Vol.2015-HPC-150 No.3
2015/8/4

# XcalableMP トランスレータをベースとした Coarray Fortran 処理系の実装と評価

岩下英俊[†,††]　中尾昌広[†]　佐藤三久[†,†††]

XcalableMP（XMP）はベース言語（Fortran と C）にディレクティブ拡張を施した並列言語である．この処理系である Omni XMP の主要な部分は，トランスレータ方式で実装されていて，入力プログラムからベース言語へのソース-to-ソース変換を行っている．Omni XMP をベースとして，Coarray Fortran 1.0 仕様の主な機能を実装し評価したので報告する．トランスレータ方式はバックエンドコンパイラに依存しないポータビリティが利点だが，coarray 変数に関わるメモリ割付けやデータの連続性の判定などの低水準の機能を実現するには独自の工夫が必要だった．結果として，Himeno ベンチマーク XL モデルによる評価では，8 から 8000 ノード並列の平均で MPI の 108%という高い性能が得られた．

## 1. はじめに

XcalableMP（XMP）は，分散メモリ環境向けの並列プログラミング言語である．XMP/F は Fortran，XMP/C は C をベース言語とする [1]．

XMP は, グローバルビューおよびローカルビューと呼ぶ二つのメモリ抽象化モデルを持つ．グローバルビューで扱うデータや計算ループは，ノードを跨いで大域的なインデックス空間をもつ．プログラマはそのようなデータのノードへの分散配置を指示し，データ分散に整合するようにループの分散を指示し，また，データ間の通信を指示することができる．一方，ローカルビューでは，インデックス空間は各ノードに固有であるので，大域的な名前を宣言された変数は，インデックス（配列添字）とノード番号の組でデータを特定する．

グローバルビューのための書式は，主にベース言語に付加された指示文（directive）の形で構成されている．ローカルビューの書式は，XMP/F では Coarray Fortran（CAF）1.0 仕様 [2]が XMP の文脈の中に取り込まれている. XMP では，CAF 文法の用語である**イメージ**を，XMP 文法で定義されるノードに 1 対 1 に対応付けている．そのため，XMP における coarray は，ノードを跨いでアクセス可能なスカラ変数や配列変数となる．これと同様な書式と考え方で，XMP/C 文法にも Coarray 機能が組み込まれている．

我々は，XMP/F のローカルビューとして CAF 1.0 仕様の主要部分を Omni XMP に実装したのでここに報告する．Omni XMP コンパイラは，XMP 仕様のフル実装を目指して開発が続けられている．その中心部分は，グローバルビューの指示文をベース言語の表現に変換するためのトランスレータである．グローバルビューとの連携機能や同期・例外処理などの実現を考えると，CAF の少なくとも一部の機能についてはトランスレータの中に一緒に作り込む必要があった. 今回は, Fortran 処理系に手を加えることなく, CAF 1.0 仕様を Omni XMP のトランスレータとランタイムライブラリだけで実現することを試みた.

CAF をトランスレータで実装することには以下のような困難があった．一つは，coarray のメモリ割付けや自動 deallocation などのメモリ管理の実現であり，もう一つは，通信を高速化するためのデータの連続性の判定であった．トランスレータ方式では，こういった一見コンパイラ依存となりそうな低水準の機能を, 標準的な Fortran コードによる表現と，Fortran から呼び出せるランタイムライブラリによって実現しなければならない．しかしトランスレータ方式で実装できれば，ベース言語コンパイラに手を入れる必要がなくなり, プラットフォームに依存しない CAF コンパイラを実現することができる.

以降，2 章では，Fortran に関わる前提知識について述べているので，必要に応じ参照していただく. 3 章では，CAF コンパイラの設計を示し，4 章では現在の実装を紹介している．5 章では評価を行い，6 章では関連研究を紹介し，7 章をまとめとする.

## 2. Fortran に関する前提

### 2.1 配列の形状

Fortran 言語仕様の大きな特徴の一つは，多次元配列の直接的な表現をもつことである．Fortran においては，すべての定数や式，部分式はスカラまたは配列であり，0 以上の次元数をもつ．n 次元の配列変数の部分実体は，スカラまたは k（≤n）次元の配列である．

配列の形状（各次元の下限と上限）は一般に実行時に決まる．例えば，形状引継ぎ配列 A を仮引数とするサブルー

† 理化学研究所 計算科学研究機構
†† 富士通(株) 次世代テクニカルコンピューティング開発本部
††† 筑波大学大学院 システム情報工学研究科

ⓒ2015 Information Processing Society of Japan

チン

```
subroutine sub(A)
real A(:, :)
…
```

があるとき， A の形状は sub の呼出し時に決まり，A が有効である間，Fortran 処理系の管理下にある．例えば

```
real B(10, 20, 30),
call sub( B(:, 3, 1:m) )
```

と呼び出された場合には，A の 1 次元目の下限は 1，上限は 10，2 次元目の下限は 1，上限は m の呼出し時の値となる．また，allocatable 配列においては，領域が割付けられるときに，その先頭アドレスと全体サイズだけでなく，形状が決定する．

これらの形状の情報について，Fortran 処理系以外の手段で決定して Fortran 処理系に伝える標準的な方法はない．

### 2.2 配列データの連続性

m 次元の配列変数

A( $L_1 : U_1$, $L_2 : U_2$, $\cdots$, $L_m : U_m$ )

があるとき，その部分配列

A( $S_1$, $S_2$, $\cdots$, $S_m$ )，ただし $S_i$ $(1 \le i \le m)$ は
スカラ，または，添字三つ組 $b_i : e_i : s_i$

について，以下のことがいえる．

- 部分配列の次元数は，部分配列のもつ添字三つ組の数であり，0 以上 m 以下である．
- 部分配列の 1 次元目から k 次元目までの添字がすべて添字三つ組であり，かつ

$$s_1 = 1, \quad b_1 = L_1, \quad e_1 = U_1,$$
$$\cdots \quad \cdots \quad \cdots$$
$$s_{k-1} = 1, \quad b_{k-1} = L_{k-1}, \quad e_{k-1} = U_{k-1}$$

かつ

$$s_k = 1$$

であるとき，部分配列の配列要素は k 次元目まで連続している．

配列要素が k 次元目まで連続していることは，データが次元を k 次元目まで隙間なく連続しているための必要条件だが，十分条件ではない．配列変数 A が形状引継ぎ配列やポインタ配列である場合には，他のデータ実体の部分実体かもしれないので，データの連続性は実行時にしか判断できないことがある．

## 3. 設計

### 3.1 通信層の選択

**図 1** にユーザプログラム実行時のソフトウェアの階層を示す. Omni XMP コンパイラが生成するオブジェクトは，独自のランタイムライブラリを呼び出し，この層で通信ライブラリの違いを吸収する．グローバルビューに関わるノード間通信は，MPI の両側通信と集団通信で実装されている．ここにローカルビューの実装を加えた．CAF では主として片側通信を用いるが，MPI-2 では機能的に不十分であるため，片側通信に強くかつ MPI と併用できる通信ライブラリを採用する．これまでの研究 [3]により，ここには GASNet を選択した．京コンピュータと富士通 PRIMEHPC FX10 等のためには，Tofu ネットワーク専用の RDMA 通信機構を使用した．

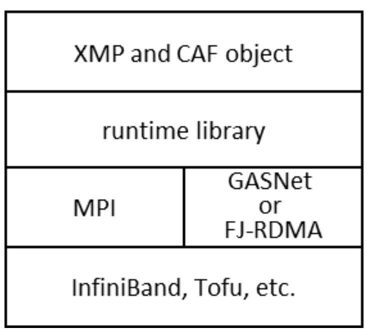



図 1　ソフトウェア階層

### 3.2 通信ライブラリによる coarray の領域獲得

GASNet で通信するデータは GASNet ライブラリによって割付けられなければならないため，coarray 変数の割付けは，静的か allocatable かに関わらず，ランタイムライブラリの呼出しで行う必要がある．Fortran には，標準ではないがデファクトとなっている Cray ポインタと呼ばれる機能があり，これを使うと処理系の外で割付けたアドレスを Fortran 変数のアドレスとして扱うことができる．この機能を使って，coarray 配列を GASNet ライブラリで割付けて，しかしその形状は Fortran ランタイムの管理下に置く方法を設計し，プロトタイピングにより検証した．

静的な coarray 配列に対するプログラム変換の原理を**図 2** に示す．入力プログラムで 2 次元配列として宣言された coarray V1 は，Fortran90 コンパイラが受け入れられる通常の配列変数に変換される（2 行目）．ただし，Cray POINTER 文（3 行目）によって Cray ポインタ cp_V1 と関連付けられているので, 通常の Fortran 処理系のメモリ割付け対象から除外される．cp_V1 は COMMON 結合（4 行目, 10 行目）により，サブルーチン FOO に対して生成される初期化手続 xmpf_init_FOO と共有されている．xmpf_init_FOO の中では，V1 のための領域 $8 \times 10 \times 20$ バイトが通信ライブラリを通じて確保され，cp_V1 に戻される．領域確保を FOO とは独立な手続として生成したのは，FOO の毎回の呼出し時のオーバヘッドコストを回避するためである. 後述のように，手続 xmpf_init_FOO は，ユーザプログラムの実行開始前に１度だけ呼び出されるように制御される．

ⓒ2015 Information Processing Society of Japan

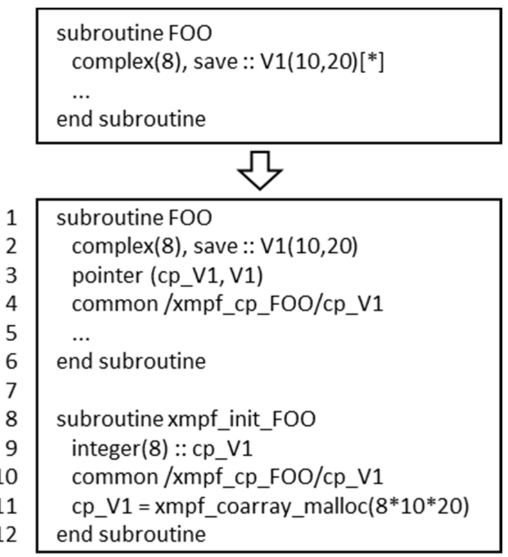



図 2 static coarray の宣言の変換の原理

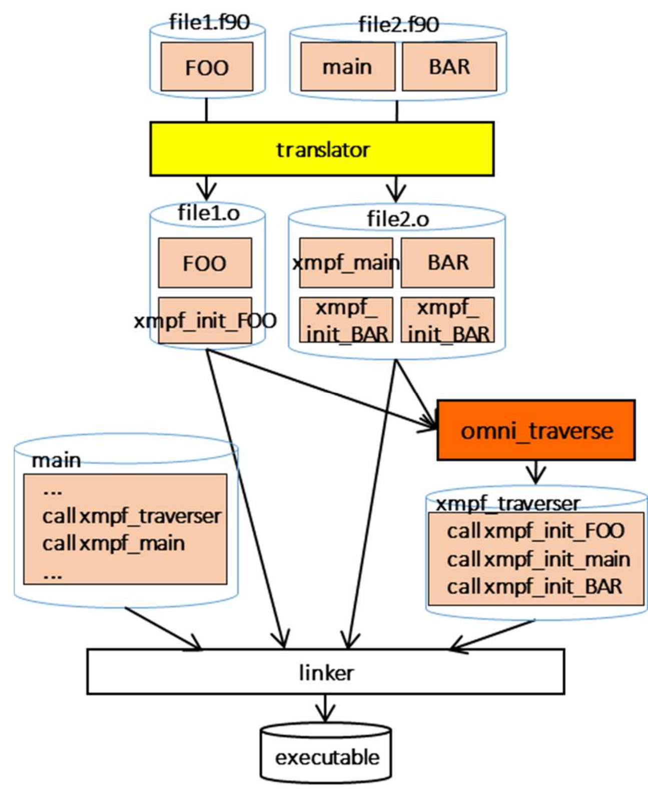



図 3 static coarray の初期化を含むコンパイラの流れ

### 3.3 Static coarray の効率的な割付け

仕様では，coarray の実体（仮引数でない）は，SAVE 属性をもつ（static である）か，ALLOCATABLE 属性をもつ（明示的な割付けと解放が可能）かのいずれかに限られる．以降，前者を static coarray，後者を allocatable coarray と呼ぶ．

通信に GASNet を用いる場合には，static coarray についても GASNet ライブラリを通じた動的な割付けが必要となる．通信に Tofu ライブラリの RDMA を用いる場合には，割り付けは Fortran 処理系に行わせてもよいが，割付けたアドレスを Tofu ライブラリに登録する必要がある．これらの処理（以降，初期化と呼ぶ）を，static coarray が最初に参照されるときに行う方法も考えられるが，以下の理由により，実行開始時にすべての static coarray に対して一度に行うべきであると考えた．

1. 初期化は，ノード間で互いに完了を知らせ合うなどの同期処理を含むため，大規模並列実行ではコストが大きい．性能のためには，実行部から追い出す方がよい．
2. GASNet の場合，static coarray の出現毎に割付けを行うと，allocatable coarray の割付けと混ざり合ってフラグメンテーションが起こりやすい．static coarray は実行開始時に一度にまとめて割付けて，実行終了まで解放しないのがよい．
3. Tofu ライブラリの場合，登録する領域の数に制限がある．同時に使用できる allocatable coarray の数を多くするために，すべての static coarray を連続する領域にまとめて 1 件として登録する方がよい．

図 **3** にこれを実現するコンパイラのフローを示す．入力は，CAF プログラムの複数のファイルである．3.2 節で述べたように，トランスレータはそれぞれの手続に対応する初期化手続を生成する．初期化手続の名前は，元の手続の名前の前に特定の文字列（この例では xmpf_init_）を付加したものとする．同時に，ユーザのメインルーチンの名前はトランスレータによって変更される．コンパイラドライバは，リンカを呼ぶ直前に，すべてのオブジェクトとアーカイブの中身を参照して手続 xmpf_traverser を生成するパス omni_traverse を呼び出す．xmpf_traverser の中身は，特定の文字列で始まりどこからも呼ばれていない初期化手続をすべて呼び出す Fortran プログラムである．xmpf_traverser とユーザのメインルーチンを呼び出す作り付けのメインルーチンと，xmpf_traverser と，トランスレータの出力をリンク結合することで，最終的な実行可能ファイルが生成される．

### 3.4 データの連続性判定と通信コードの生成

ノード間通信においては，通信立ち上がりのオーバヘッドの累積を削減するために，連続するデータはできるだけ一度に送受信する方が効率がよい．そのためには，データの連続する区間をコンパイル時または実行時に検出する必要がある．

2.2 節に説明したように，配列要素の連続性はコンパイル時または実行時に判定できるが，必要なのは配列データが実際にメモリ上で連続しているか否かである．この判断は，隣接する配列要素のアドレスを比較することによって行う．配列 A の部分配列を参照する例を図 **4** に示す．配列要素

ⓒ2015 Information Processing Society of Japan

```
real(4), save :: A(N1, N2)[*]

... A(i1b:i1e:i1s, i2b:i2e:i2s)[k] ...
```

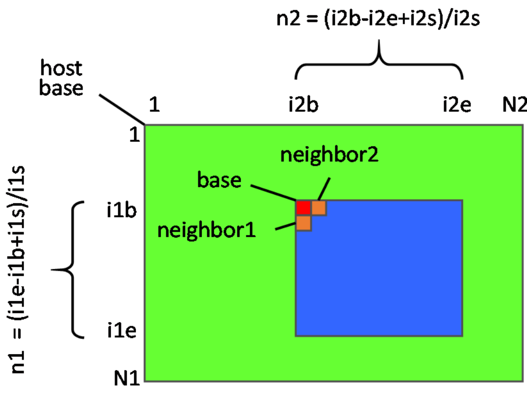



図 4　連続性の判定

の連続性は，部分配列の形状 [n1, n2] と親配列の形状 [N1, N2] などから実行時に判定できる．すなわち，i1s=1 であるなら部分配列の 1 次元目は連続であり，加えて n1=N1 かつ i2s=1 であるなら，次元を跨いで 2 次元目まで連続であると判断できる．もし親配列 A が形状引継ぎ配列であるならこの判断では不十分なので，隣り合う配列要素がメモリ上で隙間なく並んでいるかどうかの判定を加える．すなわち，同図で base と示した配列要素 A(i1b, i2b) のアドレスと，naighbor1 と示した配列要素 A(i1b+i1s, i2b) のアドレスの距離が，型のバイト数 4 と一致していなければ，1 次元目は連続していない．同様に，base のアドレスと，naibor2 と示した配列要素 A(i1b, i2b+i2s) のアドレスの距離が，4×n1 と一致していなければ，2 次元目は連続していない．

## 4. 実装

### 4.1 サポート状況

2015 年 7 月 1 日現在の CAF 1.0 仕様に対するサポート状況は，**表 1** に示す通りである．表中の「章」は，文献 1 における章である．また，CAF2.0 仕様の組込み関数 co_sum と co_min, co_max について一部実装した．この他に，XMP のグローバルビューとの連携機能の開発が予定されているが，まだサポートに至っていない．

### 4.2 Coindex 付きオブジェクトの参照

coindex 付きオブジェクトの参照は，get 通信を行うライブラリ呼出しに変換する．これは一般に配列を返す関数の呼出しとなるため，ランタイムライブラリのエントリは許される型と次元数の組合せの膨大な数となる．オブジェクト生成を簡素化するため，ランタイムライブラリの入口は

表 1　CAF1.0 仕様サポート状況（2015/7/1 現在）

| 章 | 内容 | 状況 |
|---|---|---|
| **3** | codimension の宣言 | 済 |
| | coarray の初期化 | 未 |
| | allocatable 属性の宣言 | 済 |
| **4** | coindex 付きオブジェクトの参照 | 済 |
| | coindex 付き変数の定義 | 済 |
| **5** | static coarray 仮引数 | 済 |
| | allocatable coarray 仮引数 | 済 |
| **9** | coarray に対する ALLOCATE 文 | 済 |
| | coarray に対する DEALLOCATE 文 | 済 |
| | 暗黙の割付け解放 | 済 |
| **10** | 派生型 coarray | 未 |
| | 派生型 coarray の allocatable 構成要素 | 未 |
| | 派生型 coarray のポインタ構成要素 | 未 |
| | 構造体の coarray 構成要素 | 未 |
| **12** | SYNC ALL 文 | 済 |
| | SYNC IMAGES 文 | 未 |
| | LOCK 文と UNLOCK 文 | 未 |
| | CRITICAL セクション | 未 |
| | SYNC MEMORY 文 | 済 |
| | ‘stat=’指定子と‘errmsg=’指定子 | 未 |
| **13** | 正常終了 | 未 |
| | エラー終了と ERROR STOP 文 | 未 |
| **15** | 問合せ関数 image_index, lcobound,ucobound | 未 |
| | 変形関数 num_images, this_image | 済 |
| | 変形関数 this_image(coarray [,dim]) | 済 |
| | 組込サブルーチン atomic_define, atomic_ref | 未 |

Fortran で記述した wrapper とし，型に依存しない総称名で呼び出すようにした．

実装したトランスレータで，実際に coindex 付きオブジェクトの参照を変換した例を**図 5** に示す．改行やコメントなどは説明のために手で加えている．組込み関数 log の引数として参照された coindex 付きオブジェクトは，2 次元配列を返す総称名関数 xmpf_coarray_get2d の引用に変換される．ここで，第 2，第，第 9 引数に出現する loc 関数は，Fortran のデファクトの組込み関数であり，引数のアドレスを得ている．これらは図 4 における base，neighbor1 および neighbor2 のアドレスに相当する．第 3 引数は配列要素のバイト数，第 6 引数が配列の次元数，第 8 引数と第 10 引数がそれぞれ配列の 1 次元目と 2 次元目（親配列 A の 3 次元目）のサイズである．これらの情報を受け取ったランタイムライブラリは，できるだけ長いデータの連続性を検出し，連続な区間を単位とする get 通信を反復実行する．

### 4.3 Coindex 付き変数の定義

coindex 付き変数の定義は，put 通信を行うサブルーチンの呼出しに変換する．coindex 付きオブジェクトの参照は数式の中で起こるため，プログラム中のいたるところに出現し得るのに対し，coindex 付き変数は，代入文の左辺の形でしか現れることができない．手続の引数として coindex 付き変数を記述して値を書き戻すことは文法で許されていない．

```
real(8),save:: a(10,20,30)[*]
real(8) :: z(10,15)
```

```
z = log( a(:,7,1:15)[k] )
```

```
z = log ( xmpf_coarray_get2d (               &
       xmpf_descptr_a,                       & ! 1
       loc ( a ( 1, 7, 1 ) ),                & ! 2
       8,                                    & ! 3
       xmpf_coarray_get_image_index          &
            ( xmpf_descptr_a, 1, k ),        & ! 4
       2,                                    & ! 5
       loc ( a ( 2, 7, 1 ) ),                & ! 6
       10,                                   & ! 7
       loc ( a ( 1, 7, 2 ) ),                & ! 8
       15 ) )                                  ! 9
```

図 5　coindex 付きオブジェクトの参照の変換

```
real(8),save:: a(10,20,30)[*]
real(8) :: z(10,15)
```

```
a(:,7,1:15)[k] = log( z )
```

```
CALL xmpf_coarray_put_array (              &
       xmpf_descptr_a,                     & ! 1
       loc ( a ( 1 , 7 , 1 ) ),            & ! 2
       8,                                  & ! 3
       xmpf_coarray_get_image_index        &
          ( xmpf_descptr_a, 1, k ),        & ! 4
       log ( z ),                          & ! 5
       1,                                  & ! 6
       2,                                  & ! 7
       loc ( a ( 2 , 7 , 1 ) ),            & ! 8
       10,                                 & ! 9
       loc ( a ( 1 , 7 , 2 ) ),            & ! 10
       15 )                                  ! 11
```

図 6　coindex 付き変数の定義の変換

coindex 付き変数への定義を行う代入文の変換の様子を**図 6** に示す．引数の並びは Coindex 付きオブジェクトの参照と似ているが，第 5 引数として代入文の右辺式がそのまま与えられている．第 6 引数はコンパイラの解析結果をランタイムライブラリに伝えるためのフィールドであり，ここでは右辺式の値が Tofu の RDMA 通信でアクセス可能でない領域に置かれていることを示している．

第 5 引数については改善の余地がある．このライブラリサブルーチンは Fortran77 インタフェースで呼び出されるため，第 5 引数，すなわち元の代入文の右辺式が不連続な配列であるとき，Fortran 処理系はデータを連続にするために copyin を選択し，大きなメモリコピーが発生することがある．不連続なデータであっても copyin を回避してランタイムシステムに渡すためには，第 5 引数の型と形状を意識した Fortran90 インタフェースを用いる必要がある．

## 5. 評価

Himeno ベンチマーク 98 ソースコード [4] を CAF に移植してオリジナルの MPI 版と比較し，プログラミングの容易さと実装の性能を評価した．

### 5.1 CAF プログラミング

MPI 版の Himeno ベンチマーク himenoBMTxpr.f90 を元に，**表 2** に示す 4 つの CAF プログラムを作成した．caf-wide/narrow/fit では，袖通信のためのバッファ領域だけを coarray として宣言している．caf-direct では，主変数 p の全体が coarray となる．

プログラム行数を比較すると，オリジナルの MPI 版に比べて移植後の CAF 版の方が 30%以上減少している．この主な要因は，袖通信の複雑な通信パターンを記述するための表現力の違いである．MPI 版ではデータの連続区間の長さとストライド幅を type vector で再帰的に書き下した上で mpi_isend/irecv の引数に与えなえればならないが，CAF では簡潔な配列代入文だけで同等以上のことが表現できる．

表 2　比較評価するプログラム

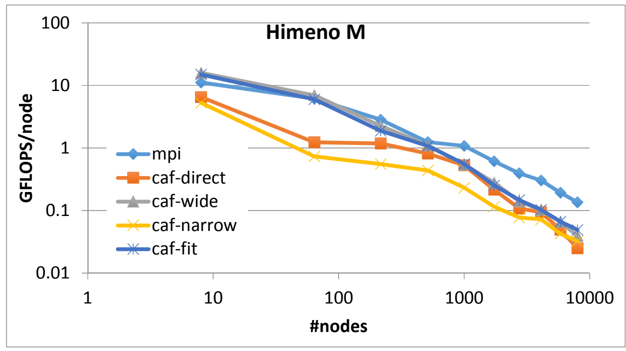
| 呼称 | プログラムの特徴 | 行数 |
|---|---|---|
| **mpi** | MPI 版オリジナル．下の 4 つはここからの移植． | 610 |
| **caf-wide** | 主変数とインデックスが揃うバッファ領域を確保．通信の連続性を重視するため余計な部分の通信を含む． | 415 |
| **caf-narrow** | バッファの形状は caf-wide と同じ．通信量の最小化を重視するため通信区間が不連続になることも． | 415 |
| **caf-fit** | 通信量は caf-narrow と同じ．バッファ領域の形状を最小化することで通信を連続にした． | 415 |
| **caf-direct** | 袖通信 3 方向のうち 1 次元以上の連続性のある 2 方向はバッファを介さない通信にした． | 402 |

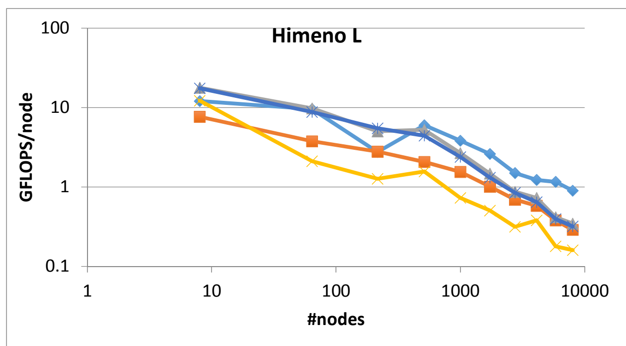
（行数はコメント行と空行を除く）

### 5.2 京コンピュータでの性能評価

京コンピュータ上で RDMA 通信を用いた場合の性能を MPI 版と比較した（**図 7**）．翻訳時オプションには，共通して -Kfast,parallel,reduction（スレッド自動並列化を含む高速実行）を用いた．測定は，M, L, XL の 3 サイズについて，ストロングスケーリングで 2 × 2 × 2，4 × 4 × 4，…，20 × 20 × 20 ノード並列に対して行った．この結果から以下のことが言える．

ⓒ2015 Information Processing Society of Japan



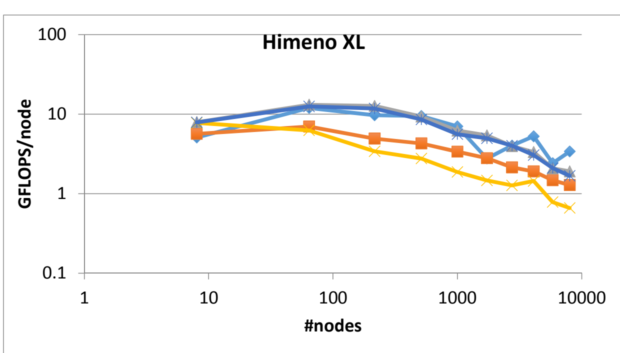




図 7　京コンピュータでのノード当り性能の比較

- CAF 版の中で最も性能が高かった caf-wide は，8 ノードから 8000 ノードの計測区間全体で平均すると，M モデルで MPI の 65%，L モデルで 84%，XL モデルで MPI を超える 108%の性能を達成した．
- 通信の粒度が小さいときの CAF 版の性能には，まだ改善の余地がある．
- CAF は，通信バッファの形状の調整など，プログラミングの少しの違いで性能が大きく違う．MPI に比べて性能チューニングの裁量の幅が大きいと言える．
- MPI 版の性能に波があるのに対して，caf-wide と caf-fit の性能は安定している．これは MPI 版以上に細かな指示を直接書き下しているためと考えられる．MPI ではデータ量によって戦略をスイッチするなどの最適化が MPI ライブラリの内部で行われている．

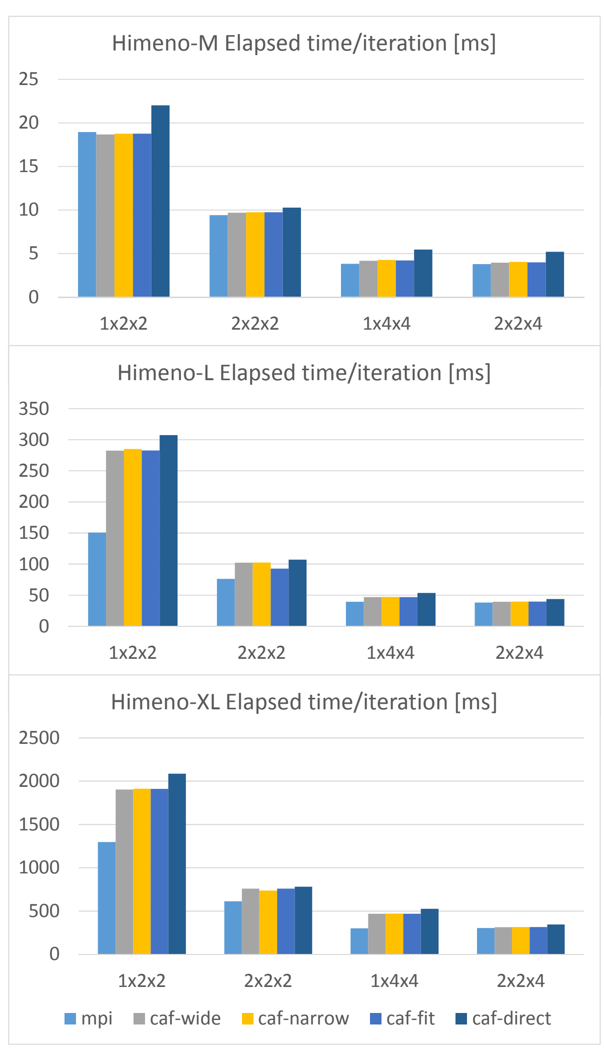



図 8　GASNet での実行時間の比較

### 5.3 HA-PACS での性能評価

筑波大学の HA-PACS 上で，MPI 版と CAF の 4 つの版を比較した．通信層には IBV-conduit でビルドした GASNet と MVAPICH2 を使い，gfortran を使った．実行時間の比較を**図 8** に示す．この結果から次のことがいえる．

- 性能で 3 つのグループに分かれる．L モデルと XL モデルでは MPI が最も早い．どのモデルでも caf-direct が最も遅い．CAF の他の 3 本はその中間である．
- 計測区間の通信をすべてコメントアウトして実行してみると，速度向上はどのプログラムでも数パーセント以下であった．つまり，MPI と CAF の速度比は，通信以外の計算実行部分で生じている．
- 通信部分を除外すると，MPI と CAF のプログラムの違いは，coarray に関する宣言と allocation の有無くらいである．
- これらのことから，MPI と CAF の性能比は，GASNet ライブラリによるメモリ割付けに起因する何らかの

オーバヘッドコストがあるためと考えられる．推測だが，メモリの過剰な pin-down が計算部分の性能を低下させたのかもしれない．caf-direct の性能がさらに低い理由も，大きな変数 p を pin-down させるためと考えると説明がつく．

### 5.4 HA-PACS での他の実装との比較

HA-PACS 上で，我々の Omni XMP コンパイラと UH-CAF と Intel コンパイラで，同じ CAF プログラムに対する性能を比較した．UH-CAF は，Houston 大学からフリーで公開されている UH Compiler の一部である [5]．XMP コンパイラと同じ GASNet と MVAPICH2 と gfortran の環境でビルドした．一方，Intel Fortran は CAF 機能を含んでいて，分散メモリ環境では Intel MPI の上で動作する．

結果は図 **9** の通りである．Omni-XMP と UH-CAF はおおむね同じような傾向を示している．1 次元目方向に分割がある 2×2×2と 2×2×4ノードのときに Omni-XMP はやや劣るので，小粒度の通信に対して性能改善の余地があると考えられる．Intel の性能は良いとは言えない．

## 6. 関連研究

Coarray Fortran は Robert Numrich と John Reid によって開発された．CAF1.0 仕様は最新の Fortran 規格である Fortran2008 の一部となっている．ベンダの中で最も開発が先行しているのは Cray であり，フリーのコンパイラとしては Rice 大学 [6]と Houston 大学のものがよく知られている．

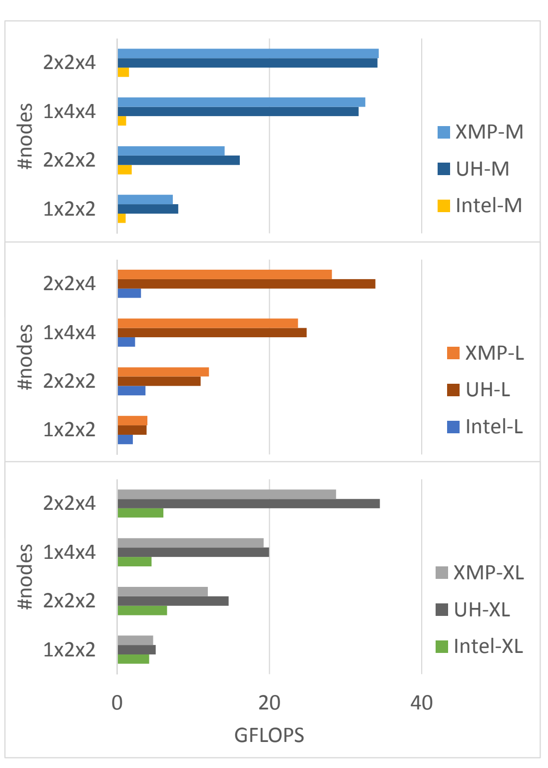



図 9 Omni-XMP，Intel，UH の CAF の性能比較

CAF1.0 仕様は完全にホモな SPMD 動作を前提とした仕様である．例えば，coarray の割付けでは，全イメージが同時に同じサイズを割付けなければならない．XMP は CAF のイメージインデックス空間を部分ノード集合から成るタスクにマッピングすることで，CAF プログラムのタスク並列実行を可能にしている．別のアプローチとして，Rice 大は CAF のイメージ空間を MPI_Comm_split のように分割する方法を CAF2.0 仕様で提案している．

CAF の実装のためには，小粒度のデータ移動に対して高速な片側通信の機能をもつ通信ライブラリ層が必要とされる．MPI2 ではこの目的には不十分と言われていて，ARMCI と GASNet がよく使われている．CAF を実装するために MPI2 に欠けていた機能は，MPI3 仕様で改善されており，今後 CAF への適用が進むと考えられる．

XMP/C は，CAF と同様な機能を C 言語をベースにして構築している．C++のテンプレート機能を使って C++に Coarray に似た機能を付加するという試みもある [7]．

## 7. まとめ

XMP 言語仕様の一部として，CAF1.0 の主要な機能を Omni-XMP コンパイラ上に実装した．Himeno ベンチマークで評価した結果，MPI よりも 30%も短いプログラムで，XL モデルでは 8 ノードから 8000 ノード並列の平均で MPI を 8%上回る性能を達成した．

CAF では Fortarn90 の配列記述を利用して通信が記述できる．性能チューニングの手段としては，むしろ MPI よりもきめ細かな記述ができる場合がある．

**謝辞** 本研究成果の一部は，筑波大学計算科学研究センターの学際共同利用プロジェクト「アクセラレータクラスタにおける高生産言語 XcalableACC の開発と評価」を利用して得られたものである．

### 参考文献

1) XcalableMP Specification Working Group: XcalableMP Language Specification Version 1.2.1, November, 2014. http://www.xcalablemp.org/
2) John Reid: Coarrays in the next Fortran Standard, ISO/IEC JTC1/SC22/WG5 N1824, April 21, 2010
3) 中尾昌広，Tran Minh Tuan，李珍泌，朴泰祐，佐藤三久: PGAS 言語 XcalableMP における coarray 機能の実装と評価，情報処理学会論文誌，Vol.41, No.6, pp1234-1242 (2000)
4) 姫野ベンチマーク 98 ソースコード http://accc.riken.jp/2460.htm
5) University of Houston: Open-source UH Compiler http://web.cs.uh.edu/~openuh/index.shtml
6) Rice University: CO ARRAY FORTRAN 2.0 http://caf.rice.edu/
7) Troy A. Johnson: Coarray C++. Proceedings of the 7th International Conference on PGAS Programming Models. October, 2013.

ⓒ2015 Information Processing Society of Japan

## 正誤表

| | 誤 | 正 |
|---|---|---|
| **p.3**<br>図 **3** の一部 | file2.o<br>xmpf_main BAR<br>xmpf_ init_BAR xmpf_ init_BAR | file2.o<br>xmpf_main BAR<br>xmpf_ init_main xmpf_ init_BAR |
| **p.7**<br>参考文献 **3)** | 3) 中尾昌広， Tran Minh Tuan， 李珍泌， 朴泰祐， 佐藤三久: PGAS 言語 XcalableMP における coarray 機能の実装と評価， 情報処理学会論文誌， Vol.41, No.6, pp1234-1242 (2000) | 3) 中尾昌広， Tran Minh Tuan， 李珍泌， 朴泰祐， 佐藤三久: PGAS 言語 XcalableMP における coarray 機能の実装と評価， 先進的計算基盤システムシンポジウム論文集,2012, pp. 289-297 (2012) |