Omni XMP Coarray Fortranの実装の状況

2016/11/7

理化学研究所計算科学研究機構（AICS）
岩下 英俊

XMP仕様の中のCoarray機能

- 仕様
  - XMPはFortranとCをベースとした並列言語
  - 「グローバルビュー」と「ローカルビュー」の両方が使用可能

Global-view

- データ分散の宣言
```
!$xmp nodes p(4)
!$xmp template t(100)
!$xmp distribute t(block) onto p
real a(100,100)
!$xmp align a(*,j) with t(j)
```
- 計算負荷の分散
```
!$xmp loop onto t(j)
do j = 1, 100
  do i = 1, 100
    a(i, j) = ...
  enddo
enddo
```
- グローバルビューの通信指示
```
!$xmp reflect (b)  ! 袖通信の指示
```

Local-view = Coarray機能

- coarray変数の宣言
```
real a(100, 25)[*]
```
- 計算は各ノードの振舞いを記述
```
do j = 1, 25
  do i = 1, 100
    a(i, j) = ...
  enddo
enddo
```
- 通信はcoarrayの参照・定義
```
k = this_image()
b(1:100, 0) = b(1:100, 25)[k-1] +...
b(1:100, 0)[k+1] = b(1:100, 25) +...
```
- 同期（sync all, sync images, ...）
- 集団通信，アトミック通信
- XMP Global-viewとの連携機能

内容

- Omni XMP Coarray Fortran これまでの成果
- メモリ管理系Ver.4
- Coarray機能のtask構文対応
- まとめ

これまでの成果(1)

- CAFの実装による性能比較
  - Himenoの性能で，Omni XMPはUH-CAF, OpenCoarray, Intelを上回った.

Fortranコンパイラの利用

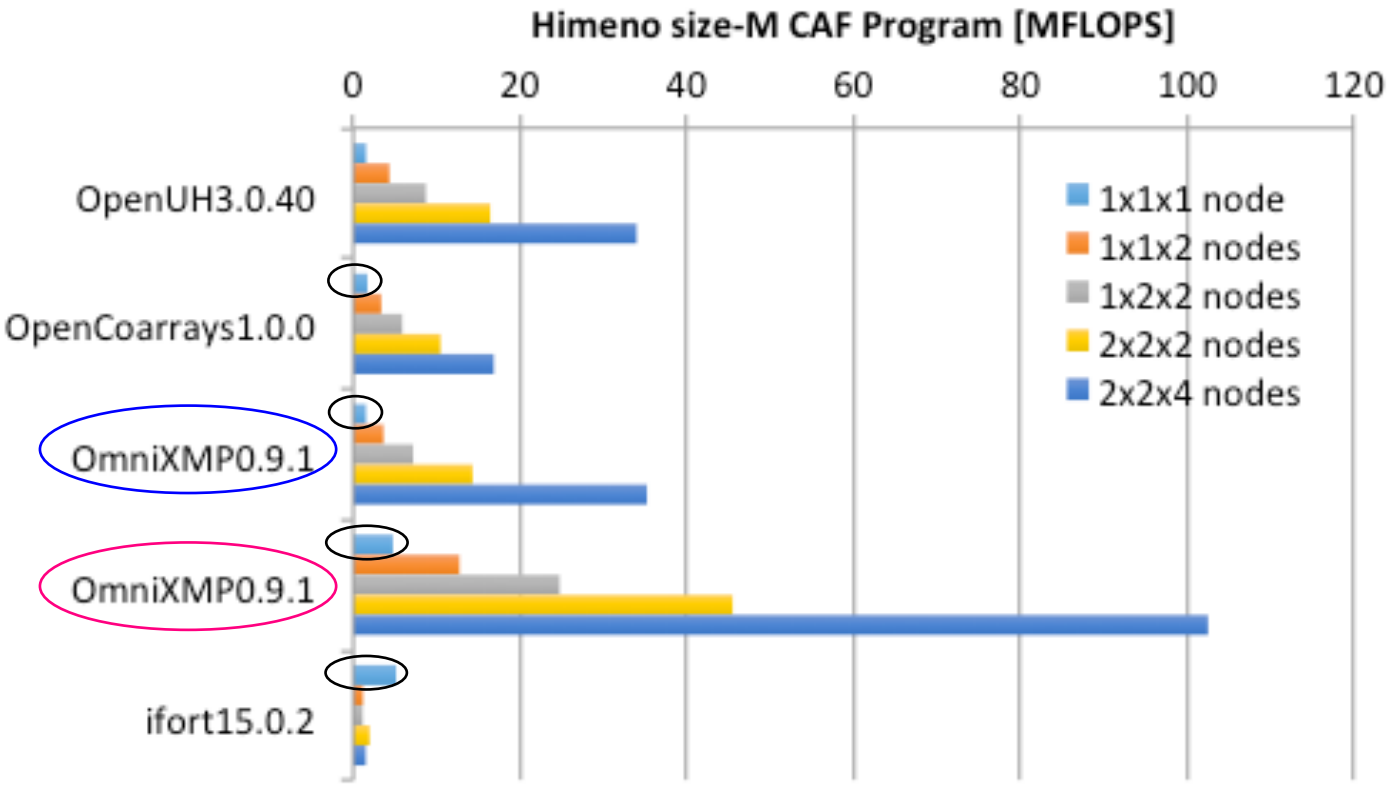
Open64(フリー)ベースの実装

GNU gfortranのライブラリ

GNU gfortranをbackendに使用

Intel ifortをbackendに使用

Intel ifortに組込み



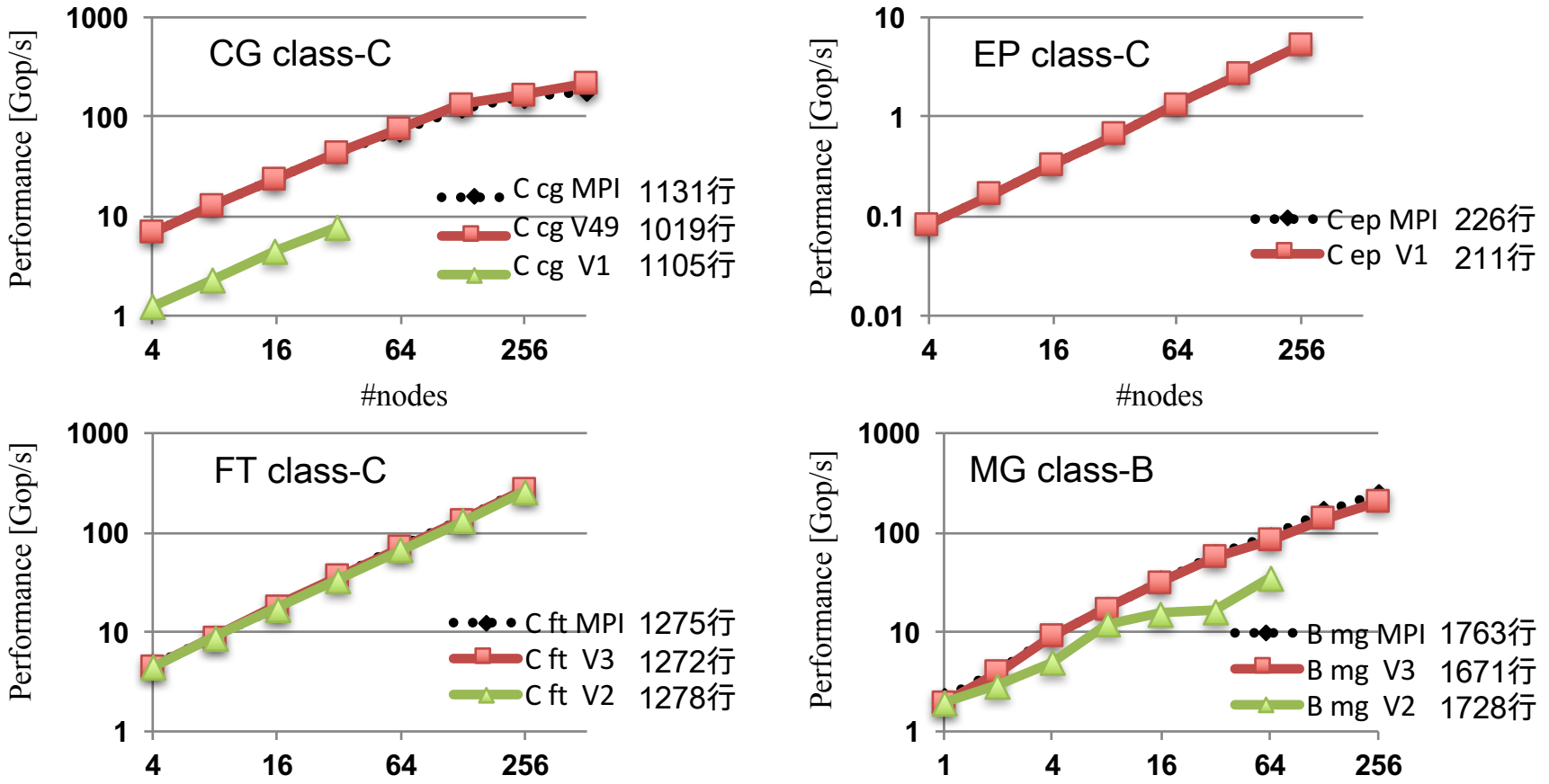
  - 主な理由は，ネイティブコンパイラの性能差
    - トランスレータ方式の良さは，Fortranコンパイラを選択できること.

測定環境 筑波大HA-PACS/TCA: Xeon E5-2680(2.8GHz,10core×2CPU/node), Mellanox IB QDR 8GB/s/node

Hidetoshi Iwashita, et al. *Implementation of Coarray Fortran Translator Based on Omni XcalableMP*. PGAS2015, Sep., 2015.

これまでの成果(2)

- MPIプログラムのCAFへのポーティング
  - 提唱:(1) MPI→CAF変換 ＋ (2) CAFプログラムとしてチューニング
  - CAFプログラムのチューニングで，MPIと同程度～同等以上の性能を示した.



測定環境 京コンピュータ: SPARC64 VIIIfx (8core, 2GHz, 128GFLOPS), 1CPU/node, 100GB/s/node
岩下英俊他. *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*. HPC研究会, 2016年3月

内容

- Omni XMP Coarray Fortran これまでの成果
- メモリ管理系Ver.4
- Coarray機能のtask構文対応
- まとめ

メモリ管理系の実装状況

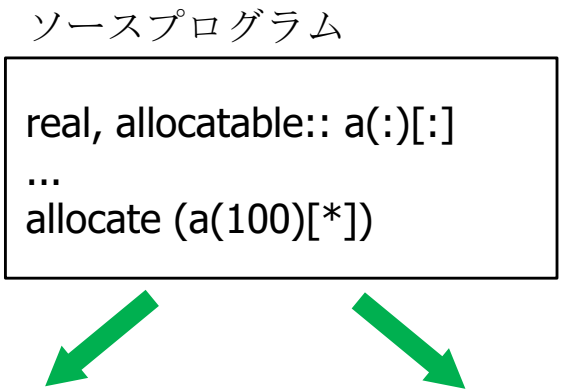
- Coarray変数（静的・動的）のメモリ管理と実装状況
  - 現在
    - 通信層に依らず，デフォルトでVer.3
  - 近日公開予定
    - FJ-RDMAの場合に限り，デフォルトをVer.4に変更
    - Ver.4: 生成コード中のポインタの使用が減り，性能改善が期待される．

Memory Management System	通信層（片側通信）		
	GASNet	FJ-RDMA	MPI-3
Ver.3 割付けと登録(*1)を 実行時ライブラリで行う．	実装済	実装済	実装済
Ver.4 割付けをFortranで行い， 登録(*1)を実行時ライブラリで行う．	不可(*2)	実装済 （次のdefault）	実装中

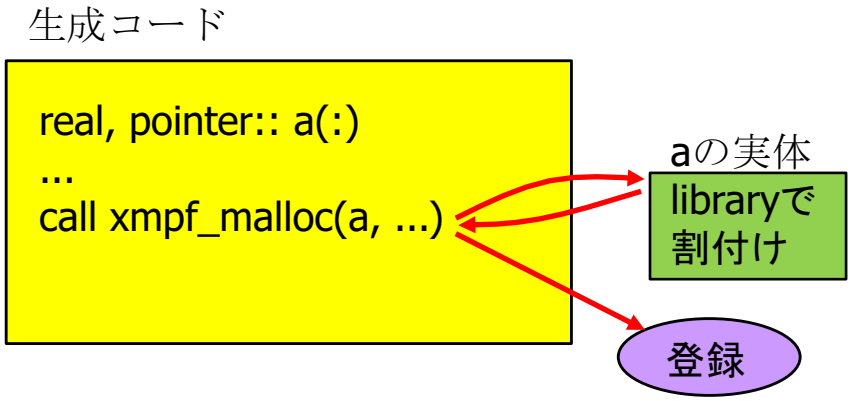
(*1) 登録とは，global addressの獲得と共有，メモリのpin-downなどの処理を含む（通信層に依存）．
(*2) GASNetで通信するデータは，gasnet_mallocで確保する必要があるため．

Ver.3とVer.4の生成コードの違い(1)

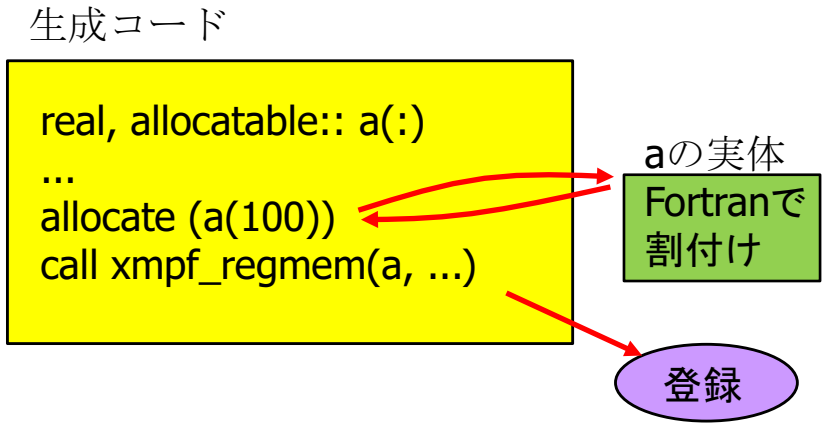
Allocatable Coarray
の場合



- Ver.3
  Coarrayを実行時libraryで割付け
  - Coarray変数はポインタに変換され，ALLOCATE文に代わるライブラリ呼出しで割付け・登録される．

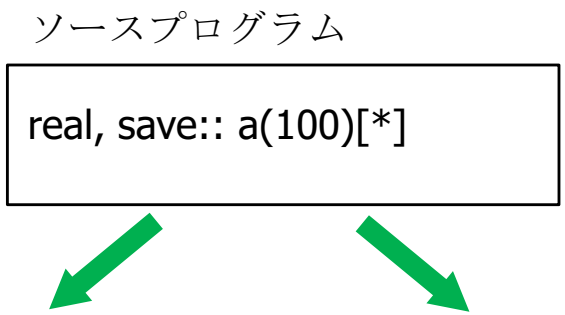


- Ver.4
  CoarrayをFortran処理系で割付け
  - Coarray変数はALLOCATE文で割付けられた後，登録される．

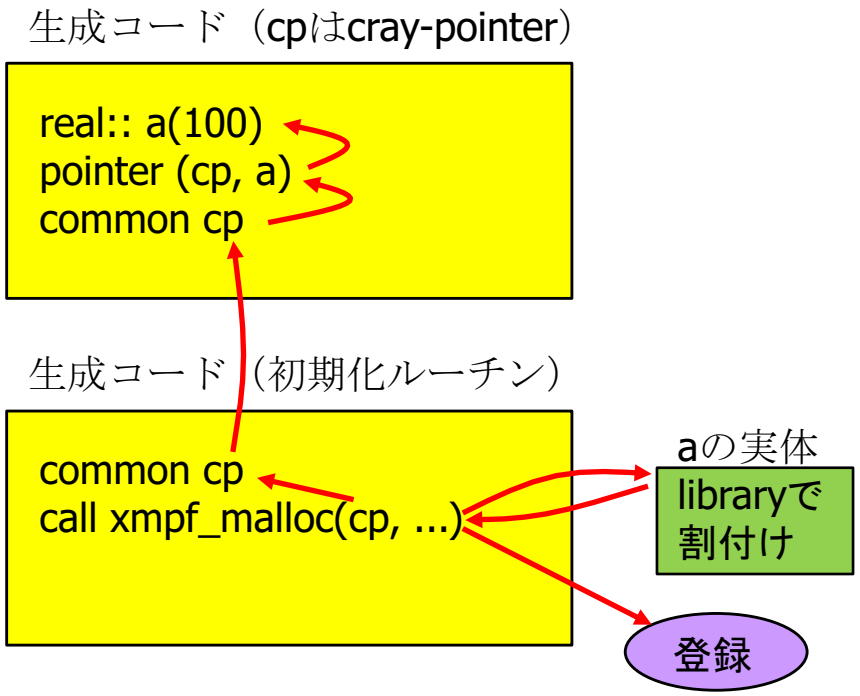


Ver.3とVer.4の生成コードの違い(2)

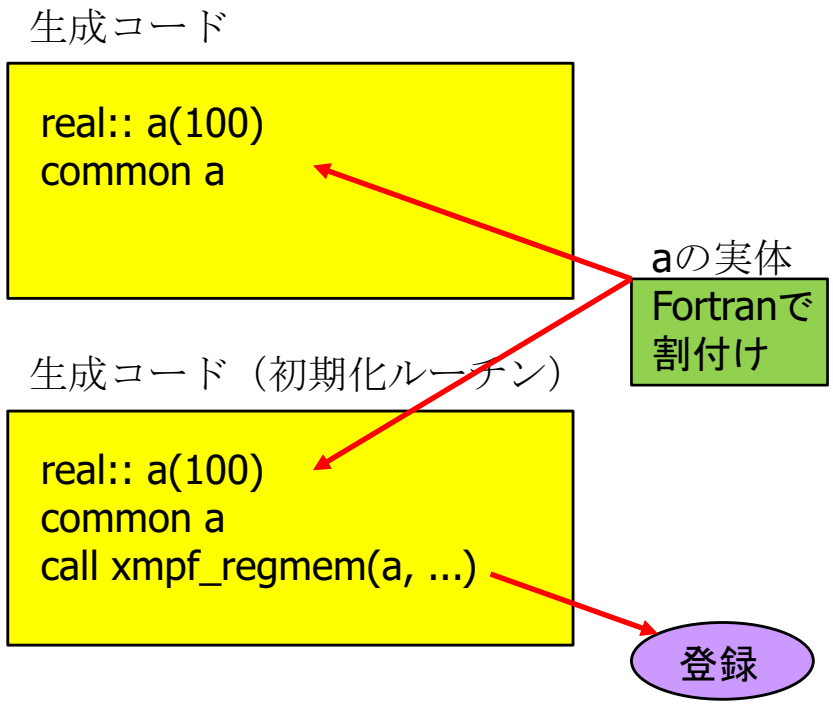
静的なCoarray
の場合



- Ver.3
  Coarrayを実行時libraryで割付け
  - Coarray変数はポインタに変換され，プログラム実行開始時に割付け・登録される．



- Ver.4
  CoarrayをFortran処理系で割付け
  - Coarray変数は静的に割付けられ，プログラム実行開始時に登録される．



Ver.3の何が問題だったのか

- 通信部分でなく計算部分の性能低下
  - Ver.3では，CAFトランスレータは，Coarray変数をポインタ変数に変換する．
    ➔計算ループ内にポインタ変数が現れる➔Fortranの最適化を妨害

HimenoベンチJacobiループ Fortran最適化の結果比較

MPI版 リスタ出力

```
                <<< Loop-information Start >>>
                <<<  [PARALLELIZATION]
                <<<    Standard iteration count:
                <<< Loop-information  End >>>
  2  pp  8          do k=2,kmax-1
                <<< Loop-information Start >>>
                <<<  [OPTIMIZATION]
                <<<    PREFETCH      : 42
                <<<      c: 6, p: 6, a: 18, wrk1: 6, wrk2: 6
                <<< Loop-information  End >>>
  3  p   8            do j=2,jmax-1
                <<< Loop-information Start >>>
                <<<  [OPTIMIZATION]
                <<<    SIMD
                <<<    SOFTWARE PIPELINING
                <<<    PREFETCH      : 14
                <<<      a: 6, wrk1: 2, p: 2, c: 2,
                <<< Loop-information  End >>>
  4  p   2v             do i=2,imax-1
  4  p   2v               s0=a(I,J,K,1)*p(I+1,J,K) &
  4                          +a(I,J,K,2)*p(I,J+1,K) &
  4                          +a(I,J,K,3)*p(I,J,K+1) &
  4                          +b(I,J,K,1)*(p(I+1,J+1,K)-p(I+1,J-1,K) &
  4                                      -p(I-1,J+1,K)+p(I-1,J-1,K)) &
  4                          +b(I,J,K,2)*(p(I,J+1,K+1)-p(I,J-1,K+1) &
  4                                      -p(I,J+1,K-1)+p(I,J-1,K-1)) &
  4                          +b(I,J,K,3)*(p(I+1,J,K+1)-p(I-1,J,K+1) &
  4                                      -p(I+1,J,K-1)+p(I-1,J,K-1)) &
  4                          +c(I,J,K,1)*p(I-1,J,K) &
  4                          +c(I,J,K,2)*p(I,J-1,K) &
  4                          +c(I,J,K,3)*p(I,J,K-1)+wrk1(I,J,K)
  4  p   2v               ss=(s0*a(I,J,K,4)-p(I,J,K))*bnd(I,J,K)
  4  p   2v               wgosa=wgosa+ss*ss
  4  p   2v               wrk2(I,J,K)=p(I,J,K)+omega*ss
  4  p   2v             enddo
  3  p   8            enddo
  2  p   8          enddo
```

2回転以上のとき並列化

プリフェッチc,p.a.wrkl,wrk2

SIMD化
ソフトウェアパイプライン
プリフェッチa,wrl1,p,c,wrk2

Ver.3 変換後の
リスタ出力

allocatable coarray p は
ポインタ変数に変換

```
                <<< Loop-information Start >>>
                <<<  [PARALLELIZATION]
                <<<    Standard iteration count: 2
                <<< Loop-information  End >>>
  2  pp  8        DO k = 2 , kmax - 1 , 1
  3  p   8          DO j = 2 , jmax - 1 , 1
                <<< Loop-information Start >>>
                <<<  [OPTIMIZATION]
                <<<    SIMD
                <<< Loop-information  End >>>
  4  p   8v           DO i = 2 , imax - 1 , 1
  4  p   8v            s0 = a ( i , j , k , 1 ) * p ( i + 1 , j , k ) + a ( i , j , k , 2 ) * p ( i ,
  4                   , k + 1 ) + b ( i , j , k , 1 ) * ( p ( i + 1 , j + 1 , k ) - p ( i + 1 , j -
  4                   , j - 1 , k ) ) + b ( i , j , k , 2 ) * ( p ( i , j + 1 , k + 1 ) - p ( i , j -
  4                   i , j - 1 , k - 1 ) ) + b ( i , j , k , 3 ) * ( p ( i + 1 , j , k + 1 ) - p ( i
  4                   + p ( i - 1 , j , k - 1 ) ) + c ( i , j , k , 1 ) * p ( i - 1 , j , k ) + c ( i
  4                   , j , k , 3 ) * p ( i , j , k - 1 ) + wrk1 ( i , j , k )
  4  p   8v           ss = ( s0 * a ( i , j , k , 4 ) - p ( i , j , k ) ) * bnd ( i , j , k )
  4  p   8v           wgosa = wgosa + ss * ss
  4  p   8v           wrk2 ( i , j , k ) = p ( i , j , k ) + 0.8 * ss
  4  p   8v         END DO
  3  p   8        END DO
  2  p   8      END DO
```

2回転以上のとき並列化

SIMD化

富士通Fortarnコンパイラ K-1.2.0-20-1
mpifrtpx, xmpf90とも，使用オプション –Qt –Kfast,parallel

Ver.4でどう改善されたか

- MPI版と同レベルのFortran最適化を取り戻した
  - Ver.4では，CAFトランスレータは，allocatable coarrayはallocatable non-coarrayに，静的なcoarrayは静的なnon-coarrayに変換する.

HimenoベンチJacobiループ Fortran最適化の結果比較

MPI版 リスタ出力

```
<<< Loop-information Start >>>
<<<  [PARALLELIZATION]
<<<    Standard iteration count:
<<< Loop-information  End >>>
2  pp  8          do k=2,kmax-1
<<< Loop-information Start >>>
<<<  [OPTIMIZATION]
<<<    PREFETCH       : 42
<<<      c: 6, p: 6, a: 18, wrk1: 6, wrk2: 6
<<< Loop-information  End >>>
3  p   8            do j=2,jmax-1
<<< Loop-information Start >>>
<<<  [OPTIMIZATION]
<<<    SIMD
<<<    SOFTWARE PIPELINING
<<<    PREFETCH       : 14
<<<      a: 6, wrk1: 2, p: 2, c: 2, w
<<< Loop-information  End >>>
4  p   2v               do i=2,imax-1
4  p   2v                  s0=a(I,J,K,1)*p(I+1,J,K) &
4                             +a(I,J,K,2)*p(I,J+1,K) &
4                             +a(I,J,K,3)*p(I,J,K+1) &
4                             +b(I,J,K,1)*(p(I+1,J+1,K)-p(I+1,J-1,K)-&
4                             +b(I,J,K,2)*(p(I,J+1,K+1)-p(I,J-1,K+1)-&
4                             +b(I,J,K,3)*(p(I+1,J,K+1)-p(I-1,J,K+1)-&
4                             +c(I,J,K,1)*p(I-1,J,K) &
4                             +c(I,J,K,2)*p(I,J-1,K) &
4                             +c(I,J,K,3)*p(I,J,K-1) &
4                             +wrk1(I,J,K)
4  p   2v                  ss=(s0*a(I,J,K,4)-p(I,J,K))*bnd(I,J,K)
4  p   2v                  wr
4  p   2v                  wr
4  p   2v               enddo
3  p   8             enddo
2  p   8          enddo
```

2回転以上のとき並列化

プリフェッチc,p.a.wrkl,wrk2

SIMD化
ソフトウェアパイプライン
プリフェッチa,wrl1,p,c,wrk2

Ver.4 変換後のリスタ出力

allocatable coarray p は
通常のallocatable変数に変換.

```
<<< Loop-information Start >>>
<<<  [PARALLELIZATION]
<<<    Standard iteration count:
<<< Loop-information  End >>>
2  pp  8          DO k = 2 , kmax - 1 , 1
<<< Loop-information Start >>>
<<<  [OPTIMIZATION]
<<<    PREFETCH       : 42
<<<      c: 6, p: 6, a: 18, wrk1: 6, wrk2: 6
<<< Loop-information  End >>>
3  p   8            DO j = 2 , jmax - 1 , 1
<<< Loop-information Start >>>
<<<  [OPTIMIZATION]
<<<    SIMD
<<<    SOFTWARE PIPELINING
<<<    PREFETCH       : 14
<<<      a: 6, wrk1: 2, p: 2, c: 2, wrk2: 2
<<< Loop-information  End >>>
4  p   2v               DO i = 2 , imax - 1 , 1
                          j , k ) + a ( i , j , k , 2 ) * p ( i , j
                          i + 1 , j + 1 , k ) - p ( i + 1 , j - 1
                          p ( i , j + 1 , k + 1 ) - p ( i , j - 1
                          ) * ( p ( i + 1 , j , k + 1 ) - p ( i -
                          k , 1 ) * p ( i - 1 , j , k ) + c ( i ,
                          k1 ( i , j , k )
                          i , j , k ) ) * bnd ( i , j , k )
                          0.8 * ss
2  p   8          END DO
```

2回転以上のとき並列化

プリフェッチc,p.a.wrkl,wrk2

SIMD化
ソフトウェアパイプライン
プリフェッチa,wrl1,p,c,wrk2

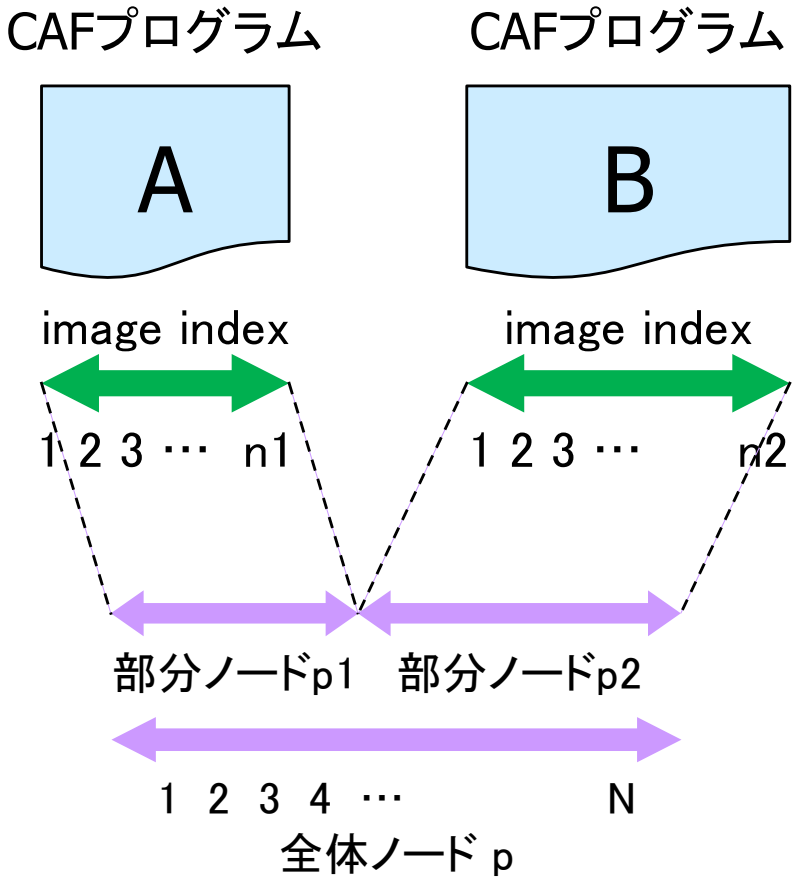
Size	Nodes	Ver.3 [Gflop/s]	Ver.4 [Gflop/s]	Ratio (Ver.4 / Ver.3)
XL	**1 x 1 x 8**	**36.9**	**97.5**	**2.64**
XL	2 x 2 x 2	22.1	29.2	1.32
XL	8 x 8 x 8	611.6	602.3	0.98

内容

- Omni XMP Coarray Fortran これまでの成果
- メモリ管理系Ver.4
- Coarray機能のtask構文対応
- まとめ

Coarrayタスク並列化 ー 考え方

- CAFのimage indexは，XMPの部分ノードにマッピングできる．
  - 個々のindex空間（1から昇順）は維持される．
- これにより，CAFプログラム（サブルーチン）を，ほとんど修正することなく，XMPのタスクとして実行できる．



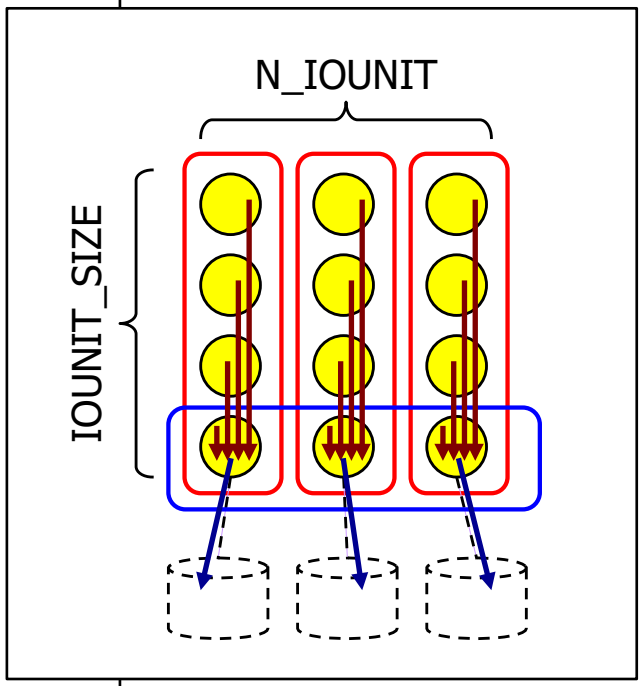

```
!$xmp nodes p(30)                    !! entire nodes
!$xmp nodes p1 = p(1:10)             !! subnodes
!$xmp nodes p2 = p(11:30)            !! subnodes

!$xmp tasks
!$xmp task on p1
      call A
!$xmp end task
!$xmp task on p2
      call B
!$xmp end task
!$xmp tasks
```

image	1	2	…	10
subnode	p1(1)	p1(2)	…	p1(10)
node	p(1)	p(2)	…	p(10)

image	1	2	…	20
subnode	p2(1)	p2(2)	…	p2(20)
node	p(11)	p(12)	…	p(30)

Corrrayタスク並列化 ー 使用例

```
integer,parameter:: IOUNIT_SIZE = 4          !! IOUNIT中のノード数
integer,parameter:: N_IOUNIT = 3             !! IOUNITの数
integer,parameter:: IO_NODE_ID = 1           !! IOUNIT中の I/O node の位置

!$xmp nodes all_nodes( IOUNIT_SIZE, N_IOUNIT )
!$xmp nodes iounit(IOUNIT_SIZE) = all_nodes( :, * )
!$xmp nodes ionodes(N_IOUNIT) = all_nodes( IO_NODE_ID, : )

type(datatype) :: mydata
integer teamdata(IOUNIT_SIZE)[*]
!$xmp coarray on iounit:: teamdata

!!------------------------------- execution
mydata = get_mydata()
sync all

!!------------------------------- gather mydata->teamdata
!$xmp task on iounit                      !! I/O unit毎の実行開始
  me = this_image()                      !! I/O unit内の自分の位置
  teamdata(me)[IO_NODE_ID] = mydata      !! I/O unit毎にデータを集める
!$xmp end task
sync all

!!------------------------------- output teamdata
!$xmp task on ionodes                     !! I/O nodeだけの実行開始
  write(*,100) this_image(), teamdata    !! データの出力
!$xmp end task
```



内容

- Omni XMP Coarray Fortran これまでの成果
- メモリ管理系Ver.4
- Coarray機能のtask構文対応
- まとめ

Coarray実装状況（対Fortran2008）

sec.	contents	stat.
3	declarations	
	• codimension	✔☐
	• initialization	not yet
	• allocatable attribute	✔☐
4	reference and definition	
	• ref. of coindexed obj.	✔☐
	• def. of coindexed var.	✔☐
5	coarray dummy argument	
	• static coarray	✔☐
	• allocatable coarray	✔☐
9	allocation and deallocation	
	• allocate stmt.	✔☐
	• deallocate stmt.	✔☐
	• automatic deallocation	✔☐
10	structure and component	
	• derived type coarray	✔☐
	• allocatable comp. of coarray	not yet
	• pointer comp. of coarray	not yet
	• coarray component	not yet

sec.	contents	stat.
12	synchronization	
	• sync all stmt.	✔☐
	• sync images stmt.	✔☐
	• sync memory	✔☐
	• lock/unlock	not yet
	• critical section	not yet
	• stat= & errmsg= spec.	not yet☐
13	control	
	• termination	✔☐
	• error stop stmt.	not yet
15	intrinsic procedures	
	• image_index, cobound	✔☐
	• num_images, this_image	✔☐
	• atomic_define/ref	not yet

以下の文献の章立てで分類した.
John Reid. *Coarrays in the next Fortran Standard*, ISO/IEC JTC1/SC22/WG5 N1824, April 21, 2010

まとめ

- Omni XMP Coarray Fortranのこれまでの成果
  - Himenoで，他のフリーのCoarray処理系を上回る性能
    - トランスレータ方式は，Fortranを選択できることが利点．
  - NAS Parallelで，MPI-CAF移植版が元のMPI版と同等の性能
    - CAFプログラムの性能チューニングに成功．
- メモリ管理系Ver.4
  - Ver.3で計算部分が遅くなる性能問題を解決
    - Ver.3ではCoarray変数がポインタに変換されていた．
  - FJ-RDMA用（京，FX-10向け）を近々リリース
    - MPI-3用も開発中．GASNet用にはこの改善は使えない．
- タスク並列対応
  - 関連仕様をサポート．Task構文中でCoarrayが参照できる．
  - Fiber NTChem と NAS Parallel BT, SP での利用に期待