Omni XcalableMP V1.0 における Coarray 機能
—ランタイムライブラリ編—

岩下英俊[†1]　中尾昌広[†1]　村井均[†1]　佐藤三久[†1]

概要：Omni XcalableMP (XMP)コンパイラ 1.0 版は 5 月にリリースされた．この中に含まれる Coarray Fortran 機能の実装について，ランタイムライブラリを中心に紹介する．動的なメモリ管理には，Fortran の実行に適した方法を採用した．ランタイムライブラリの入口に抽象度の高い Fortran インタフェースを使うことで，スマートな実装で実行効率を上げた．ソース-to-ソース変換の生成コードが後続の Fortran コンパイラの最適化を妨げないよう配慮することで，実行効率の改善が進められている．

キーワード：Fortran，Coarray，コンパイラ，NAS Parallel ベンチマーク，MPI

1. はじめに

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[3]と，Rice 大学の ROSE コンパイラを開発基盤とした caft[4]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，最近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 機能の提供を Fortran2003 の完全な実装よりも優先させた．

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．

本稿は，Omni XMP の coarray 機能の実装を紹介する．この後に続く 2 章では，CAF 言語仕様の特徴と，それに伴う実装上の課題を紹介する．3 章が本稿の中心であり，課題を解決するために我々がどのようにランタイムライブラリを実装したかを述べる．4 章で性能評価を行い，5 章で今後の課題を述べ，6 章で関連研究を紹介して，7 章をまとめとする．

2. CAF とその実装上のポイント

Omni XMP で実装した coarray 機能について，仕様を紹介し実装上の課題を挙げる．

Omni XMP の一部である CAF トランスレータは，ソース-to-ソース変換によって，CAF プログラムを coarray 機能を含まない Fortran プログラムに変換する．Omni XMP から呼ばれる後続の MPI/Fortran コンパイラ（以降，ネイティブコンパイラと呼ぶ．coarray 機能に対応していなくてもよい）は，それを mpiexec で実行可能なオブジェクトに変換する．なお，Omni XMP は CAF 仕様に準じた C 向けの coarray 機能もサポートしている．

2.1 Coarray 変数

CAF の並列実行の主体は「イメージ」と呼ばれる．CAF のプログラムはすべてのイメージによって SPMD 実行される．coarray であると宣言された変数（スカラでも配列でもよい）は，すべてのイメージから互いに参照・定義することが許される．XMP 仕様ではイメージを分散メモリ環境の各ノードに対応付ける．

coarray 変数には静的な coarray と割付け（allocatable）coarray がある．静的な coarray はプログラマからは最初から存在するデータ領域に見えるが，後述のように実装上は通信ライブラリによる動的な割付けで獲得されることがある．その場合，ネイティブコンパイラからはポインタ変数に見えるので，性能を落とさない工夫が必要になる．

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．CAF の仕様上の制約として，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．それにより処理系は coarray のメモリ配置をイメージ間で対称に保つことができ，処理の高速化と作業領域の削減になる．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

†1 理化学研究所 計算科学研究機構

ⓒ2016 Information Processing Society of Japan

同名の変数の情報や状態を参照すればよいので，通信を起こして他イメージの情報を受け取る必要も，大きなテーブルに保管しておく必要もない．

Fortran 仕様では，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．coarray 変数についてもこの機能が要求されている．以降これを自動 deallocation と呼ぶ．

2.2 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．[ ]で囲まれた式は coindex と呼ばれ，一般に多次元であってもよい．

この参照によって起こる通信は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 Put 通信

以下のような，[ ]の付いた左辺をもつ組込み代入文は，イメージ k のもつ a への自ノードの式 expr の値の代入，すなわち Put 通信を意味する．

a [i] = expr

Get 通信と同様，a は配列名にも部分配列にもなれるので，Put 通信でも次元を跨いだ連続性の抽出が必要である．また右辺 expr には任意のスカラ式または配列式が許されるため，右辺（データ実体，または，評価した値が一時的に格納されている領域）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．

2.4 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．

2.5 形状引継ぎ配列

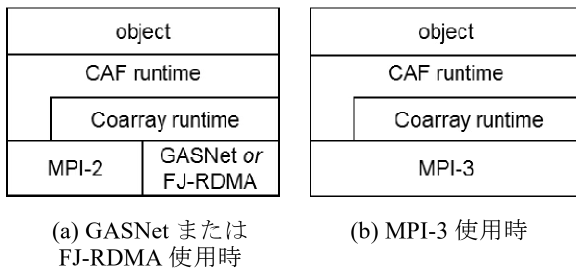
Fortran は 2 通りの手続呼出しインタフェースを文脈によって使い分けている．従来のインタフェースは単純なアドレス渡しであるが，Fortran90 から導入されたインタフェースでは，実引数の次元数と形状，割付け状態，省略可能引数が省略されたかどうかなどの情報を呼出し側からサブプログラムに引き継ぐことができる．特に形状引継ぎ配列に関しては，割付け時の配列（親配列）の一部である部分配列を引数渡しすると，添字式のパターンによって copy-in/out が不要なら行わないでサブプログラムに伝え，効率よくアクセスする方法が, 多くの Fortran コンパイラで実装されている．このときに裏で受け渡される情報は dope ベクトルと呼ばれ，引数の次元数，各次元の原点からのオフセット値，各次元のサイズ，各次元の要素間のストライド幅などの情報を含んでいて，その具体的な表現は実装に依存する．

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．前者はトランスレータの実装にコストがかかり，後者は性能低下を起こすと予想される．そこで我々は，coarray 変数を同じ形状をもつ非 coarray 変数に変換することで，Fortran 処理系に dope ベクトルの扱いをそのまま任せる方法を採った．その結果生じた課題については 3.3 節で議論する．

3. ライブラリ階層

図 1 にプログラム実行時のソフトウェア階層を示す．最下層の片側通信のための通信ライブラリには 3 通りの方法を試していて，(a)は InfiniBand 向けに GASNet を使用する場合と富士通 Tofu ネットワーク向けに富士通固有の RDMA 通信を使用する場合，(b)は MPI-3 を使用する場合である．なお，XMP のグローバルビューのためのライブラリはここでは省略されている．

同図で，object はネイティブコンパイラの出力であり，mpiexec で実行される．object から呼び出される CAF ランタイム手続には，C で書かれたものと Fortran90 で書かれた



(a) GASNet または FJ-RDMA 使用時　(b) MPI-3 使用時

図 1　実行時のソフトウェア階層

ものがある．CAF ランタイムは，通信・制御のために主に Coarray ランタイム関数を使用するが,集団通信の一部では MPI ライブラリを直接呼び出す．これらの違いは Coarray ランタイムで吸収する．

3.1 通信ライブラリ

coarray 機能の実装には，片側通信と局所的な同期に強い通信層が適しているが，Fortran 2015 の集団通信．グローバルビューでは両側通信と集団通信を多用するため，通信に MPI を使用している．一方でローカルビューである XMP でグローバルビューとローカルビューの同時利用を実現するため，通信層にはプラットフォームに合わせて以下の 3 通りの方法を選択的に使用した．

(1) MPI と GASNet の併用

グローバルビューと CAF の集団通信･集団同期の実装には MPI を使用し，CAF の片側通信・局所同期の実装には GASNet を使用する方法．InfiniBand で結合された並列計算環境を対象とする．

GASNet では，対象となるデータは GASNet ライブラリが提供する gasnet_malloc を使って割付けなければならないことが CAF の実装を難しくしている．また，GASNet を ibv-conduit でビルドした場合には，MPI との同時利用についての制限に配慮した実装が必要になる．例えば，GASNet による片側通信と MPI による集団通信が続けて実行される場合には，バリア同期で区切るなど，通信区間が重なり合わないようにしなければならない．部分ノード内で片側通信と集団通信が絡み合う場合には，安定した動作を保証する方法が分かっていない．mpi-conduit で GASNet をビルドした場合には問題が起こっていないが，MPI を超える性能を狙うためには ibv-conduit を選択したい．

(2) MPI と富士通固有 RDMA の併用

グローバルビューと CAF の集団通信･集団同期の実装には MPI を使用し，CAF の片側通信･局所同期の実装には富士通の Tofu ネットワーク向け RDMA（Remote Direct Memory Access）通信ライブラリを使用する方法（FJ-RDMA)．京コンピュータと富士通 FX10 を対象とする．

FJ-RDMA は，GASNet のようにライブラリ内で対象データの割付けと登録を同時に行うのではなく，割付け済のデータのアドレスを与えてそれを登録することができる．ただし，登録できる領域には制限があり，テキスト領域など書き込みのできない領域は，たとえ参照のみのデータでも登録できない．実際に，Put 通信で右辺式が定数の場合と，配列式や関数呼出しの場合（結果変数の割付けは Fortran 処理系が決める）には，登録が許される領域へのバッファリングが必要であった．また，対象データは 4 バイト境界にアラインしていなければならないことと，対象データの数は 210 に限られていることにも注意が必要である．

(3) MPI-3 の使用

MPI-3 は MPI-2 よりも片側通信機能が改善されているので，別の片側通信用ライブラリを使うことなく実装できている．MPI は GASNet などの低レベルな通信ライブラリに比べて使いやすいインタフェースをもつが，内部でどのような通信やバッファリングが起こるか分からないので，性能が予測しにくい．

3.2 Coarray ランタイム

通信ライブラリ層の違いをできる限り吸収するための層である．CAF と Coarray/C の両方で利用する．上位層の CAF ランタイムから現在使用しているインタフェースを表 1 に示す．

表 1　使用した Coarray ランタイムライブラリ関数
（未サポート機能に関連するものを除く）

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

(1) 割付け・解放と登録

表 1 の 1,2 によってサイズなどを設定した後，3 を使用して coarray 変数の領域を割付け，同時に通信ライブラリに登録して，片側通信が可能な状態にする．3 は割付けたアドレスと，coarray の記述子を返す．通信ライブラリからの要請により，全ノードによる集団実行を前提とする．FJ-

RDMA の場合には, Fortran システムによって既に割付けられている coarray 変数について，1,2 の後に 4 を使って通信ライブラリに登録することで片側通信を可能にすることもできる．4 は割付けアドレスを入力とし coarray の記述子を返す．MPI-3 の場合にも同様に 4 を使うことができると考えるが，未実装である．

3 で割付けられた領域は 5 で解放できるが，複数の割付けがある場合，割付けと逆の順序で解放しなければならないという制約がある．この制約は DEALLOCATE 文を使わず手続出口での自動的な解放だけを利用するプログラムには自然であるが，DEALOCATE 文を使用するか，save 属性付きの（自動的に解放されない）coarray を使用する場合には不都合なので，上位層の CAF ランタイムで解決する．

(2) 片側通信

表 1 の 6 と 7 はそれぞれ Get と Put の通信を行う．どちらも，通信先のイメージ番号，通信先データの記述子とベースアドレスからのオフセット値，自イメージデータの記述子とベースアドレスからのオフセット値，および通信量を引数とし，連続データから連続データへの通信に絞って高速性を保証するインタフェースである．このインタフェースは，自イメージのデータについても記述子が必要である点が特徴で, Coarray ランタイムは記述子を必要としないインタフェースも用意しているが，3.3 節(3)の考察の結果，現在の実装ではこのインタフェースだけを使用している．

(3) その他

表 1 の 8 から 12 までは，SYNC ALL 文，SYNC IMAGES 文，SYNC MEMORY 文にそれぞれ対応して準備されたインタフェースである．13 から 16 までは，組込みサブルーチン ATOMIC_DEFINE と ATOMIC_REF に対応し，対象データが自イメージの場合と他イメージの場合のために用意されている．17 は XMP グローバルビューの全ノードの数を問い合わせる関数であるが，XMP 仕様では CAF のイメージはノードに対応するので，全イメージの数の問合せに流用している．19 は XMP システムの共通のエラーハンドリングのインタフェースである．

3.3 CAF ランタイム

CAF ランタイムライブラリは Coarray ランタイムの上位層に位置し，CAF トランスレータが生成する Fortran コードから呼ばれる． CAF ランタイム手続は C または Fortran で書かれた関数またはサブルーチンである．

(1) 概観

現在使用しているライブラリ手続を表 2 に挙げる．オブジェクトとの間で形状引継ぎ配列や省略可能引数を使用するものは Fortran で書かれている. 加えて, 名前に ”_generic” と付いているものは，総称名呼出しを利用している．

同表の 1 から 3 は，ユーザプログラムの実行開始前に実施される静的な coarray の割付けと登録に使われる．4 と 5 は割付け coarray の自動 deallocation のために手続の入口と出口で呼び出される．

静的な coarray の割付けと登録を行う 6 は，初期化ルーチン 3 からコールバックされる．Fortran システムで割付けた領域を登録する場合には 7 を使うが，これは通信ライブラリに FJ-RDMA を使うときだけサポートしている．8，9 と 10 は割付け coarray に対してそれぞれ，割付けと登録，登録，解放に使う．割付け coarray のメモリ管理は(2)で詳しく説明する．11 と 12 は 6 から 10 に対して補助的に用いられ，デバッグ用の情報などをランタイムに伝える．

13 については(3)で詳しく説明する．

以降，14 から 29 は，CAF の Get 通信と Put 通信，同期文，および組込み手続にそれぞれ対応する．30 と 31 は，XMP 指示文と同時利用するための機能である．

表 2 使用した CAF ランタイムライブラリ手続
（未サポート機能に関連するものを除く）

分類	手続
初期化	1. xmpf_traverse_countcoarray 2. xmpf_coarray_count_size 3. xmpf_traverse_initcoarray 4. xmpf_coarray_prolog 5. xmpf_coarray_epilog
割付け・解放と登録	6. xmpf_coarray_alloc_static 7. xmpf_coarray_regmem_static 8. xmpf_coarray_malloc_generic 9. xmpf_coarray_regmem_generic 10. xmpf_coarray_dealloc_generic 11. xmpf_coarray_set_coshape 12. xmpf_coarray_set_varname
記述子問合せ	13. xmpf_coarray_find_descptr
片側通信	14. xmpf_coarray_get_generic 15. xmpf_coarray_put_generic
同期	16. xmpf_sync_all 17. xmpf_sync_all_auto 18. xmpf_sync_images 19. xmpf_sync_memory
組込み手続：集団通信	20. xmpf_co_broadcast_generic 21. xmpf_co_sum_generic 22. xmpf_co_max_generic 23. xmpf_co_min_generic
組込み手続：atomic 通信	24. xmpf_atomic_define_generic 25. xmpf_atomic_ref_generic
組込み手続 問合せ関数	26. xmpf_num_images_generic 27. xmpf_this_image_generic 28. xmpf_image_index 29. xmpf_cobound_generic
XMP 指示文連携	30. xmpf_coarray_set_nodes 31. xmpf_coarray_set_image_nodes

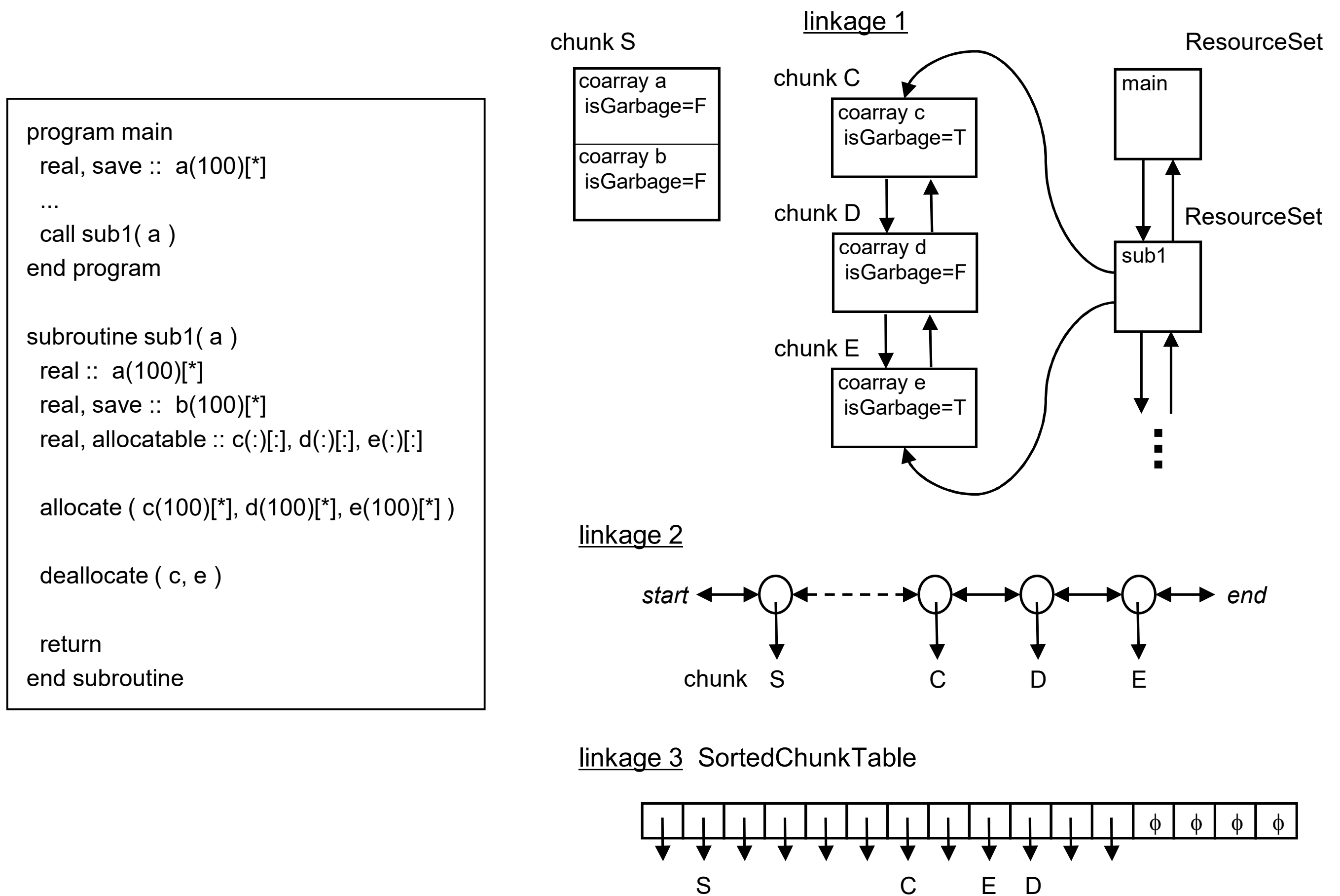


図 2 実行時のメモリ管理の様子
右の図は左のプログラム中で DEALLOCATE 文まで実行が進みガーベジコレクションが行われる前の状態.

(2) メモリ管理

FJ-RDMA では登録するメモリ領域の数に制限があるので，これを緩和するため，また，後述のように記述子の検索を速くするため，同じタイミングで割付けと解放ができる coarray は，まとめて一つの領域に割付ける．この個々の領域をチャンクと呼ぶ．チャンクには次のものがある．

- 静的な coarray を一つにまとめた領域．サイズが閾値を超える場合には複数のチャンクに分ける．
- 割付け coarray
- 通信用バッファ領域
- lock 変数，同期タグ，エラーコード収集などのための領域（現在未使用．開発中）

チャンクを管理するチャンク構造体は，以下の情報を保持する．

- orgAddr：自イメージのチャンクの先頭アドレス
- nsize：チャンクのサイズ（バイト）
- Coarray ランタイムの記述子
- isGarbage：データが有効か否か

チャンク構造体は以下の 3 つのリンケージで管理される．

1. 手続に対応する ResourceSet 構造体がもつ双方向リスト．手続出口での自動 deallocation の実現のために，手続と coarray の対応付けを保持する．
2. 割付けられた順番に沿った 1 本の双方向リスト．Coarray ランタイムのインタフェースでは，coarray の解放は，割付けの逆順に行われる必要があるので，このリストによって割付けの順序を保存する．
3. SortedChunkTable：チャンク構造体へのポインタの配列．orgAddr の昇順にソートされている．アドレスからチャンクを検索するときに用いる．（用途は次項で説明する．）

これらの構造を使ったプログラム実行時の様子を図 2 に示す．CAF トランスレータが自動生成する初期化ルーチンにより，coarray a と b は一つのチャンク S にプログラム実行開始前に割付けられる[1]．プログラム実行中に手続呼出しのネストに対応して ResourseSet 構造体が作成され，スタック状にリンクされる．手続き sub1 に局所的な割付け coarray 変数，例えば c について， ALLLOCATE 文に出会うとチャンク構造体 C が生成されて Coarray ランタイムを通して領域が確保されると同時に，上述の 1, 2 に対応して linkage 1 と 2 に追加され, SortedChunkTable に挿入される．DEALLLOCATE 文に出会うと，チャンク構造体の要素 isGarbage が真に設定される．

ⓒ2016 Information Processing Society of Japan

ガーベジコレクション（GC）は，DEALLOCATE 文の直後と，手続の出口で自動 deallocateion が実行された後に実施される．対象は，linkage 2 の末尾から順に，isGarbage が真であるすべてのチャンク構造体である．この例の DEALLOCATE 文の直後では，チャンク E のみが対象となり，Coarray ランタイムを通してチャンク領域が解放され，チャンク構造体が linkage 1,2,3 から削除される．手続 sub1 の出口では自動 deallocation によってチャンク D の isGarbage も真となるため，チャンク D の解放に続いてチャンク C も解放される．

この GC のアルゴリズムでは，複数の割付け coarray 変数が ALLOCATE 文と DEALLOCATE 文を反復して実行する場合，DEALLOCATE の順序によっては反復の度にガーベジを増やし続ける可能性がある．しかしそのようなガーベジも手続出口での自動 deallocation で完全に解放される．

(3) 記述子の検索

CAF トランスレータは， coarray 変数 v を同じ型と形状をもつ非 coarray 変数に変換し，同時に v に対する記述子 xmpf_descptr_v を生成する．CAF ランタイムは，coarray 変数の割付け・解放や Get/Put 通信などを，記述子を引数で受け取ることで実施できる．従って CAF トランスレータの生成コードには，v に対応して必ず xmpf_descptr_v が存在し，そこに正しい情報が保持されていることが原則である．しかし以下の場面では，記述子の情報を常に維持・伝播するのは困難であるかコストが大きいと分かった．

- coarray 変数の引数渡し
  記述子を引数渡しするために引数拡張する方法は，総称名や省略可能引数への対応が難しく，できたとしても実行時オーバヘッドが大きいかもしれない．COMMON 変数など他のルートで受け渡す方法では，関数呼出しの入れ子などを考慮する実現を考えると実行時オーバヘッドが大きい．

そこで，データ実体のアドレスから対応する記述子を得る find_descptr 関数（表 2 の 13）を，CAF ランタイム層に作成した．この関数は図 2 の SortedChunkTable をバイナリサーチで検索し，アドレスの値が orgAddr 以上 (orgAddr + nsize) 未満となるチャンク構造体を記述子として返す．テーブルのサイズについては，初期状態のサイズ n に対して n/2 まで使用したら n を 2 倍にし，逆に使用量が n/4 を切ったら n を半分にする．

この関数は，以下のような場面にも利用しているか利用予定であり，実行性能向上と，開発コスト削減に役立っている．

- Get/Put 通信における自イメージ側の記述子
  Put 通信では，右辺式が登録済みの片側通信可能なデータ実体である場合には，バッファリングを省略して直接通信の対象にできる．右辺式が登録済みである可能性のある場合に find_descptr 関数を使うことで，記述子を持ち回す必要がなくなる．Get 通信においても，データを受け取る側のデータ実体が登録済であると分かれば，バッファリングを省略して直接通信にできる．
- リダクション演算の引数
  文法では co_sum，co_broadcast などの通信の対象となる引数は coarray でなくてもよいが，実際には登録済みの coarray であるケースも多いと考えられる．

対象のデータ実体は登録済データの部分配列かもしれないので，1 チャンクに 1 つの coarray だけを持たせたとしても，データ実体のアドレスと orgAddr の値は一致するとは限らない．バイナリサーチではチャンクの数 n に対して log n 回の比較と分岐が起こる．静的な coarray を 1 つのチャンクにまとめることは，この検索の高速化にも役立っている．実プログラムで使われるチャンクの数が高々数十だとすれば，手続を跨いで記述子を持ち回るオーバヘッドコストとあまり変わらないと考えている．

(4) Fortran 90 インタフェースの利用

割付け，Get/Put 通信，集団通信など，配列を引数とするライブラリ手続の多くは，オブジェクトから直接呼ばれる手続は Fortran で記述し，形状引継ぎ配列（手続側では配列の各次元のサイズを抽象化する）のインタフェースを利用した．これには以下の利点がある．

- 実引数が，どの次元もスカラか添字三つ組だけで表される部分配列であるとき（例えば a(:,2:n-1, 3)），copy-in/out を起こすことなくそのまま CAF ランタイムに伝わる．各次元のサイズとストライドも CAF ランタイム側で知ることができる．CAF ランタイムはデータ実体の連続性を解析して，無駄なバッファリングを除いた通信を起こすことができる．
- 実引数が，前者以外の部分配列であったり（例えば a(ix) や a(:, f())，ix は整数型 1 次元配列変数，f は整数型 1 次元配列を返す関数），配列式であるとき，copy-in/out によって扱いやすい連続なデータとなって CAF ランタイムに伝わる．

例として，Put 通信のランタイムライブラリインタフェースを図 3 を使って説明する．このインタフェースブロックは，CAF トランスレータがサポートする 11 の型・型パラメタ（i2, i4, i8, l2, l4, l8, r4, r8, z8, z16, c）について，スカラ（0 次元）から 7 次元までの配列（XMP コンパイラの制限による）の 88 の組合せに対応する 88 のサブルーチンインタフェースを定義している．同図に示したのはその一部である．CAF トランスレータは coindex variable への代入文

```
a(i1:i2, j1:j2, 1)[k] = expr
```

を見つけると，以下のようにソース-to-ソース変換する．

```
call xmpf_coarray_put_generic
( xmpf_descptr_a, k, a (i1:i2, j1:j2, 1), expr )
```

第 1 引数は coarray 変数 a の記述子，第 2 引数はイメージ

番号である，第3と第4引数に形状引継ぎ配列が使われていて，これらの型・型パラメタと次元数によってインタフェースブロック内の個別名のサブルーチンが Fortran コンパイラによってコンパイル時に選択される．a と expr が2バイト整数型であるなら，同図の3番目のサブルーチン名 xmpf_coarray_put2d_i2 で CAF ライブラリが呼び出される．ここで第3引数 mold は，自イメージのデータであるが，イメージ k の同じ形状のデータの身代わりとして次元毎の連続性を調べるために使われ，値が参照されたり更新されたりすることはない．送信元データである第4引数についても形状引継ぎ配列であるので，まずデータの連続性を調べ，条件によってはバッファリングなしで Put 通信を行う（DMA 通信）.

従来の Put 通信のランタイムライブラリインタフェース[1][2]では，形状引継ぎ配列を使う代わりに，いくつかの配列要素のアドレスを直接 CAF ランタイムに伝えることで，連続性の判定を行っていた．同じ例なら，

a(i1, j1, 1) のアドレス
a(i1+1, j1, 1) のアドレス
a(i1, j1+1, 1) のアドレス

を伝えることで，データの連続性を実行時に判断できる．この方法では，右辺が一般の配列式のとき連続性を知ることができなかった．加えて，形状引継ぎ配列を使えば，右辺が間接参照を含むなど複雑な場合に，Fortran システムの copy-in は高速なバッファリングの意味を持つので，仮引数をそのまま Put 通信のソースに利用することができる．

```
interface xmpf_coarray_put_generic

   subroutine xmpf_coarray_put0d_i2        &
       (descptr, coindex, mold, src)
     integer(8), intent(in) :: descptr
     integer, intent(in) :: coindex
     integer(2), intent(in) :: mold
     integer(2), intent(in) :: src
   end subroutine

   subroutine xmpf_coarray_put1d_i2        &
       (descptr, coindex, mold, src)
     integer(8), intent(in) :: descptr
     integer, intent(in) :: coindex
     integer(2), intent(in) :: mold(:)
     integer(2), intent(in) :: src(:)
   end subroutine

   subroutine xmpf_coarray_put2d_i2        &
       (descptr, coindex, mold, src)
     integer(8), intent(in) :: descptr
     integer, intent(in) :: coindex
     integer(2), intent(in) :: mold(:,:)
     integer(2), intent(in) :: src(:,:)
   end subroutine
   …

 end interface
```

図 3　Put 通信のランタイム手続インタフェースの定義（一部）

4. 評価

SWoPP2015 で評価した CAF 版 Himeno ベンチマーク[2]を用いて，ランタイムライブラリの改善の効果を見る．評価に利用した計算機とソフトウェアの環境を表 3 に示す．翻訳時オプションはそのまま富士通 Fortran に伝えられ，自動最適化と自動並列化･SIMD 化を促す．今回使用した CAF 版のコードは表 4 に示す2つである．

表 3　評価環境

計算機：京コンピュータ

ノード当りコア数	8
ノード当り主記憶容量	16GB
ノード当り理論性能	128GFLOPS
プロセッサ	SPARC64 VIIIfx 2.0GHz
ノード間ネットワーク	Tofu インターコネクト
ネットワーク性能	5GB/秒×双方向×4 方向同時

コンパイラ：
Omni XcalableMP Version 1.0.2
富士通 MPI/Fortran 1.2.0，オプション –Kfast.parallel

表 4　評価に使うプログラム
（行数はコメント行と空行を除く）

呼称	特徴	行数
mpi	姫野ベンチマークの MPI 版．移植元ソース．	610
nobuf	計算ループで頻出する変数 p を coarray で宣言し，袖通信は隣接するノードの変数 p の間の put 通信で直接記述する．	412
wide	袖通信のためのバッファを coarray 変数として確保．バッファサイズはインデックス計算が単純になるように必要サイズより多めに確保している	438

4.1 スケーラビリティ

mpi と wide で台数効果を比較する．図 4 に見られるように，MPI とほぼ同じ性能が出せるようになったと言える．これはこの半年ほどの性能改善の成果である．

4.2 メモリ管理方法の改善の効果

静的な coarray と割付け coarray の両方について，Fortran システムで割付けて通信ライブラリに登録する方法の効果を検証する．従来の方法（Ver.3）では coarray 変数をソース-to-ソース変換でポインタにしてネイティブコンパイラに渡していたが，新しい方法（Ver.4）ではネイティブコンパイラの静的または allocatable 配列に変換するので，ループ

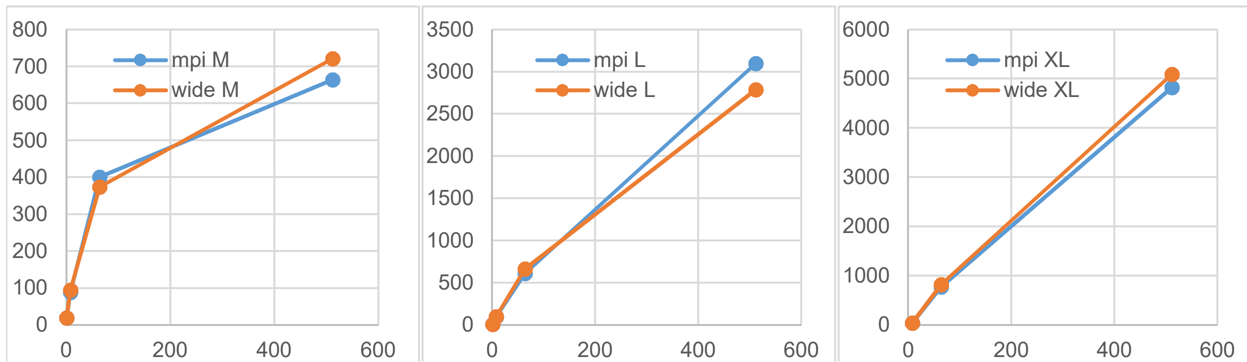


図 4 Himeno ベンチマーク（M, L, XL モデル）の CAF プログラムの性能
横軸はノード数（イメージ数），縦軸は性能で GFLOPS 単位．MPI から移植した CAF プログラムを移植元と比較．

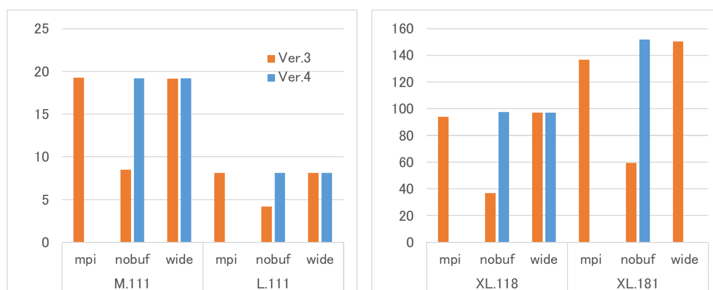


図 5 CAF コンパイラの改善

の自動並列化やプリフェッチなどの最適化を止めずに済むことが期待できる．図 5 はイメージ数（ノード数）1 と 8 のときの性能を Ver.3（赤）と Ver.4（青）で比較している．このバージョンの違いは既にハンドチューニングされている wide にはほとんど影響しないが，nobuf は Ver.4 を適用することで mpi や wide と同等の性能にまで改善された．これは計算ループである jacobi ルーチンで頻出する変数 p の変換後の状態が，ポインタから allocatable 変数に変わったためであると考えられる．

5. 今後

Omni XMP コンパイラはフリーで公開されているが，coarray 機能のうち派生型 coarray，critical section などの同期，stat 指定子など未対応の項目が残っているので，開発を継続する．性能面では，ネイティブコンパイラの性能を引き出す工夫がもっと可能であると思っている．

6. 関連研究

Rice 大学の CAF は ROSE コンパイラ基盤を利用していて，我々と同じソース-to-ソース変換方式を採用している．彼らは dope ベクトルを独自で管理している[4]．Hoeston 大学の UH-CAF は Open64 をベースとする OpenUH コンパイラに実装されている．彼らは Fortran2015 で現れる TEAM をサポートするために独自の動的なメモリ管理方法を提案している[5].

通信ライブラリにはまだ多くの選択肢があり，最新の動向を見ていく必要がある．GASNet は高速だが，MPI と同時に使うという前提では資源競合の回避が難しい．GASNet 側に集団通信の機能を取り込んでいく動きもある．

7. まとめ

トランスレータ（ソース-to-ソース変換）方式で coarray 機能の実装を行っている．開発に当って，Fortran の配列代入や形状引継ぎ配列などの仕様と，dope ベクトルなどの Fortran 固有の機能をうまく使っていけることを示した．メモリ管理については，多くの Fortran 利用者はあまり DEALLOCATE 文を使わず，割付けたきりか自動 deallocation に任せることが多いという前提で効率的な方法と考えている．

今回，ランタイムライブラリの話を中心にまとめた．トランスレータについては次の機会に報告する．

参考文献

[1] Hidetoshi Iwashita, Masahiro Nakao, Mitsuhisa Sato. *Preliminary Implementation of Coarray Fortran Translator Based on Omni XcalableMP.* Proc. of 9th International Conference on Partitioned Global Address Space Programming Models (PGAS2015), P.70-75, Washington, D.C. USA, September, 2015.

[2] 岩下英俊，中尾昌広，佐藤三久．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．第 150 回 HPC 研究会（SWoPP2015），2015 年 8 月．

[3] Deepak Eachempati, Hyoung Joon Jun and Barbara Chapman. *An Open-Source Compiler and Runtime Implementation for Coarray Fortran.* PGAS'10 Proc. of 4th Conference on PGAS Programming Models, No.13, 2010

[4] Cristian Coarfa. Portable High Performance and Scalability of Global Address Space Languages. Ph.D. Thesis, Rice University, January 2007.

[5] Shiyao Ge, Deepak Eachempati, Dounia Khaldi and Barbara Chapman, *An Evaluation of Anticipated Extensions for Fortran Coarrays,* 9th International Conference on Partitioned Global Address Space Programming Models (PGAS2015) , P47-58, Washington, D.C. USA, September, 2015.

ⓒ2016 Information Processing Society of Japan