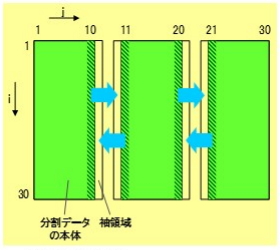
5　通信

XMP は，性能透過性―どこでどのような実行が行われるか利用者に分かりやすい―を重視する言語仕様です．これは，利用者の予期しない性能低下を避けることで，性能チューニングを容易にすることを狙っています．そのため，指示文などによる利用者の明示的な指示がない限り，通信を自動的に生成することはありません．その代わりに，できる限り簡単な記述で通信を指示できるように検討されています．その代表的なものが，Reflect 指示文と，Gmove 指示構文です．

5.1　袖通信の使い方

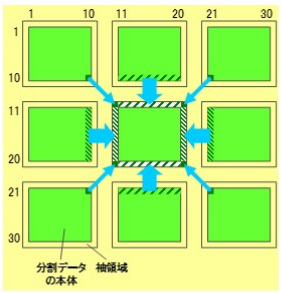
分散配列について Shadow 指示文で袖領域を宣言する方法は 4.2.2 項で説明しました．袖領域に隣接ノードが持つ値をセットするには，Reflect 指示文を使います．下図にその例を示します．Shadow 指示文と Reflect 指示文は，




```
#include<stdio.h>
int main()
{
#pragma xmp nodes p(3)
#pragma xmp template t(30)
#pragma xmp distribute t(block) onto p
      double a[30][30];
#pragma xmp align a[*][j] with t(j)
#pragma xmp shadow a[0][1]
/*** 袖領域に値を設定 ***/
#pragma xmp reflect (a)
int i,j;
/*** ループ内で袖領域を参照 ***/
#pragma xmp loop on t(j)
      for( j=0;j<30;j++)
      for( i=0;i<30;i++)
          a[i][j] = a[i][j]+a[i][j-1]+a[i][j+1];
return 0;
}
```

図 22: プログラム 22

多次元分割の配列についても使うことができます．2 次元分割の例を下に示します．中央のノードは，前後左右と斜め方向に隣接する 8 ノードから袖のデータを受け取りますが，このアクセスパターンに対して Reflect 指示行 1 行だけで記述することができるので，非常に簡単です．

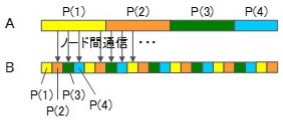



```
#include<stdio.h>
int main(){
#pragma xmp nodes p(3,3)
#pragma xmp template t(30,30)
#pragma xmp distribute t(block,block) onto p
 double a[30][30];
#pragma xmp align a[i][j] with t(i,j)
#pragma xmp shadow a[1][1]
/*** 袖領域に値を設定 ***/
#pragma xmp reflect (a)
/*** ループ内で袖領域を参照 ***/
int i,j;
#pragma xmp loop (i,j) on t(i,j)
      for( j=0; j<30 ;j++)
      for( i=0; i<30 ;i++)
         a[i][j] = a[i][j]+a[i-1][j]+a[i][j+1]+a[i+1][j]+a[i][j-1];
return 0;
}
```

図 23: プログラム 23

5.2 Gmove 指示構文の使い方

Gmove 指示構文だけで様々なパターンの通信を記述できます．下図にその例を示します．

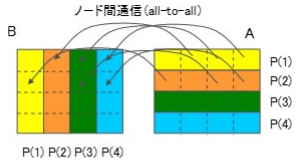



```
#include <stdio.h>

int main()
{
int N=100;
double A[N], B[N];
#pragma xmp nodes P(4)
#pragma xmp template TA(0:N-1)
#pragma xmp template TB(0:N-1)
#pragma xmp distribute TA(block) onto P
#pragma xmp distribute TB(cyclic) onto P
#pragma xmp align A[i] with TA(i)
#pragma xmp align B[i] with TB(i)


#pragma xmp gmove
B[:] = A[:];
return 0;
}
```

図 24: プログラム 24(a)

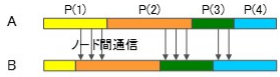



```
#include<stdio.h>

int main()
{
int N=100,M=100;
double A[N][M], B[N][M];
#pragma xmp nodes P(4)
#pragma xmp template T1(0:M-1)
#pragma xmp template T2(0:N-1)
#pragma xmp distribute T1(block) onto P
#pragma xmp distribute T2(block) onto P
#pragma xmp align A[*][i] with T1(i)
#pragma xmp align B[j][*] with T2(j)


#pragma xmp gmove
B[:][:] = A[:][:];
return 0;
}
```

図 25: プログラム 24(b)

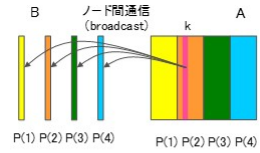



```
#include<stdio.h>

int main()
{
int a[4] = {6,8,4,4};
int b[4] = {3,8,5,6};
double A[22], B[22];
#pragma xmp nodes P(4)
#pragma xmp template TA(0:21)
#pragma xmp template TB(0:21)
#pragma xmp distribute TA(gblock(a)) onto P
#pragma xmp distribute TB(gblock(b)) onto P
#pragma xmp align A[i] with TA(i)
#pragma xmp align B[i] with TB(i)

#pragma xmp gmove
B[:] = A[:];
return 0;
}
```

図 26: プログラム 24(c)




```
#include<stdio.h>

int main()
{
int M=100,N=100;
double A[N][M], B[M];
#pragma xmp nodes P(4)
#pragma xmp template T2(0:N-1)
#pragma xmp distribute T2(block) onto P
#pragma xmp align A[i][*] with T2(i)
#pragma xmp align B[i] with T2(i)

int k=1;
#pragma xmp gmove
B[:] = A[k][:];
return 0;
}
```

図 27: プログラム 24(d)