並列プログラミング入門 MPI版（2012年6月1日版）正誤表

2012年6月1日

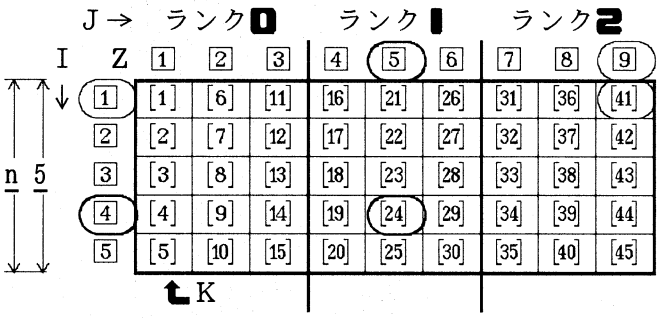
「並列プログラミング入門 MPI版」(2012年6月1日版)に掲載されておらず、今後の改訂版に掲載する予定の項目を本資料に示します。

■３－３－６－１節で説明したMPI_MAXLOCは、最大値の入った位置のデータを1つしか指定できません。位置のデータが2つの場合、通信しながら行う演算を新たに作成する方法を、P3-26で説明しました。以下では別の方法を説明します。

下図(P3-26の図3-3-12(1)と同じ)のように、左上から順に通し番号([1],[2],･･･)を付け、これを変数Kとします。2つの位置のデータの組(I,J)をKに変換する方法を下記の(1)に示します。(1)の下線部のnは、I方向の要素数(本例では5)を示します。変換したKは1つなので、MAXLOCを使用して送信することができます。

送信先では、受信したKを元の(I,J)に復元します。この方法を下記の(2)と(3)に示します。

なお、この方法であれば、図3-3-12(1)のように最大値が複数の場合、下記の[24]と[41]のように位置の大小が決まるので、位置の小さい方の最大値を取ることができます。



(1) (I,J)をKに変換する方法

K = I + (J-1)*n

(例) K = [4] + ([5]-1)*5 = [24]

(2) KからIを復元する方法

I = MOD(K +n-1,n) + 1

(例) I = MOD([24]+5-1,5) + 1 = [4]

(3) KからJを復元する方法

J = (K+n-1)/n (整数の割り算)

(例) J= ([24]+5-1)/5 = [5]

■３－４節で説明した1対1通信ルーチンの指定方法について2点補足します。

(1) 図1の④に示す複数の「CALL MPI_WAIT」を1行にまとめることができます。
(注) 本例では2行の「CALL MPI_WAIT」をまとめるので、以下の(注)の部分は2となります。
●④を図2の⑧の「CALL MPI_WAITALL」に変更し、⑧の最初の引数に(本例では)2(注)を指定します。
●①の代わりに⑤で(本例では)大きさ2(注)の配列IREQ(名前は任意)を宣言し、③,④のIREQ1,IREQ2を⑦,⑧のように配列IREQに変更します。
●②の配列ISTATUSを、⑥のように2次元目の大きさが(本例では)2(注)の2次元配列に変更します。

(2) 図1の③の通信が完了すると、④で、MPIによって、②で宣言した配列ISTATUSに戻り値が設定されますが、通常、④の後で配列ISTATUSを参照することはあまりありません。そのような場合、④の配列ISTATUSの代わりに図3の⑨のように「MPI_STATUS_IGNORE」を指定すると、②の配列宣言は不要となり、④での配列ISTATUSへの設定も行われません。

なお、この機能はMPI-2で提供された機能で、MPI-1しか使用できない環境では使用することができません。MPI-1しか使用できない環境で⑨を指定した場合、「MPI_STATUS_IGNORE」は通常のスカラー変数と見なされますが、⑨では④の配列ISTATUSへの設定と同様に複数の値がMPIによって設定されるため、記憶域保護例外になってしまうので注意して下さい。

図2の⑧のように「CALL MPI_WAITALL」の場合は、図3の⑨の「MPI_STATUS_IGNORE」の代わりに、図4の⑩の「MPI_STATUSES_IGNORE」を指定します(この機能も、前述のようにMPI-1しか使用できない環境では使用することができません)。

なお、図3、図4の機能は、1対1ブロッキング通信ルーチン「CALL MPI_RECV」で配列ISTATUSを指定する引数にも使用することができます。

```
INTEGER IREQ1,IREQ2                          ①
INTEGER ISTATUS(MPI_STATUS_SIZE)             ②
CALL MPI_ISEND(～,IREQ1,IERR)                ③
CALL MPI_IRECV(～,IREQ2,IERR)                ③
CALL MPI_WAIT(IREQ1,ISTATUS,IERR)            ④
CALL MPI_WAIT(IREQ2,ISTATUS,IERR)            ④
```

図1

```
INTEGER IREQ1,IERQ2
CALL MPI_ISEND(～,IREQ1,IERR)
CALL MPI_IRECV(～,IREQ2,IERR)
CALL MPI_WAIT(IREQ1,MPI_STATUS_IGNORE,IERR)      ⑨
CALL MPI_WAIT(IREQ2,MPI_STATUS_IGNORE,IERR)      ⑨
```

図3 (MPI-2のみで使用可能)

```
INTEGER IREQ(2)                              ⑤
INTEGER ISTATUS(MPI_STATUS_SIZE,2)           ⑥
CALL MPI_ISEND(～,IREQ(1),IERR)              ⑦
CALL MPI_IRECV(～,IREQ(2),IERR)              ⑦
CALL MPI_WAITALL(2,IREQ,ISTATUS,IERR)        ⑧
```

図2

```
INTEGER IREQ(2)
CALL MPI_ISEND(～,IREQ(1),IERR)
CALL MPI_IRECV(～,IREQ(2),IERR)
CALL MPI_WAITALL(2,IREQ,MPI_STATUSES_IGNORE,IERR)⑩
```

図4 (MPI-2のみで使用可能)

■３－６節で説明した「新グループの作成」の他の例を紹介します。図1のようにノード内が共有メモリー型で複数のCPU(またはコア)が含まれていて、ノード間はネットワークで結合された分散メモリー型の環境で集団通信ルーチンを実行する場合、全プロセス間で1回の通信を行うよりも、異なるノードのプロセス間の通信と、ノード内の各プロセス間の通信の2回(集団通信ルーチンの種類によっては3回)に分けて行った方が速くなる場合があります。

例えばMPI_BCAST(送信元がランク0)を2回に分けて通信する場合の動作は次の通りです。
(1回目) 図1の①に示すように、ランク0から各ノードの代表プロセス(ランク2,4)に通信します。
(2回目) 図1の②に示すように、各ノードの代表プロセスから、各ノードのその他のプロセスに通信します。

プログラム例を図2に示します。(1)～(5)で、３－６節で紹介したMPI_COMM_SPLIT(付録参照)を使用して、図3、図4に示す2種類のグループを作成します。

●(1)で、ノード内のCPU数(またはコア数)を設定します。
●(2),(3)を実行すると、図3の枠に示すように、(2)のICOLOR1の値が同一のプロセスが同じグループになります。グループ名(正式名はコミュニケータ)を(3)でIGLOBALとします。(3)の3つ目の引数に「0」を指定した場合、グループIGLOBAL内の各プロセスのランク値は、図3の0,1,2となります。
●同様に(4),(5)を実行すると、図4に示すように、(4)のICOLOR2の値が同一のプロセスが同じグループになります。グループ名を(5)でILOCALとします。グループILOCAL内の各プロセスのランク値は、図4の[0],[1]となります。
●図2の⓪のIF文により、図3のICOLOR1が「0」のプロセスのみ、①でグループIGLOBALの各プロセス間で集団通信MPI_BCASTを実行します。その結果、図1の①の通信が行われます。①の4つ目の引数(送信元のプロセスのランク)の「0」は、MPI_COMM_WORLD内のランク0ではなく、図3の○内のランク0を意味します。
●図2の②により、図4の各□内でそれぞれ集団通信MPI_BCASTを実行します。②の4つ目の引数(送信元のプロセスのランク)の「0」は、MPI_COMM_WORLDのランク0ではなく、図4の○内のランク[0]を意味します。

以下に注意点を述べます。
●ランク0～5の値がノード内の各プロセスで連続する場合(例えば0,1)のみ、この方法を使用することができます((2),(4)を指定するため)。
●マシン環境、集団通信ルーチンの種類、メッセージ長によって、効果のある場合とない場合があります。

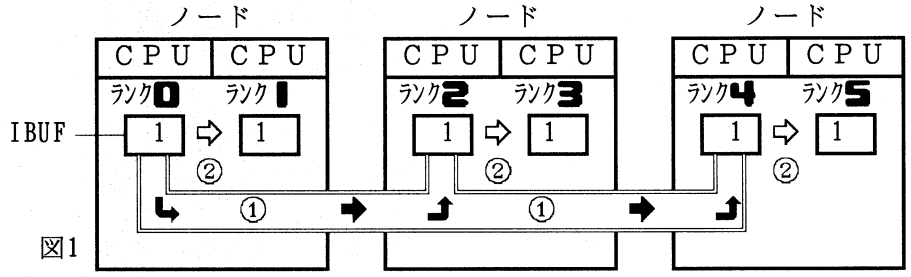


図1

```
   :
 ICPU = 2 (ﾉｰﾄﾞ内のCPU数またはｺｱ数を設定)                    (1)
 ICOLOR1 = MOD(MYRANK,ICPU)                                  (2)
 CALL MPI_COMM_SPLIT(MPI_COMM_WORLD,ICOLOR1,0,IGLOBAL,IERR)  (3)
 ICOLOR2 = MYRANK/ICPU (整数の割り算)                        (4)
 CALL MPI_COMM_SPLIT(MPI_COMM_WORLD,ICOLOR2,0,ILOCAL,IERR)   (5)
 IF (ICOLOR1==0)                                             ⓪
&CALL MPI_BCAST(IBUF,1,MPI_INTEGER,0,IGLOBAL,IERR)           ①
 CALL MPI_BCAST(IBUF,1,MPI_INTEGER,0,ILOCAL ,IERR)           ②
   :
```
図2

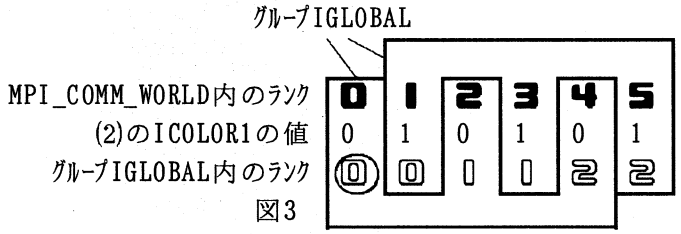


図3

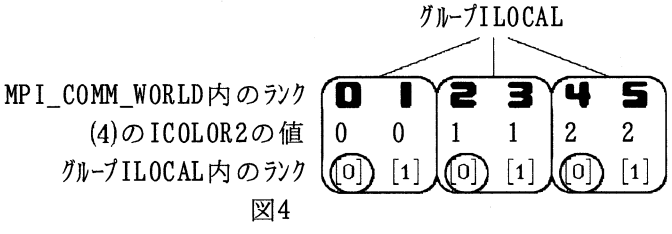


図4

■P4-28の「■ 例３」の三角行列を分割する別の方法を示します。以下の図1の三角行列を4プロセスで分割するとします。★の要素が図2の☆の部分にあると見なして図2のようにブロック分割し、各プロセスが担当する列を決定します。それに基づいて、最終的に図3のように分割を行います。

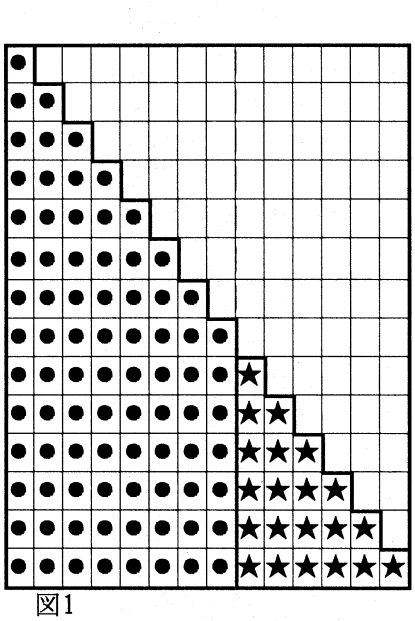


図1

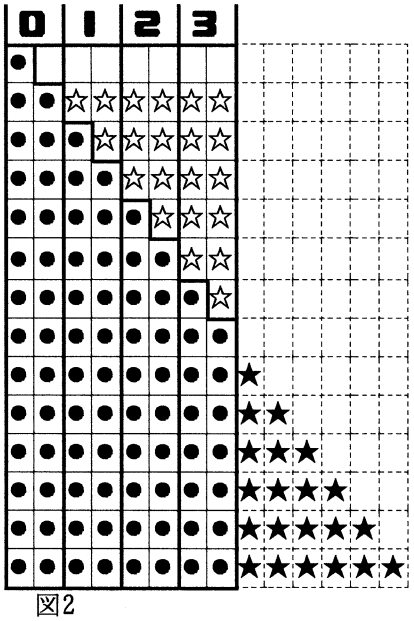


図2

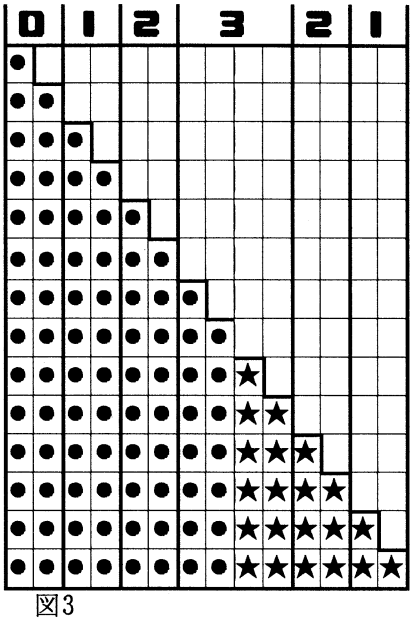
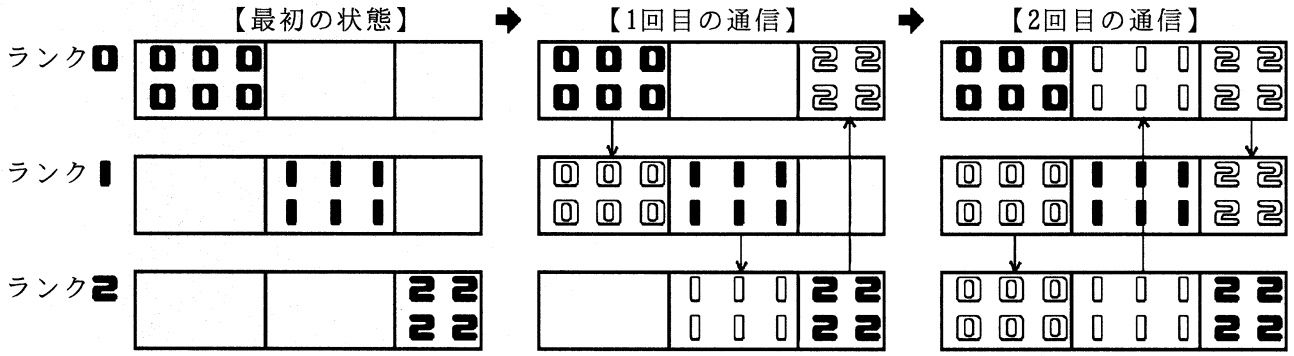


図3

■P4-63とP4-64では、MPI_ALLGATHERVの代替として、MPI_BCASTで通信を行いました。その後、例えば図4-6-11(1)の場合、1対1通信ルーチン(MPI_SENDRECVなど)を利用して、下記のように通信した方が速いことが分かりました。



■P4-108の図4-8-8(5)で、特にプロセスの数が多い場合、2回目のMPI_BARRIERが実行される時刻がプロセス間で若干異なるので、各プロセスの「測定対象部分」の最長の時間が若干不正確になります。この場合、2回目のMPI_BARRIERをやめて、以下の①～③のように、各プロセスの「測定対象部分」の最長の時間をMPI_REDUCEで求めて下さい。

```
  REAL*8 ELP1,ELP2,ELP                                    ①
  CALL MPI_BARRIER(MPI_COMM_WORLD,IERR)
  ELP1 = MPI_WTIME()
  ////////測定対象部分////////
  ELP2 = MPI_WTIME()
  CALL MPI_REDUCE(ELP2-ELP1,ELP,1,MPI_REAL8,
 &    MPI_MAX,0,MPI_COMM_WORLD,IERR)                     ②
  IF (MYRANK==0)
 &    PRINT *,'ELAPSE = ',ELP                            ③
```