XcalableACC概要

村井 均 (理研)

はじめに

- アクセラレータ・クラスタ (e.g. GPUクラスタ) の普及
  - TOP500のトップ10のうち8つ
- MPI+CUDA/OpenACCによるプログラミングが主流
  - 生産性の問題 (難しい)
- 指示文ベースのクラスタ向けプログラミングモデル
- 指示文ベースのアクセラレータ向けプログラミングモデル

➡ XcalableMPとOpenACCを組み合わせよう！

➡ *XcalableACC*

もくじ

1. クラスタ向け言語XcalableMP
2. アクセラレータ向け言語OpenACC
3. アクセラレータ・クラスタ向け言語XcalableACC

もくじ

1. クラスタ向け言語XcalableMP
2. アクセラレータ向け言語OpenACC
3. アクセラレータ・クラスタ向け言語XcalableACC

並列プログラミング言語XcalableMP

- 次世代並列プログラミング言語検討委員会 / PCクラスタコンソーシアムXcalableMP規格部会で検討中。
- MPIに代わる並列プログラミングモデル
- 目標:
  - Performance
  - Expressiveness
  - Optimizability
  - Education cost

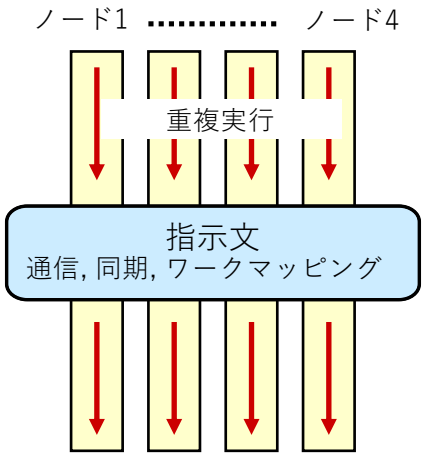
www.xcalablemp.org

XMPの特徴

- Fortran/Cの拡張 (指示文ベース)
  → 逐次プログラムからの移行が容易
- SPMDモデル
  - 各ノード (並列実行の主体) が独立に (重複して) 実行を開始する。
- 明示的な並列化と通信
  - ワークマッピング (並列処理)、通信および同期は「集団的」な指示文によって明示される → チューニングが容易
- 2つのプログラミングモデル
  - グローバルビュー
  - ローカルビュー

XMPの実行モデル (SPMD)

- 各ノードは、同一のコードを独立に (重複して) 実行する。
- 指示文の箇所では、全ノードが協調して動作する (集団実行)。
  - 通信・同期
  - ワークマッピング（並列処理）



XMPのメモリモデル

- 各ノードは、自身のローカルメモリ上のデータ (ローカルデータ) のみをアクセスできる。
- 他のノード上のデータ (リモートデータ) にアクセスする場合は、特殊な記法による明示的な指定が必要。
  - 通信指示文
  - coarray
- 「分散」されないデータは、全ノードに重複して配置される。

プログラム例（MPIとの比較）

XMP/Cプログラム

```
int array[MAX];
#pragma xmp nodes p(*)
#pragma xmp template t(0:MAX-1)
#pragma xmp distribute t(block) onto p
#pragma xmp align array[i] with t(i)

main(){
#pragma xmp loop on t(i) reduction(+:res)
   for (i = 0; i < MAX; i++){
      array[i] = func(i);
      res += array[i];
   }
}
```

MPIプログラム

```
int array[MAX];

main(int argc, char **argv){
    MPI_Init(&argc, &argv);
    MPI_Comm_rank(MPI_COMM_WORLD, &rank);
    MPI_Comm_size(MPI_COMM_WORLD, &size);

    dx = MAX/size;
    llimit = rank * dx;
    if (rank != (size -1)) ulimit = llimit + dx;
    else ulimit = MAX;
    temp_res = 0;

    for (i = llimit; i < ulimit; i++){
       array[i] = func(i);
       temp_res += array[i];
    }

    MPI_Allreduce(&temp_res, &res, 1, MPI_INT,
                  MPI_SUM, MPI_COMM_WORLD);

    MPI_Finalize( );
}
```

XMPのグローバルビュー・プログラミング

- 解くべき問題全体を記述し、それをN個のノードが分担する方法を示す。
  - 「問題1～100を4人で分担して解け」

ローカルビュー: 各ノードが解くべき問題を個別に示す。「ノード1は問題1～25を解け。ノード2は……」

- 分かりやすい (基本的に指示文を挿入するだけ)。
- 「分担」を指定する方法
  - データマッピング
  - ワークマッピング
  - 通信・同期

XMP指示文の記法

- XMPの指示文は、「#pragma xmp」または「!$xmp」から始まる。

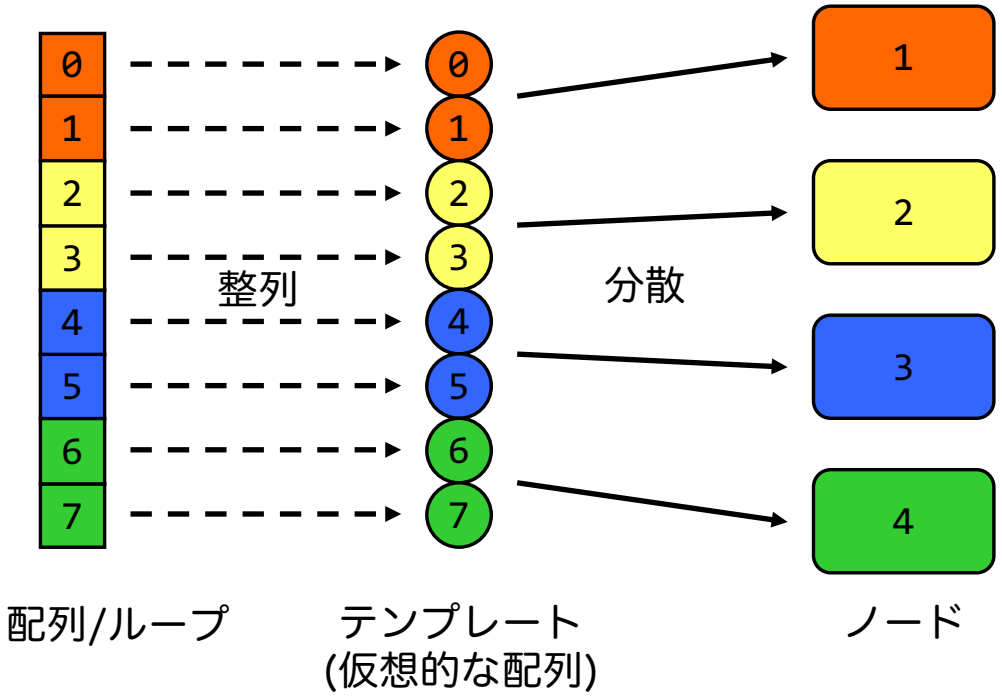
例

[C]
```
#pragma xmp align a[i] with t[i]
```

[F]
```
!$xmp align a(i) with t(i)
```

XMPのデータマッピング

- 整列 + 分散による2段階の処理



・配列はテンプレートに整列され、
・テンプレートはノードに分散される。

データマッピング指示文 (1) nodes指示文

- XMPプログラムの実行者である「ノード」のサイズと形状を宣言する。
  - データやワークを割り当てる対象。
  - プロセッサ (マルチコア可) とローカルメモリから成る。
- 実際のpのサイズを、実行時に決めることもできる。
  - mpiexecコマンドの引数など。

[C]
```
#pragma xmp nodes p[4][4]
```

[F]
```
!$xmp nodes p(4,4)
```

[C]
```
#pragma xmp nodes p[*]
#pragma xmp nodes p[*][4]
```

[F]
```
!$xmp nodes p(*)
!$xmp nodes p(4,*)
```

最後の次元に「*」を指定できる。

データマッピング指示文 (2) template指示文

- XMPプログラムの並列処理の基準である「テンプレート」のサイズと形状を宣言する。
  - データやワークの整列の対象。

[C]
```
#pragma xmp template t[64][64]
```

[F]
```
!$xmp template t(64,64)
```

データマッピング指示文 (3) distribute指示文

- ノード集合pに、テンプレートtを分散する。

[C]
```
#pragma xmp distribute t[block] onto p
```

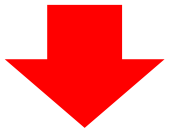
[F]
```
!$xmp distribute t(block) onto p
```

分散形式として、ブロック、サイクリック、ブロックサイクリック、不均等ブロックを指定できる。

データマッピングの例

例1: ブロック分散

```
#pragma xmp nodes p(4)
#pragma xmp template t(0:19)
#pragma xmp distribute t(block) onto p
```



ノード	インデックス
p(1)	0, 1, 2, 3, 4
p(2)	5, 6, 7, 8, 9
p(3)	10, 11, 12, 13, 14
p(4)	15, 16, 17, 18, 19

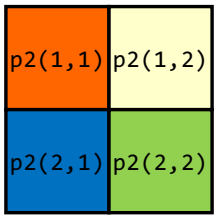
例2: サイクリック分散

```
#pragma xmp nodes p(4)
#pragma xmp template t(0:19)
#pragma xmp distribute t(cyclic) onto p
```

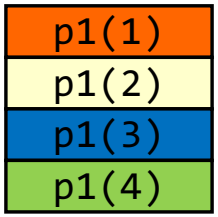
ノード	インデックス
p(1)	0, 4, 8, 12, 16
p(2)	1, 5, 9, 13, 17
p(3)	2, 6, 10, 14, 18
p(4)	3, 7, 11, 15, 19

多次元テンプレートの分散

```
#pragma xmp nodes p2(2,2)
#pragma xmp distribute t(block,block) onto p2
```




```
#pragma xmp nodes p1(4)
#pragma xmp distribute t(block,*) onto p1
```



「*」は非分散を意味する。

データマッピング指示文 (4) align指示文

- 配列aの要素iを、テンプレートtの要素iに整列させる。

[C]
```
#pragma xmp align a[i] with t[i]
```

[F]
```
!$xmp align a(i) with t(i)
```

- 多次元配列も整列可能。

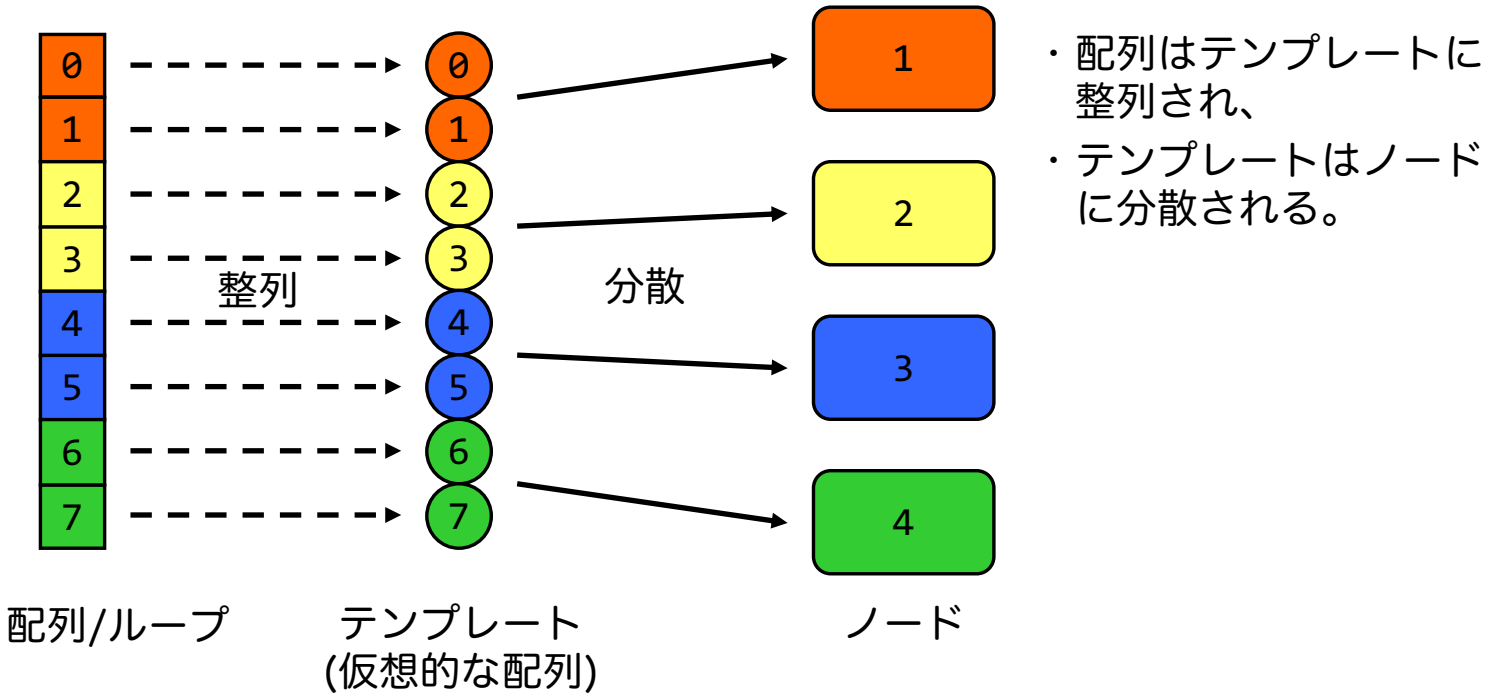
[C]
```
#pragma xmp align a[i][j] with t[i][j]
```

[F]
```
!$xmp align a(i,j) with t(i,j)
```

XMPのデータマッピン

```
#pragma xmp nodes p(4)
#pragma xmp template t[8]
#pragma xmp distribute t[block] onto p
float a[8];
#pragma xmp align a[i] with t[i]
```

- 整列 + 分散による2段階の処理



・配列はテンプレートに整列され、
・テンプレートはノードに分散される。

ワークマッピング指示文 (1) loop指示文

- ループの並列化を指示する。
  - t(i,j)を持つノードが、繰り返し(i,j)においてa[i][j]への代入を実行する。

```
#pragma xmp loop (i,j) on t[i][j]
for (i = 0; i < n; i++)
for (j = 0; j < n; j++)
   a[i][j] = ...;
```

loop指示文 (続き)

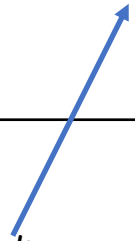
- アクセスされるデータが、その繰り返しを実行するノードに割り当てられていなければならない。
  - 下の例では、t(i,j)を持つノードが、a[i][j]を持たなければならない。
  - そうでない場合、事前に通信を行っておく。

```
#pragma xmp loop (i,j) on t[i][j]
for (i = 0; i < n; i++)
for (j = 0; j < n; j++)
   a[i][j] = ...;
```

loop指示文 (続き)

- reduction節
  - 並列ループの終了時に、各ノードの値を「集計」する。
  - 提供している演算は+, max, minなど。

```
#pragma xmp loop (i) on t(i) reduction(+:sum)
   for (i = 0; i < 20; i++)
      sum += i;
```



各ノード上のsumの値(部分和)を合計した
値で、各ノード上のsumを更新する。

ワークマッピング指示文 (2) task指示文

- 直後の処理を、指定したノードが実行する。

```
#pragma xmp task on p(1)
{
   func_a();
}

#pragma xmp task on p(2)
{
   func_b();
}
```

ノードp(1)が
func_aを実行する。

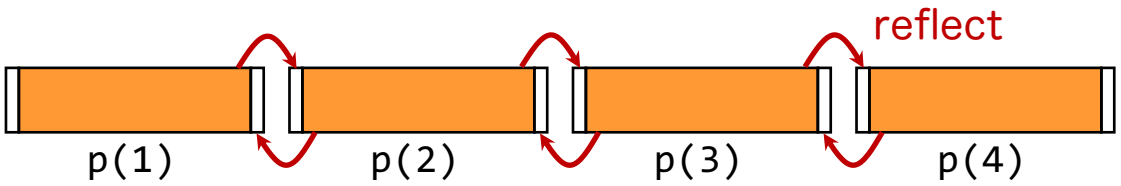
ノードp(2)が
func_bを実行する。

通信指示文 (1) shadow/reflect指示文

aの上下端に幅1のシャドウを付加する。

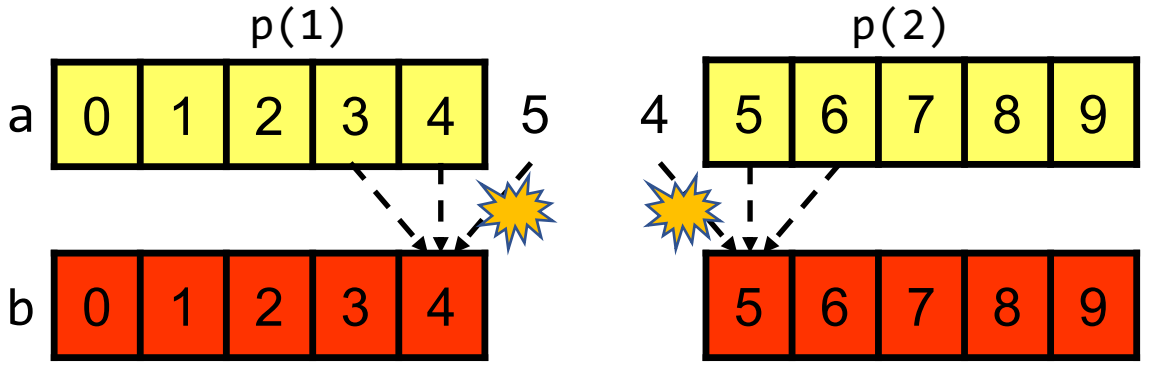
```
#pragma xmp distribute t(block) onto p
#pragma xmp align a[i] with t(i-1)
#pragma xmp shadow a[1:1]
...
#pragma xmp reflect (a)
```

aに対する隣接通信を実行する。



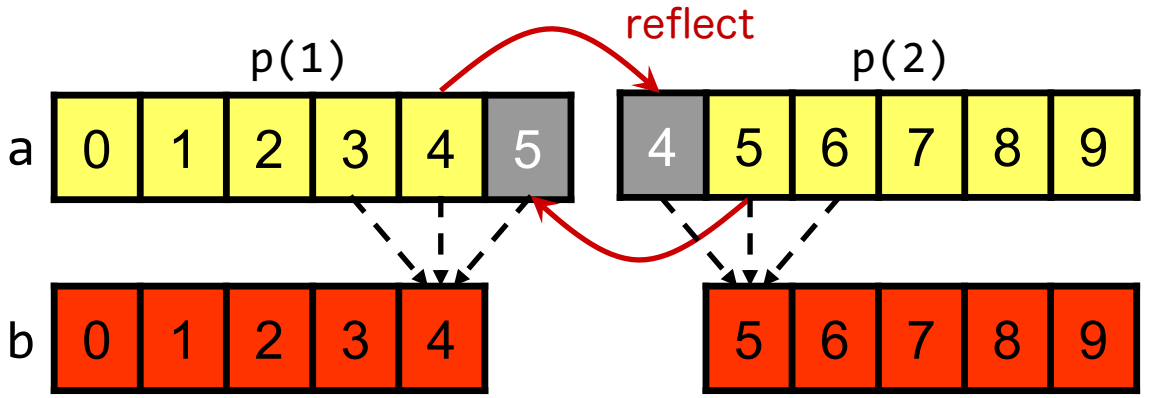
shadow/reflect指示文の例

```
#pragma xmp loop on t(i)
for (i = 1; i < 9; i++)
   b[i] = a[i-1] + a[i] + a[i+1];
```



shadow/reflect指示文の例

```
#pragma xmp shadow a[1:1]

#pragma xmp reflect (a)
#pragma xmp loop on t(i)
for (i = 1; i < 9; i++)
   b[i] = a[i-1] + a[i] + a[i+1];
```

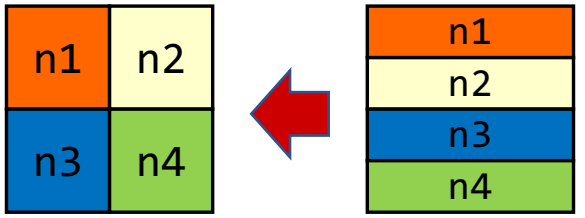


通信指示文 (2) gmove指示文

- 通信を伴う任意の代入文を実行する。

```
#pragma xmp gmove
a[:][:] = b[:][:];
```

※ Cで「部分配列」も記述できる。



a[block][block]　　b[block][*]

通信指示文 (3)

- bcast指示文
  - 特定のノードが、指定したデータを他のノードへブロードキャストする (ばらまく)。

```
#pragma xmp bcast (s) from p(1)
```

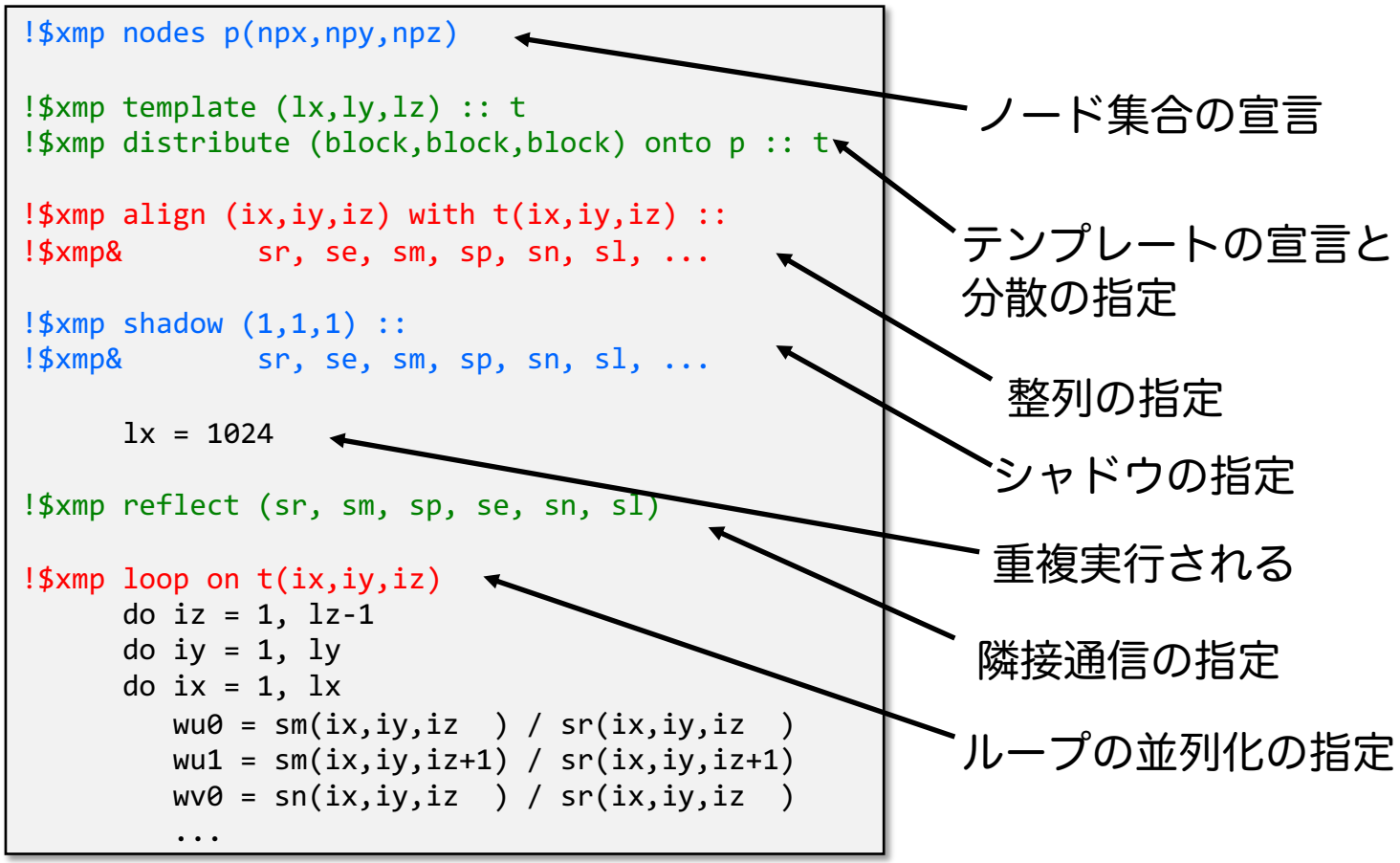
※ from p(1) は省略可

- barrier指示文
  - ノードが互いに待ち合わせる (バリア同期)。

```
#pragma xmp barrier
```

XcalableMPプログラムの例

```
!$xmp nodes p(npx,npy,npz)

!$xmp template (lx,ly,lz) :: t
!$xmp distribute (block,block,block) onto p :: t

!$xmp align (ix,iy,iz) with t(ix,iy,iz) ::
!$xmp&        sr, se, sm, sp, sn, sl, ...

!$xmp shadow (1,1,1) ::
!$xmp&        sr, se, sm, sp, sn, sl, ...

      lx = 1024

!$xmp reflect (sr, sm, sp, se, sn, sl)

!$xmp loop on t(ix,iy,iz)
      do iz = 1, lz-1
      do iy = 1, ly
      do ix = 1, lx
         wu0 = sm(ix,iy,iz  ) / sr(ix,iy,iz  )
         wu1 = sm(ix,iy,iz+1) / sr(ix,iy,iz+1)
         wv0 = sn(ix,iy,iz  ) / sr(ix,iy,iz  )
         ...
```



もくじ

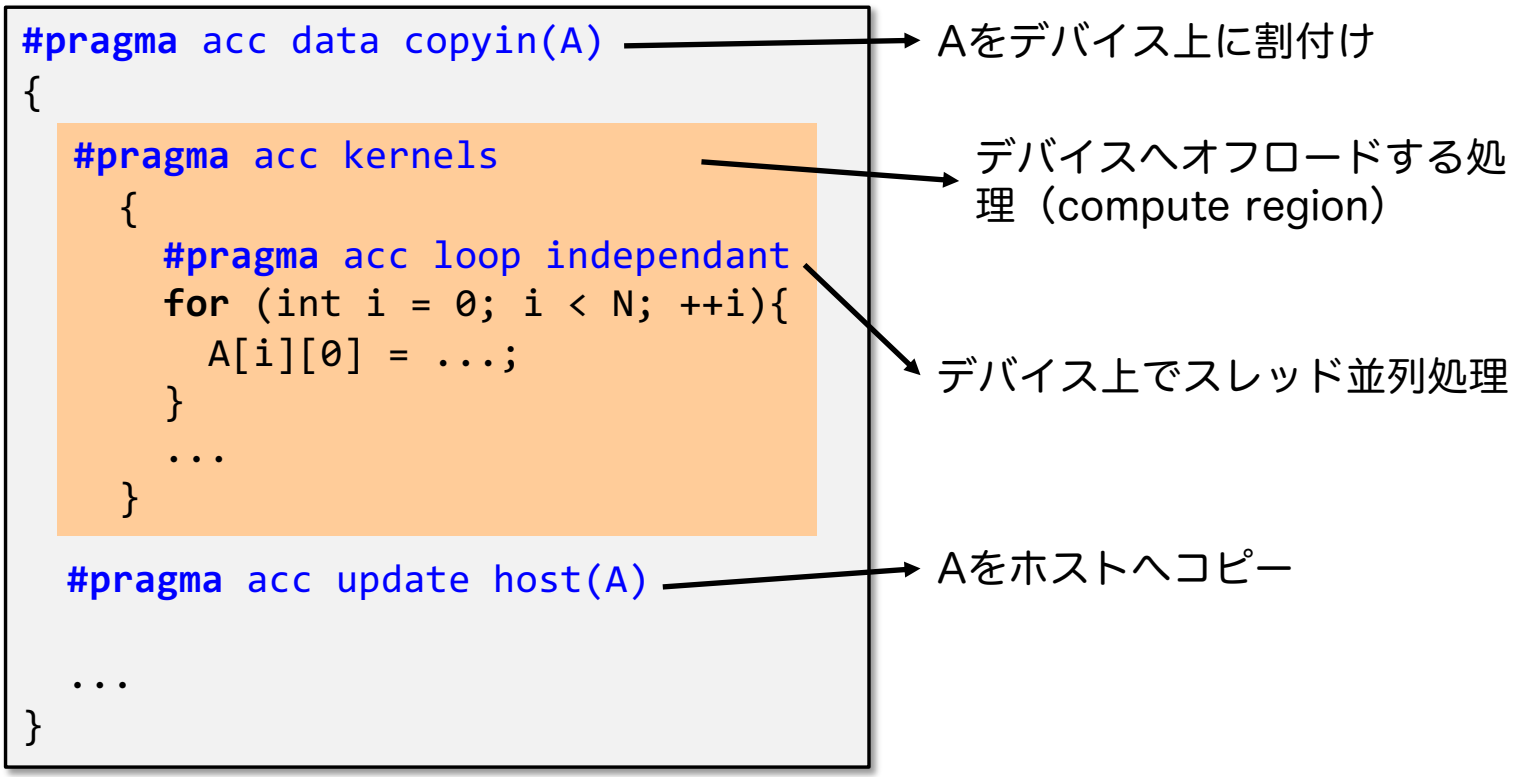
1. クラスタ向け言語XcalableMP
2. アクセラレータ向け言語OpenACC
3. アクセラレータ・クラスタ向け言語XcalableACC

OpenACC

- オフロードモデルに基づくアクセラレータのためのプログラミングモデル
  - ホスト(CPU)が、データおよびワークをデバイスに「オフロード」する。
- 指示文ベース
- C/C++/Fortran
- 最新版 Version 2.7 (Nov. 2018)
- Cray, (CAPS), PGI (NVIDIA)がサポート。

cf. OpenMP4.0以降で同様の機能が導入されている (デバイス構文)。

サンプルコード


#pragma acc data copyin(A)
{
#pragma acc kernels
{
#pragma acc loop independant
for (int i = 0; i < N; ++i){
A[i][0] = ...;
}
...
}
#pragma acc update host(A)
...
}
Aをデバイス上に割付け
デバイスへオフロードする処理（compute region）
デバイス上でスレッド並列処理
Aをホストへコピー

OpenACC指示文 (1)

- parallel/kernels構文
  - デバイスにオフロードするべきコード(compute region)を指定する。
  - parallel ― 全体を一つのカーネルとして、内部の個々のループの並列化はloop構文を用いて明示する。
  - kernels ― コンパイラが複数のカーネルに分割する。loop構文で明示することも可能。
- loop構文
  - 直後のループをデバイス上で並列実行する方法(gang、worker、vectorの3つの並列性※)を指定する。

※ NVidia GPUの場合: gang⇔thread block、worker⇔warp, vector⇔thread

OpenACC指示文 (2)

- data構文
  - 直後のブロックにおいて、指定したデータをデバイス上に配置することを指定する。
  - ブロックの入口と出口における、データ転送の方法を指定できる（e.g. `copy`, `copyin`, `copyout`)
- update指示文
  - ホスト上のデータを、対応するデバイス上のデータの値で更新する。

もくじ

1. クラスタ向け言語XcalableMP
2. アクセラレータ向け言語OpenACC
3. アクセラレータ・クラスタ向け言語XcalableACC

XcalableACC

- ポストペタCREST「ポストペタスケール時代に向けた演算加速機構・通信機構統合環境の研究開発(～2017年度)」の成果
  - FPGAベースの通信デバイス*PEACH2* による *Tightly Coupled Accelerator (TCA)* アーキテクチャを提案。
- アクセラレータ・クラスタのための新しいプログラミング言語
  - (多階層並列性)
  - デバイス間直接通信
- Omni XcalableACCコンパイラの開発

➡ 高性能と高生産性の両方を実現

XACCの基本方針（1）

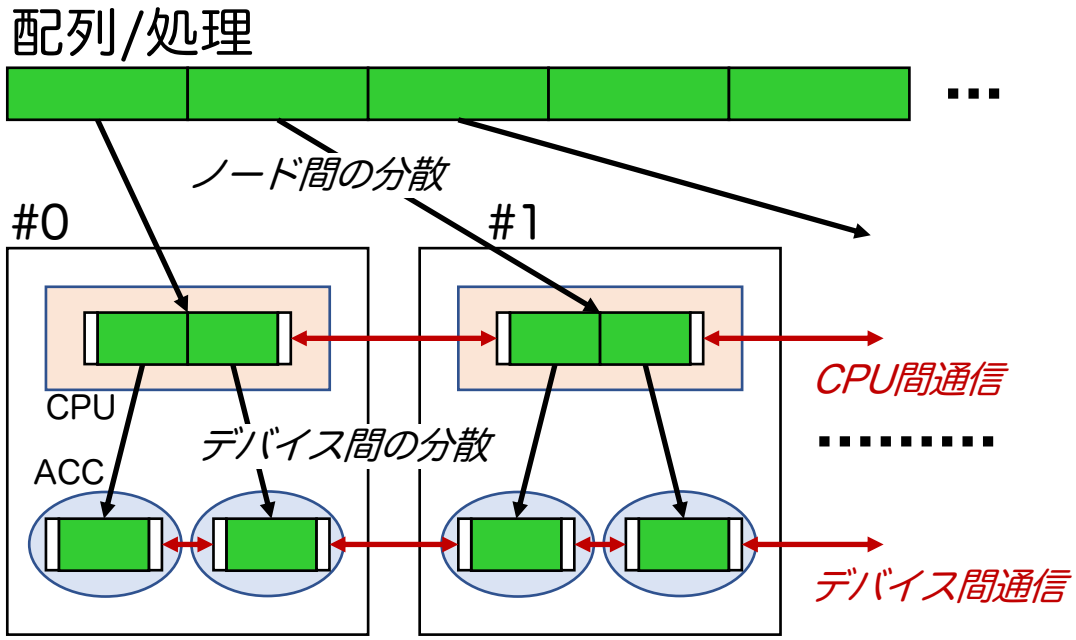
- XACC = XMP + OpenACC※ + XACC拡張

※ OpenACC1.0および2.0の一部

XMP指示文	ノード間の分散メモリ並列処理
OpenACC指示文	デバイスへのオフロード処理
XACC拡張	階層並列性、デバイス間直接通信

- デバイス間直接通信
  - デバイスメモリ上のデータに対するノード間通信を指定可能。
  - cf. デバイスデータに対する通常の通信: デバイス0 →(データ転送)→ CPU0 →(ノード間通信)→ CPU1 →(データ転送)→ デバイス1

XACCの実行モデル



XMP通信指示文の拡張 (acc節)

- acc節を伴うXMP通信指示文は、デバイス上のデータを対象にする。

```
#pragma xmp reflect (a) acc
```

※ acc節がない場合は、処理系が選択する。

- acc節を伴うbarrier指示文は、デバイス間のバリア同期を行う。

```
#pragma xmp barrier acc
```

サンプルコード (HIMENO)

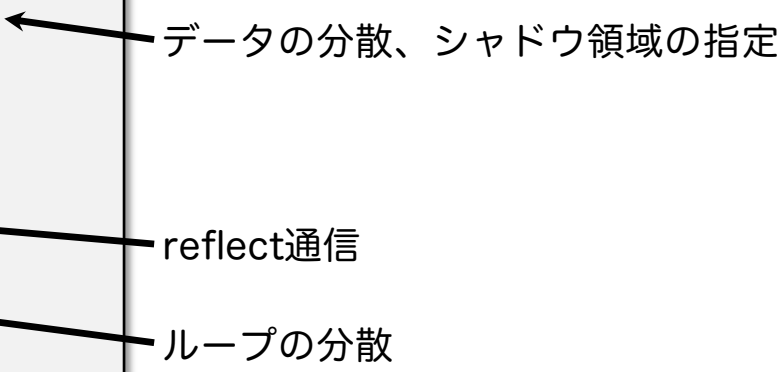
逐次コード

```
float p[MIMAX][MJMAX][MKMAX];




...

...


for(i=1 ; i<MIMAX ; ++i)
  for(j=1 ; j<MJMAX ; ++j){
    for(k=1 ; k<MKMAX ; ++k){
      S0 = p[i+1][j][k] * ..;
```

サンプルコード (HIMENO)

XMPコード

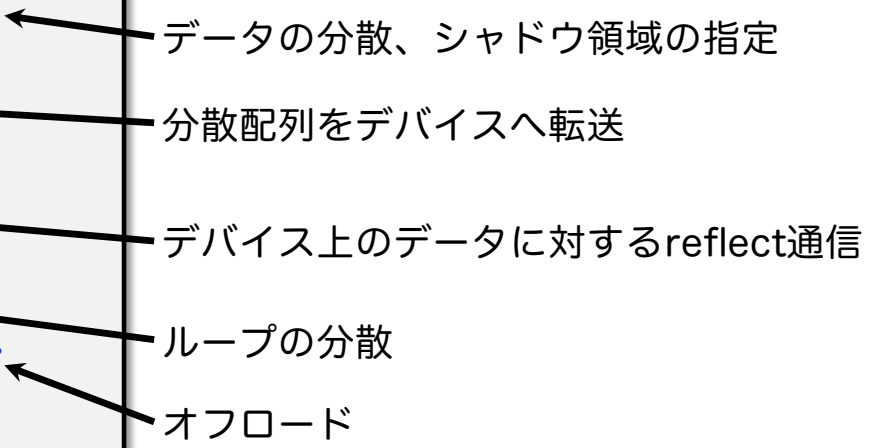
```
float p[MIMAX][MJMAX][MKMAX];
#pragma xmp align p[i][j][k] with t[i][j][k]
#pragma xmp shadow p[1:1][1:1][0]




...
#pragma xmp reflect (p)
...
#pragma xmp loop (k,j,i) on t(k,j,i)

for(i=1 ; i<MIMAX ; ++i)
  for(j=1 ; j<MJMAX ; ++j){
    for(k=1 ; k<MKMAX ; ++k){
      S0 = p[i+1][j][k] * ..;
```



サンプルコード (HIMENO)

XACCコード

```
float p[MIMAX][MJMAX][MKMAX];
#pragma xmp align p[i][j][k] with t[i][j][k]
#pragma xmp shadow p[1:1][1:1][0]

#pragma acc data copy(p) ..
{
...
#pragma xmp reflect (p) acc
...
#pragma xmp loop (k,j,i) on t(k,j,i)
#pragma acc parallel loop collapse(3) ...
for(i=1 ; i<MIMAX ; ++i)
  for(j=1 ; j<MJMAX ; ++j){
    for(k=1 ; k<MKMAX ; ++k){
      S0 = p[i+1][j][k] * ..;
```

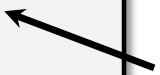


サンプルコード (HIMENO)

OpenACCコード

```
float p[MIMAX][MJMAX][MKMAX];



#pragma acc data copy(p) ..
{
...

...

#pragma acc parallel loop collapse(3) ...
for(i=1 ; i<MIMAX ; ++i)
  for(j=1 ; j<MJMAX ; ++j){
    for(k=1 ; k<MKMAX ; ++k){
      S0 = p[i+1][j][k] * ..;
```

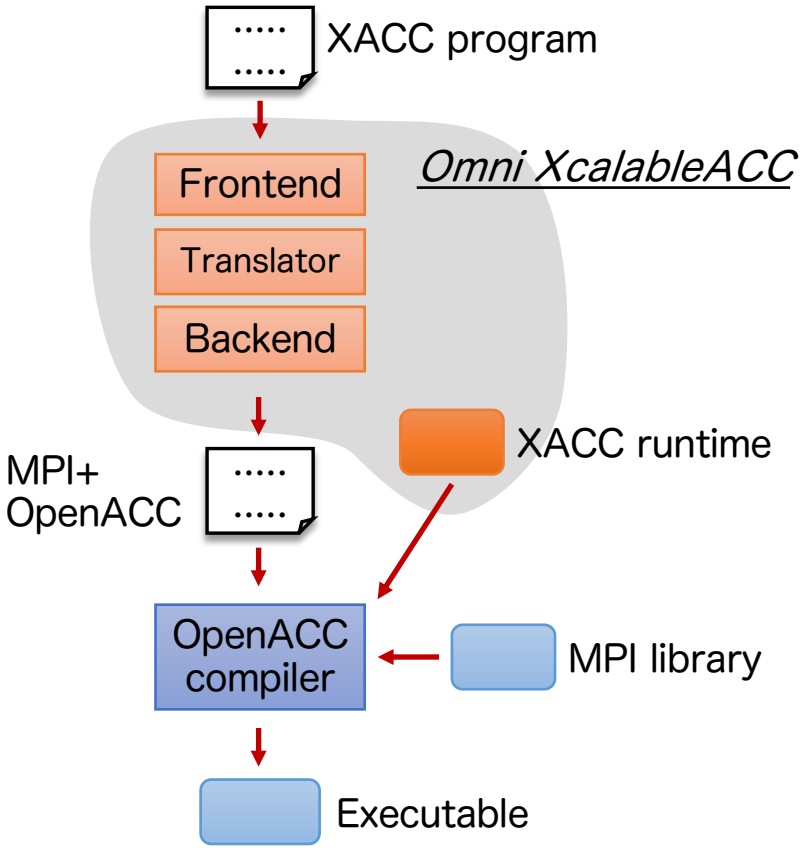
分散配列をデバイスへ転送



オフロード

Omni XcalableACC

- Omni XMPの「拡張機能」として開発
- ソース-to-ソースのトランスレータ
- バックエンドとして、任意のOpenACCコンパイラ (e.g. PGI) を用いる。
  ※ Omni自身もOpenACCコンパイラとしての機能を持つ。
- XACCのデバイス間直接通信に対する実装を、環境に応じて選択
  - PEACH2のTCA通信
  - GPU Direct RMA



まとめ

- XACC = XMP + OpenACC + XACC拡張
- 指示文ベースで、アクセラレータ・クラスタ向けプログラムを書ける。
- デバイス間直接通信を記述できる。
  - Omni XACCでは、環境に応じて実装を選択。

➡ アクセラレータ・クラスタにおける高性能と高生産性の両方を実現