

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 communication 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徴から、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】

【要修正】 This chapter describes how the compiler and runtime techniques are implemented in the XMP compiler with some evaluation. In the rest of this chapter, Section 2 shows the basic implementation for coarray features, Section 3 describes how the implementation has solved the issues shown above, Section 4 evaluates the implementation with basic and practical applications, and Section 5 concludes.

2 Coarray Features and the Key of the Implementation

XMP Fortran language specification 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. And also XMP C language specification extended to support coarray features.

This section introduces the coarray features and what is required to the compiler in order to implement the coarray features.

2.1 Images Mapped to XMP Nodes

In the Fortran standard, an **image** is defined as a instance of a program. Each image executes the same program and has its own data individually (SPMD: Single Program/Multiple Data). Each image has a different image index, which is one of $1, \dots, n$, where n is the number of images. While Fortran standard does not specify actually where each image is executed, XMP defines that the images are mapped in tern to the executing nodes. Therefore, image k is always executed simply on executing node k , where $1 \leq k \leq n$ and n is the number of images and also the number of the executing nodes.

Note that the executing nodes can be a subset of the entire (initial) node set. For example, two distinct node sets can execute two coarray subprograms concurrently. When the execution encounters a TASK directive specified with a subset of nodes, the corresponding subset of the images will be the execting images for the TASK region. The number of images and my image number, which are given by inquire functions `num_images` and `this_image`, also match with the executing images, and the SYNC_IMAGES statement synchronizes among the executing images. When the execution encounters the END TASK directive corresponding to the TASK directive, the set of executing image is reinstated. Unless the TASK and END TASK directives are used, coarray features are compatible to the ones of the Fortran standard.

Requirement to the compiler. Memory management is necessary to cope with changes of the executing images.

2.2 Allocation of Coarrays

A **coarray** or a coarray variable is a variable that can be referred from the other images. A coarray with the ALLOCATABLE attribute is called an **allocatable coarray**, otherwise called a non-allocatable coarray. A non-allocatable coarray may not be a pointer and must have an explicit shape and the SAVE attribute. In order to help intuitive understanding, we call a non-allocatable coarray as a **static coarray**. The lifetime of a static coarray is throughout execution of the program. On the other hand, an allocatable coarray is allocated with the ALLOCATE statement and freed either explicitly with the DEALLOCATE statement or implicitly at the end of the scope in which the ALLOCATE statement is executed (**automatic deallocation**).

Static coarrays can be declared as scalar or array variables as follows:

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

The square bracket notation in the declaration distinguishes coarray variables from the others (non-coarrays). It declares the virtual shape of the images and the last dimension must be deffered (as '*').

Allocatable coarrays can be declared as follows:

```
real(8), allocatable :: b(:,:)[:]
```

```
type(user_defined_type), allocatable :: t[:, :, :]
```

A notable constraint is that at any synchronization point in program execution, coarrays must have the same dimensions (sizes of all axes) between all images (**synmetric memory allocation**). Therefore, an static coarray must have the same shape between all images during the program execution, and an allocatable coarray must be allocated and deallocated collectively at the same time with the same dimensions between the executing images. Thanks to the synmetric memory allocation rule, all executing images can have the same symmetrical memory layout, which makes it possible to calculate the address of the remote coarray with no prior interimage communication.

Requirement to the compiler. The language specification suggests the compiler to maintain the synmetric memory allocation rule and to use it for efficient remote address calculation.

automatic deallocation の生成が必要なことをどこかに。deregister と一緒に。片側通信を使って remote coarray をアクセスするためには、register があることを先に。

2.3 Communications

Coarray features include three types of communications between images, i.e.,

- reference and definition to remote coarrays (GET and PUT communication),
- collective communication (intrinsic subroutines CO_SUM, CO_MAX, CO_MIN and CO_BROADCAST), and
- atomic operations (ATOMIC_DEFINE and ATOMIC_REF).

Omni compiler implements the latter two communications using the corresponding functions in MPI. The rest of this section describes the former communication.

2.3.1 PUT communication

PUT communication is caused by an assignment statement with a **coindexed variable** as the left-hand side expression, e.g.,

$$a(i, j)[k] = \alpha * b(i, j) + c(i, j)$$

This statement is to cause the PUT communication to the array element $a(i, j)$ on image k with the value of the left-hand side.

Fortran の配列代入文を使えば、配列から配列への PUT 通信を記述することもできる。例えば以下の配列代入文は、 MN 要素の PUT 通信を生じさせる。

$$a(1:M, 1:N)[k] = \alpha * b(1:M, 1:N) + c(1:M, 1:N)$$

Requirement to the compiler. 配列代入文では、一般には右辺の評価結果を Fortran システムが一時領域に置くが、通信バッファをできる限り多重にしない工夫が必要である。右辺が配列の参照だけの場合はゼロコピー通信とすることが望ましい。また、次の image 制御文によって同期が行われるまで PUT されたデータは他の image から参照されないので、実行は PUT 通信の完了を待たないで次に進むことができる (nonblocking communication)。

2.3.2 GET communication

GET communication is caused by referencing the **coindexed object**, which is represented by a coarray variable with cosubscripts enclosed by square brackets, e.g., $s[1, 2]$ and $a(i, j)[k]$, where s and a are scalar and two-dimensional array coarrays, respectively. Coindexed objects can appear in most expressions.

値を配列とする配列式を使うと配列に対する GET 通信を行うことができる。例えば coindexed-object $a(1:M, 1:N)[k]$ の値は a と同じ型で形状 $[M, N]$ の 2 次元配列である。

Requirement to the compiler. ローカル側は、獲得したデータがすぐに消費されることから nonblocking 化は適さない。できる限りバッファリングを回避してゼロコピーとすることで効率化したい。受信側 (local 側) をゼロコピーにするため、最適化が有効。右辺に 1 つの coindexed-object しかない配列代入文、例えば、

$$a(1:M, 1:N) = b(1:M, 1:N)[k]$$

では、GET 通信の受信先をバッファでなく直接変数 a にすることで、ゼロコピーにできる場合がある。

2.4 Subarrays and Data Cotiguity

In Fortran, if an array, except dummy argument, is declared as `real a(1:M, 1:N)`, the reference to the whole array (i.e., a or $a(:, :)$ or $a(1:M, 1:N)$) is defined *contiguous*. We defined a term *contiguous length* as the length how long the data is partially contiguous. For example, the contiguous lengths of $a(2, 3)$ and $a(2:5, 3)$ are 1 and 4 respectively. $a(1:M, 1:3)$ is two-dimensionally contiguous and has contiguous length $2 \times M$. $a(1:M-1, 1:3)$ is one-dimensionally contiguous and has contiguous length $(M-1)$.

Requirement to the compiler. 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。【fx100-latency のグラフから数千バイトを言ってもよい。】

Get 通信と同様、 $a(i,j)[k]$ は代わりに全体配列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。それに加えて、ローカル側、すなわち、右辺式データの連続性も意識に入れなければならない。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ある。

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使った。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

3.1 Omni XMP Compiler Framework

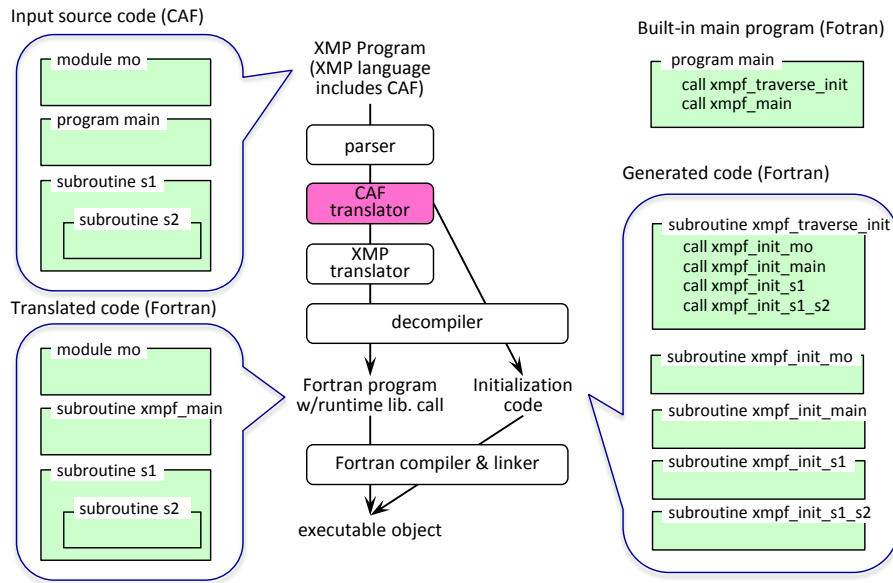


Fig. 1 XMP compiler and an example of coarray program compilation
CAF translator → coarray translator

The CAF translator was added into the Omni XMP compiler as shown in Figure 1. The Omni XMP compiler is a source-to-source translator that converts XMP programs into the base language (Fortran or C). The component ‘coarray translator’ is located in front of the XMP translator to solve coarray features previously. The output of the decompiler is a standard Fortran/C program that may include calls to the XMP runtime library.

In order to allocate and register static coarray variables prior to the execution of the user program, the built-in main program is made in order to call the initialization routine `xmpf_traverse_init` followed by user's main program. `xmpf_traverse_init` is generated by the coarray translator to call initialization routines (e.x., `xmpf_init_foo`) corresponding to user-defined procedures (e.x., `foo`).

3.2 PUT/GET Communication

To avoid disturbing the execution on the remote image, PUT and GET communications are implemented always using Remote Direct Memory Access (RDMA) provided by the communication library (except coarrays with pointer/allocatable structure components). In contrast, local data access is selective between using Direct Memory Access (DMA) or using a local buffer. And the suitable algorithm differs depending on the relationship of the three parameters, the size of the local buffer B and the local and remote contiguous lengths N_L and N_R . The Fortran syntax guarantees that N_L is a multiple of N_R or N_R is a multiple of N_L .

Table 1 summarizes our algorithm for PUT/GET communication for five cases. The unit size is the chunk length of the PUT/GET communication. Case 0 shows the algorithm using RDMA-DMA PUT/GET communication and Cases 1 through 4 shows the algorithms using RDMA and local-buffering. Due to its strict condition, the DMA scheme is rarely used. And it is not always faster than the buffering scheme cases 2 and 3 because of the difference of the unit sizes. The merit of cases 2 and 3 is that the unit size is extended to a multiple of N_L by gathering number of short contiguous data in the buffer, or by scattering from the buffer into number of short contiguous data.

Table 1 Summary of the PUT/GET algorithm related to N_L , N_R and B

case	scheme	condition	unit size
0	DMA	Local data is registered.	$\min(N_L, N_R)$
1	buffering	$N_R \leq B, N_R \leq N_L$	N_R
2		$N_L < N_R \leq B$	N_R
3		$N_L < B < N_R$	multiple of N_L and up to B
4		$B < N_R, B \leq N_L$	B (or less than B at last)

case	PUT action for every unit	GET action for every unit
0	put once	get once
1	buffer once and put once	get once and unbuffer once
2	buffer for each N_L , and put once	get once, and unbuffer for each N_L
3	buffer for each N_L , and put once	get once, and unbuffer for each N_L
4	buffer once and put once	get once and unbuffer once

3.2.1 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

3.2.2 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

3.2.3 contiguity 検出による高速化

次元を跨ぐ連続性抽出

3.2.4 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 Libraries

4.1 Layered Communication Libraries

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。

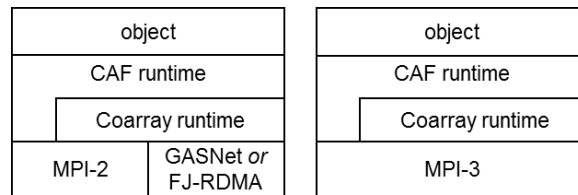


Fig. 2 Layered Communication Libraries(左の絵だけでよい)

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

To implement one-sided communication, the XcalableMP (XMP) runtime library selects one of the *communication libraries*, MPI-3, GASNet, or Fujitsu's native interface FJ-RDMA, as specified at build time. All coarray data must be *registered* to the communication library to be referenced or defined via the communication library.

MPI-3 can be selected for all platform on which MPI-3 is implemented. Coarrays are registered and deregistered at the start and end point of the MPI window. Coarrays are performed one-sided communication by `MPI_Put` and `MPI_Get`, and synchronized by `MPI_Win_fence`. Implementation on MPI incurs certain costs for dynamic allocation of coarrays and waiting for communication completion.

GASNet can be selected for more advanced implementation over InfiniBand. Since allocation and registration of are inseparable and can be done only once on GASNet, the implementation allocates and registers a huge pool at the startup of program execution to contain all coarray variables. The XMP runtime should allocate and deallocate coarrays not using the Fortran library but using the memory manager made for the pool.

FJ-RDMA can be selected for the implementation over Tofu interconnect of the K computer and Fujitsu PRIMEHPC FX10 and FX100. Basically, each coarray is allocated by the Fortran library and registered with `FJMPI_Rdma_reg_mem`. And it is deregistered with `FJMPI_Rdma_dereg_mem` before deallocated by the Fortran library. One-sided communication is performed with `FJMPI_Rdma_put` and `FJMPI_Rdma_get`, which include confirmation of communication completion.

4.2 Intermediate Communication Library Interface

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

下位通信層は通信ライブラリを隠蔽する。例えば `register` は
しかしあらわになるものがあってオブジェクトが通信ライブラリを意識するモードで分ける。

5 Evaluation

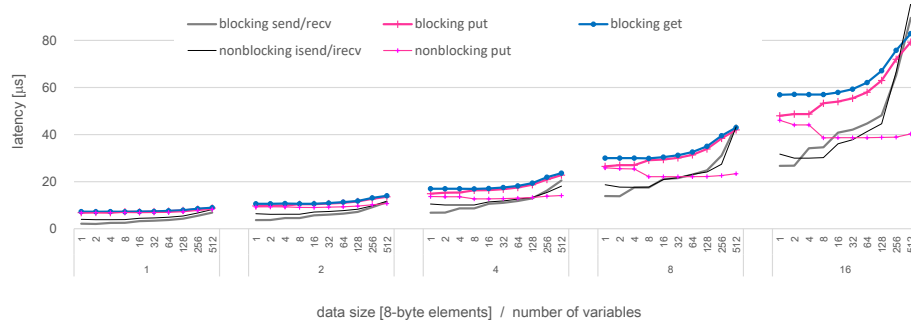
5.1 Aggregated Nonblocking Communication

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 2 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. <code>_XMP_coarray_malloc_image_info_1</code> 2. <code>_XMP_coarray_malloc_info_1</code> 3. <code>_XMP_coarray_malloc_do</code> 4. <code>_XMP_coarray_regmem_do</code> 5. <code>_XMP_coarray_lastly_deallocate</code>
片側通信	6. <code>_XMP_coarray_shortcut_get</code> 7. <code>_XMP_coarray_shortcut_put</code>
同期	8. <code>xmp_sync_all</code> 9. <code>xmp_sync_image</code> 10. <code>xmp_sync_images</code> 11. <code>xmp_sync_images_all</code> 12. <code>xmp_sync_memory</code>
atomic 通信	13. <code>_XMP_atomic_define_0</code> 14. <code>_XMP_atomic_define_1</code> 15. <code>_XMP_atomic_ref_0</code> 16. <code>_XMP_atomic_ref_1</code>
問合せ	17. <code>xmp_all_num_nodes</code>
エラー処理	18. <code>_XMP_fatal</code>

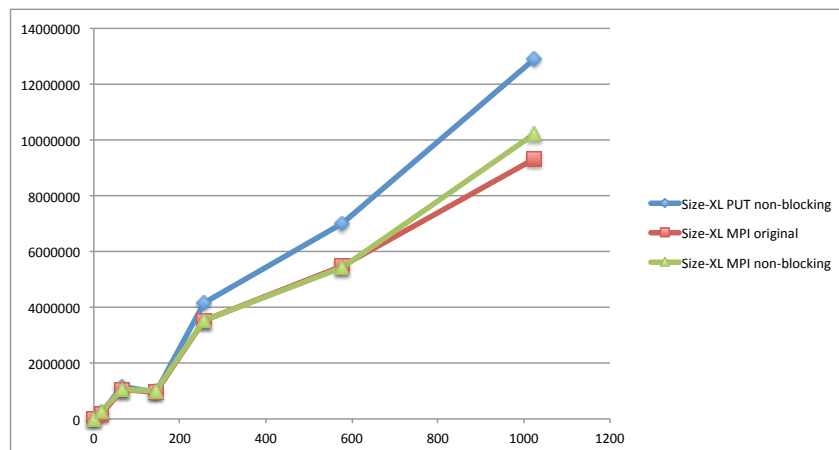


latency-16var.pdf
2 からの 4 つにして図を大きくする。

Fig. 3 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. 4 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、**l 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 specifications [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-association (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 cannot be referenced 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. Intrinsic 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

ズム

CompilerAllocationFortran 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 nigon

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