Coarrays in the Context of XcalableMP

H. Iwashita and M. Nakao

Abstract @@@ XcalableACC (XACC) is an extension of XcalableMP for accelerated clusters. It is defined as a diagonal integration of XcalableMP and OpenACC, which is another directive-based language designed to program heterogeneous CPU/accelerator systems. XACC has features for handling distributed-memory parallelism, inherited from XMP, offloading tasks to accelerators, inherited from OpenACC, and two additional functions: data/work mapping among multiple accelerators and direct communication between accelerators.

Hidetoshi Iwashita
Fujitsu Limited, 140 Miyamoto, Numazu-shi, Shizuoka 410-0396, Japan, e-mail: iwashita.hideto@fujitsu.com

Masahiro Nakao
RIKEN Center for Computational Science, 7-1-26 Minatojima-minami-machi, Chuo-ku, Kobe, Hyogo 650-0047, Japan, e-mail: masahiro.nakao@riken.jp

1 Introduction

Omni XMP は，PC クラスタコンソーシアムの XcalableMP 規格部会が制定する並列言語 XcalableMP（XMP）の実装である．XMP は，Fortran と C をベースとし，ディレクティブ行の挿入によって並列化を記述するが，Fortran 2008 で定義される coarray 機能も仕様として含んでいる．前者は「逐次プログラムに指示を与えて並列化する」という考え方からグローバルビューと呼ばれ，並列プログラミングが容易にできることを狙う．後者は「個々のノード（イメージ）の挙動を記述する」という考え方でローカルビューと呼ばれる。後者は、前者では記述困難な領域のアプリケーションを記述するため、それと、局所的に性能を出したい部分を MPI よりも容易なプログラミング言語という手段で記述するために導入された。そのため、XMP の文脈の中で自然な解釈ができて同時に使用できることと、同時に、MPI に匹敵する高い性能が求められた。

我々は、XMP コンパイラの一つの機能として、Fortran 2008 仕様で定義された Coarray 機能を実装した。また、言語仕様を Fortran に準じて、C 言語ベースの Coarray 機能もサポートした。Fortran で定義される image は XMP 仕様で定義される node に 1 対 1 に mapping されるため、coarray 変数の定義・参照は node 間の put/get の片側通信としてそれぞれ実現される。片側通信と同期のための communiation library として GASNet，MPI3 または Fujitsu 固有の通信ライブラリを使用する。

本章では、coarray 機能を高性能で実現するために要求された技術と、その実現方法を示す。通信ライブラリの特質から、Coarray データの割付けは可能な限り利用者プログラムの実行前に集約すべきである。片側通信ベースなので、遠隔側のアドレス計算を低コストで実現する工夫が必要となる。通信が簡単に記述できることから、小粒度の頻繁な通信が起こりがちになるので、それらを aggregate して高速化する技術が要求される。

これらに対応した実装により、【評価の成果を】

この章の続く構成は以下の通りである。。。。

2 Language Specifications and the Key of the Implementation

The XcalableMP/Fortran compiler supports a major part of coarray features defined in Fortran 2008 standard [6], and intrinsic procedures CO_SUM, CO_MAX, CO_MIN and $CO_BROADCAST$ defined in Fortran 2018 standard [7] were supported.

This section introduces some coarray features and the issues toward the implementation.

2.1 Images in the context of XcalableMP

An *image* is a instance of a program. Each image executes the same program and has its own data individually (SPMD: Single Program/Multiple Data). As the default,

data is private to each image and is not allowed to be accessed from the other images. *Coarray* is only the variable that is allowed to be accessed from the other images in the usual Fortran contexts.

In XcalableMP laguage specification defines that the images are mapped in tern to the nodes in the executing task. It means that the images are not always mapped to the entire nodes but mapped to the subset nodes. Coarrays that are declared in a subprogram to be executed on a subset of nodes are mapped to the subset of nodes, and the $SYNC_IMAGES$ statement synchronizes among the subset of nodes.

多重タスク実行がある場合、それぞれのタスクで coarray プログラムが原則として独立に実行される。あるタスクで allocate した coarray は、そのタスクの有効範囲を超えない範囲で、タスクを実行する node 間で共有されるのが原則である。同期の範囲も原則としてその中である。

これを入れるかどうか　タスク実行のときのスタックの使い方、構文が入れ子なので同じレベルに戻ることができること。

2.2 Non-allocatable and allocatable coarrays

The coarray variables can be declared as scalar or array variables with the SAVE attribute (the static attribute of C) as follows:

```
real(8), save :: a(100,100)[*]
type(user_defined_type), save :: s[2,2,*]
```

The square bracket notation in the declaration distingishes coarray variables from the usual (non-coarray) variables. It declares the virtual shape of the images and the last dimension must be defferred (as '*'). This type of coarrays are called as *non-allocatable coarrays*.

Alternatively, coarrays can be declared as *allocatable coarrays* with the following form:

```
real(8), allocatable :: b(:,:)[:]
type(user_defined_type), allocatable :: t[:,:,:]
```

割付け coarray は ALLOCATE 文を使って割付けて，DEALLOCATE 文を使って解放することができる．Fortran では，手続に局所的で save 属性を持たない割付け変数は，手続からの復帰時に割付け状態であると，自動的に解放される．CAF では coarray 変数についてもこの機能 (automatinc deallocation) が要求されている．

共通の制約として，これらは全イメージで集団的に実行されなければならず，全イメージで同じ形状を割付けなければならない．これらの制約により，処理系は coarray のメモリ配置をイメージ間で synmetric に保つことができるので，リモート coarray のアドレス計算をローカル側の情報だけで行うことができる。例えば，他イメージの変数の多次元の添字式から相対アドレスを計算するには，自イメージの同名の変数の情報を参照すればよいので，他イメージ

のインデックス情報を受け取る必要も，事前に収集してテーブルに保存しておく必要もない．これにより MPI の lendezbous 通信のような通信前のアドレス交換のための通信を不要にできる。

communication library によっては、Fortran ランタイムシステムが割付けた領域を効率的な片側通信の対象にすることができない。その場合には、communication libary が提供する方法で割付けた領域を Fortran ランタイムシステムに渡して Fortran の変数であると認識させる実装が求められる。また、データを片側通信の対象とするためには、commnication library にそれを *register* しなければならない。commnucation library はその領域を RDMA の対象とするための pin-down などの必要な前処理を行う必要があるからである。registration コストの削減のためには、すべての non-allocatable coarray は 1 カ所にまとめて利用者プログラムの実行開始前に 1 回だけ行うようにするのが望ましい。

2.3 Communication

one-sided と collective と atomic の 3 つ。

2.3.1 Get 通信

イメージ k における coarray 変数 a の値を参照するには a[k] と記述する．これはイメージ k からの Get 通信を引き起こす．a が配列名の場合には配列の全要素に対する通信を意味する．a の部分配列を得るには配列記述を使って a(:, j1:j2)[k] などと書くこともできる．これらは coindexed object と呼ばれる．coindexed object の参照によって起こる通信は一般に連続データであるとは限らない．部分配列の場合はもちろん，配列名であっても形状引継ぎ配列（仮引数の一種）の場合には実引数次第で不連続になる．通信の高速化のためには，十分に長い範囲の連続性を実行時に抽出する必要がある．一般にノード間通信で立ち上がりレイテンシ時間がほぼ無視できるようになるデータ量は数千バイトのオーダーであることから，配列や部分配列の 1 次元め（Fortran では 1 番左の添字）が連続であっても数千要素に満たない場合には，次元を跨いだ連続性まで抽出して，十分に長い連続データとして通信する技術が必要である．

2.3.2 Put 通信

以下のように coindex object と同じ形を代入文の左辺に書くと，Put 通信の表現となる．a [i] = 式この左辺式は coindexed variable と呼ばれる．Get 通信と同様，a は配列名にも部分配列にもなれるので，coindexed variable についても次元を跨いだ連続性の抽出が必要である．右辺式には任意のスカラ式または配列式が許されるため，右辺式データの連続性にも配慮しなければならない．高速な通信を実現するには，左辺と右辺で共通に連続な区間を検出してその単位で

通信を反復するか，右辺データは連続区間に pack して左辺の連続区間を単位として通信を反復するなどの戦略が考えられる．

2.3.3 集団通信

XMP 1.0 では Fortran2008 仕様の Coarray 機能までを使用範囲としているため，イメージ間のリダクション演算とブロードキャストを含まない．これらは必ず必要になると考えるので，Omni XMP では Fortran2015 で定義されている組込みサブルーチン co_sum, co_max, co_min と $co_broadcast$ の一部の機能を実装した．ランタイムライブラリの実装は，下位通信層に MPI を使うなら，対応する MPI 手続を呼ぶだけであるが，配列や部分配列のリダクション演算を copy-in/out をできるだけ起こさないように効率よく実装するには，CAF トランスレータ側に工夫が必要である．

2.4 Array Notation and Contiguity

MPI とくらべた coarray の優位性は array section を効率よく表現できることだ。Fortran 90 以降 Fortran に導入された配列式、配列代入の機能はそのまま coarray put/get communication の表記に活かされている。つまり array section はデータの部分を表現するとともに通信の範囲やパターンを表現する。MPI では後者は MPI_type などを使って次元毎に再帰的に表現し直さなければならない。

また、RDMA 可能かどうかはコンパイラと runtime 上位で判断できる。

これを効率よく実装する方法については次の章で。

2.5 Argument Passing

coarray 変数の引数渡しをトランスレータ方式で実現するには大きな選択肢があった．coarray 変数を 1 つの記述子に変換し，記述子を通して実体のアドレスとその他の情報を取り出せるようにすれば，引数渡しは記述子の受渡しだけとなる．しかしその場合には，データ実体が Fortran システムから見えなくなるので，dope ベクトルも我々が生成・管理しなければならない．そして，サブプログラム内での coarray 変数の通常の参照・定義に対して，dope ベクトルを使ったインデックスの計算式をソース-to-ソースで組み上げるか，実行時ライブラリ呼出しに変換しなければならない．これは大きな性能低下が予想される．我々は，coarray 変数を同じ形状をもつ非 coarray 変数に変換することで，Fortran システムの dope ベクトルの仕組みをそのまま使う方法を採った．その結果新しく生じた課題については 3.3 節で議論する．

2.6 Loose synchronization

各イメージでは従来通りのデータフローを守りながら、image 間では明示的な同期を書かない限り順序が保証されない仕様である。この仕様が連続通信の自由度と制限を決める。図を使って説明。参照は blocking になる。定義はデータ依存関係の許す限り、次の同期までの non-blocking が許される。ここに最適化のチャンスがある。

2.7 Coarray C Language Specifications

C 言語にも配列式、配列代入を導入することで Fortran と同様な coarray features を仕様定義した。

coarray はデータ実体に限る。ポインタは coarray になれない。形式的には、1) 基本型、2) ポインタを含まない構造型、または、3) 1or2or3 の配列とする。C の coarray においても static と allocatable がある。ファイル直下で宣言するか static 指示した coarray は static coarray である。ライブラリ関数 xxx をつかって割付けたデータ領域は allocatable coarray であり、ライブラリ関数は allocatable coarray へのポインタを返す。C の coarray は、通常の C の変数と同じように、引数渡しや cast 演算によって自由にその型と形状の解釈を変えることができる。

これらの仕様と制限は、C プログラマにとっての使いやすさを考えて、C らしいプログラミングスタイルを認めた。Coarray C++とは違うアプローチである。

cf. air:/Users/iwashita/Desktop/coarray/Project_Coarray/の下にいくつか

3 Compiler Implementation for high performance

3.0.1 static coarray の高速化

全部まとめて先に alloc するコンパイラ技術
定数評価、構造体の大きめな見積り

3.0.2 allocatable coarray の高速化

lower level communication library に合わせて allocation method を選択
runtimelibshare 事前に巨大領域を取る
実行時 alloc のコストが大きいとき
runtimeliballocation 実行時 alloc のコストが小さいとき

3.0.3 contiguouity 検出による高速化

次元を跨ぐ連続性抽出

3.0.4 バッファリング

有限サイズ、基礎データから、nlocal, nremote, nbuf の関係でアルゴリズム commpilerallocationfortran が allocate して register できるとき

4 Communication Library

implementation software layer

３種：one sided, collective, atomic. このうち collective と atomic は MPI の機能をそのまま使う。それ以上の工夫はない。one sided は allocate/free と put/get と同期。lower-level 通信層のバリエーションを吸収するライブラリ層を設けた。階層の図。

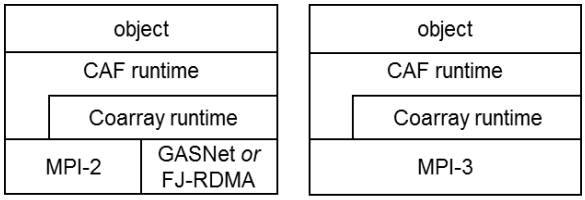


Fig. 1 Software Layer

cf. air:/Users/iwashita/Desktop/coarray/Project_Coarray/coarray implement 6-1.docx 他

次元の概念と contiguity を上位層で解決するため結果的に必要なインタフェースは少なくなった。表

5 Evaluation

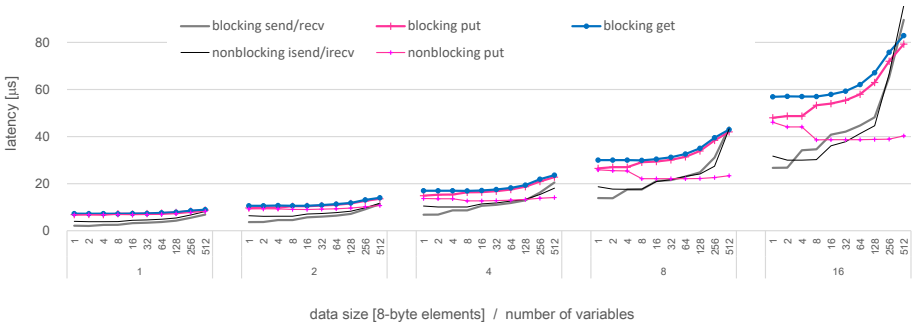
5.1 Aggregated Nonblocking Communication

n-var の効果があった。実際のアプリでもこれはよく起こるパターンである。

どういうパターンか分析して説明。n 個の contiguous な nonblocking comm.

Table 1 使用した Coarray ランタイムインタフェース（未公開機能関連を除く）

割付け・解放と登録	1. `_XMP_coarray_malloc_image_info_1` 2. `_XMP_coarray_malloc_info_1` 3. `_XMP_coarray_malloc_do` 4. `_XMP_coarray_regmem_do` 5. `_XMP_coarray_lastly_deallocate`
片側通信	6. `_XMP_coarray_shortcut_get` 7. `_XMP_coarray_shortcut_put`
同期	8. `xmp_sync_all` 9. `xmp_sync_image` 10 `.xmp_sync_images` 11 `.xmp_sync_images_all` 12 `.xmp_sync_memory`
atomic 通信	13 `._XMP_atomic_define_0` 14 `._XMP_atomic_define_1` 15 `._XMP_atomic_ref_0` 16 `._XMP_atomic_ref_1`
問合せ	17 `.xmp_all_num_nodes`
エラー処理	18 `._XMP_fatal`

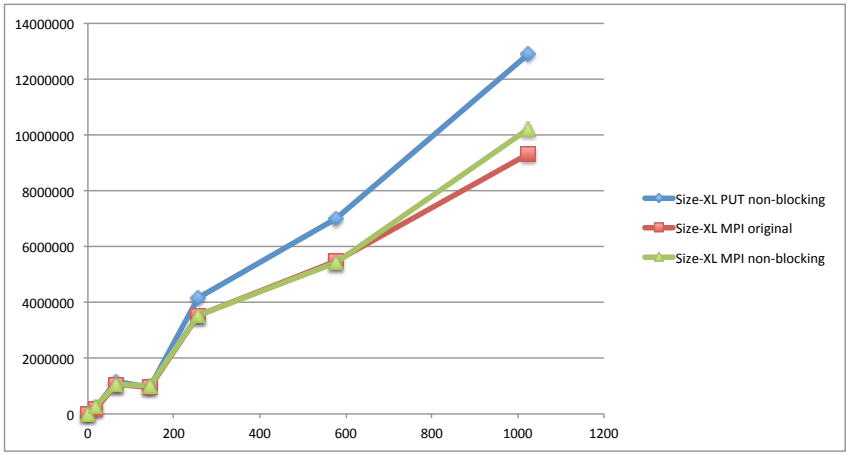


latency-16var.pdf
２からの４つにして図を大きくする。

Fig. 2 n-var latency of pingpong

6 Related Work

Coarray C++は



棒グラフにして L を加えて 2 つにするか M まで加えて 3 つにする
cf. air:/Users/iwashita/Desktop/coarray/Project_Coarray/coarray_runtime.pdf

Fig. 3 Himeno XL

task 間は F2018 の coarray にある。違いは …
Cray は get を directive で実装

7 Conclusion

8 MEMO

MPI は PGAS に向かない、という論文がどこかに。

9 Fusen

`synmetric memory`
リモートのアドレスをローカルで計算できる
これがあるから、片側通信が。。。

下位層：通信 `lib` の差異を吸収し、上位層にプリミティブを提供する。
`XMP coarray` 実装のプリミティブ `i/f`
`put operation`

get operation
- どちらも連続データのみ/nonblocking。
sync memory operation
- 発行した put をすべて remote に書き込み、get をすべて local に読み込むまで待つ。

put には nonblocking 技術：runtime で十分
get には prefetch 技術：コンパイラ技術

生産性の観点の評価
1. リンク時に static 配列を集めるところが対 MPI 片側通信で重要。言語仕様／言語処理系だからできる。単にライブラリ群／ライブラリ呼出しではできない。
2. F90 配列記述が MPI_Type の定義を不要にしている。

```
klogin7$ which xmpf90
~/Project/OMNI-clone/bin/xmpf90
klogin7$ xmpf90 --version
Version:1.3.1, Git Hash:4a5736e
klogin7$
```

sync memory がいつ必要か
- ユーザ記述
- get のとき、1 subarray 参照に対して syncmemory を 1 回にする。
- get の前に、同じ remote アドレスへの put があれば syncmemory で待つ

alignment の問題か
64 バイト未満で put の latency が高い。
src5, RUN5 で改善　… しなかった
メモリ割付けの alignment の問題ではない。
残るは Tofu の put の単位の問題と推測
マスク付き書き込みになる？
この点では RDMA よりバッファリングが有利

A Source-to-Source Translation of Coarray Fortran with MPI for High Performance
https://dl.acm.org/citation.cfm?id=3155888&dl=ACM&coll=DL

Preliminary Implementation of
Coarray Fortran Translator Based on Omni XcalableMP
https://ieeexplore.ieee.org/document/7306099

10 STATUS-CAF

```
1.2  Interoperability with the global-view features (NEW since V1.0)
  Coarray features can be used inside the TASK directive blocks. As default,
  each coarray image is mapped one-to-one to a node of the current executing
  task. I.e., num_images() returns the number of nodes of the current executing
  task and this_image() returns each image index in the task.
     There are two directives to change the default rule above. A COARRAY
  directive corresponding to a coarray declaration changes the image index set
  of the specified coarray with the one of the specified nodes. An IMAGE
  directive corresponding to one of a SYNC ALL statement, a SYNC IMAGES
  statement, a call statement calling CO_SUM, CO_MAX, CO_MIN or CO_BROADCAST
  changes the current image index set with the one of the specified nodes.
  See the language spacifications [3].

2. Declaration
  Either static or allocatable coarray data objects can be used in the program.
  Use- and host-associations are available but common- or equivalence-
  association are not allowed in conformity with the Fortran2008 standard.
  Current restrictions against Fortran2008 coarray features:
    * Rank (number of dimensions) of an array may not be more than 7.
    * A coarray cannot be of a derived type nor be a structure component.
    * A coarray cannot be of quadruple precision, i.e., 16-byte real or 32-byte
      complex.
    * Interface block cannot contains any specification of coarrays. To describe
      explicit interface, host-assocication (with internal procedure) and use-
      association (with module) can be used instead.
    * A pointer component of a derived-type coarray is not allowed.
    * An allocatable component of a derived-type coarray cannnot be referrenced
      as a coindexed object.
    * A derived-type coarray cannot be defined as allocatable.

2.1  Static Coarray
  E.g.
      real(8) :: a(100,100)[*], s(1000)[2,2,*]
      integer, save :: n[*], m(3)[4,*]
  The data object is allocated previously before the execution of the user
  program.  A recursive procedure cannot have a non-allocatable coarray without
  SAVE attribute.
  Current restrictions against Fortran2008 coarray features:
```

```
    * Each lower/upper bound of the shape must be such a simple expression that
      is an integer constant literal, a simple integer constant expression, or
      a reference of an integer named constant defined with a simple integer
      constant expression.
    * A coarray cannot be initialized with initialization or with a DATA
      statement.

2.2  Allocatable Coarray
  E.g.
      real(8), allocatable :: a(:,:)[:], s(:)[:]
      integer, allocatable, save :: n[:], m(:)[:,:]
  The data object is allocated with an ALLOCATE statement as follows:
      allocate ( a(100,100)[*], s(1000)[2,2,*] )
  The allocated coarray is deallocated with an explicit DEALLOCATE statement or
  with an automatic deallocation at the end of the scope of the name unless it
  has SAVE attribute.
  Current restrictions against fortran2008 coarray features:
    * A scalar coarray cannot be allocatable.
    * An allocatable coarray as a dummy argument cannot be allocated or
      deallocated inside the procedure.

3. Reference and definition of the remote coarrays
  For the performance of communication, it is recommended to use array
  assignment statements and array expressions of coindexed objects as follows:
      a(:) = b(i,:)[k1] * c(:,j)[k2]    !! getting data from images k1 and k2
      if (this_image(1))  d[k3] = e     !! putting data d on k3 from e
  Current restrictions on the K computer and Fujitsu PRIMEHPC FX10:
    * The coindexed object/variable must be aligned with the 4-byte boundary
      and the size of the array elements of them must be a multiple of 4 bytes.

4. Image control statements
  SYNC ALL, SYNC MEMORY and SYNC IMAGES statements are available.
  Current restrictions against Fortran2008 coarray features:
    * LOCK, UNLOCK, CRITICAL and END CRITICAL statements are not supported.
    * STAT= and ERRMSG= specifiers of image control statements are not
      supported.
    * ERROR STOP statement is not supported.

5. Incrinsic Functions
  Inquire functions NUM_IMAGES, THIS_IMAGE, IMAGE_INDEX, LCOBOUND and LUBOUND
  are supported.
  Current restrictions against Fortran2008 coarray features:
    * ATOMIC_DEFINE and ATOMIC_REF subroutines are not supported.

6. Intrinsic Procedures in Fortran2015
```

```
Argument SOURCE can be a coarray or a non-coarray.
Intrinsic subroutines CO_BROADCAST, CO_SUM, CO_MAX and CO_MIN can be used
only in the following form:
  * CO_BROADCAST with two arguments SOURCE and SOURCE_IMAGE
    E.g.,  call co_broadcast(a(:), image)
  * CO_SUM, CO_MAX and CO_MIN with two arguments SOURCE and RESULT
    E.g.,  call co_max(a, amax)
```

11 mac-air memo

introduction
coarrays in XcalableMP
support F2008 coarrays
image set を task に対して map
natural extension to C
basic implementation and issues
lower-level interface
evaluation
related work
Coarray C++は
task 間は F2018 の coarray にある。違いは …
Cray は get を directive で実装
conclusion

implementation for high performance
static coarray の高速化
全部まとめて先に alloc するコンパイラ技術
定数評価、構造体の大きめな見積り
allocatable coarray の高速化：lowlevel に合わせて選択
RuntimeLibShare 事前に巨大領域を取る
実行時 alloc のコストが大きいとき
RuntimeLibAllocation 実行時 alloc のコストが小さいとき
contiguouity 検出による高速化
次元を跨ぐ連続性抽出
バッファリング
有限サイズ、基礎データから、Nlocal, Nremote, Nbuf の関係でアルゴリズム
CommpilerAllocationFortran が allocate して register できるとき

implementation software layer
３種:one sided，collective，atomic．このうち collective と atomic は MPI の機能をそのまま使う。それ以上の工夫はない。one sided は allocate/free と put/get と同期。lower-level 通信層のバリエーションを吸収するライブラリ層を設けた。階層の図。
次元の概念と contiguity を上位層で解決するため結果的に必要なインタフェースは少なくなった

12 nigun

12.1 intro より related work の方がよいか？

PGAS 言語の一つである Coarray Fortran（CAF）は，Fortran 2008 仕様の一部として採用されたことも追い風となって，近年研究開発が盛んに進められている．フリーのものでは，Houston 大学の OpenUH コンパイラを開発基盤とした UH-CAF[5] と，Rice 大学の ROSE コンパイラを開発基盤とした caft[6] が有名である．近年リリースされた OpenCoarray は，GNU gfortran にリンクできるライブラリである．我々の開発する Omni XMP もまた，CAF コンパイラとして利用することができる．ベンダでは Cray と Intel が古くから提供しており，近年富士通からもリリースされている．これら 3 社は，Fortran 2008 と 2015 に含まれる coarray 機能の実装を Fortran2003 のフル実装よりも先行させたことになる．Omni XMP は，PC クラスタコンソーシアムの XcalableMP 規格部会が制定する並列言語 XcalableMP（XMP）のパイロット実装である．XMP は，Fortran と C をベースとし，ディレクティブ行の挿入によって並列化を記述するが，Fortran 2008 で定義される coarray 機能も仕様として含んでいる．前者は「逐次プログラムに指示を与えて並列化する」という考え方からグローバルビューと呼ばれ，並列プログラミングが容易にできることを狙う．後者は「個々のノード（イメージ）の挙動を記述する」という考え方でローカルビューと呼ばれ，MPI に匹敵する性能が MPI よりも容易に出せることを狙っている．

References

1. XcalableMP Language Specification, http://xcalablemp.org/specification.html (2017).
2. The OpenACC Application Programming Interface, http://www.openacc.org (2015).
3. MPI: A Message-Passing Interface Standard, http://mpi-forum.org (2015).
4. John Reid, JKR Associates, UK. Coarrays in the next Fortran Standard. ISO/IEC JTC1/SC22/WG5 N1824, April 21, 2010.
5. ISO/IEC TS 18508:2015, Information technology – Additional Parallel Features in Fortran, Technical Specification, December 1, 2015.

6. Fortran 2008
7. Fortran 2018