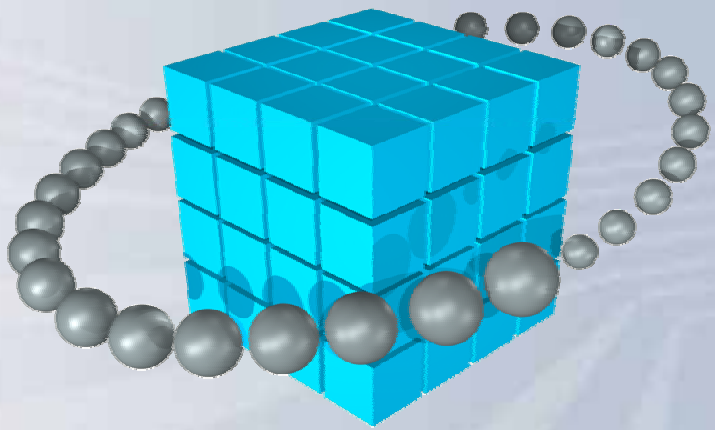
Co-Array Fortran 文法概説

Cray Japan, Inc.



CRAY
THE SUPERCOMPUTER COMPANY

前書き

本プレゼンテーションは2006**年**10**月**31**日に東京大学山上会館で行われた『**Fortran Day@**東京大学**』での講演、**

Dr. John Reid, ISO Fortran Convener
“Co-arrays to be included in the Fortran 2008 Standard”

を基に作成しています

背景と目的

- Co-ArrayはCrayが開発し、T3E, X1/X1Eで実装している分散メモリ型プログラミングのためのFortran拡張文法
- ISO Fortran CommmitteeはCo-Arrayを次期標準言語規格（Fortran 2008）として採用することを決定
- このプレゼンテーションでは、
  - Co-Array機能の説明
  - 分散メモリ型プログラミングの効率的かつ容易な開発
  - 高速なデータ転送
  - プログラムのメンテナンス容易性

  に関して説明
- Fortran2003に関する予備知識は必要としない Fortran95の範囲内で説明

Co-Arrayモデル

- SPMD – Single Program Multiple Data
- **イメージと呼ばれる実行単位複数から構成される**
- **イメージは1から始まる連続した番号付け** 1, 2, 3, …, np
- **実行中のイメージ数は固定**
- **各イメージはローカル変数と呼ばれる独自データセットを持つ**
- **イメージは明示的に同期を取らない限り、非同期実行される**
  sync_all, sync_team, notify, query
- Co-Array**として宣言されたデータは[ ]で括られるインデックスの指定により、他のイメージからアクセス可能**
- **組込み関数**: `this_image, num_images`…
- **集積演算**: `co_sum`…
- `critical`**コンストラクト**

Co-Array文法の例

```
real :: r[*]    ! Scalar co-array
real :: x(n)[*] ! Array  co-array
! Co-arrays always have assumed
! co-size (equal to number of images)

real :: t       ! Local scalar
integer :: p    ! Local scalar

! Remote array references
t = r[p]
x(:) = x(:)[p]

! Reference without [] is to local part
x(:)[p] = r
```

実装モデル

- 通常、各イメージは1プロセッサに1つづつ配置される
- しかし、幾つかのイメージがプロセッサを共有する (e.g. デバッグ) 場合や、一つのイメージがマルチ・プロセッサ上に配置される (e.g. OpenMPとの併用) 場合がある
- Co-Arrayは全てのイメージ上で同じデータサイズなので、コンパイラは各イメージ中のアドレスが同じになるようにデータ配置をアレンジすることが可能
- 共有メモリマシンでは、Co-Arrayは単一の巨大な配列として実装することが可能
- どの様なマシン上でも、他のイメージ上のCo-Arrayデータアドレスを、各イメージで独自に算出可能となるように実装することができる

同期

- 幾つかの例外を除いて、各イメージは非同期的に実行される
- もし、同期が必要であれば、明示的に同期を指示する
  - 全イメージの同期: `sync_all()`
  - 同一チーム内の同期: `sync_team(team)`
  - バリアのマーク付け (同期無し): `notify(image-set)`
  - 他イメージが`notify`を呼ぶのを待つ: `query(image-set)`
- 例:イメージ1でデータを読み、他のイメージへ分配

```
if(this_image()==1) read(*,*)p
sync_all()
p=p[1]
```

criticalセクション

- コードのある部分に関して、複数のイメージが同時に実行することが無いように制限する必要がある場合：

```
critical
    p[6]=p[6]+1
    …
end critical
```

集積演算サブルーチン

- 集積演算サブルーチンは標準提供され、暗黙的な同期を含む
- 全てのサブルーチンはオプションでteam引数をとる

  全イメージ上で宣言
  ```
  real :: x[*], y(n)[*]
  real :: sum, sums(n)
  ```
  全イメージ上でのxの総和をsumに代入
  ```
  call co_sum(x,sum)
  ```
  全イメージ上でyの各要素の和を配列sumsに代入
  ```
  call co_sum(y(:),sums(:))
  ```
- その他の集積演算
  - 全ての値が真であるか　co_all
  - 少なくとも一つが真であるか　co_any
  - 真値の数をカウント　co_count
  - 最大値を持つイメージを特定　co_maxloc
  - 最大値　co_maxval
  - 全ての値の積　co_product

動的メモリ確保のCo-Array

- 動的メモリ確保によるCo-Arrayで唯一許される形式はAllocatable文によるものである
- Allocate/Deallocate文では、暗黙的な同期がとられる
  これは、全イメージ上で同じメモリオペレーションを同じ順序で行うことを保障し、データ配置を同一にするためである
  データ配置は全イメージで同一でなければならない
- Co-Arrayの自動変数、Co-Arrayを返り値とする関数を実現するには全イメージで自動的に同期をとる機構が必要となる
  故に、許されていない

Co-ArrayとSAVE

- Allocatableか仮引数でない限り、Co-ArrayはSAVE属性が与えられなければならない
- これはreturn文でCo-Arrayが変数スコープから外れるとき、同期を取らなくてもよい様にするためである

手続き

- [ ]を付けていないCo-Arrayデータのサブオブジェクトを、手続き（サブルーチン/関数）のCo-Array引数として渡すことが可能
  - Fortran95の通常の文法ルールがローカル変数部分には適用され、`co-rank, co-bounds`は新たに定義される
  - `INTERFACE`は明示的でなければならない
  - copy-in, copy-outの機能はない
- 一般手続き参照の文法規則に変更はない
- ピュアファンクション（純関数）にはCo-Array文法を使用することはできない

構造体

- Co-ArrayはAllocatable, Pointerを含む構造体として定義されても良い
- ポインタはローカルイメージ上のデータをポイントしなくてはならない (リモートデータはポイントできない)

```
q=>z[i]%p                ! これは不可
allocate(z[i]%p)         ! これも不可
```

- 構造体メンバーのポインタ操作時に同期は行われず、各イメージは独立して動作する
- 各イメージ毎にサイズが異なるデータを扱う場合に、Co-Array構造体は最適化を損なうことなく、単純かつ強力な機構を実現する

入出力

- チーム (幾つかのイメージの集合) が一つのファイルに対してアクセスするには以下の通り (ローカルバッファリングも可能)
  - ファイルを開く

    OPEN(unit,…,TEAM=team,…)

    暗黙的にteam内の同期sync_team(team)がとられる

    このunitはteam外のimageから開くことはできない

- Co-Array入出力は下記場合に可能
  - Sequential Write: あるイメージがレコードを書き込んでいるとき、他のイメージはブロックされるため、各レコードは一つのイメージから書き込まれることが保障される
  - Direct Access: 一つのレコードに対して、複数のイメージからのアクセスをプログラマが制御

最適化

- 殆どの場合、コンパイラはCo-Arrayをローカル変数であるかの如く最適化することが可能 (キャッシュ適用、レジスタウインドの適用、プリフェッチ…)
- 同期をとるべき場所以外では、データのコヒーレンシを保障する必要はない
- データ通信は代入文として記述されるため、コンパイラがデータ通信を最適化する余地がある

MPIとの比較 (1)

- MPIは分散メモリ型並列プログラミングの*事実上*の標準であるものの、とても厄介な代物である
- 以下の例はAHPCRC-NCSIのJef Dawsonによる比較である
  - Co-Arrayで、あるイメージから他のイメージへ配列のm要素を通信

```
real :: a(n)[*]
if (this_image().eq.2) a(1:m)=a(1:m)[1]
sync_all()
```

  - MPIの場合

```
real :: a(n)
call MPI_Comm_Rank(COMM,myrank,ierr)
if (myrank.eq.0)
  call MPI_Send(a,m,MPI_FLOAT,1,tag,COMM,ierr)
if(myrank.eq.1)
  call
  MPI_Recv(a,m,MPI_FLOAT,0,tag,COMM,stat,ierr)
```

MPIとの比較 (2)

- 多数のプロセッサを使用する分散メモリ型並列処理において、Co-Arrayは性能向上のための多くの利点があることが、Cray X1上により実証されている
- 例えばDawson(2004)は60プロセッサを使用し、ステンシルアップデート処理を行う場合のMPIとの比較を報告している

Dawson, Jef(2004). *"Co-array Fortran for productivity and performance."* In Army HPC Research Center Bulletin, 14, 4.

Co-Arrayの利点

- コードを書きやすい　ー　コンパイラがデータ通信を認識
- ローカル変数の参照が明らか
- コードをメンテナンスし易い　ー　MPIより直感的で判りやすい
- Fortran言語標準に統合される　ー　型認識、型変換…
- コンパイラがデータ通信を最適化可能
- ローカル変数に対する最適化が適用可能
- コンパイラに過度の負担をかけない
  e.g. データコヒーレンシ

参考文献

Numerich, Robert W. and Reid, John (2005).
“Co-Arrays in the next Fortran Standard.”
ACM Fortran Forum, 24, 2, 4-17.

ftp://ftp.nag.co.uk/sc22wg5/N1601-N1650/N1642.pdf


CRAY
CRAY XT4
SCALABLE COMPUTING AT WORK
AMD 64