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

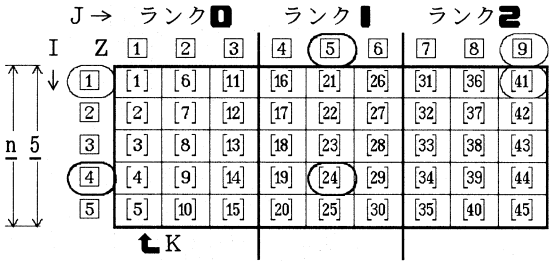
「並列プログラミング入門 MPI 版」(2012 年 6 月 1 日版) に掲載されておらず、今後の改訂版に掲載する予定の項目を本資料に示します。

■ 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、P 38 で説明しました。以下では別の方法を説明します。
下図 (P 40 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 1、2、3 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。

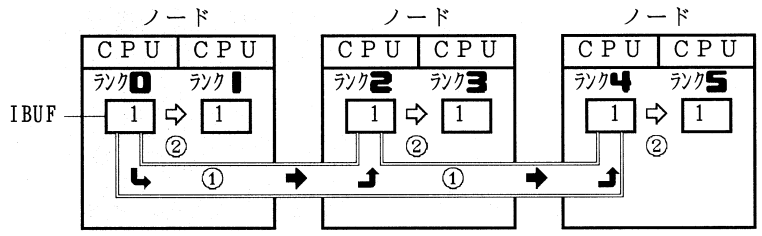


図 1

```
   :
 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

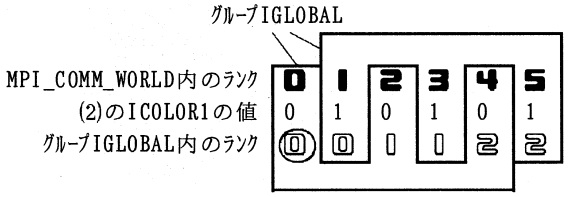


図 3

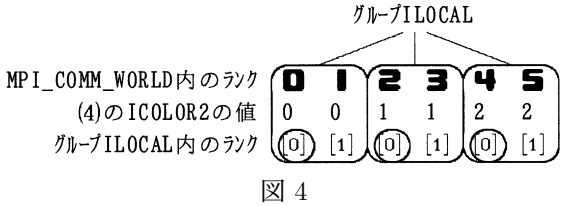


図 4

■ P 96 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。

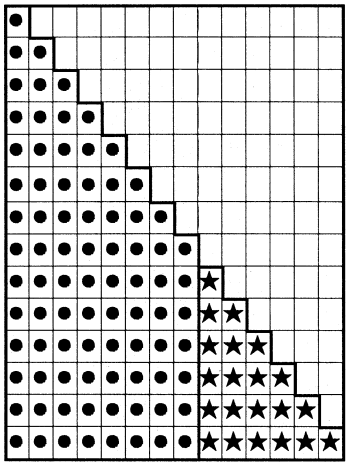


図 1

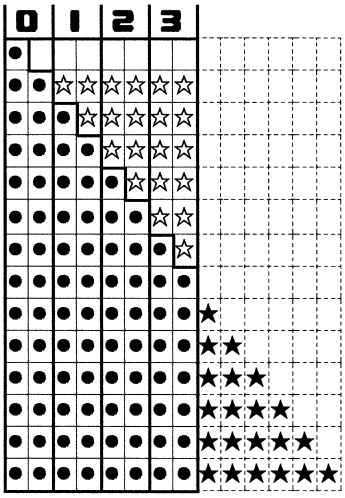


図 2

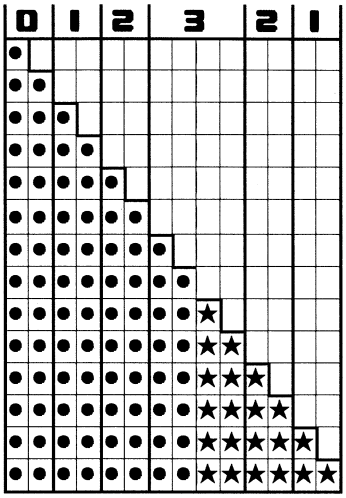
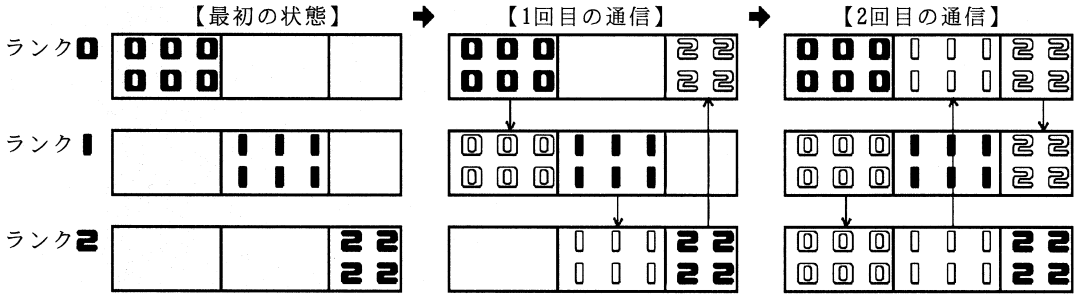


図 3

■ P 131 と P 132 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。



■ P 176 の図 4-8-8 (5) で、特にプロセスの数の多い場合、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
```