OpenMPによる並列計算の基本

2017年2月3日（金）

佐藤健太郎

アウトライン

- 並列計算の種類

- OpenMPによる並列計算
  - 並列計算の指示
  - ループの並列化
  - 足し合わせ
  - ループの計算順序の調整

計算の並列化

Fortranのループ例

```
do i = 1, 200
   a(i) = b(i) + c(i)
end do
```

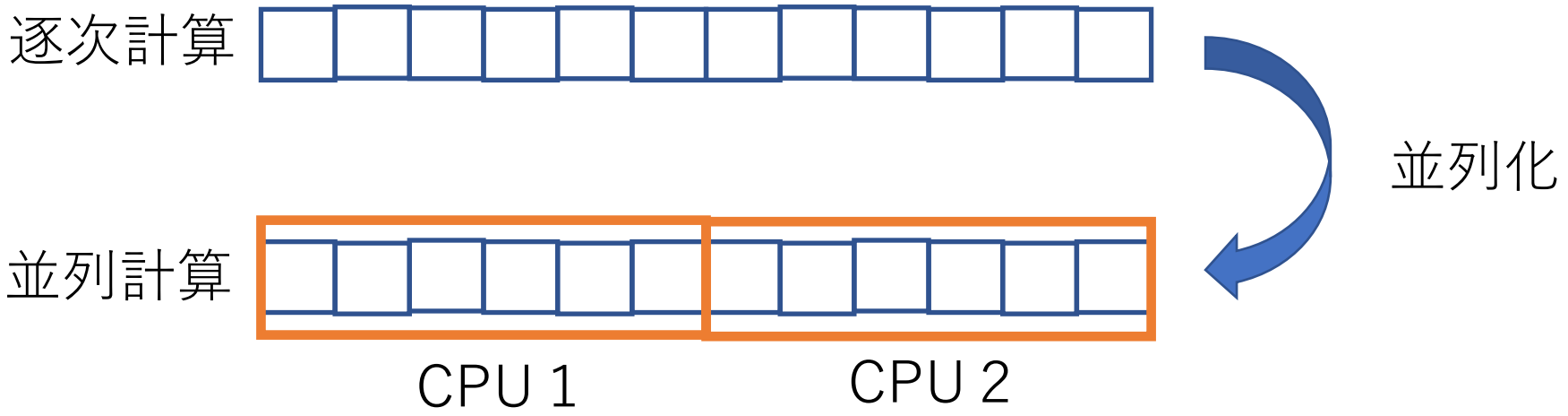
逐次計算

CPU 1

a(1)=b(1)+c(1)
↓
a(2)=b(2)+c(2)
↓
……

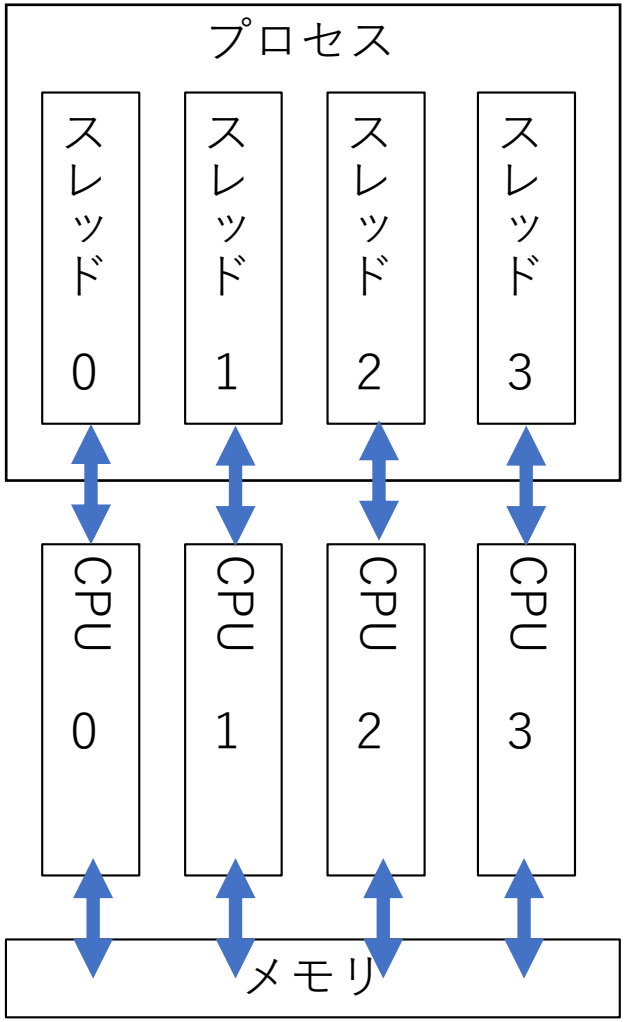
並列計算

CPU 1	CPU 2
a(1)=b(1)+c(1) ↓ …… ↓ a(100)=b(100)+c(100)	a(101)=b(101)+c(101) ↓ …… ↓ a(200)=b(200)+c(200)



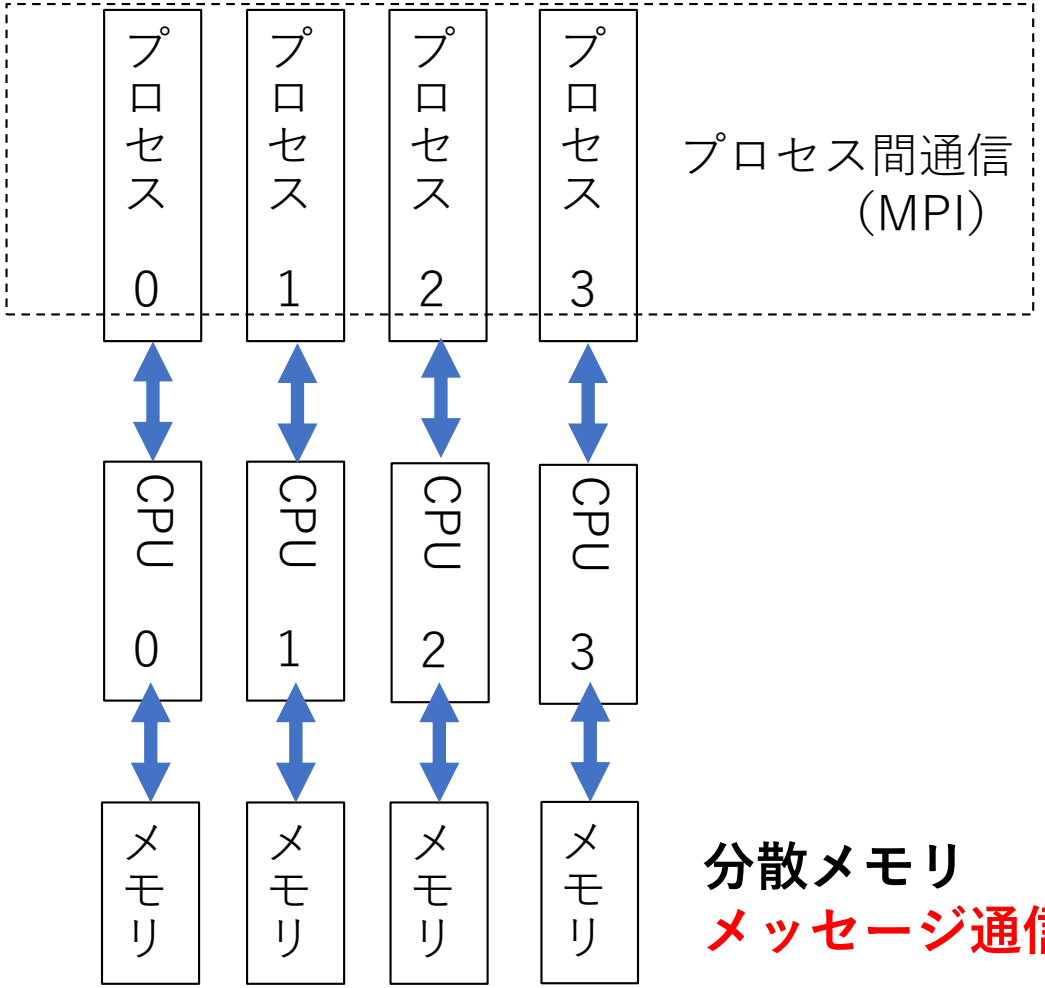
スレッドとプロセス

マルチスレッド（OpenMP）



共有メモリ

マルチプロセス（MPI）



分散メモリ

メッセージ通信

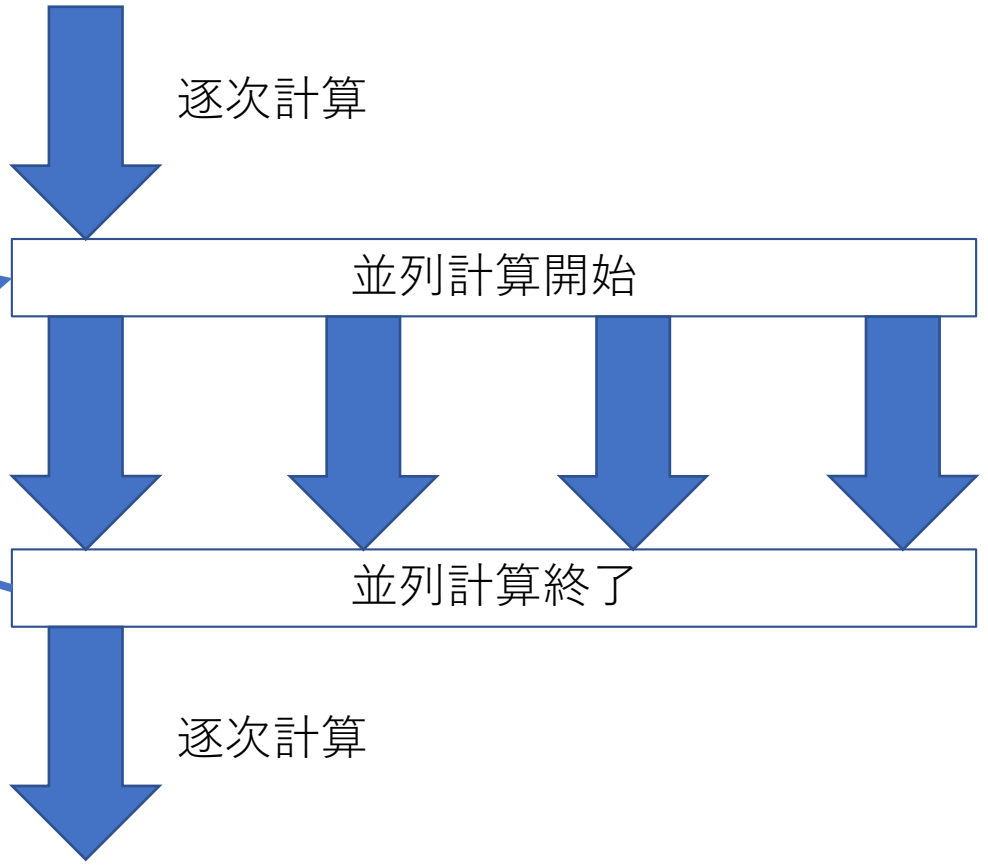
並列プログラミングの特徴

	実装	**データの移動法**	**データ移動の制御**	**プログラミングの難易度**	**使える環境**	**CPU数と並列化性能の関係**
メッセージ通信	MPI	メッセージ(データ)を送受信	プログラマ	難しい	分散メモリ環境 共有メモリ環境	CPUが増えれば性能上昇
共有メモリ	OpenMP	共有メモリを使う	システム	簡単	共有メモリ環境	16個くらいまでは性能上昇

OpenMPの基本

sample1.f90

```
program sample
  !$ use omp_lib
  implicit none

  write(*,*) "start"

  !$omp parallel
  write(*,*) "hello"
  !$omp end parallel

  write(*,*) "stop"

end program sample
```



- OpenMPは「!$omp」ではじまる指示文で制御
- 指示文では大文字･小文字は区別されない

OpenMPの基本

sample1.f90

```
program sample
  !$ use omp_lib
  implicit none

  write(*,*) "start"

  !$omp parallel
  write(*,*) "hello"
  !$omp end parallel

  write(*,*) "stop"

end program sample
```

コンパイル、実行、結果

```
> ifort -qopenmp sample1.f90 -o sample1.out
> env OMP_NUM_THREADS=4 ./sample1.out
 start
 hello
 hello
 hello
 hello
 stop

> ifort sample1.f90 -o sample1.out
> ./sample1.out
 start
 hello
 stop
```

-qopenmpがあると並列計算

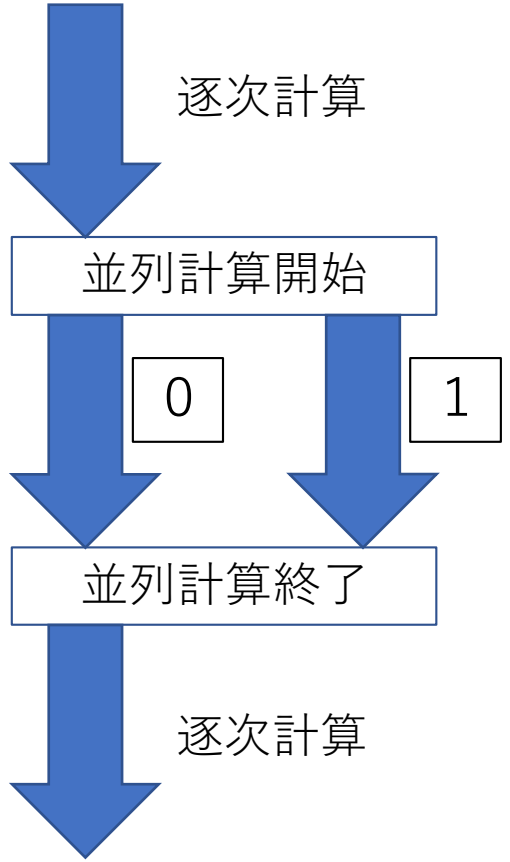
-qopenmpがないと逐次計算

- コンパイル時にOpenMP用のオプション（-qopenmp）がないと指示文はコメントになり無視される
- 並列数は環境変数「OMP_NUM_THREADS」で指定する

ループの並列化（1）

sample2.f90

```
program sample
  !$ use omp_lib
  implicit none

  integer, parameter :: MAX = 8
  integer :: i

  !$omp parallel do
  do i = 1, MAX
     write(*,FMT='(2I4)') i, omp_get_thread_num()
  end do
  !$omp end parallel do

end program sample
```



- 「*!$omp parallel do*」の直後のループが並列化される

ループの並列化（1）

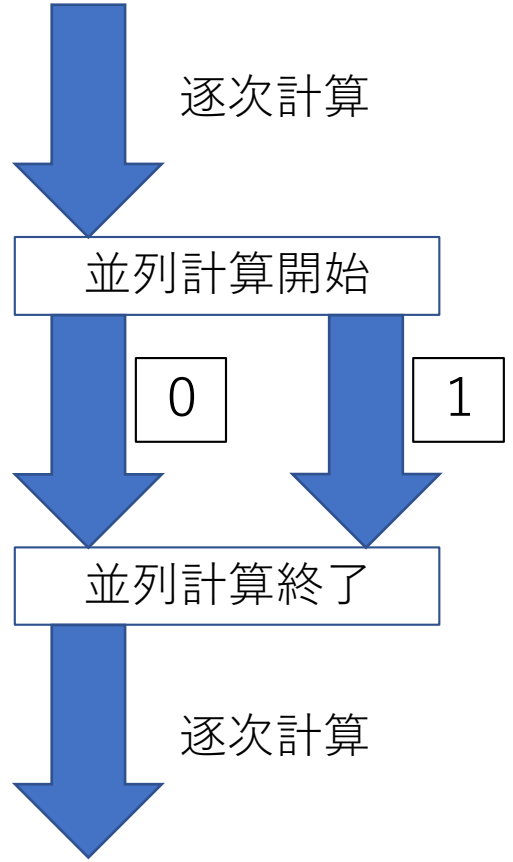
```
!$omp parallel do
do i = 1, MAX
   write(*,FMT='(2I4)') i, omp_get_thread_num()
end do
!$omp end parallel do
```

スレッド番号を取得する

コンパイル、実行、結果

```
> ifort -qopenmp sample2.f90 -o sample2.out
> env OMP_NUM_THREADS=2 ./sample2.out
```

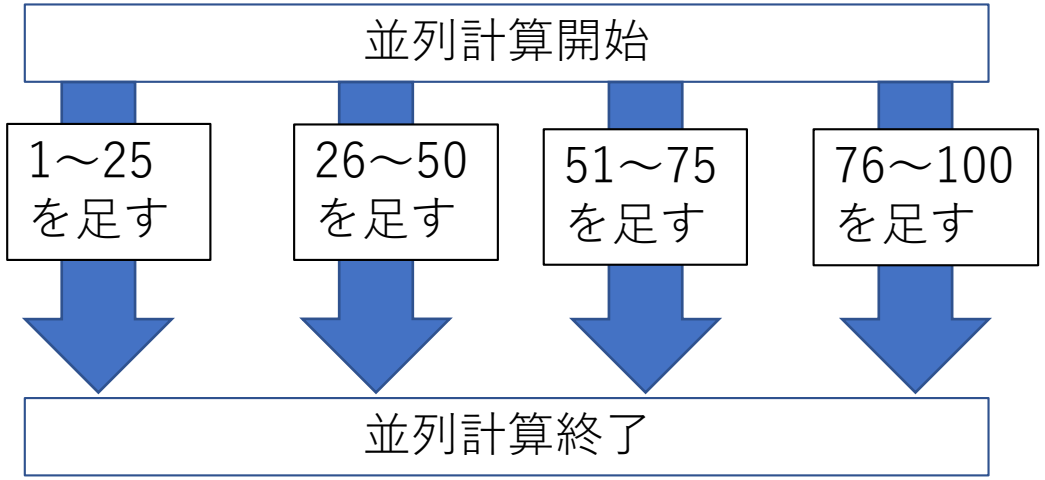
ループの順番	スレッド番号
1	0
5	1
2	0
6	1
3	0
7	1
4	0
8	1



ループの並列化（2）reductionの使い方

sample3.f90

```
program sample
  !$ use omp_lib
  implicit none

  integer :: i, sum

  sum = 0
  !$omp parallel do
  do i = 1, 100
     sum = sum + i
  end do
  !$omp end parallel do

  write(*,*) sum

end program sample
```

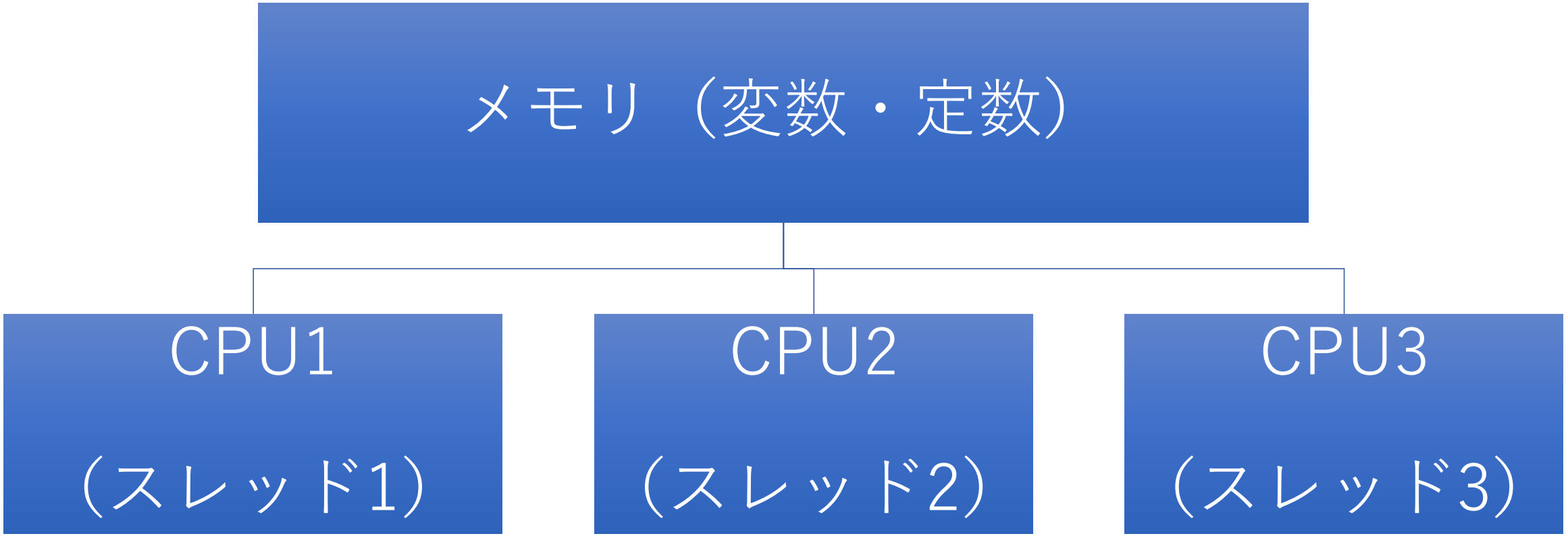


コンパイル、実行、結果

```
> ifort -qopenmp sample3.f90 -o sample3.out
> env OMP_NUM_THREADS=4 ./sample3.out
      4207
> env OMP_NUM_THREADS=4 ./sample3.out
      5050
> env OMP_NUM_THREADS=4 ./sample3.out
      3833
```

?

OpenMPはスレッド間でメモリを共有する

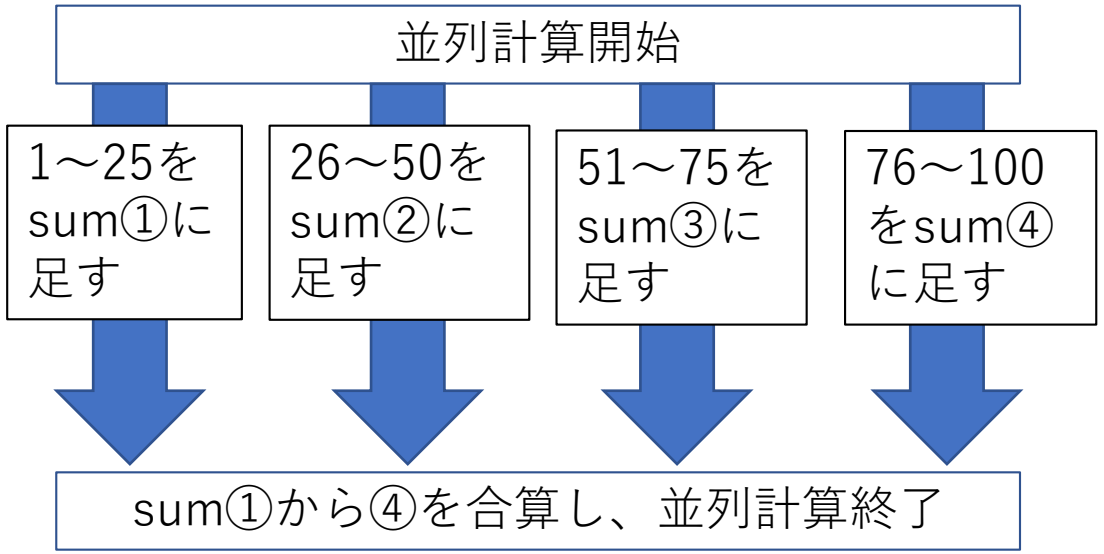


ループの並列化（2）reductionの使い方

正しいsample3.f90

```
program sample
  !$ use omp_lib
  implicit none

  integer :: i, sum

  sum = 0
  !$omp parallel do reduction(+:sum)
  do i = 1, 100
     sum = sum + i
  end do
  !$omp end parallel do

  write(*,*) sum

end program sample
```

スレッドごとにsumを準備し、最後に合算する



ループの並列化（3）privateの使い方

sample4.f90　円周率πの値を計算するプログラム

```
program sample
  !$ use omp_lib
  implicit none

  integer, parameter :: num_rects = 1000000
  double precision :: mid, height, width, sum, area
  integer :: i

  width = 1.0D0 / dble(num_rects)
  sum = 0.0D0
  !$omp parallel do reduction(+:sum) private(mid,height)
  do i = 0, num_rects-1
     mid = (dble(i) + 0.5D0)*width
     height = 4.0D0 / (1.0D0 + mid*mid)
     sum = sum + height
  end do
  !$omp end parallel do

  area = width * sum
  write(*,*) "PI =", area
end program sample
```

円周率πの値を計算する

$$\pi = \int_0^1 \frac{4}{1+x^2} dx$$

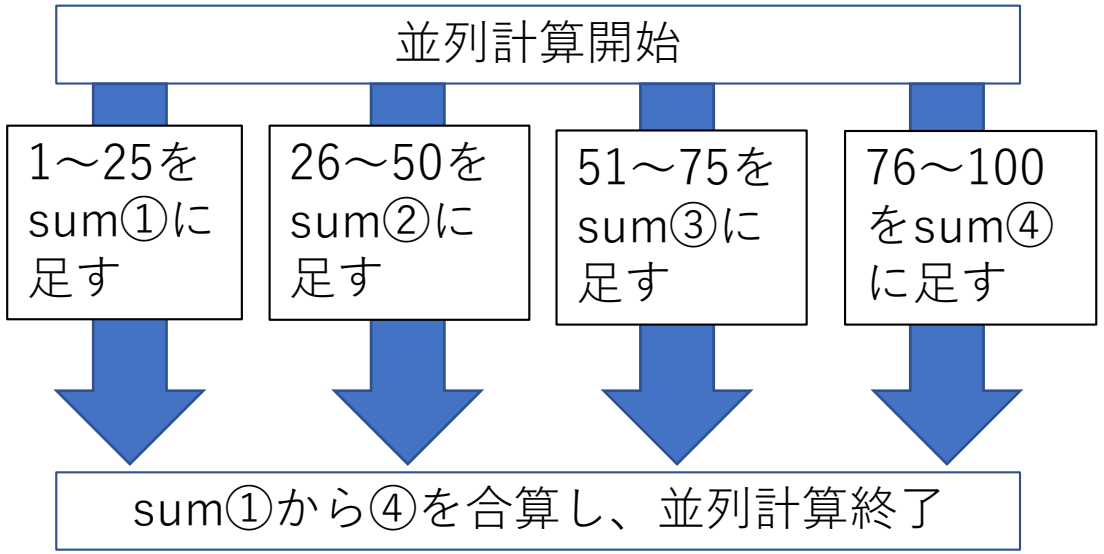
privateで指定した変数はローカルな変数になり、スレッドごとに別の値をとる

ループの並列化（3）privateの使い方

正しいsample3.f90

```
program sample
  !$ use omp_lib
  implicit none

  integer :: i, sum

  sum = 0
  !$omp parallel do reduction(+:sum)
  do i = 1, 100
     sum = sum + i
  end do
  !$omp end parallel do

  write(*,*) sum

end program sample
```

スレッドごとにsumを準備し、最後に合算する

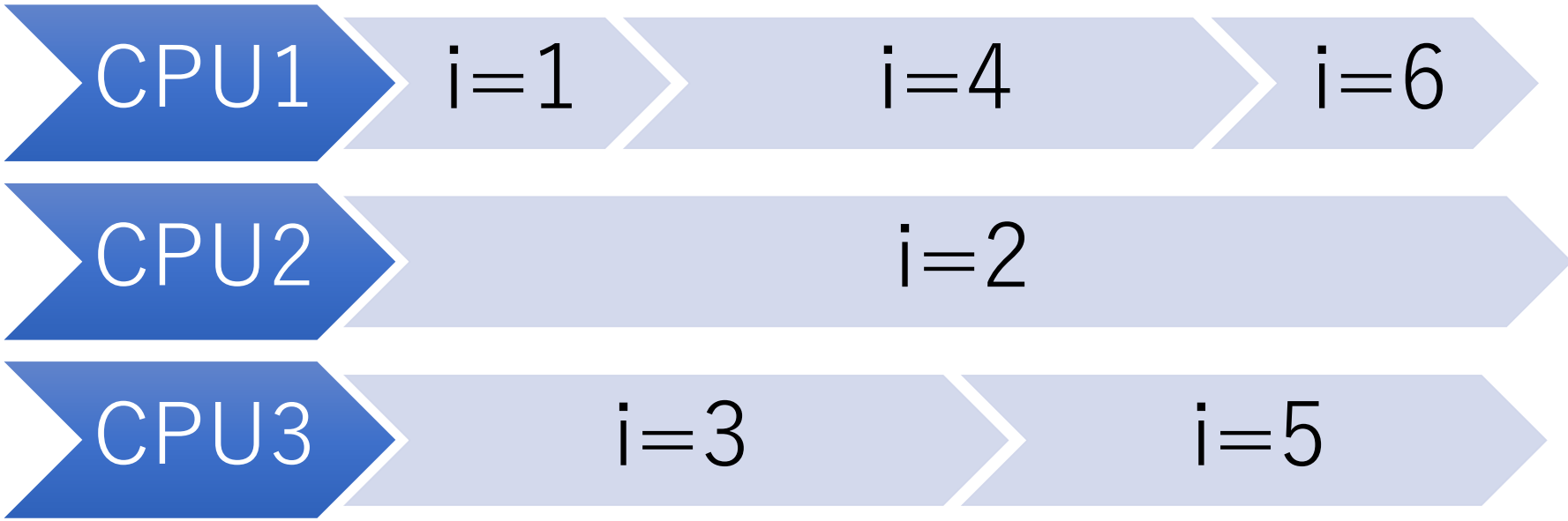


ループの並列化（4）実行順序の制御

サブルーチンfuncの終了時間が引数aに依存する

```
!$omp parallel do schedule(dynamic)
do i = 1, 100
   call func(a(:))
end do
!$omp end parallel do
```

処理が終わったスレッドは次の処理にかかる



まとめと参考文献

- OpenMPを使うと任意のループを簡単に並列化できる

- 参考文献
  - 並列コンピューティング技法　Clay Breshears　O'REILY
  - FortranプログラマのためのOpenMP入門
    https://web.kudpc.kyoto-u.ac.jp/Archives/PDF/NewsLetter/2003-6_openmp.pdf
  - Intel Fortran Compiler Developer Guide and Reference