列

表 1 評価環境のノード構成

CPU	Intel(R) Core(TM)2 Quad CPU Q9650 3.00GHz
Memory	8 GB
Network	1000 BASE-T Ethernet
OS	Linux kernel 2.6.28 x86_64
MPI	OpenMPI 1.3.2

```
#pragma xmp nodes on p(NPCOL,NPROW)
#pragma xmp template t(0:na+1, 0:na+1)
#pragma xmp distribute t(BLOCK, BLOCK) on p

double x[na+2], z[na+2], p[na+2],
       q[na+2], r[na+2], w[na+2];

#pragma xmp align [i] with t(i,*) :: x,z,p,q,r
#pragma xmp align [i] with t(*,i) :: w

static void conj_grad ( ... )
{
  . . .
#pragma xmp loop on t(i,*)
  for(j = 0; j < lastcol–firstcol; j++) {
    . . . // ベクトルの計算
  }
  . . .
#pragma xmp loop on t(*,i)
  for (j = 0; j <= lastrow-firstrow; j++) {
    sum = 0.0;
    for (k = rowstr[j]; k <= rowstr[j+1]-1; k++) {
      sum = sum + a[k]*p[colidx[k]];
    }
    // 元はq[j] = sum;
    w[j] = sum;
  }

#pragma reduction(+:w) on p(*,:)

#pragma xmp gmove
  q[:] = w[:];
  . . .
}
```

図 7 XcalableMP による NPB-CG ベンチマークの並列化

として宣言される．行列へのアクセスのため，各行と列の index を収納する配列 rowstr と colidx が生成される．これらの分割は指示文で記述することが困難であるため，NPB-CG の XcalableMP 版では行列 a とその index 配列 rowstr，colidx の分割を手動で行う．そのため，a，rowstr，colidx をローカル配列として各ノードで宣言する．生成される疎行列のデータが，割り当てられた template 空間の上に存在する時，それを a に収納して indexh 配列に情報を記録する（これらの手順は MPI 版の疎行列の生成と同じである）．

NPB-CG ではベクトル x，z，p，q，r を用いて計算を行う．ベクトル p，z は疎行列との掛け算に用いられるため，行列の各行の分割に整合してベクトルを分割することが望ましい．ベクトル x，q，r はその計算に p や z の値を利用する．ノード間通信を記述せず，なるべくローカルな処理のみでベクトル間のデータ交換を行うため，これらのベクトルも p と z に整合して分割する．align 指示文を記述し，行列の行の分割（template t(i,*)）に整合してベクトルを分割させる．

6.2.2 ループ文の並列実行

図 7 に示すように，ベクトルに対する初期化や値の代入は単一ループ文で行われる．その並列化は loop 指示文を用いて記述する．ループ文の並列実行とデータの分割を整合させるため，データの分割に用いた template の次元を記述する．

行列・ベクトル積は 2 重ループ文で処理される．rowstr のローカルな宣言と生成によって，内側のループ文は手動で並列化される．外側のループ文を並列化するために，loop 指示文を

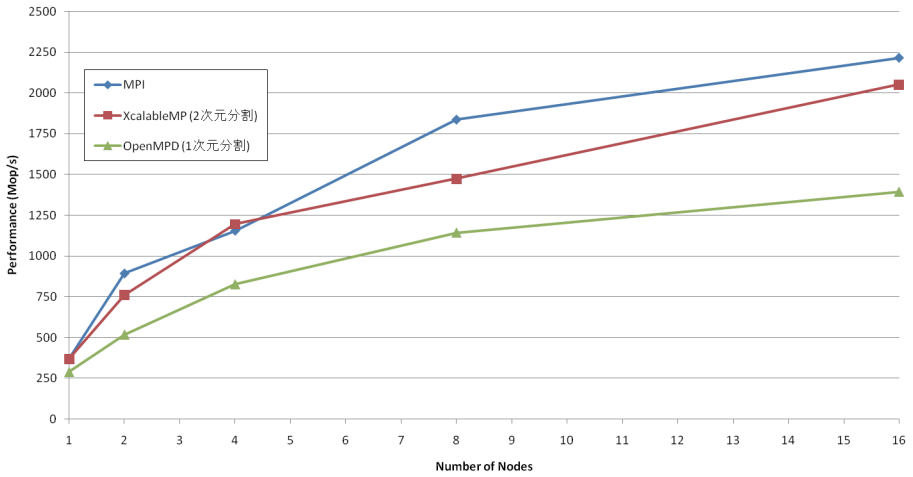

図 8 NPB-CG の評価結果

記述する．loop 指示文の記述により，行列の各行とベクトルの内積が並列に求められる．逐次版の NPB-CG では内積 sum の値が配列 r や q に直接代入される．しかし，XcalableMP 版の CG ではこれらの配列が行列の行の分割に整合して分割されているため，データの分割と並列実行の整合性が取れない．この問題を解決するため，一時バッファーの配列 w を宣言し，行列の列の分割に整合して分割を行う．内積を w に代入することで，ワークシェアリングとデータの分割を整合させることができる．

6.2.3 通信の記述

内積を計算するループ文（内側のループ文）が並列化されているため，w の各要素は不完全な値を持つ．内積の値を完成させるため，リダクション操作を行う．図 7 では reduction 指示文の on 節で p(*,:) と記述し，p(1,y) から p(NPCOL,y) の間でリダクション操作が行われるように記述している（y の値が異なるとリダクション操作も独立に行われる）．

次に，w のデータを r や q に代入する．図 7 では部分配列機能を用いて w の全ての要素を q に代入している．これは異なる次元で分割されたベクトル間のデータ交換（転置）であり，ノード間通信が必要となる．gmove 指示文を記述し，転置に必要なノード間通信を生成するようにコンパイラに指示を行う．

6.3 評価結果と考察

図 8 に NPB-CG の評価結果を示す．NPB-CG を XcalableMP で並列化し，MPI 版との比較を行う．参考のため，OpenMPD による NPB-CG の 1 次元分割の評価結果を一緒に示す．OpenMPD は指示文によるグローバルビューの並列化と明示的な通信の記述など，XcalableMP と同じコンセプトを持つもので，XcalableMP の設計のベースとなっている言語モデルの一つである．XcalableMP で CG を 1 次元分割した場合，OpenMPD と類似したコードが生成される．従って，1 次元分割の結果をより正確に示すため，OpenMPD の実物のコンパイラを用いた評価結果を掲載する．XcalableMP 版の NPB-CG はコンパイラの挙動を考慮し，手動で並列化したものである．コンパイラが挿入するランタイムライブラリを実装し，手動でコードの変換を行っている．通信ライブラリには MPI を用いる．

ノード数が 1 の時の値は MPI と XcalableMP，OpenMPD 共に逐次バージョンのものである．MPI と XcalabeMP は Fortran で書かれた NPB 3.2 のコードを用いているが，OpenMPD の場合は独自に C 言語で書き直された NPB 2.3 のコードを利用しているため，ベンチマークの絶対性能が異なる．全てのケースでノード内並列化は行っていないので，各ノードでのスレッドの数は一つである．

6.3.1 OpenMPD による 1 次元分割の評価結果

OpenMPD を用いた CG の 1 次元分割は行列・ベクトルの掛け算の計算において，2 重ループ文の外側のループ文のみを並列に実行するものである．内積を求める内側のループ文が分割されていないため，ベクトル p と z の全ての領域の値が必要となる．従って，OpenMPD 版ではこれらの配列の full shadow を宣言し，参照前に同期を行っている．full shadow の同期は配列全体に対する通信が必要となるため，コストが高い．図 8 の結果から分かるように，高い通信コストが性能に影響するため，他のモデルと比べて並列化の効率が悪い．

6.3.2 XcalableMP による 2 次元分割の評価結果

XcalableMP 版では内側のループ文も並列化されているため，full shadow の宣言や同期は不要である．その代わり，ベクトルのリダクション操作とデータの転置のための通信が必要である（この通信パターンは MPI 版と同じである）．2 ノードと 8 ノードで実行した場合，XcalableMP 版は MPI 版と比べて性能が低下する．その理由は MPI 版が後に転置を行う要素のみをリダクション操作の対象にするのに対し，XcalableMP 版は割り当てられた全ての要素に対してリダクション操作を行っているからである．NPCOL と NPROW が同じ値を持つ 4 ノード（2 × 2）と 16 ノード（4 × 4）の場合は，割り当てられた全ての要素が転置に用いられるため，リダクション操作のための通信コストが MPI 版と同じである．この場合，MPI 版の MPI_Send() と MPI_Irecv() で記述された Binary Exhange より，XcalabelMP のリダクション操作で用いられる MPI_Allreduce() の方が効率的な通信

を行うため，4 ノードの場合は XcalableMP 版が MPI 版より高い性能を示す．16 ノードの場合はリダクション操作を効率的に行えるものの，後述するように gmove の実装がまだ十分に最適化されていないため，MPI 版より低い性能を示す．

試作では gmove 指示文による通信の生成を MPI の片側通信で実装している．グローバルな index を解析し，自分に割り当てられた領域の取得を行うコードが生成される．代入される元となる領域が自分に割り当てられている場合，ローカルなメモリコピーでデータを取得するが，他のノードに存在する場合は MPI_Get() を用いた片側通信を行う．データを持っているノードが複数存在する場合，現在の設計では選択できるノードの集合を代表し，特定のノードがデータを提供する（無論，より効率的に通信を分散させる方法は色々考えられるが，コンパイラが実装途中の段階のため，最もシンプルで確実に実現できる手法を選択した）．そのため，8 と 16 ノードの場合は片側通信の要求が一つのノードに集中し，MPI 版のような効率的な転置ができない（4 ノードの場合はデータを持ってくるノードが 1 対 1 に決まり，MPI 版と同じ通信を行う）．

6.3.3 評価結果の考察

評価結果は XcalableMP が少ないプログラミングコストで MPI に近い性能を達成することを示している．性能低下の原因はリダクション操作の通信範囲の違いと gmove の非効率的な実装である．2 ノードと 8 ノードでの評価結果で，MPI 版の性能が高い主な理由はリダクション操作の範囲を調整する（XcalableMP 版と比べてより多くの記述が必要な）チューニングが施されているからで，XcalableMP 版でもユーザが明示的な通信を記述することで性能をチューニングすることができる．gmove に関しては通信をシステム全体に分散させるようにするなど，より効率的な実装が課題となる．4 ノードでの結果から，gmove を効率的に実装することで，MPI 版と同等かより高い性能を達成することが期待される．

7. 現状と今後の課題

本稿では PC クラスタなど分散メモリ環境のための並列プログラミングモデルとして C や Fortran 言語の並列拡張である XcalableMP を提案した．既存の分散メモリ向け並列プログラミングモデルではパフォーマンスチューニングができない，またはチューニングのための自由な記述ができるがプログラミングコストが高いという問題があった．XcalableMP は二つのプログラミングモデルを同一の言語で併用できるようにして，プログラミングコストと記述力を両立させている．

グローバルビューモデルでは典型的な並列化手法を OpenMP-like な指示文で記述する．MPI と比べてプログラミングコストが少ないというメリットがあり，逐次コードからのシームレスな並列化を行うことができる．2 次元分割で並列化された NPB-CG の性能評価では，MPI で並列化した場合と同等の処理をより少ないプログラミングコストで記述することができ，MPI 版に近い性能向上が得られることが分かった．

グローバルビューモデルで十分な性能を達成できないアプリケーションに対してはローカルビューモデルを利用することができる．ローカルビューモデルでは CAF-like な記述による片側通信を行う．MPI と同等に自由度が高く，MPI 関数呼び出しより直感的な通信の記述が可能である．

更なるパフォーマンスチューニングのために OpenMP や MPI と XcalableMP の併用が可能である．グローバルビューモデルは OpenMP と親和性が高く，実行や通信モデルは MPI と同じであるため，これらのモデルが違和感なく混在することができる．

現在，言語仕様の細部をつめている．今後，タスク並列化や集団通信を記述するための指示文の設計を行う．それと同時に，コンパイラのバックエンド部やランタイムライブラリの実装を進め，プロトタイプコンパイラの完成を目指す．今回の性能評価はグローバルビューを用いた場合を想定しているため，ローカルビューモデルを用いた場合の性能評価が必要である．また，gmove 指示文やランタイムライブラリの効率的な実装が課題となる．

謝辞 本研究の一部は，文部科学省「e-サイエンス実現のためのシステム統合・連携ソフトウェアの研究開発・高生産・高性能計算機環境実現のための研究開発・シームレス高生産・高性能プログラミング環境」による．XcalableMP の仕様は，検討している次世代並列プログラミング言語検討委員会によるものである．

参 考 文 献

1) XcalableMP, http://www.xcalablemp.org/
2) C. Bell, D. Bonachea, R. Nishtala, K. Yelick, ”Optimizing Bandwidth Limited Problems Using One-Sided Communication and Overlap”, 20th International Parallel & Distributed Processing Symposium (IPDPS), 2006.
3) UPC Language Specifications V1.2, http://upc.lbl.gov/docs/user/upc_spec_1.2.pdf
4) Stephen Olivier, Jan Prins, ”Scalable Dynamic Load Balancing Using UPC”, The 37th International Conference on Parallel Processing (ICPP08), pp.123-131, 2008.
5) Robert W. Numrich, John Reid, ”Co-array Fortran for parallel programming”, ACM SIGPLAN Fortran Forum Volume 17 Issue 2, pp.1-31, 1998.
6) Yuri Dotsenko, Cristian Coarfa, John Mellor-Crummey, ”A Multi-platform Co-Array Fortran Compiler”, Proceedings of the 13th International Conference of Par-

allel Architectures and Compilation Techniques (PACT 2004), pp.29-40, 2004.
7) Cristian Coarfa, Yuri Dotsenko, Jason Eckhardt, John Mellor-Crummey, "Co-array Fortran Performance and Potential: An NPB Experimental Study", The 16th International Workshop on Languages and Compilers for Parallel Computing (LCPC 2003), 2003.
8) Cristian Coarfa, et al., "An Evaluation of Global Address Space Languages: Co-Array Fortran and Unified Parallel C", The Tenth ACM SIGPLAN Symposium on Principles and Practice of Parallel Programming, pp.36-47, PPoPP 2005.
9) High Performance Fortran 言語仕様書 Version 2.0, http://www.hpfpc.org/jahpf/spec/hpf-v20-j10.pdf
10) 太田 寛, 西谷 康仁, 小林 篤, 布広 永示, "HPF 処理系 Parallel FORTRAN による NAS Parallel ベンチマークの並列化", Transactions of Information Processing Society of Japan 38(9), pp.1830-1839, 1997.
11) 村井 均, 岡部 寿男, "地球シミュレータ上の HPF による NAS Parallel Benchmarks の実装と評価", In Proc. of SACSIS2004, pp.389-396, 2004.
12) Jinpil Lee, Mitsuhisa Sato, Taisuke Boku, "OpenMPD: A Directive Based Data Parallel Language Extensions for Distributed Memory Systems", In Proc. of ICPP08 - Workshops, pp.121-128, 2008.
13) 岩下 英俊, 進藤 達也, 岡田 信, "VPP Fortran:分散メモリ型並列計算機言語", Transactions of Information Processing Society of Japan 36(7), pp.1542-1550, 1995.