物性物理で使える誰でもできるMPI並列計算の初歩

永井佑紀

平成29年7月6日

1　はじめに

最近は一つのCPUに複数のコアが積まれていることが多く、気軽にMPI並列計算ができるようになってきた。しかし、MPIになんとなく苦手な感情を持っており、食べず嫌いをしてしまって並列計算をしない人や、やりたいがどうしたらいいかわからない人がいるらしい。このPDFノートでは、「運動量空間での積分の並列化」を例にして、最小限の労力で並列計算を行うための方法を示す。なお、言語はFortran90とする。このPDFノートはあくまでMPI並列計算を使ってみるということに重点を置いており、MPIの詳細は別の文献を参照すること。最後にサンプルコードを載せる。

2　MPIの基本的な考え方

MPI並列計算は、異なる部屋での作業に似ている。並列計算の数だけ部屋があり、その中で何かの作業をしている。別々の作業をしている場合もあれば、異なる作業をしてい場合もある。もし、別の部屋で計算した結果が欲しかった場合、その部屋への電話（MPI通信）をかけて、持ってきてもらう。他の部屋に電話をかけない限りそれぞれの作業は完全に独立（メモリが独立）である。最後に結果が欲しかった場合は、全員に連絡して、データをかき集めてもらう。

3　コンパイル

まず、プログラムのコンパイルについて。Fortran90のコンパイルにはifortやgfortranなどが使われる。MPI並列計算を行いたい場合、これらの代わりにmpif90やmpiifort(多くの場合mpif90と思われる)を使うだけで、MPI並列計算用のコンパイルができる。

4　おまじない：初期化

まず、使いたいプログラムのmainの冒頭に、以下のmoduleを書いておこう。

ソースコード 1: 初期化。mainの前に置いておく。

```
1  module para
2    implicit none
3    include "mpif.h"
4    integer::myrank,nprocs
5
6  contains
7
8    subroutine mpi_initialize
9      implicit none
10     integer::ierr
11
12     call mpi_init(ierr)
```

```
13
14      call mpi_comm_rank(mpi_comm_world,myrank,ierr)
15      call mpi_comm_size(mpi_comm_world,nprocs,ierr)
16
17      if(myrank == 0) then
18          write(*,*) "Num␣of␣Parallel␣calculations:", nprocs
19      end if
20
21
22    end subroutine mpi_initialize
23
24    subroutine mpi_fin
25      implicit none
26      integer::ierr
27
28      call mpi_finalize( ierr )
29    end subroutine mpi_fin
30  end module para
```

ここには二つのサブルーチンを入れておいた。MPI 並列をしたいサブルーチンあるいはメインのコードの冒頭に

```
use para
```

とモジュールの呼び出しを書く。通常、プログラムの最初の main の部分で呼び出しておき、初期化のサブルーチンを

```
call mpi_initialize
```

と呼んでおく。これを呼ぶことで、それぞれのプロセスの名前「myrank」と、全部でいくつの並列数で計算をしているかという「nprocs」をセットできる。
なお、MPI 計算の最後には、

```
call mpi_fin
```

を付け加えておいたほうが良い。

5 Do ループの並列化：和

5.1 1 次元

以下のような Do ループを考える (ソースコード 2)。

ソースコード 2: 初歩的な Do ループの例

```
1  subroutine test(wa)
2    implicit none
3    real(8),intent(out)::wa
4    integer::i,N
5
6    N= 100
7    wa = 0d0
8
9    loop:do i = 1,N
10       wa = wa + dble(i**2)
11   end do loop
12   return
13
14 end subroutine test
```

このコードでは、サブルーチン test は、$i = 1$ から $i = N$ までの i^2 の和を計算する。このコードを MPI 並列化してみよう。具体的なコードは、こちら（ソースコード 3）である。

ソースコード 3: 初歩的な Do ループの MPI 並列計算

```
1  subroutine test_mpi(wa)
2    use para
3    implicit none
4    real(8),intent(out)::wa
5    integer::i,N
6  ,   ,!,For, ,MPI
7    integer::ista,iend
8    integer::nbun
9    real(8)::wa_tmp
10   integer::m
11   integer::ierr
12
13   N= 100
14   wa = 0d0
15   nbun = N/nprocs
16   ista = myrank*nbun+1
17   iend = ista + nbun - 1
18   m = 1
19
20   loop:do i = ista,iend
21       wa = wa + dble(i**2)
22   end do loop
23
24   call mpi_allreduce(wa,wa_tmp,m,MPI_double_precision,MPI_SUM,MPI_COMM_WORLD,ierr)
25   wa = wa_tmp
26
27   return
28
29 end subroutine test_mpi
```

このコードでは、ループの変数 i を各 MPI プロセスごとに割り振りをしており、Do ループを分割している。そして、それぞれのプロセスでの計算が終わった後、最後にそれぞれのプロセスの和をとってきている。なお変数 m は集めたい変数の数で、今は変数 wa は一要素だけなので $m = 1$ である。もし、wa が配列であれば、m には配列の数が入る。このコードでは、Do ループの回数 N はプロセス数 nprocs で割り切れることを仮定していることに注意。MPI 並列による Do ループの並列計算はこれだけである。

5.2 2 次元

具体的な物理の計算においては、二次元での運動量空間での積分をすることがよくある。例えば、以下のコード（ソースコード 4）である。

ソースコード 4: 二重 Do ループ

```
1  subroutine test2d(wa)
2    implicit none
3    real(8),intent(out)::wa
4    integer::Nx,Ny
5    integer::ikx,iky
6    real(8)::pi,dkx,dky
7    real(8)::kx,ky
8
9    Nx= 100
10   Ny = 100
11   wa = 0d0
12   pi = atan(1d0)*4d0
13   dkx = 2d0*pi/dble(Nx-1)
14   dky = 2d0*pi/dble(Ny-1)
15
16   loopx:do ikx = 1,Nx
17       kx = dble(ikx-1)*dkx - pi
18       loopy:do iky = 1,Ny
19           ky = dble(iky-1)*dky - pi
20           wa = wa + (cos(kx) + cos(ky))
21       end do loopy
22   end do loopx
23   wa = wa/dble(Nx*Ny)
```

```
24
25    return
26
27  end subroutine test2d
```

このコードでは、変数 k_x と k_y が $-\pi$ から π まで動いている。このコードの MPI 並列計算のコードは、こちら（ソースコード 5）である。

ソースコード 5: 二重 Do ループの MPI 並列計算

```
1   subroutine test2dmpi(wa)
2     use para
3     implicit none
4     real(8),intent(out)::wa
5     integer::Nx,Ny
6     integer::ikx,iky,i
7     real(8)::kx,ky
8     real(8)::pi,dkx,dky
9   ,     ,!,For, ,MPI
10    integer::ista,iend
11    integer::nbun
12    real(8)::wa_tmp
13    integer::m
14    integer::ierr
15
16
17    Nx= 100
18    Ny = 100
19    nbun = Nx*Ny/nprocs
20    ista = myrank*nbun+1
21    iend = ista + nbun - 1
22    m = 1
23
24
25
26    wa = 0d0
27    pi = atan(1d0)*4d0
28    dkx = 2d0*pi/dble(Nx-1)
29    dky = 2d0*pi/dble(Ny-1)
30
31  ,   ,!,i,=(,iky,-1)*,Nx,+,ikx
32    loop:do i = ista,iend
33        ikx = mod(i - 1,Nx) + 1
34        iky =(i-ikx)/Nx + 1
35
36        kx = dble(ikx-1)*dkx - pi
37        ky = dble(iky-1)*dky - pi
38        wa = wa + (cos(kx) + cos(ky))
39    end do loop
40    call mpi_allreduce(wa,wa_tmp,m,MPI_double_precision,mpi_sum,MPI_COMM_WORLD,ierr)
41    wa = wa_tmp/dble(Nx*Ny)
42
43    return
44
45  end subroutine test2dmpi
```

ここで、Do ループを分割するために、二重だったループを一重に変更している。このコードでの wa を計算する部分を、自己エネルギーなりグリーン関数なりの計算に置き換えれば、運動量空間の積分を実行することができる。次に、wa が複素数の行列の場合のコードも載せておく（ソースコード 6）

ソースコード 6: 二重 Do ループの MPI 並列計算

```
1   subroutine test2dmpi_green(mat_wa)
2     use para
3     implicit none
4     complex(8),intent(out)::mat_wa(1:2,1:2)
5     integer::Nx,Ny
6     integer::ikx,iky,i
7     real(8)::pi,dkx,dky
8     real(8)::kx,ky
9   ,     ,!,For, ,MPI
```

```
10    integer::ista,iend
11    integer::nbun
12    complex(8)::mat_wa_tmp(1:2,1:2)
13    integer::m
14    integer::ierr
15
16
17    Nx= 100
18    Ny = 100
19    nbun = Nx*Ny/nprocs
20    ista = myrank*nbun+1
21    iend = ista + nbun - 1
22    m = 4
23
24
25
26    mat_wa = 0d0
27    pi = atan(1d0)*4d0
28    dkx = 2d0*pi/dble(Nx-1)
29    dky = 2d0*pi/dble(Ny-1)
30
31 ,   ,!,i,=(,iky,-1)*,Nx,+,ikx
32    loop:do i = ista,iend
33       ikx = mod(i - 1,Nx) + 1
34       iky =(i-ikx)/Nx + 1
35
36       kx = dble(ikx-1)*dkx - pi
37       ky = dble(iky-1)*dky - pi
38       mat_wa(1,1) = mat_wa(1,1) + (cos(kx) + cos(ky))
39       mat_wa(2,2) = mat_wa(2,2) - (cos(kx) + cos(ky)) +1
40
41    end do loop
42    call mpi_allreduce(mat_wa,mat_wa_tmp,m,MPI_double_complex,mpi_sum,MPI_COMM_WORLD,ierr)
43    mat_wa = mat_wa_tmp/dble(Nx*Ny)
44
45 end subroutine test2dmpi_green
```

この場合、MPI で通信する要素数は 2×2 行列なので、$m = 4$ である。

6 Do ループの並列化：並列要素ごとの作業

次に、物性物理でよく使われる、配列要素ごとの並列計算について述べる。並列計算しない元のコードはこちら（ソースコード 7）である。

ソースコード 7: 配列の計算

```
1  subroutine test_h(nvec,vec_a)
2     implicit none
3     integer,intent(in)::nvec
4     real(8),intent(out)::vec_a(1:nvec)
5     integer::i,N,j
6
7     N=nvec
8     vec_a = 0d0
9     loop:do i = 1,N
10       do j = 1,100000
11          vec_a(i) = vec_a(i) + cos(dble(i**2))*sin(dble(j**2))
12       end do
13
14    end do loop
15
16    return
17 end subroutine test_h
```

このコードでは、Do ループのそれぞれの i が配列のそれぞれの配列要素の計算をしており、ループの一つ一つは独立である。このような独立なループの場合、MPI 並列化は容易である。例えば、以下のようなコード（ソースコード 8）となる。

ソースコード 8: 配列の並列計算

```
subroutine test_hmpi(nvec,vec_a)
  use para
  implicit none
  integer,intent(in)::nvec
  real(8),intent(out)::vec_a(1:nvec)
  integer::i,N,j
!     ,!,For, ,MPI
  integer::ista,iend,ii
  integer::nbun
  real(8),allocatable::vec_atemp(:)
  integer::m
  integer::ierr

  N=nvec
  nbun = N/nprocs
  ista = myrank*nbun+1
  iend = ista + nbun - 1
  m = nbun
  allocate(vec_atemp(1:m))

  vec_a = 0d0
  ii = 0
  loop:do i = ista,iend
     ii = ii + 1
     do j = 1,100000
        vec_atemp(ii) = vec_atemp(ii) + cos(dble(i**2))*sin(dble(j**2))
     end do
  end do loop
  call mpi_allgather(vec_atemp,m,MPI_DOUBLE_precision,vec_a,m,MPI_DOUBLE_precision,
       MPI_COMM_WORLD,ierr)

  return
end subroutine test_hmpi
```

この allgather を用いると、最終的にすべてのプロセスで同じ配列を持つことになる。

7 サンプルコード

サンプルコードにはここで紹介したすべてのコードが入っている。また、test.f90 を

```
mpif90 test.f90
```

でコンパイルした後、

```
mpirun -np 1 ./a.out
```

で実行すると並列数 1、

```
mpirun -np 2 ./a.out
```

で実行すると並列数 2 となる。数字を増やしていけば計算が速くなることが実感できるはずである。