MPI プログラミング入門

野口 孝明[1], 曽我 隆[1], 金野 浩伸[2], 撫佐 昭裕[2],
大泉 健治[3], 小野 敏[3], 伊藤 英一[3],
岡部 公起[4], 江川 隆輔[4], 小林 広明[4]

1) NEC システムテクノロジー株式会社 2) 日本電気株式会社
3) 東北大学情報部情報基盤課 4) 東北大学情報シナジー機構情報シナジーセンター

1. MPI プログラムの概要

1.1. 並列処理

並列処理とは, 並行して同時に実行可能な処理を複数の CPU を用いて実行することである. 並列処理を行っていないプログラムを逐次処理プログラムと呼び, 逐次処理プログラムを並列処理プログラムに書き換えることを並列化と言う. 並列化する目的は, 逐次処理プログラムの実行時間を短縮することである. ここでは, 並列処理を行うための MPI プログラミングについて幾つかの実例を示し, 並列化プログラミング技術を紹介する.

並列処理には, コンピュータアーキテクチャに応じた以下の並列処理方式がある.

(1)分散メモリ型並列処理

分散メモリ型並列処理とは, 情報シナジーセンターにおける SX-7C システムのように複数のコンピュータがネットワークを介して接続されているコンピュータ群で行なわれる並列処理のことであり, MPI 等の並列処理ライブラリを用いてプログラムの並列化を行うものである. PC クラスタなどで実行される並列処理も同様である.

(2)共有メモリ型並列処理

共有メモリ型並列処理とは, 情報シナジーセンターにおける SX-7 システムと TX7 システムのようにノード内での並列処理を行うものであり, 並列化コンパイラによる自動並列化や OpenMP などの指示行により並列化を行うものである. ここでノードとは, 1つのメモリシステムを複数の CPU で共有しているコンピュータのことである.

1.2. MPI とは

MPI (Message Passing Interface) とは, MPI フォーラムにより開発・規格化された分散メモリ型並列処理におけるデータ通信のための標準規格であり, 複数のプロセス間でのデータをやり取りするために用いるメッセージ通信操作の仕様標準である.

主な特徴を以下に示す.

- FORTRAN, C から呼び出すサブプログラムである
- 標準化されたライブラリインターフェースによって, 様々な MPI 実装環境で同じソースを利用できる

➢ データの分割及び通信の記述がプログラマに任されているため，プログラマの負担が比較的大きい

1.3. MPI による並列化

ここでは，総和計算を行うFORTRANプログラムを例にMPIプログラムの簡単な説明を行う．

(1)プログラム例

図1は，1から100までの整数の和を求める逐次処理プログラムである．

```
program example1
isum = 0
do i = 1, 100
   isum = isum + i
end do
write(6,*)'sum=',isum
stop
end program
```

図 1.　逐次処理プログラム例(総和)

(2)処理の分割

図1の総和計算は，DOループにより1から100まで順に足されているが，足す順序は結果に影響しないため，このDOループは分割して並行に実行することができる．

図2は，ループを均等に4分割し，4プロセスで分担した場合の例である．各プロセスがそれぞれDOループのどの部分を担当するか，プロセスの識別id.(0～3)で表している．MPIでは，プロセスの識別id.のことをrank番号と呼ぶ．

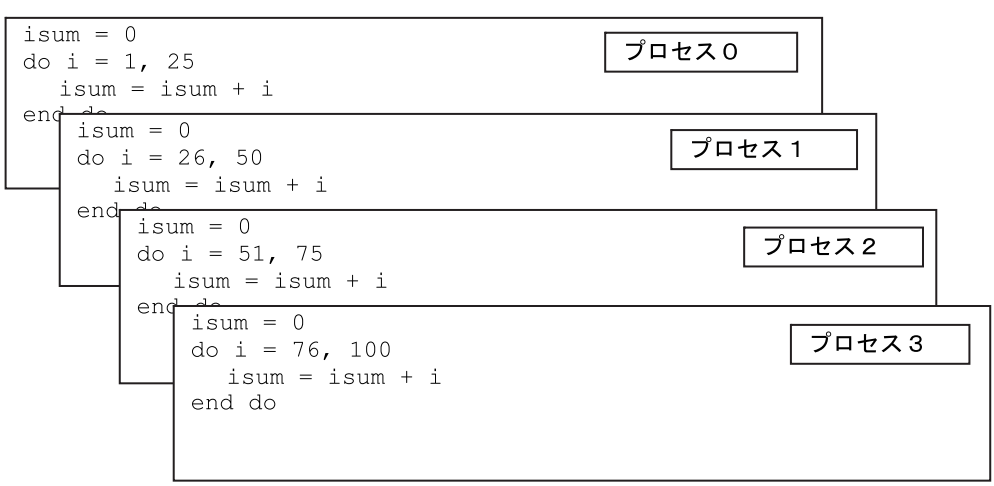


図 2.　DOループの分割イメージ(総和)

図2は並列数を 4 に固定した例であるが，ここでは並列数を自由に変えることができる

例を示す. 並列実行数を nprocs とした場合, rank 番号 myrank の各プロセスが担当する等分割された部分ループの始点と終点は以下の式で求めることができる.

➢ 始点

```
ist = ((100-1)/nprocs+1)*myrank+1
```

➢ 終点

```
ied = ((100-1)/nprocs+1)*(myrank+1)
```

全並列数が全プロセス数であることから, 各プロセスが全プロセス数 nprocs と rank 番号 myrank で表される ist から ied までループを繰り返すことで, 元のループ全体の実行ができる. 図 3 は, DO ループを分割したソースイメージである.

```
ist = ((100-1)/nprocs+1)*myrank+1
ied = ((100-1)/nprocs+1)*(myrank+1)
isum = 0
do i = ist, ied
 isum = isum + i
end do
```

図 3.　DO ループの分割例(総和)

(3)プロセス間通信

各プロセスで求められた部分和を合計して全体の総和を求めるため,MPI による通信(コミュニケーション)が必要である. 図 4 は, 通信プログラムの例であり, データの送受信を行う MPI のサブルーチン(MPI_RECV, MPI_SEND)を用いている. 図4では, rank 番号 0 番のプロセスが, 自プロセス以外のすべてのプロセスから送られる部分和を(nprocs−1)回

```
 if (myrank.eq.0) then
    do i=1, nprocs-1
       source=i
       tag=i
       call MPI_RECV(itmp,1,MPI_INTEGER,source,tag,MPI_COMM_WORLD,
&                          status, ierr)
       isum = isum + itmp
    enddo
 else
    tag=myrank
    call MPI_SEND(isum,1,MPI_INTEGER,0,tag,MPI_COMM_WORLD,ierr)
 endif
```

図 4.　通信プログラム例(総和)

受信し, 受信した部分和 itmp を isum に足し込んでいる. rank 番号が 0 番以外のプロセス

は, それぞれの部分和を rank 番号 0 番に送信している.

(4)MPI プログラム例

以上を総合して, 図 1 の逐次処理プログラムを並列化したプログラムを図5に示す.

```
 1   program example1
 2   include 'mpif.h'
 3   integer status(MPI_STATUS_SIZE)
 4   integer nprocs,myrank,source,tag,ierr,ist,ied,i,isum,itmp
 5   call MPI_INIT(ierr)
 6   call MPI_COMM_SIZE(MPI_COMM_WORLD,nprocs,ierr)
 7   call MPI_COMM_RANK(MPI_COMM_WORLD,myrank,ierr)
 8   ist = ((100-1)/nprocs+1)*myrank+1
 9   ied = ((100-1)/nprocs+1)*(myrank+1)
10   isum = 0
11   do i=ist, ied
12      isum = isum + i
13   end do
14   if (myrank.eq.0) then
15     do i=1, nprocs-1
16        source=i
17        tag=i
18        call MPI_RECV(itmp,1,MPI_INTEGER,source,tag,MPI_COMM_WORLD,
19  &                           status, ierr)
20        isum = isum + itmp
21     enddo
22   else
23     tag=myrank
24     call MPI_SEND(isum,1,MPI_INTEGER,0,tag,MPI_COMM_WORLD,ierr)
25   endif
26   if (myrank.eq.0) write(6,*)'sum=',isum
27   call MPI_FINALIZE(ierr)
28   stop
29   end
```

図 5.　並列化プログラム例(総和)

図 5 において,赤字で記載された部分が MPI ライブラリの宣言とサブルーチンである. 以下, 簡単に紹介する.

■ mpif.h (2 行目)

MPI で使用されるすべての定数が定義されている. MPI のサブルーチン・定数が含まれるサブルーチンすべてにこの include 文が必要である.

■ status(MPI_STATUS_SIZE)　(3 行目)

下記, MPI_RECV の第七引数で使用されるメッセージ情報.

■ MPI_INIT (5 行目)

MPI_INIT は, MPI の実行開始を宣言するサブルーチンであり, すべての MPI サブルーチンに先立ち, 呼び出される必要がある.

■ MPI_COMM_SIZE（6 行目）

MPI のプロセス管理を行うサブルーチンであり，総プロセス数の問い合わせを行う．第一引数は MPI 通信のためのコミュニケータというものであり“MPI_COMM_WORLD”を指定する．第二引数（nprocs）に総プロセス数，第三引数に実行結果の状態が戻る．

■ MPI_COMM_RANK（7 行目）

MPI のプロセス管理を行うサブルーチンであり，自プロセスの rank 番号の問い合わせを行う．第二引数（myrank）に自 rank 番号を受け取る．

■ MPI_RECV（18～19 行目）

MPI の通信処理を行うサブルーチンであり，データの受信をブロッキングモードで行う．このサブルーチンは，一対一通信と呼ばれる分類に属し，通信相手が必要になる．送信側が送ったデータを受信するために呼び出される．第一引数（itmp）は受信データの開始アドレス，第二引数(1)は受信データの要素数，第三引数(MPI_INTEGER)は受信データのタイプ，第四引数(source)は通信相手のランク，第五引数(tag)はタグ（送受信は同じタグ間で行われる），第六引数(MPI_COMM_WORLD)はコミュニケータ，第七引数（status）はメッセージ情報を示す．

■ MPI_SEND（24 行目）

MPI の通信処理を行うサブルーチンであり，データの送信をブロッキングモードで行う．このサブルーチンは，一対一通信と呼ばれる分類に属し，通信相手が必要になる．受信側にデータを送信するために呼び出される．第一引数（isum）は送信データの開始アドレス，第二引数(1)は送信データの要素数，第三引数(MPI_INTEGER)は受信データのタイプを示す．以降の引数は MPI_RECV と同じである．

■ MPI_FINALIZE（27 行目）

MPI_FINALIZE は，MPI_INIT で始まった MPI の実行終了を宣言するものであり,すべての MPI サブルーチンが呼ばれた後で呼び出す必要がある．

1.4. プログラムのコンパイル・実行

ここでは，SX-7 用のコンパイルと実行に関して簡単に説明する．詳細については 3 章に示す．コンパイルのために，クロス環境（gen. isc. tohoku. ac. jp）と会話型環境（super. isc. tohoku. ac. jp）が用意されている．図 6（A）は，クロス環境において，コンパイルオプションをすべて既定値とした場合のコマンドイメージである．

実行は，会話型環境(super.isc.tohoku.ac.jp)，もしくはバッチ型環境(gen.isc.tohoku.ac.jp)を利用する．図 6 (B)は，会話型環境を利用した場合の実行イメージである．-np 4 は、生成するプロセス数を指定している．

```
% sxmpif90 example1.f
% ls
a.out  example1.f
%
```

(A) クロス環境でのコンパイル例

```
% mpirun -np 4 ./a.out
sum=  5050
%
```

(B) 会話型環境での実行例

図 6.　コンパイル・実行例

2. MPI プログラムの具体例

ここでは，MPI ライブラリの具体的な利用方法とプログラムのテクニックについて例を示す．1 章の総和計算では，各プロセスで求められた部分和を集計するために MPI による通信を行い，総和計算を完成させた．1 章で見た通信処理を含め，MPI には以下の 2 種類の通信処理が用意されている．

2.1. プロセス間通信

2.1.1. 一対一通信

MPI に用意された最も基本的な通信方式は，一対一通信である．図 7 に示すように，2 つのプロセスが通信処理に関与し，それぞれ送信処理と受信処理を行う．代表的な手続きに1章で総和を並列化する際に用いた MPI_SEND や MPI_RECV などがある．

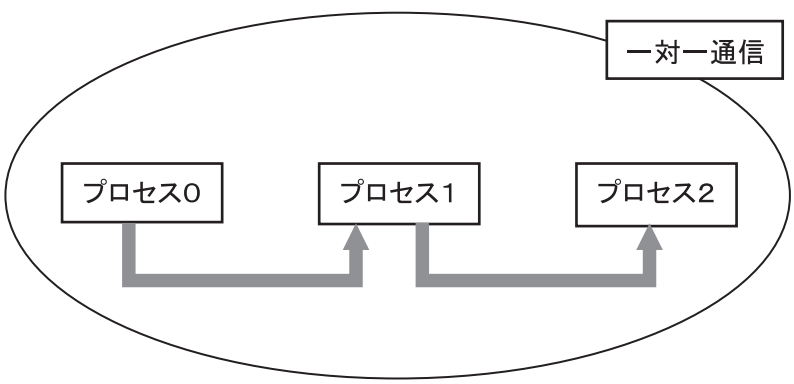


図 7.　一対一通信

2.1.2. 集団通信

集団通信とは，すべての MPI プロセスが参加して行う通信方式であり，入力データの配信などのブロードキャストや総和などのリダクション演算，FFT などで用いられる転置，ファイル入出力で用いられる収集，分配などの処理がある．図8はブロードキャストの概念図である．プロセス0から各プロセスにデータを一斉に配布することができる．また，図9はリダクション

演算の総和を求める例である. リダクション演算とは, 複数のデータを一つに縮約する演算のことを指す. 以下で, 集団通信を用いた並列化の例を示す.

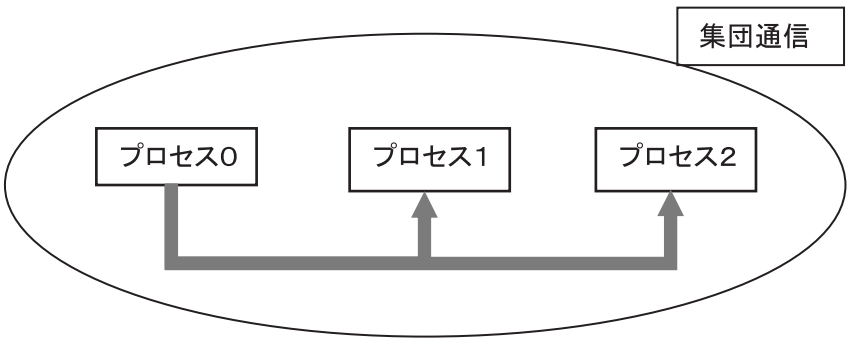


図 8.　集団通信

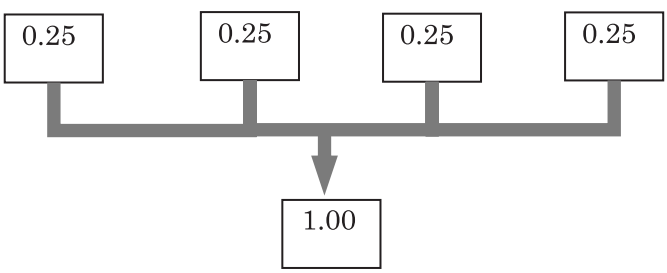


図 9.　リダクション演算例(総和)

(1)逐次処理プログラム例

図 10 は, ベクトルの内積を行う逐次処理プログラムである. ベクトルの内積はベクトルの要素ごとの積を足し合わせる. つまり, 和の順序は不同であり, 並列に計算可能である. 1 章同様, DO ループの並列性に着目して並列化を行う.

```
program example2
integer,parameter::nn=100
real(8):: a(nn), b(nn), c
integer:: i
c = 0.0d0
do i=1, nn
  a(i) = 1.0d0
  b(i) = 0.01d0
  c = c + a(i) * b(i)
end do
write(*,*)'inner product=',c
stop
end program
```

図 10.　逐次処理プログラム例(内積)

(2)データの分割

1.3 節同様に積和部分のループを等分割し，並列に計算する．分割するプロセスの数を nprocs 個とし，各プロセスの rank 番号を myrank とする．分割されるループの始点と終点は，ループのサイズ nn を用いて以下の様に表される．

➢ 始点

```
ist = ((nn-1)/nprocs+1)*myrank+1
```

➢ 終点

```
ied = ((nn-1)/nprocs+1)*(myrank+1)
```

(3)MPI プログラミング例

内積の並列化は，総和と同じく各プロセスでベクトル内積の部分和を求めて，それらを足し合わせる方法を取る．1 章の総和計算では，MPI_SEND と MPI_RECV のサブルーチンを組み合わせることで各プロセスの部分和を足し合わせたが， reduction 演算を用いた総和計算の例を図 11 に示す．

```
 1    program example2
 2    include 'mpif.h'
 3    integer,parameter::nn=100
 4    real(8):: a(nn), b(nn), c, d
 5    integer:: i,ist,ied,nprocs,myrank,ierr
 6    call MPI_INIT(ierr)
 7    call MPI_COMM_SIZE(MPI_COMM_WORLD,nprocs,ierr)
 8    call MPI_COMM_RANK(MPI_COMM_WORLD,myrank,ierr)
 9    ist = ((nn-1)/nprocs+1)*myrank+1
10    ied = ((nn-1)/nprocs+1)*(myrank+1)
11    d = 0.0d0
12    do i=ist, ied
13      a(i)=1.0d0
14      b(i)=0.01d0
15      d = d + a(i) * b(i)
16    end do
17    call MPI_ALLREDUCE(d, c, 1, MPI_REAL8,
18   &                   MPI_SUM, MPI_COMM_WORLD, ierr)
19    if (myrank.eq.0) write(*,*)'inner product=',c
20    call MPI_FINALIZE(ierr)
21    stop
22    end program
```

図 11.　並列処理プログラム例(内積)

以下，この例題で用いた MPI のサブルーチンを簡単に説明する．

■ MPI_ALLREDUCE（17～18 行目）

MPI の集団通信を行うサブルーチンの一つで，複数のプロセスからそれぞれデータを

受信し，一つの演算を行う．集団通信は，全プロセスが対象になる．MPI_ALLREDUCE の引数に MPI_SUM を与えると，各プロセスから送信されたデータを第二引数に足し込む演算を行う．図 12 では，4 つのプロセスから渡された 0.25 を加えて，変数 c に 1.00 が代入される．

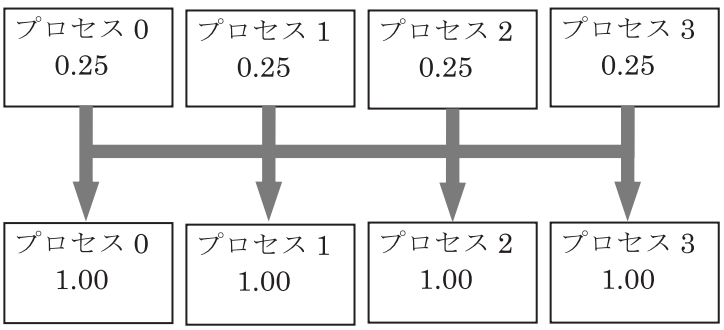


図 12.　MPI_ALLREDUCE の例

2.2. 隣接領域間転送

DO ループを分割すると，ループを分割した境界上で，演算に必要なデータが不足する場合がある．本節では，不足するデータを MPI による通信で補う例として，差分式の並列化を紹介する．図 13 は差分計算を行うプログラムである．

```
 1    program example3
 2    integer,parameter::nn=100
 3    integer::a(0:nn+1),b(nn),i
 4    do i=0,nn+1
 5       a(i)=i
 6    enddo
 7    do i=1,nn
 8       b(i)=a(i+1)-a(i)+a(i-1)
 9    enddo
10    write(10,'(10I8)')(b(i),i=1,nn)
11    stop
12    end program
```

図 13.　逐次処理プログラム例（差分）

図 14 に図 13 の 7 行目から 9 行目の DO ループを等分割して，各プロセスで並列に処理するイメージを示す．

```
do i = 1, 25
   b(i) = a(i+1) - a(i) + a(i-1)
end do
```

```
do i = 26, 50
   b(i) = a(i+1) a(i) + a(i-1)
end do
```

```
do i = 51, 75
   b(i) = a(i+1) - a(i) + a(i-1)
end do
```

```
do i = 76, 100
   b(i) = a(i+1) - a(i) + a(i-1)
end do
```

図 14.　DO ループの分割例(差分)

図 15 に領域を分割した場合に境界領域で不足するデータのイメージを示す．プロセス 0 では 26 番目のデータが，プロセス 1 では 25 番目と 51 番目のデータが不足している．不足するデータは，MPI による通信により補う．

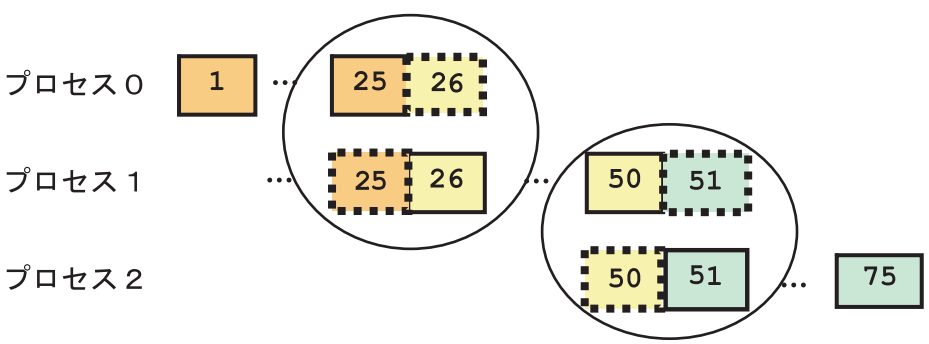


図 15.　領域の分割例(差分)

図 16 に不足するデータを補う MPI 通信のプログラム例を示す．送受信の対象となる配列 a の添え字が少し複雑になるが，添え字は，境界のデータを指し示しており，境界のデータを送受信することで不足しているデータを補う．

```
 if (myrank.ne.0) then
    dest=myrank-1
    tag=myrank
    call MPI_ISEND(a(25*myrank+1),1,MPI_INTEGER,dest,tag,
&                  MPI_COMM_WORLD,request,ierr)
    source=myrank-1
    tag=myrank-1
    call  MPI_RECV(a(25*(myrank+1)+1,1,MPI_INTEGER,source,tag,
&                 MPI_COMM_WORLD,status,ierr)
 endif
 if (myrank.lt.nprocs-1) then
    dest=myrank+1
    tag=myrank
    call MPI_ISEND(a(25*(myrank+1),1,MPI_INTEGER,dest,tag,
&                  MPI_COMM_WORLD,request,ierr)
    source=myrank+1
    tag=myrank+1
    call MPI_RECV(a(25*myrank+1,1,MPI_INTEGER,source,tag,
&                 MPI_COMM_WORLD,status,ierr)
 endif
```

図 16. 通信処理プログラム例(差分)

図 17 に図 13 に示した差分計算プログラムを MPI により並列化したプログラムの例を示す.

```
 program example3
 include 'mpif.h'
 integer,parameter::nn=100
 integer::a(0:nn+1),b(nn),it,ist,ied,i
 integer::myrank,nprocs,request,status(MPI_STATUS_SIZE),
&         source,dest,tag,ierr
 call MPI_INIT(ierr)
 call MPI_COMM_SIZE(MPI_COMM_WORLD,nprocs,ierr)
 call MPI_COMM_RANK(MPI_COMM_WORLD,myrank,ierr)
 it = (nn-1)/nprocs+1
 ist = ((nn-1)/nprocs+1)*myrank+1
 ied = ((nn-1)/nprocs+1)*(myrank+1)
 do i=ist, ied
    a(i)=i
 enddo
 if (myrank.eq.0) a(0)=0
 if (myrank.eq.nprocs-1) a(nn+1)=nn+1
 if (myrank.ne.0) then
    dest=myrank-1
    tag=myrank
    call MPI_ISEND(a(it*myrank+1),1,MPI_INTEGER,dest,tag,
&                  MPI_COMM_WORLD,request,ierr)
    source=myrank-1
    tag=myrank-1
    call MPI_RECV(a(it*myrank),1,MPI_INTEGER,source,tag,
&                   MPI_COMM_WORLD,status,ierr)
 endif
 if (myrank.lt.nprocs-1) then
    dest=myrank+1
    tag=myrank
    call MPI_ISEND(a(it*(myrank+1)),1,MPI_INTEGER,dest,tag,
&                  MPI_COMM_WORLD,request,ierr)
    source=myrank+1
    tag=myrank+1
    call MPI_RECV(a(it*(myrank+1)+1),1,MPI_INTEGER,source,tag,
&                 MPI_COMM_WORLD,status,ierr)
 endif
 do i=ist, ied
    b(i)=a(i+1)-a(i)+a(i-1)
 enddo
 write(11+myrank,'(10I8)')(b(i),i=ist,ied)
 call MPI_FINALIZE (ierr)
 stop
 end program
```

図 17.　並列処理プログラム例(差分)

2.3. 計算量の均一化

並列処理では, 各プロセスが処理する計算量にばらつきがあると各プロセスの処理時間にばらつきが生じ, 処理時間短縮の弊害になる. 本節では, このような典型的な計算量の不均衡の例と, その均一化について紹介する.

(1)逐次処理プログラム例

図 18 は，計算量インバランスの例として三角行列とベクトルとの積を行う逐次処理プログラム例である．

```
program example4
integer,parameter::nn=100
integer ::a(nn,nn),b(nn),c(nn),i,j
do i=i,nn
 do j=1,i
  a(i,j)=j+(i-1)*nn
 enddo
 b(i)=1
enddo
do i=1,nn
 c(i)=0
 do j=1,i
  c(i)=c(i)+a(i,j)*b(j)
 enddo
enddo
write(10,'(10I8)')(c(i),i=1,nn)
stop
end
```

図 18.　逐次処理プログラム例（三角行列）

(2)データの分割（サイクリック分割）

行列の行ベクトルとベクトルの掛け算は，並列に行うことが可能であるので，この部分を並列化する．図 19 に各プロセスが処理する計算量を示す．行列の形が下三角になっているので，等分割では各プロセスが処理する計算量に不均衡が生じる．

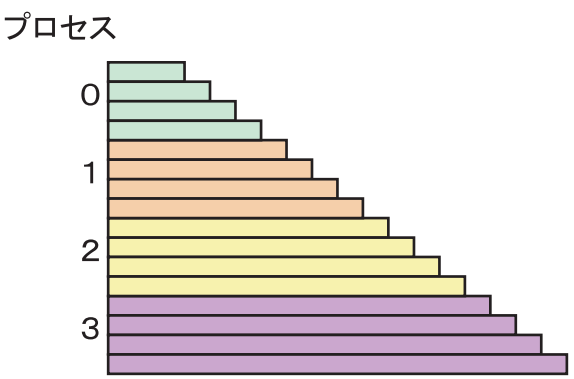


図 19.　等分割の例（三角行列）

図 20 に各プロセス間の計算量が均一になるように，分割幅をより細かくした例を示す．特に，分割した領域を連続して 1 つのプロセスが担当せず，担当領域を巡回的に割り当てる方

法はサイクリック分割と呼ばれる. 例えば, 1から100までのループがあった場合に, プロセス0が1〜25までを担当するのではなく, 1, 5, 9, 13, …という風に4飛びで担当するようにループを分割する.

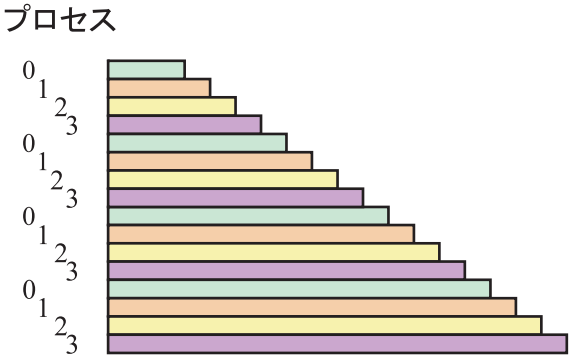


図 20.　サイクリック分割(三角行列)

図21にサイクリック分割によるDOループの分割を示す. サイクリック分割により, 各プロセスが担当する計算量がほぼ均一化される.

```
do i = 1, 97, 4
 do j=1,i
  c(i)=c(i)+a(i,j)*b(j)
 enddo
enddo
```

```
do i = 2, 98, 4
 do j=1,i
  c(i)=c(i)+a(i,j)*b(j)
 enddo
enddo
```

```
do i = 3, 99, 4
 do j=1,i
  c(i)=c(i)+a(i,j)*b(j)
 enddo
enddo
```

```
do i = 4, 100, 4
 do j=1,i
  c(i)=c(i)+a(i,j)*b(j)
 enddo
enddo
```

図 21.　DO ループの分割例(三角行列)

プロセスの数をnprocs個とし, 各プロセスのrank番号をmyrankとすると, 分割されるループの始点と終点および間隔は, ループのサイズnnを用いて以下の通り表される.

➢ 始点

```
ist = myrank+1
```

➢ 終点

```
ied = nn-nprocs+myrank+1
```

➢ 間隔

```
int = nprocs
```

これらを用いて並列化したプログラムを図 22 に示す.

```
 program example4
 include 'mpif.h'
 integer,parameter::nn=100
 integer::a(nn,nn),b(nn),c(nn),i,j
 integer::nprocs,myrank,ierr
 call MPI_INIT (ierr)
 call MPI_COMM_SIZE(MPI_COMM_WORLD,nprocs,ierr)
 call MPI_COMM_RANK(MPI_COMM_WORLD,myrank,ierr)
 do i=1,nn
  do j=1,i
   a(i,j)=j+(i-1)*nn
  enddo
  b(i)=1
 enddo
 do i=myrank+1, nn-nprocs+myrank+1, nprocs
  c(i)=0
  do j=1,i
   c(i)=c(i)+a(i,j)*b(j)
  enddo
 enddo
 write(21+myrank,'(5I8)')
&      (c(i),i=myrank+1,nn-nprocs+myrank+1,nprocs)
 call MPI_FINALIZE(ierr)
 stop
 end
```

図 22.　並列処理プログラム例(三角行列)

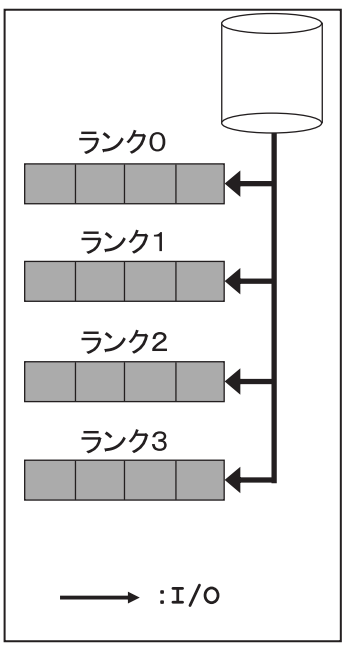
2.4. ファイル入出力

本節では, ファイル入出力を取り上げる. プログラムの並列化を行うと, プロセスの 0 番がファイルリードを行っても, プロセスの 1 番はそのデータを利用することができない. そこで, 並列化されたプログラムのファイル入出力には幾つかのパターンを考えることができる. それぞれの特徴を実際のプログラム例を元に以下に示す.

2.4.1. ファイル入力

(1)全プロセス同一ファイル入力

すべてのプロセスが同一の(逐次処理と同じ)ファイルにアクセスを行う. すべてのプロセスがすべてのデータを読み込むため, ファイルを分割する必要がなく, また通信も発生しない. しかし, 複数のプロセスから同じファイルを同時にアクセスするため, ファイル入出力の競合が発生する場合がある.

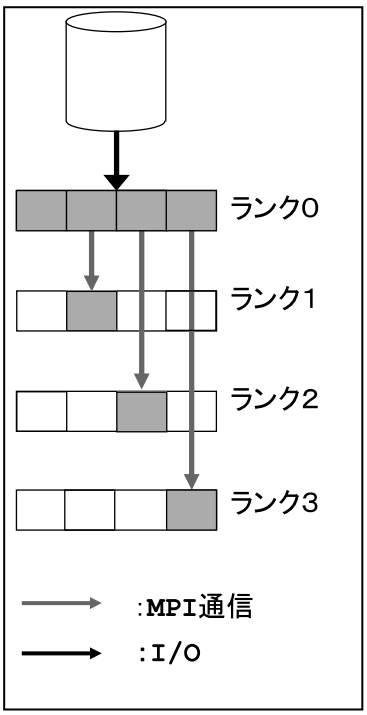



```
program example5
include 'mpif.h'
parameter (numdat=100)
dimension idat(numdat)
integer ist,ied,isum,i
integer myrank,nprocs,ierr
call MPI_INIT(ierr)
call MPI_COMM_SIZE(MPI_COMM_WORLD,nprocs,ierr)
call MPI_COMM_RANK(MPI_COMM_WORLD,myrank,ierr)
ist=((numdat-1)/nprocs+1)*myrank+1
ied=((numdat-1)/nprocs+1)*(myrank+1)
read(10,*) idat
isum=0
do i=ist,ied
   isum=isum+idat(i)
enddo
write(6,*) myrank,':partial sum=',isum
call MPI_FINALIZE(ierr)
stop
end
```

図 23. 全プロセス同一ファイル入力

(2)代表プロセス入力

逐次処理と同じファイルに特定のプロセス(図 24 では, rank0)だけが, アクセスを行う. 各プロセスへは, MPI の通信機能を用いて配信する. 1 つのプロセスがファイル全体をアクセスするため, 逐次処理と同じファイルを利用でき, ファイルの分割を行う必要がない.




```
 program example6
 include 'mpif.h'
 parameter (numdat=100)
 integer senddata(numdat),recvdata(numdat)
 integer i,icount,isum
 integer myrank,nprocs,ierr
 call MPI_INIT(ierr)
 call MPI_COMM_SIZE(MPI_COMM_WORLD,nprocs,ierr)
 call MPI_COMM_RANK(MPI_COMM_WORLD,myrank,ierr)
 if (myrank.eq.0) read(10,*) senddata
 icount=(numdat-1)/nprocs+1
 call MPI_SCATTER(senddata,icount,MPI_INTEGER,
&        recvdata(icount*myrank+1),icount,
&        MPI_INTEGER,0,MPI_COMM_WORLD,ierr)
 isum=0
 do i=1,icount
    isum=isum+recvdata(icount*myrank+i)
 enddo
 write(6,*)myrank,':partial sum=',isum
 call MPI_FINALIZE(ierr)
 stop
 end
```

図 24. 代表プロセス入力

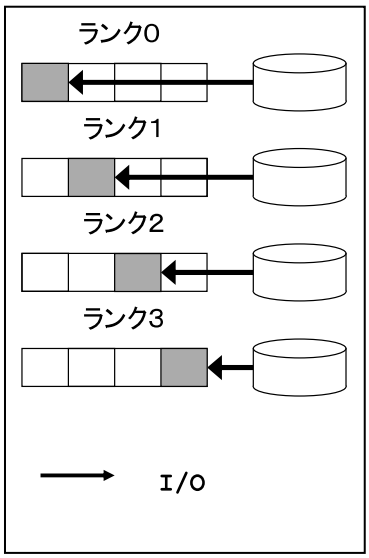
以下, この例題で用いた MPI のサブルーチンを簡単に説明する.

■ MPI_SCATTER

MPI の通信処理を行うサブルーチンの一つで, 一つのプロセスからその他の全プロセスへデータを分配する機能を持つ. このサブルーチンは, 集団通信と呼ばれる分類に属し, 全プロセスが対象になり, 特定のプロセスで設定された第一引数を等分割して全プロセスに配信される.

(3)分散ファイル入力

分散配置された複数のファイルから, 各プロセスがそれぞれ別のファイルにアクセスを行う. 並列化により DO ループを分割したため, 各プロセスは, データの一部分だけが必要である. そこで,各プロセスが必要な部分だけを収めたファイルを分散配置し, 各プロセスは, 必要な部分だけ読み込む. 分割数が変わるたびにファイルを新たに用意する必要がある. ただし, データ配信のための通信がなくなり, 処理時間の短縮につながる.



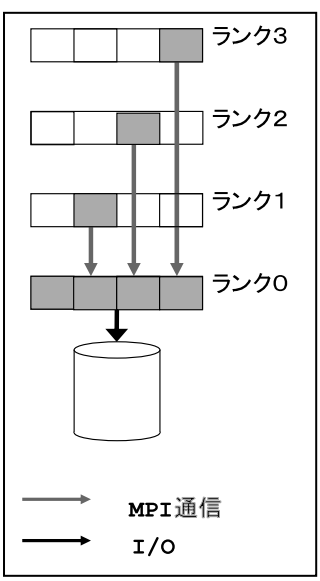

```
program example7
include 'mpif.h'
parameter (numdat=100)
integer buf(numdat),ist,ied,isum,i
integer myrank,nprocs,ierr
call MPI_INIT(ierr)
call MPI_COMM_SIZE(MPI_COMM_WORLD,nprocs,ierr)
call MPI_COMM_RANK(MPI_COMM_WORLD,myrank,ierr)
ist=((numdat-1)/nprocs+1)*myrank+1
ied=((numdat-1)/nprocs+1)*(myrank+1)
read(11+myrank,*) (buf(i),i=ist,ied)
isum=0
do i=ist,ied
   isum=isum+buf(i)
enddo
write(6,*)myrank,':partial sum=',isum
call MPI_FINALIZE(ierr)
stop
end
```

図 25. 分散ファイル入力

2.4.2. ファイル出力

(1)代表プロセス出力

特定のプロセスだけがファイルにアクセスする方法である. 作成されるファイルは, 逐次処理の時と同じものが作成される. 特定のプロセスへは, MPI の通信機能を用いてデータを集める. 生成されるファイルが一つになるため, ファイルの扱いが容易になる.




```
program example8
include 'mpif.h'
parameter (numdat=100)
integer senddata(numdat),recvdata(numdat)
integer myrank,nprocs,ierr,icount,i
call MPI_INIT(ierr)
call MPI_COMM_SIZE(MPI_COMM_WORLD,nprocs,ierr)
call MPI_COMM_RANK(MPI_COMM_WORLD,myrank,ierr)
icount=(numdat-1)/nprocs+1
do i=1,icount
   senddata(icount*myrank+i)=icount*myrank+i
enddo
call MPI_GATHER(senddata(icount*myrank+1),
&        icount,MPI_INTEGER,recvdata,
&        icount,MPI_INTEGER,0,MPI_COMM_WORLD,ierr)
if (myrank.eq.0) write(10,'(10I8)') recvdata
call MPI_FINALIZE(ierr)
stop
end
```

図 26.　代表プロセス出力

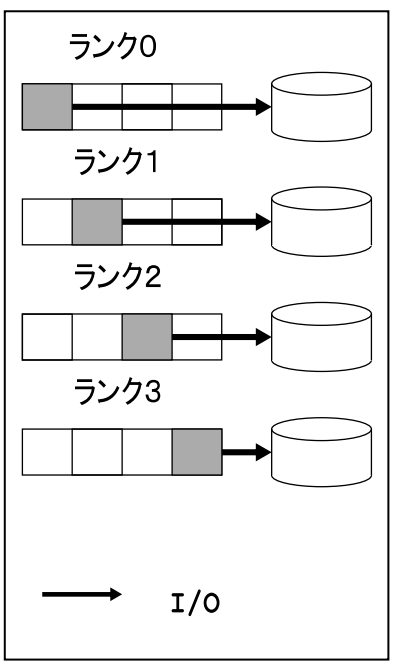
以下，この例題で用いた MPI のサブルーチンを簡単に説明する．

■ MPI_GATHER

MPI の通信処理を行うサブルーチンの一つで，すべてのプロセスから一つのプロセスへデータを収集する機能を持つ．このサブルーチンは，集団通信と呼ばれる分類に属し，全プロセスが対象になり，すべてのプロセスで設定された第一引数を収集して一つのプロセスに集められる．

(2)分散ファイル出力

各プロセスから別々のファイルにアクセスを行う．各プロセスは，直接ファイルに書き出すので通信の必要がない．また，各プロセスは，必要な分だけファイルに書き出せば良いので，入出力等に要する時間を短縮することもできる．ただし，書き出されるファイルが複数になるので取り扱いが面倒になり，別のプログラムで再利用する場合は分割数を合わせる必要がある．




```
program example9
include 'mpif.h'
parameter (numdat=100)
integer buf(numdat),ist,ied,i
integer myrank,nprocs,ierr
call MPI_INIT(ierr)
call MPI_COMM_SIZE(MPI_COMM_WORLD,nprocs,ierr)
call MPI_COMM_RANK(MPI_COMM_WORLD,myrank,ierr)
ist=((numdat-1)/nprocs+1)*myrank+1
ied=((numdat-1)/nprocs+1)*(myrank+1)
do i=ist,ied
   buf(i)=i
enddo
write(11+myrank,'(10I8)') (buf(i),i=ist,ied)
call MPI_FINALIZE(ierr)
stop
end
```

図 27. 分散ファイル出力

以上，本章では，MPI を用いた並列処理プログラミングを行うための様々なプログラミングテクニックについて述べた．次章では，コンパイルから実行，および性能の解析を行う手順を示す．

3. 実行方法と性能解析

本章では，MPI プログラムのコンパイルから実行および性能解析までの手順について説明する．

3.1. 情報シナジーセンターのシステム

(1)シングルノード型

複数の CPU が同一のメモリを共有しているコンピュータであり,MPI ライブラリの利用に加え,コンパイラによる自動並列化機能や OpenMP なども利用できる．

- SX-7
- TX7

(2)マルチノード型

複数のシングルノード型コンピュータがネットワークを介して接続されており,ノード間は，MPI のみによる並列化を行う．

- SX-7C

3.2. SX におけるコンパイル・実行

(1)SX-7/SX-7C におけるコンパイル方法

【形式】 sxmpif90　オプション　MPI ソースファイル名

■ 主なオプション

- -sx8　　SX-7C 向けの命令を生成する
- -pi　　インライン展開を行う
- -R5　　ベクトル化／並列化状況を表示した編集リストの出力
- -ftrace　手続きごとの性能情報の取得
- MPI ソースファイル名

  FORTRAN のソースプログラムファイル名を指定する．複数のファイルを指定する場合は，空白で区切る．ソースファイル名には，サフィックス「.f90」か「.F90」（自由形式），または「.f」か「.F」（固定形式）が必要．

(2)SX-7 における実行

SX-7 ではノード内の MPI 実行が可能であり，バッチキューで実行可能な CPU 数まで MPI プロセスを実行することができる．

【形式】 mpirun -np プロセス数　実行形式ファイル名

■ バッチリクエストスクリプトイメージ

```
# test job-MPI
#PBS -q キュー名                    →  SX-7 でのキュー名
#PBS -jo -N reqname                    (キューの詳細はホームページ参照)
unsetenv F_PROGINF
setenv MPIPROGINF detail
setenv MPICOMMINF detail
setenv MPISEPSELECT 3
cd $PBS_O_WORKDIR
mpirun -np 8 /usr/lib/mpi/mpisep.sh a.out
                                →  8 プロセス実行の指定
```

図 28. バッチスクリプトイメージ(SX-7)

(3)SX-7C における実行

SX-7C では 5 ノード 40 プロセスまでの実行が可能である.

【形式】 mpirun -np プロセス数 -nn 使用ノード数 実行形式ファイル名

■ バッチリクエストスクリプトイメージ

```
# test job-M40
#PBS -q px                    →  SX-7C での MPI 専用キュー名
#PBS -jo -N reqname
unsetenv F_PROGINF
setenv MPIPROGINF detail
setenv MPICOMMINF detail
setenv MPISEPSELECT 3
cd $PBS_O_WORKDIR
mpirun -np 40 -nn 5 /usr/lib/mpi/mpisep.sh a.out
                                →   5 ノード 40 プロセス実行の指定
```

図 29. バッチスクリプトイメージ(SX-7C)

3.3. SX における実行時の環境変数

(1)MPIPROGINF

実行性能情報を MPI プロセス毎, または全 MPI プロセスの情報を集計編集して表示させることを指定する環境変数である. 表示は, MPI プログラムの実行において, MPI_FINALIZE 手続きを呼び出した際に MPI_COMM_WORLD(MPIUNIVERSE=0)のランク 0 の MPI プロセスから標準エラー出力に対して行われる.

MPIPROGINF の値と表示内容は以下の通り.

➢ NO　　実行性能情報を出力しない(既定値)

- YES　　基本情報を集約形式で出力
- DETAIL　　詳細情報を集約形式で出力
- ALL　　基本情報を拡張形式で出力
- ALL_DETAIL　詳細情報を拡張形式で出力

■ 出力例(DETAIL 指定時)

```
MPI Program Information:
========================
Note: It is measured from MPI_Init till MPI_Finalize.
      [U,R] specifies the Universe and the Process Rank in the Universe.

Global Data of 2 processes  :          Min [U,R]          Max [U,R]      Average
==========================

     Real   Time (sec)           :        2.239 [0,1]        2.240 [0,0]        2.239
     User   Time (sec)           :        2.002 [0,1]        2.228 [0,0]        2.115
     System Time (sec)           :        0.005 [0,0]        0.053 [0,1]        0.029
     Vector Time (sec)           :        1.825 [0,1]        1.846 [0,0]        1.835
     Instruction Count           :    196086710 [0,1]    216572111 [0,0]    206329411
     Vector Instruction Count    :     51758642 [0,1]     52015179 [0,0]     51886911
     Vector Element Count        :  12988808180 [0,1]  12993777771 [0,0]  12991292976
     FLOP Count                  :   5922735934 [0,1]   5922746854 [0,0]   5922741394
     MOPS                        :      5905.744 [0,0]     6558.563 [0,1]     6232.154
     MFLOPS                      :      2658.256 [0,0]     2957.758 [0,1]     2808.007
     Average Vector Length       :       249.807 [0,0]      250.950 [0,1]      250.378
     Vector Operation Ratio (%)  :        98.749 [0,0]       98.901 [0,1]       98.825
     Memory size used (MB)       :        65.045 [0,0]       65.045 [0,0]       65.045
     Global Memory size used (MB):        16.000 [0,0]       16.000 [0,0]       16.000
     MIPS                        :        97.202 [0,0]       97.924 [0,1]       97.563
     Instruction Cache miss (sec):         0.029 [0,0]        0.053 [0,1]        0.041
     Operand     Cache miss (sec):         0.013 [0,0]        0.030 [0,1]        0.022
     Bank      Conflict Time (sec):         0.012 [0,1]        0.012 [0,0]        0.012
```

図 30.　MPIPROGINF 出力例

(2)MPICOMMINF

本環境変数の指定により，全 MPI 手続き実行所要時間，MPI 通信待ち合わせ時間，送受信データ総量，および主要 MPI 手続き呼び出し回数が表示される．

MPI_COMM_WORLD(MPI_UNIVERSE=0)のランク 0 の MPI プロセスが MPI_FINALIZE 手続き中で標準エラー出力に対して行う．

MPICOMMINF の値と表示内容は以下の通り．

- NO　通信情報を出力しない(既定値)
- YES 最小値，最大値，および平均値を表示する
- ALL 最小値，最大値，平均値，および各プロセス毎の値を表示する

■ 出力例(YES 指定時)

```
MPI Communication Information:
------------------------

       Real MPI Idle Time (sec)    :        0.012 [0,0]        0.018 [0,1]        0.015
       User MPI Idle Time (sec)    :        0.011 [0,0]        0.018 [0,1]        0.015
       Total real MPI Time (sec)   :        0.196 [0,0]        0.236 [0,1]        0.216
       Send       count            :         1088 [0,0]         1088 [0,0]         1088
       Recv       count            :         1088 [0,0]         1088 [0,0]         1088
       Barrier    count            :            2 [0,0]            2 [0,0]            2
       Bcast      count            :            0 [0,0]            0 [0,0]            0
       Reduce     count            :            0 [0,0]            0 [0,0]            0
       Allreduce  count            :         5441 [0,0]         5441 [0,0]         5441
       Scan       count            :            0 [0,0]            0 [0,0]            0
       Exscan     count            :            0 [0,0]            0 [0,0]            0
       Redscat    count            :            0 [0,0]            0 [0,0]            0
       Gather     count            :            0 [0,0]            0 [0,0]            0
       Gatherv    count            :            0 [0,0]            0 [0,0]            0
       Allgather  count            :            0 [0,0]            0 [0,0]            0
       Allgatherv count            :            0 [0,0]            0 [0,0]            0
       Scatter    count            :            0 [0,0]            0 [0,0]            0
       Scatterv   count            :            0 [0,0]            0 [0,0]            0
       Alltoall   count            :            0 [0,0]            0 [0,0]            0
       Alltoallv  count            :            0 [0,0]            0 [0,0]            0
       Alltoallw  count            :            0 [0,0]            0 [0,0]            0
       Number of bytes sent        :      6266880 [0,0]      6266880 [0,0]      6266880
       Number of bytes recv        :      6266880 [0,0]      6266880 [0,0]      6266880
       Put        count            :            0 [0,0]            0 [0,0]            0
       Get        count            :            0 [0,0]            0 [0,0]            0
       Accumulate count            :            0 [0,0]            0 [0,0]            0
       Number of bytes put         :            0 [0,0]            0 [0,0]            0
       Number of bytes got         :            0 [0,0]            0 [0,0]            0
       Number of bytes accum       :            0 [0,0]            0 [0,0]            0
```

図 31.　MPICOMMINF 出力例

注意事項として，本機能は，プロファイル版 MPI ライブラリをリンクした場合に有効となる．プロファイル版 MPI ライブラリは，MPI プログラムのコンパイル/リンク用コマンド(mpif90 等)の -mpitrace，-mpiprof，-ftrace のいずれかのオプション指定によりリンクされる．

(3)MPISEPSELECT

本環境変数の指定により，標準出力および標準エラー出力の出力先を制御することができる．

- ➢ 値が 1　標準出力だけを stdout.$ID に出力する
- ➢ 値が 2　標準エラー出力だけを stderr.$ID に出力する
- ➢ 値が 3　標準出力を stdout.$ID に，標準エラー出力を stderr.$ID に出力する

➢ 値が 4　標準出力および標準エラー出力を std.$ID に出力する
➢ その他　標準出力も標準エラー出力もファイルに出力しない

3.4. SX の簡易性能解析機能

本機能は，サブルーチンや関数ごとの性能情報を採取するものである．本機能を利用するためには，測定対象のソースプログラムをコンパイルオプション -ftrace を指定してコンパイル・リンクする必要がある．プログラムを実行すると，カレントディレクトリに解析情報ファイルとして ftrace.out が生成される(MPI プログラムの場合は，グループ ID，ランク番号が付与された名前となる)．生成されたファイルを ftrace(super.isc.tohoku.ac.jp) または sxftrace(gen.isc.tohoku.ac.jp) コマンドに入力することによって，解析リストが標準出力ファイルに出力される．

【形式】 sxftrace f f trace.out

実行時オプションとして，環境変数 F_FTRACE を値 {YES FMT0 FMT1 FMT2} と設定することにより，ftrace コマンドを使用せず，プログラムの終了時に解析リストを標準エラーファイルへ出力することもできる．

■ 出力例(FMT1 指定時)

```
*----------------------*
 FLOW TRACE ANALYSIS LIST
*----------------------*

Execution : Thu May 17 11:59:51 2007
Total CPU : 0:00'03"677


 FREQ.     EXCLUSIVE      AVER.TIME     MOPS   MFLOPS  V.OP   AVER.   VECTOR  I-CACHE  O-CACHE  BANK     PROG.UNIT
           TIME[sec]( %)     [msec]                    RATIO  V.LEN   TIME    MISS     MISS     CONF

  5441      1.476( 40.1)      0.271   7725.2  3824.1  99.01  255.9    1.463   0.0018   0.0031   0.0001   inner_prod
     1      1.306( 35.5)   1306.130   6516.5  2591.0  99.42  256.0    1.301   0.0003   0.0006   0.0000   CG
  1088      0.891( 24.2)      0.819   7047.3  3162.0  98.72  239.3    0.884   0.0003   0.0012   0.0241   mut_vec
     1      0.004(  0.1)      4.455   2522.6     0.2  93.46  251.6    0.001   0.0007   0.0006   0.0000   main
-------------------------------------------------------------------------------------------------------------------
  6531      3.677(100.0)      0.563   7125.3  3221.0  99.07  251.8    3.649   0.0030   0.0055   0.0242   total


    ELAPSE  COMM.TIME COMM.TIME  IDLE TIME IDLE TIME AVER.LEN     COUNT     TOTAL LEN PROG.UNIT
     [sec]      [sec]  / ELAPSE      [sec]  / ELAPSE                      [byte]    [byte]

     1.477      0.004     0.003      0.000     0.000     16.0      5441     85.0K inner_prod
     1.308      0.000     0.000      0.000     0.000      0.0         0      0.0  Conjugate_Gradient
     0.891      0.000     0.000      0.000     0.000      0.0         0      0.0  mut_vec
     0.016      0.016     0.971      0.000     0.000      0.0         1      0.0  main
-------------------------------------------------------------------------------------------------------------------
```

図 32.　ftrace 出力例

注意事項として，翻訳時オプション -ftrace 指定でコンパイルされた手続きから，翻訳時オプション -ftrace 指定なしでコンパイルされた手続を呼び出している場合，呼び出し回数以外の測定値は，呼び出し先の手続の性能情報を含んだ値となる．

また，測定ルーチンには以下の定量的な制限がある．

- 呼び出される手続きの数の最大は 10,000
- 呼び出される手続きのネストの最大は 200

3.5. TX7 におけるコンパイル・実行

(1)TX7 におけるコンパイル方法

【形式】 mpif95 オプション MPI ソースファイル名

■ 主なオプション

- -parallel 自動並列化機能を利用する
- -O3/-O2/-O1 最適化のレベルを指定する
- MPI ソースファイル名
  FORTRAN のソースプログラムファイル名を指定する．複数のファイルを指定する場合は，空白で区切る．ソースファイル名には，サフィックス「.f90」か「.F90」(自由形式)，または「.f」か「.F」(固定形式)が必要．

(2)TX7 における実行方法

TX7 ではノード内の MPI 実行が可能である．また，バッチキューで実行可能な CPU 数まで MPI のプロセスを実行することができる．

【形式】 mpirun -np プロセス数 実行形式ファイル名

■ バッチリクエストスクリプトイメージ

```
# test job-MPI
#PBS -q キュー名              →  TX7 でのキュー名
#PBS -jo -N reqname                (キューの詳細はホームページ参照)
cd $PBS_O_WORKDIR
mpirun -np 8 ./a.out          →  8 プロセス実行の指定
```

図 33. バッチスクリプトイメージ(TX7)

4. まとめ

本記事は，並列化プログラミングの初心者を想定している．1 章では，並列化の概念から始めて，簡単な例題を通して，逐次処理プログラムを並列化するための検討方法について述べた．2 章では，逐次処理には無い，並列処理特有のテクニックを実際の例題を使って説明した．さらに 3 章では，作成した MPI プログラムのコンパイルから実行並びに性能解析までの操作ができるよう基本的なオプションの説明を行った．並列化プログラミング初心者の方でも一通りのことができるよう配慮したつもりである．本記事が，少しでも初学者の助けになれば幸いである．

なお，センターではMPIプログラムの相談を受け付けておりますので，MPIプログラムの開発等でご不明な点は是非 sodan05@isc.tohoku.ac.jp へお問い合わせ下さい．

参考文献

MPI に関する資料として以下がある．

・http://phase.hpcc.jp/phase/MPI-j/ml/mpi-j-html/contents.html

・東北大学情報シナジー機構情報シナジーセンター
講習会テキスト「MPI プログラミング入門」

・MPI 並列プログラミング，Peter S. Pacheco 著，秋葉　博訳

・並列プログラミング入門　MPI 版　青山幸也　著

ホームページ（大規模科学計算システム）

http://www.cc.tohoku.ac.jp/