Vol.2016-HPC-153 No.25
2016/3/2

NAS Parallel ベンチマークの CAF への移植と Omni XcalableMP CAF コンパイラを用いた評価

岩下英俊[Á1]　中尾昌広[Á1]　佐藤三久[Á1, Â2]

概要：PGAS 言語の研究開発が盛んだが，中でも Coarray Fortran（CAF）は MPI に匹敵する性能が出せる高級言語として期待されている．本稿では，NAS Parallel ベンチマークの MPI 版を題材に，MPI プログラムから CAF プログラムへの正確な移植を行う一つの方法を紹介する．そして両者の性能比較を行い，その結果を考察する．CAF プログラムの翻訳には，OMNI XcalableMP コンパイラの一部として開発中のソース to ソースのコンパイラを用いる．

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

1. はじめに

PGAS 言語の一つとして Coarray Fortran (CAF) の研究開発が盛んになっている．CAF は Fortran に少しだけの仕様拡張を施した HPC 向けの並列言語である．John Reid らが古くから提唱していたこの言語仕様は，近年になってその基本部分である CAF1.0 仕様[1]が標準の Fortran2008 の一部として取り込まれ, さらに Fortran2015 で拡張される方向にある[2].

我々は, 並列言語 XcalableMP（XMP）の処理系 Omni XMP を開発中である．XMP はその仕様の一部として CAF 1.0 仕様を含んでいて, Omni XMP でも CAF の大部分の仕様が利用できるようになってきた. XMP のディレクティブ拡張による仕様は, Fortran と C のプログラムの容易な並列化を助けるように考えられているが，CAF 仕様はより低レベルな記述を許し, MPI に匹敵する性能を MPI よりもやさしい記述で実現することを目指している．MPI プログラムが既に多数存在し実用レベルであるのに対し，CAF プログラムはまだ無いに等しく，プログラミングのノウハウも一般的でない.

本稿では，CAF によるプログラミングの実用性を試し，同時にプログラミングノウハウを得るため, 既存の MPI プログラムからの CAF への移植を試し評価する. この後に続く章の構成は以下の通りである. 2 章では準備として，MPI と CAF の仕様を比較して紹介する．3 章では一般的な MPI から CAF への移植の手順を提案し留意点を述べる．4 章は NAS Parallel の 4 つのカーネルアプリについて，どう移植したかを紹介している．移植に使用した Omni XMP CAF コンパイラは 5 章で紹介する．6 章は京コンピュータでの評価を示し，7 章で関連研究を紹介し，8 章でまとめとする.

2. MPI プログラムと CAF プログラム

2.1 MPI による通信の表現

MPI の基本は 1 対 1 通信である．あるノードがデータ送信のためのライブラリ（mpi_send, mpi_isend など）を呼出し（以降，Send），別のノードがそのデータを受信するためのライブラリ（mpi_recv, mpi_irecv など）を呼び出す（以降，Recv）ことで通信が成立する. Send と Recv の対応付けは，互いに相手のランク番号を指定し，同じ値のタグを指定することで指定する．ノンブロッキング通信の場合には，リクエスト番号で対応付けられる待ち合わせのライブラリ（mpi_wait など）とも対応付けられ，それら全体でひとまとまりの通信処理が表現される.

2.2 CAF による通信の表現

CAF の並列実行の主体は「イメージ」と呼ばれる．CAF ではすべてのデータ実体（スカラまたは配列）はすべてのイメージに同じように割付けられるが, coarray 宣言がある場合に限って他のイメージ上のものも定義と使用が許される.

リモートの coarray の定義は，左辺の変数に coindex をもつ代入文で表現できる. たとえば式の文脈で a[i]と書けば，イメージ i における変数 a の値の引用を意味し, それは Get 通信の発生を意味する．以下のように代入文の左辺にこれを書けば,

a [i] = 式

式の値をイメージ i における変数 a に代入することを意味し，つまりは Put 通信の発生を意味する．この Put は非同期なので，代入文が実行完了した時点でデータがイメージ i の変数 a に保存されている保障はない．これを保証するためには利用者の責任で同期を明示しなければならない. よく使われる同期は sync all 文であり，これは通信の保証と同時に全イメージでのバリア同期を行う．また，1 対 1 同期の手段として sync images 文

sync images(イメージ番号[の並び])

がある．これは指定した番号のイメージと 1 対 1 の同期を取るものであり，同期相手が自イメージを指定した sync images 文を実行したときに同期が成立し待ちが解ける.

Á1 理化学研究所 計算科学研究機構
Â2 筑波大学大学院 システム情報工学研究科

2.3 集団通信の比較

MPI にはリダクション，ブロードキャストなどの集団通信が用意されている．一方 CAF では，Fortran2008 で定義されている範囲には集団通信はない．co_sum, co_max などのリダクションや co_broadcast などの仕様は Fortran2015 規格で定義されている．MPI プログラムから CAF への移植ではこれらの機能は必須と言えるので，Omni XMP CAF はこれらの機能を一部サポートしている．

2.4 MPI 仕様との対応

よく使われる MPI ライブラリと，それに近い意味を持つ CAF の書式の対応を表 1 に示す．MPI の 1 対 1 通信と coarray の使用・定義は単純な対応付けにならないので，移植するときにはひとまとまりの Send/Recv/Wait の組として捉えて CAF 表現に変換する．mpi_reduce（結果を 1 ノードに集める）に対応する CAF の表現はないので，無駄な通信は発生するが mpi_allreduce（同じ結果を全ノードにもつ）に相当する CAF の組込み関数で代用する．

表 1　MPI と CAF の表現の対応

	MPI ライブラリ サブルーチン	CAF の書式 （Fortran2008 仕様）
開始，終了	mpi_init/finalize	（不要）
異常終了	mpi_abort	ERROR STOP 文
1 対 1 通信	mpi_send, mpi_recv, mpi_isend, mpi_irecv mpi_wait, mpi_waitall	リモート coarray の使用または定義（片側通信），SYNC IMAGE 文
集計演算	mpi_reduce	（なし）
	mpi_allreduce	組込み関数 co_sum, co_max など （Fortran 2015 仕様）
放送	mpi_bcast	組込み関数 co_broadcast
一斉同期	mpi_barrier	SYNC ALL 文
問合せ	mpi_comm_rank, mpi_comm_size	組込み関数 this_image, num_images

3. MPI から CAF への移植

3.1 方針

移植の作業は，効率的で確実なものとするため，以下の二段階に分けて考える．

1. プログラム変換：個々の MPI の呼出し，または意味的なひとまとまりの呼出しの組に着目し，その単位で対応する CAF 表現に置き換えていく．
2. 性能チューニング：無駄な通信の削減と，無効となったコードの削除を行う．

第 1 段階では，対象プログラムの大域的なアルゴリズムの理解を必要としない．元のプログラムのアルゴリズムはそのまま残し，通信パターンだけに着目して，CAF の表現に変換するというアプローチである．

このアプローチでは，コンパイラの最適化によって生じる精度誤差と，ノード間リダクション演算の誤差などを除けば，移植前後のプログラムで出力結果が全く同じになるので，両者を実行して比較することで容易に誤りの検出ができる．また，MPI が混在した CAF プログラムに対応できるコンパイラを用いるなら，第 1 段階の途中においても結果の確認を行いながら作業を進めることができる．

3.2 1 対 1 通信の変換

MPI の 1 対 1 通信は，Send と Recv の組によって成り立つ．これらはそれぞれ以下の引数をもつ．

- 送信バッファ sendbuf，データ数，データ型，宛先ランク番号 dest，メッセージタグ，コミュニケータ
- 受信バッファ recvbuf，データ数，データ型，送り主ランク番号 source，メッセージタグ，コミュニケータ

CAF 記述への変換を考える場合，どの Send のどのノードでの実行と、どの Recv のどのノードでの実行が組を成すかを、ソースコードから見つけなければならない．これには以下のような場合がある．

- Send の宛先と Recv の送り主のランク番号が相互に指し合っていて，かつ，メッセージタグが一致している組がソースから見て取れる場合，以下のどちらかの CAF 記述に変換できる場合がある．
  - recvbuf を coarray 変数として宣言し，Send を
    recvbuf[dest] = sendbuf
    のような coarray の定義に変換する．Recv の呼出しは抹消する．
  - sendbuf を coarray 変数として宣言し，Recv を
    recvbuf = sendbuf[source]
    のような coarray の参照に変換する．Send の呼出しは抹消する．
- 上記の場合でも，相手側バッファのアドレスを決めるパラメタが相手側ノードにローカルな値である場合，二段階の通信が必要となる．例えば，
  call MPI_Recv(a(:, i), é)
  で受信バッファ a の添字 i が Send 側ノードの知らない値である場合，i の値を coarry の Get または Put で獲得した後に目的の通信を行う必要がある．
  CG, MG でこの例が見られる．
- ランク番号とメッセージタグから Send と Recv のペアが読み取れない場合や，対応が一意でない場合，複数の Send と Recv を対象に CAF 表現への変換を考えなければならない．
  MG でこの例が見られる．

3.3 同期

CAF ではリモート変数の参照と定義は非同期である．MPI の 1 対 1 通信は同期を内在するので，これを CAF 表現に展開するときには同期処理を注意深く挿入しなければならない．

この目的で利用できる CAF の同期には，SYNC ALL 文と SYNC IMAGES 文がある．前者はバリア同期の機能を含み，ハードサポートなどによる高速性が期待できるが，利用できる局面が限られる．後者は

sync images(相手イメージ番号)

という形式で Send-Recv ペアに対応する 1 対 1 の同期を表現することができるが，タグを持たず相手先の相互一致だけで出現順にカップリングされてしまうため，Send-Recv ペアの表現をそのまま投影することができない．場合によっては，元のソースの実行文の順序を入れ替えるなどしなければならない．

この例は MG で見られる．

3.4 その他の MPI 呼出しの変換

表 1 に示したように，1 対 1 通信の関連を除けば，MPI でよく使われているものには概ね対応する CAF の表現があると言える．ただし以下の注意が必要である．

- MPI のランク k を CAF のイメージ(k+1)に対応付ける．ランク番号のバウンダリは 0，イメージインデックスのバウンダリは 1 である．
- CAF は Fortran2008 の範囲では全イメージに対する集団通信しかサポートしないので，COMM_WORLD 以外のコミュニケータが現れると対応が難しい．ただし，ノードを矩形上に仮想配置して列や行を表現するためのコミュニケータに対しては，多次元イメージインデックスで自然に対応できる場合がある．（この例は FT で見られる．）

3.5 Coarray のメモリ割付け

MPI は任意のデータ実体を通信の対象とできるが，CAF 表現の Put または Get の対象となるデータは，coarray 変数として宣言されなければならない．1 対 1 通信を coarray の参照に変換する場合には，リモート側となる送信バッファを coarray 変数として宣言し，coarray の定義に変換する場合には受信バッファを coarray 変数として宣言すればよい．CAF のリダクションとブロードキャストは非 coarray 変数についても利用することができる．

coarray 変数には，静的な変数と，割付け変数が利用できる．文法により，これらは同じ型と形状を持つように全イメージに割付けられなければならない．coarray 変数はモジュールで宣言して使用することや，引数渡しが許されるが，COMMON 変数になることはできない．この点が NPB のようなレガシーなコードでは問題になることがある．CG の移植の初期の版では，common ブロックの中に coarray 宣言したい変数が 3 つあり，うち 1 つは common ブロックの外に出せなかったので，同じ形状の coarray 変数配列を定義して通信を仲介するようにした．

4. NAS Parallel プログラムの CAF 化

サイトからダウンロードした NPB3.3.1 の MPI 版の Fortran のアプリケーションについて，出現する MPI ライブラリの呼出しを表 2 に整理する．

カーネルライブラリの 4 本について CAF プログラムへの移植を行った．

表 2 NAS Parallel で使用される MPI ライブラリ

	kernels				applications		
	CG	EP	FT	MG	BT	SP	LU
mpi_init/finalize/abort	6	3	7	3	3	3	8
mpi_comm_rank/size	2	2	3	2	3	3	3
mpi_[i]send/[i]recv/wait	30			15	61	31	32
mpi_reduce/allreduce	4	7	4	10	7	6	9
mpi_bcast	1	1	5	7	9	4	10
mpi_barrier	1	1	2	9	4	2	1
mpi_comm_split/dup			2		3	3	
mpi_alltoall			3				
mpi_type_*					11		
mpi_file_*					20		
mpi_info_create/set					4		
total	44	14	26	46	125	52	63

4.1 CG

CG の主な通信は，mpi_irecv ï send ï wait のパターンで記述されている．他には単純なリダクションとブロードキャストがあるが，それらは表 1 に沿った単純な置換でよい．

(3) **プログラム変換（MPI→V1）**

例えば MPI コードの 1192 行目に以下のような記述がある．

```
call mpi_irecv(  q(reduce_recv_starts(i)),  ! sendbuf
                 reduce_recv_lengths(i),
                 dp_type,
                 reduce_exch_proc(i),       ! source
                 i,                         ! tag
                 é , request, é  )
call mpi_send(   w(reduce_send_starts(i)), ! recvbuf
                 reduce_send_length(i),
                 dp_type,
                 reduce_exch_proc(i),       ! dest
                 i,                         ! tag
                 é  )
call mpi_wait(   request, é  )
```

この mpi_irecv と mpi_wait の呼出しが同じノードのインスタンスどうしで対応付くことは同じ request 引数をもつことから分かる．mpi_irecv と mpi_send の対応は，source 引数と dest 引数が同じ値をもつことから，自ノードからの送信相手が自ノードへの送信元でもあることが分かる．配列 reduce_exch_proc は

```
    j = reduce_exch_proc(i)
```
のとき
```
    i = reduce_exch_proc(j)
```
という性質をもつので，2 ノード間の送受信が同時に双方向に起こることが分かる．これらのことから，最初のバージョン V1 では上記のパターンを以下の CAF 記述に変換した．

```
double precision q( é )[*]
integer exproc, start, length
é
exproc = reduce_exch_proc(i) + 1
sync images(exproc)
start = reduce_recv_starts(i)[exproc]
length = reduce_recv_lengths(i)[exproc]
q(start : start + length - 1)[exproc] = w(
    reduce_send_starts(i) :
    reduce_send_starts(i) + reduce_send_lengths(i) - 1)
sync images(exproc)
```

送信バッファ d は coarray として宣言した．exproc は送受信先のランク番号に 1 を加えたものであり，送受信先のイメージ番号となる．最後から 2 行目の d への代入文が目的の Put 通信である．ここで問題となるのは，送信先のアドレスとデータ長を得るのに必要なパラメタ reduce_recv_starts(i)と reduce_recv_length(i)を送信者が保持していないことである．そのため，これらのパラメタを Put 通信の直前に Get 通信によって獲得している．これらの通信のために必要最小限の 2 ノード間の同期を，2 つの sync image 文で記述している．

(4) **性能チューニング（V1→V3, V3.5）**

通信量を削減するために，上記の 2 段階の通信を 1 段の通信に変えることを考えた. テーブル reduce_recv_starts は，「ステップ i における自ノードの受信バッファの先頭位置」を保持している．ソースの目視により，このテーブルはプログラム初期化時に設定され変更されないことが分かったので，代わりに「ステップ i における送信先ノードの受信バッファの先頭位置」を意味するテーブル rrstarts を初期化時に用意するようにした．このテーブルは送信者がローカルに保持するので，以下のように Put の前の通信を消すことができる．

```
exproc = reduce_exch_images(i)
sync images(exproc)
q(rrstarts(i) : rrends(i))[exproc] = w(
    reduce_send_starts(i) :
    reduce_send_starts(i) + reduce_send_lengths(i) - 1)
sync images(exproc)
```

ここではさらに細かな工夫として，ランク番号のテーブルだった reduce_exch_proc をイメージ番号のテーブル reduce_exch_images に替え，受信データ長の代わりにデータの終わり位置を示す rrends を使用している．この版を V3.5 とした．

この sync images 文は，すべてのノードで同時に実行されることがソースの解析で分かったので，sync all 文に置き換えることもできる．性能比較のため sync all 文に置き換えた版を V3 として作成した．sync images はノード数が増えても通信が局所的だという利点があるのに対し，sync all 文はハードサポート等による高速性が期待できるので，どちらが高速かは一概に言えない．

4.2 EP

EP には Send/Recv 通信はない．リダクション，ブロードキャストとバリアは，単純な置換のみである．作成した版を V1 としたが，特に性能チューニングできる部分も見つからなかった．ソースを見る限り，性能は MPI と全く同等と考えられた．

4.3 FT

FT にも Send/Recv 通信は見られないが，代わりに all-to-all の集団通信が使われている. これに対応する機能は CAF にはない. MPI が混在できる CAF 処理系を使うなら，実用上はこのまま MPI 呼出しとして残しておくのが賢明かもしれないが，ここでは敢えて CAF の記述に変換してみた．

(5) **プログラム変換（MPI→V2）**

一般に，

```
MPI_Alltoall( sendbuf, sendcount, sendtype,
                recvbuf, recvcount, recvtype,  é )
```
は，すべてのノードに対する Send とすべてのノードに対する Recv の組に書き下せるはずなので，まずそのように変換した後，CAF に変換しようと当初は考えた．しかし実際に MPI で書こうとすると，デッドロックを回避しながら効率的なプログラムにすることが意外に難しいと感じ断念した. そこで直接 CAF で書いてみたのが以下のコードである．なお，ここで使われているコミュニケータ commslice1 は，全ノードを 2 次元配列状に仮想配置したときの列または行ベクトルに属するノードの集まりを表現する．

- 変換前(MPI)

```
double complex xin(ntdivnp)
double complex xout(ntdivnp)
é
call mpi_alltoall(xin, ntdivnp/np, dc_type,
                  xout, ntdivnp/np, dc_type,
                  commslice1, ierr)
```

- 変換後(V2)

```
double complex xin(ntdivnp/np, 0:np-1)[0:np2, 0:*]
double complex xout(ntdivnp/np, 0:np-1)[0:np2, 0:*]
é
sync all
do ne2 = 0, np2-1
    if (ne2 == me2) then
        xout(:, me2) = xin(:, ne2)
    else
        xout(:, me2)[ne2, me1] = xin(:, ne2)
    endif
enddo
sync all
```

ここで変数 me2 は，2 次元イメージ配列（0 バウンダリとした）の 2 次元目の自イメージのインデックスである．送信先イメージインデックス ne2 が me2 と異なる場合には Put 通信を行う．バッファ変数 xin と xout は仮引数で与えられるが，Fortran では対応する実引数とは異なる形状で宣言してもよいので，記述が自然になる 2 次元配列とした．ここでは xin はリモート参照されないので，非 coarray であってもよい．

(6) **性能チューニング（V2→V3）**

上記の変換後のコードでは，多数ノードのとき，ne2=0 となるイメージから順に Put 通信の集中砲火を受けることになるので，ネットワークの衝突によって性能が低下する恐れがある．そこで以下のように，ネットワーク負荷を均等にするため，通信の順序を自ノードの隣から順に回すように変更した（V3）．

```
sync all
xout(:, me2) = xin(:, me2)
do i = 1, np2-1
    ne2 = mod(me2 + i, np2)
    xout(:, me2)[ne2, me1] = xin(:, ne2)
enddo
sync all
```

4.4 MG

MG には mpi_irecvï sendï wait のパターンで複雑な並列制御をもつサブルーチン comm3 と comm3_ex がある．comm3 は複数の箇所から呼び出され，以下のサブルーチンを呼び出す．

- ready: 1 つの mpi_irecv を含む．
- give3: 6 つに場合分けされ，それぞれのブロックにバッファリングと mpi_send を含む．
- take3: 1 つの mpi_wait と，6 つの選択的なアンバッファリングを含む．

comm3_ex は 1 箇所で呼ばれ，サブルーチン ready, give3_ex と take3_ex を呼び出す. ready は共通に呼ばれている. give3 と give3_ex，take3 と take3_ex はそれぞれよく似ている．

(1) **プログラム変換（MPI→V2）**

comm3 と comm3_ex の制御構造は大きく違うので，一つずつ対応することとした．まずサブルーチン ready を comm3 用と comm3_ex 用にバージョニングし，comm3 用の ready と give3 と take3 を合わせて CAF に変換し実行確認した後，comm3_ex 用の ready と give3_ex と take3_ex を合わせて CAF に変換した．最終的に 2 つの ready は同じものになったので共通化した．その他のリダクションやブロードキャストについては, 特に問題なく対応する CAF 記述に変換できた．以下では comm3 系の変換について紹介する．

```
if (一部のノードで成り立つ条件) then
    do axis = 1, 3
        call ready; call ready          !! mpi_irecv
        call give3; call give3          !! mpi_send
        call take3; call take3          !! mpi_wait
    end do
else
    call zero3              !! 通信無し
end if
```

2 組の mpi_irecvï sendï wait のパターンが同時進行するが，基本的には CG のパターンに似ているので，同様な 2 段階通信でまず動作確認できた．ただし，送信者と受信者の対応付けがソース目視では難しかったので, sync images による 1 対 1 同期ではなく sync all による一斉同期を活用した．結果として以下のようなコードとなった．

```
if (一部のノードで成り立つ条件) then
    do axis = 1, 3
        rtag = 0                !! 初期化
        sync all
        call ready; call ready  !! パラメタの受渡し
        sync all
        call give3; call give3  !! データの Put
        sync all
        call take3; call take3  !! アンバッファリング
        sync all
    end do
else
    call zero3              !! 通信無し
    do axis = 1, 3
        sync all; sync all; sync all; sync all
    end do
end if
```

ready の中の通信は Put で記述したので，前後の同期 sync all を容易に外すことはできない．その他の理由で結果的に毎反復に対して 4 回の sync all が必要となった．また，一部のノードは if then ブロックを通らず else ブロックを通るが，デッドロック回避のためにそれらのノードに対しても同じ数だけの sync all が必要となる．

(2) **性能チューニング（V2→V3）**

ソースコードの目視での解析により，最終的に CG と同じように通信の 1 段化による高速化に成功した．結果として，以下のようにサブルーチン ready が不要となった．

```
if (一部のノードで成り立つ条件) then
    do axis = 1, 3
        sync all
        call give3; call give3     !! データの Put
        sync all
        call take3; call take3   !! アンバッファリング
    end do
else
    call zero3                        !! 通信無し
    do axis = 1, 3
        sync all; sync all
    end do
end if
```

5. Omni XcalableMP CAF コンパイラ

Omni XMP コンパイラの現在の CAF 1.0 仕様（Fortran2008 仕様に含まれる）に対するサポート状況は，表 3 に示す通りである．表中の「章」は，文献[1]における章である．

この範囲に加え，MPI でありがちなプログラムに対応するため, Fortran2015 仕様である組込み関数 co_sum, co_min, co_max と co_broadcast の一部の仕様範囲について実装している．

6. 評価

6.1 ソースコード量

4 章で述べた MPI から移植した CAF プログラムについて，コード量を表 4 に示す．ここで LOC はコメント行と空行を除くソース行数であり，乱数生成やパラメタ設定のための共通のコードを含まない．

共通して以下の傾向が見られる．

- MPI から CAF への最初の移植では，行数に大きな増減はない．これは，通信が MPI 呼出し（NAS Parallel では 1 文を複数の行で表現することが多い）から coarray のコンパクトな表現に変わることで削減される量と，同期の追加や二段階通信の生成によって増加する量でほぼ相殺されているように見られる．
- チューニングが進むとコード量が減る傾向がある．これはそのまま，通信や同期が削減された効果とみる．

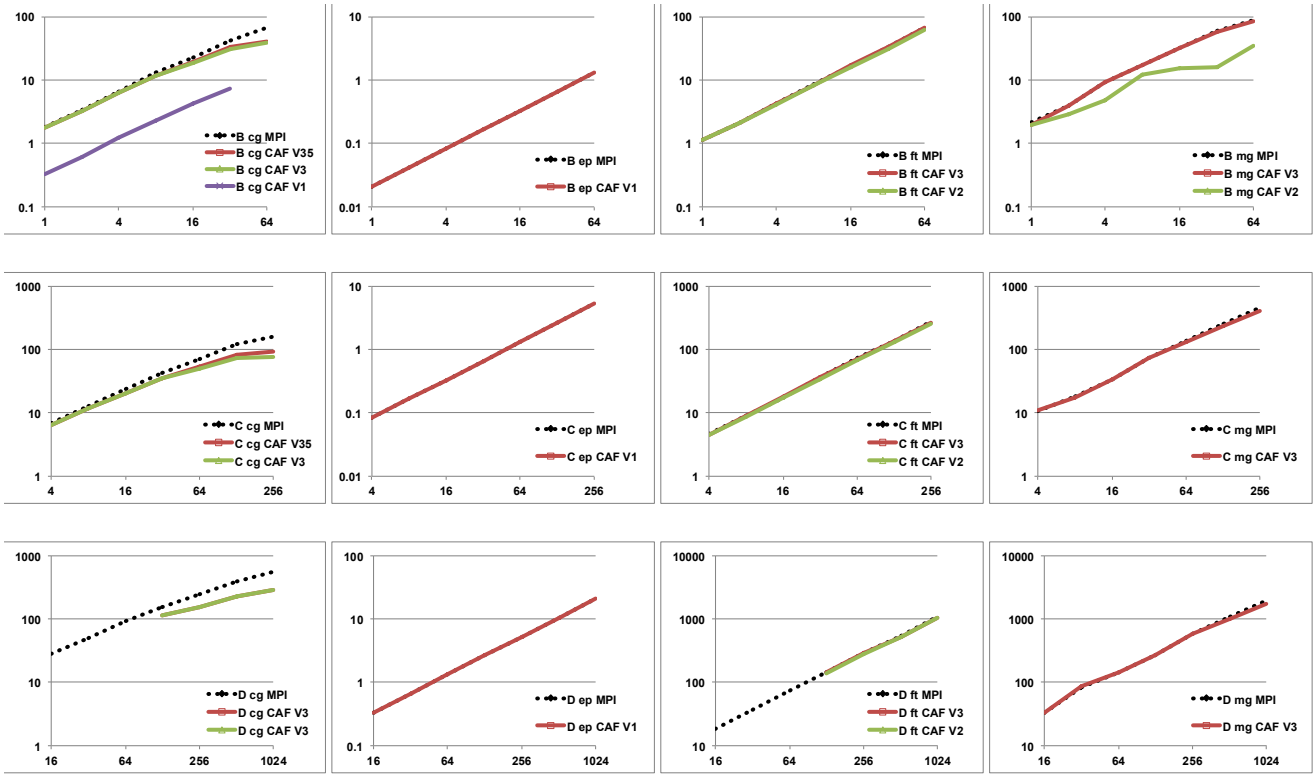
なお，FT については特別であり，mpi_alltoall の呼出しを敢えて CAF 表現を含むループ実行に変換したため, コード量が増加している．

表 3　Omni XMP CAF の CAF1.0 仕様サポート状況
（2016/1/末現在）

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	済
	LOCK 文と UNLOCK 文	未
	CRITICAL セクション	未
	SYNC MEMORY 文	済
	ǿstat=ǿ指定子とǿerrmsg=ǿ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	済
	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

表 4　コード量の比較

		LOC	MPI 比
CG	MPI	1131	
	CAF V1	1105	-2%
	CAF V3	1009	-11%
	CAF V3.5	1008	-11%
EP	MPI	226	
	CAF V1	211	-7%
FT	MPI	1275	
	CAF V2	1278	+0.2%
	CAF V3	1272	-0.2%
MG	MPI	1763	
	CAF V2	1728	-2%
	CAF V3	1671	-5%



6.2 性能

図 1 に，MP 版と，チューニング後の CAF 版の性能を比較する. 測定には京コンピュータを用いた. Omni XMP CAF コンパイラのバックエンドは富士通 Fortran を使い. そのオプションは -Kfast,parallel,reduction（スレッド自動並列化を含む）とした.

CG, EP, FT, MG について，問題サイズの異なる 3 つのクラス B, C, D で測定した結果を示している．クラス B の CG と MG では，チューニング前の版も同時にプロットしている.

最終的に得られた性能は，EP, FT, MG では MPI とほぼ同等と言える. 測定したクラス B,C,D の全測定区間で，Mop/s 値は数点のプロットを除けば MPI の性能の 96%から 103%の範囲に入る．CG では高並列で飽和する傾向が若干見られるため，全測定範囲では MPI の性能の 50%から 100%だった．スケーラビリティが阻害される原因を調べる必要がある.

クラス D の CG と FT では，64 ノード以下のときにメモリ不足となり実行できなかった．Omni XMP CAF の実装に課題がないか確認する必要がある.

7. 関連研究

Omni XMP CAF 以外にフリーの CAF コンパイラとしては，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. 昨年リリースされた OpenCoarray は，GNU gfortran にリンクできるライブラリである.

XMP 仕様は C 版の Coarray 仕様を含んでいて, Omni XMP CAF の低位のランタイムシステムは C 版と共通化されている．Omni XMP の C 版 Coarray によるプログラミングの例は[4]にある.

8. まとめ

NAS Parallel の 4 つのカーネルプログラムを題材に，MPI プログラムから CAF への移植を試みた．ここで提案した CAF 表現への変換と CAF プログラムとしての性能チューニングの 2 段階の作業手順により，多くの MPI プログラムは CAF に移植できると考えられる．CAF のコンパクトな表現により，性能チューニングが進むとソースコード行数が少なくなる傾向が見られた.

最終的に，EP, FT, MG では MPI と数%上下に振れるだけのほとんど同じ性能が得られた．MPI プログラムからの並

列アルゴリズムを変えない移植なので, 原理的に MPI の性能を超えることは難しいが, 今後 CAF 固有の並列アルゴリズムの研究が進み, 処理系の改善も進めば, 高級言語で MPI の性能を超えることも近いうちに不思議ではなくなるだろう.

参考文献

[1] John Reid. Coarrays in the next Fortran Standard, ISO/IEC JTC1/SC22/WG5 N1824, April 21, 2010
[2] Information technology -- Additional Parallel Features in Fortran ISO/IEC TS 18508:2015
[3] 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．
[4] 中尾昌広，村井均，岩下英俊，下坂健則，朴泰祐，佐藤三久．PGAS 言語 XcalableMP を用いた HPC Challenge ベンチマークの実装と評価，第 148 回 HPC 研究会，2015 年 3 月．
[5] Deepak Eachempati, Hyoung Joon Jun and Barbara Chapman. *An Open-Source Compiler and Runtime Implementation for Coarray Fortran.* PGASô10 Proc. of 4th Conference on PGAS Programming Models, No.13, 2010
[6] Cristian Coarfa. Portable High Performance and Scalability of Global Address Space Languages. Ph.D. Thesis, Rice University, January 2007.