

計算科学演習 I (第10回)

MPIを用いた並列計算 (II)

神戸大学大学院システム情報学研究科

横川 三津夫

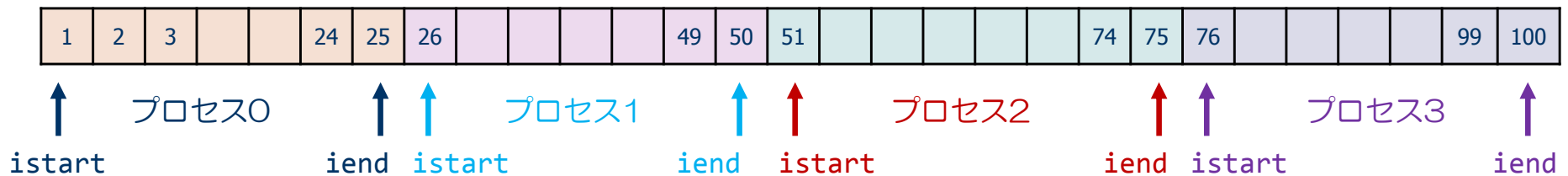
yokokawa@port.kobe-u.ac.jp

今週の講義の概要

1. 前回課題の解説
2. 計算時間の計測
3. 集団通信 (`mpi_allreduce`関数)

演習8-3 (提出課題1) 【再掲】

- 1 から 100 までの整数の和を求めるプログラムを、4並列で実行できるように修正し、4プロセスで実行せよ。
 - ◆ `mpi_send`, `mpi_recv`関数だけを使うこと。
- プログラム改良のポイント
 - ◆ 各プロセスの部分和を計算する範囲を、`myrank` をうまく使って求める。



- ◆ `myrank` ≠ 0 以外のプロセスから、プロセス0 (`myrank=0`) に部分和を送信する。
- ◆ プロセス0 (`myrank=0`) は、他の3つのプロセスから送られた部分和を受信し、全体の和を計算する。

演習8-3 プログラム例

```
program sum100_by_mpi
use mpi
implicit none
integer :: i, istart, iend, isum_local, isum_tmp, isum, src
integer :: nprocs, myrank, ierr
integer :: istat(MPI_STATUS_SIZE)
call mpi_init( ierr )
call mpi_comm_size( MPI_COMM_WORLD, nprocs, ierr )
call mpi_comm_rank( MPI_COMM_WORLD, myrank, ierr )
istart = myrank*25 + 1
iend   = (myrank+1)*25
isum_local = 0
do i = istart, iend
    isum_local = isum_local + i
enddo
if( myrank /= 0 ) then
    call mpi_send( isum_local, 1, MPI_INTEGER, 0, 100, MPI_COMM_WORLD, ierr )
else
    isum = isum_local
    do src = 1, 3
        call mpi_recv( isum_tmp, 1, MPI_INTEGER, src, 100, MPI_COMM_WORLD, istat, ierr )
        isum = isum + isum_tmp
    end do
end if
if( myrank == 0 ) print *, 'sum =', isum
call mpi_finalize( ierr )
end program sum100_by_mpi
```

解答例

```
if( myrank == 1 ) then
  call mpi_send( isum_local, 1, MPI_INTEGER, 0, 100, MPI_COMM_WORLD, ierr )
end if

if( myrank == 2 ) then
  call mpi_send( isum_local, 1, MPI_INTEGER, 0, 200, MPI_COMM_WORLD, ierr )
end if

if( myrank == 3 ) then
  call mpi_send( isum_local, 1, MPI_INTEGER, 0, 300, MPI_COMM_WORLD, ierr )
end if

if( myrank == 0 ) then
  call mpi_recv( isum_tmp1, 1, MPI_INTEGER, 1, 100, MPI_COMM_WORLD, istat, ierr )
  call mpi_recv( isum_tmp2, 1, MPI_INTEGER, 2, 100, MPI_COMM_WORLD, istat, ierr )
  call mpi_recv( isum_tmp3, 1, MPI_INTEGER, 3, 100, MPI_COMM_WORLD, istat, ierr )
end if
```

演習8-5（提出課題2）【再掲】

- プログラム M-3を以下のように書き換え，8プロセスで実行せよ.
 - ◆ 変数 `isum_local`, `isum`を倍精度実数 `sum_local`, `sum`に変更
 - ◆ `mpi_reduce`でのリダクション演算も倍精度で行うように変更
- 修正の方針
 - ◆ 倍精度実数型変数の宣言の仕方を思い出す.
 - ◆ `mpi_reduce`内の`datatype`を`MPI_REAL8`に変更する.

演習8-5 プログラム例 → MPIプログラムM-4

```
program dsumn
use mpi
implicit none
integer, parameter :: SP = kind(1.0)
integer, parameter :: DP = selected_real_kind(2*precision(1.0_SP))
integer :: n, i, istart, iend
real(DP) :: sum_local, sum
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 )

if( myrank==0) n=10000
call mpi_bcast( n, 1, MPI_INTEGER, 0, MPI_COMM_WORLD, ierr )
istart = (n/nprocs)*myrank + 1
iend   = (n/nprocs)*(myrank+1)
sum_local = 0.0
do i = istart, iend
    sum_local = sum_local + i
enddo
call mpi_reduce( sum_local, sum, 1, MPI_REAL8, MPI_SUM, 0, MPI_COMM_WORLD, ierr )

if( myrank == 0 ) print *, 'sum (double precision) =', sum

call mpi_finalize( ierr )
end program sum_real8
```

計算時間の計測

- 並列計算の目的は，計算時間の短縮にある。
- 同じ結果が得られるが，アルゴリズムや書き方が異なったプログラムのうち，どれが一番良いか？
 - ◆ （正しい結果が得られるならば）計算時間の短いものが良い。
- 計算時間を計って比較する。

計算時間を計測する方法

```
real(DP) :: time0, time2
```

```
·  
·
```

```
call mpi_barrier( MPI_COMM_WORLD, ierr )  
time0 = mpi_wtime()
```

(計測する部分)

```
call mpi_barrier( MPI_COMM_WORLD, ierr )  
time1 = mpi_wtime()
```

(time1-time0 を出力する)

計測のための変数を倍精度実数型で宣言する。

MPI_barrier関数で、計測開始の足並みを揃える。
mpi_wtime関数で開始時刻をtime0に設定

全プロセスで終了の足並みを揃える。
mpi_wtime関数で終了時刻をtime1に設定

time1-time0が計測した部分の計算時間となる。

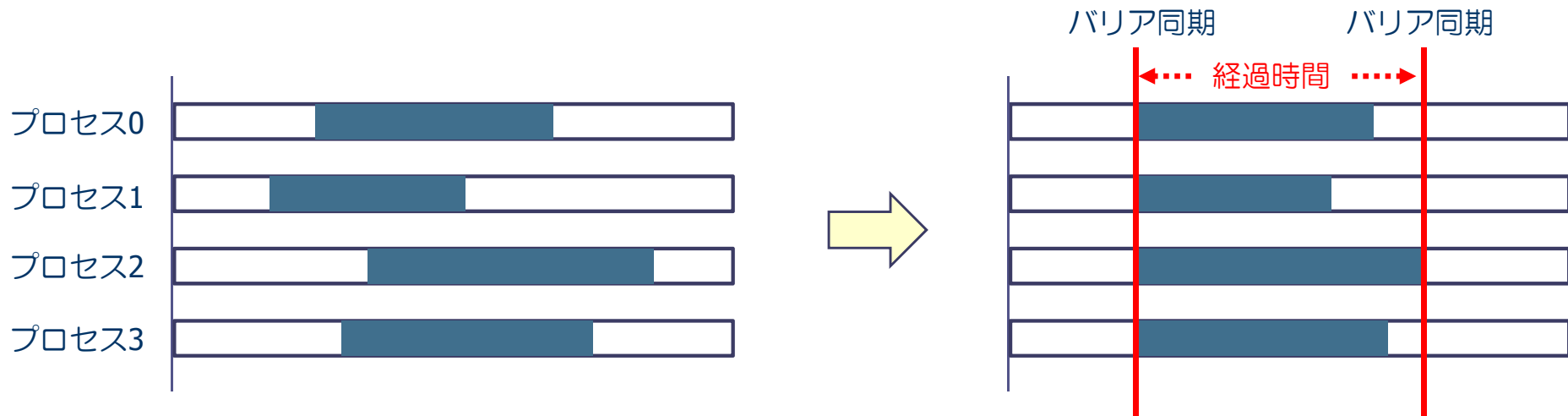
`mpi_barrier(comm, ierr)` : バリア同期関数

- ◆ `comm`: コミュニケータ (例えば, `MPI_COMM_WORLD`)
- ◆ `ierr`: 戻りコード (整数型)

`var = mpi_wtime()` : 倍精度実数を返す関数

時間計測のイメージ

- 各プロセスでの計算時間の測定関数
 - ◆ `mpi_wtime()`
 - ある時点を基準とした経過秒数を倍精度実数型で返す関数
- プログラムのある区間の計算時間の測定
 - ◆ プログラムの実行は各プロセスで独立なので、開始時間や終了時間が異なる。
 - ◆ ある部分の計算時間の計測では、**バリア同期** (MPI_barrier) により測定開始と測定終了の**足並みを揃えて**、計測する。



演習9-1

- 演習8-5で作成したプログラムM-4について、次の区間の計算時間をプロセス数を変えて計測せよ。
 - ◆ mpi_bcastの開始からmpi_reduceの終了までの時間
 - ◆ 計測した時間は、ランク0のプロセスに出力させる。
 - ◆ $n=10,000,000$ として、1, 2, 4, 8プロセスで実行し、それぞれ結果が正しいことを確かめる。
 - ◆ 計算時間を、gnuplotを使って図示せよ。
 - Xming, ssh forwarding を使う。
 - x軸をプロセス数, y軸を計算時間とする。
 - x軸, y軸ともログスケールにする。 `$ set logscale xy`

※ /tmp/cpmmpi /M-4/dsumn.f90 を参考にせよ。

集団通信（mpi_allreduce関数）

- mpi_reduce関数は、ひとつのプロセスに、すべてのプロセスの同じ変数のデータを集め、（足し算や掛け算などの）リダクション演算を行った結果を得る関数
- すべてのプロセスで、この結果を共有したいときは、どうすれば良いか？
- プログラム方針
 - ① mpi_reduce関数により、ひとつのプロセスで結果をもとめ、その結果を mpi_bcastで全プロセスに放送する。
 - ② mpi_allreduce関数を使う。

集団通信 — mpi_allreduce()

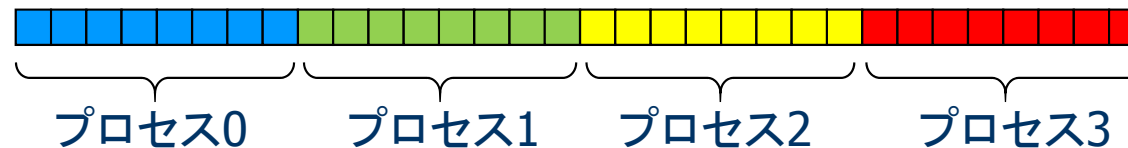
```
mpi_allreduce( sendbuff, recvbuff, count, datatype, op, comm, ierr )
```

※ mpi_reduceとmpi_bcastを同時に行える関数。すべてのプロセスで同じ結果（総和など）が得られる。

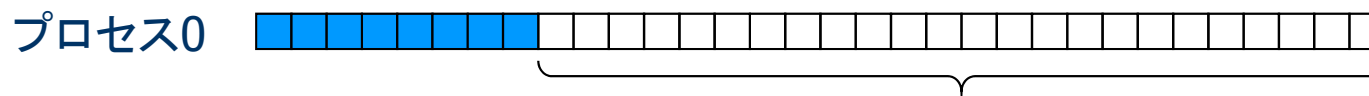
- ◆ sendbuff: 送信するデータの変数名（先頭アドレス）
- ◆ recvbuff: 受信するデータの変数名（先頭アドレス）
- ◆ count: データの個数（整数型）
- ◆ datatype: 送信するデータの型
 - MPI_INTEGER, MPI_REAL8, MPI_CHARACTER など
- ◆ op: 集まってきたデータに適用する演算の種類
 - MPI_SUM（総和）, MPI_PROD（掛け算）, MPI_MAX（最大値）など
- ◆ comm: コミュニケータ（例えば, MPI_COMM_WORLD）
- ◆ ierr: 戻りコード（整数型）

演習9-2：ベクトルの正規化

- n 次元ベクトル x の第 i 要素を i とする ($x(i) = i$) .
- このとき, x を正規化したベクトル $x/\|x\|_2$ を求めるプログラムを作成せよ.
 - ◆ $\|x\|_2$ は x の各要素の2乗和の平方根である.
 - ◆ ベクトルは, ブロック分割で各プロセスに配置する.
- 各プロセスの担当する要素 ($nprocs$ はMPIプロセス数)
 - ◆ $istart = (n/nprocs)*myrank + 1$
 - ◆ $iend = (n/nprocs)*(myrank+1)$



- ベクトルの格納方法
 - ◆ 各プロセスは長さ n の配列を持ち, そのうち自分の担当部分のみを使う



プロセス0では, この部分が使われない

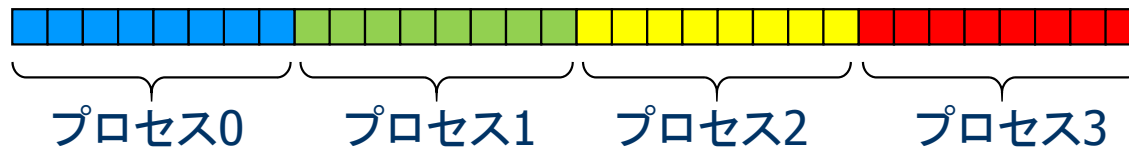
演習9-2：ベクトルの正規化（方針）

- プログラムM-4をベースに修正する.
- まず、各プロセスが自分の担当分の要素について、2乗和を計算する.
- 全プロセスの総和を `mpi_allreduce`関数で求める.
- 各プロセスは `mpi_allreduce` の結果を用いて、自分の担当する要素について正規化を行う.
- $n=1000$ としてプロセス数を1, 2, 4, 8と変えて計算せよ.
- 正規化されたベクトルの要素は $\tilde{x}(i) = i / \sqrt{n(n+1)(2n+1)/6}$ であるので、これと比較をして、計算が正しくできていることを確かめよ.

データの分割方法

1. ブロック分割

- ◆ 全体のデータ（配列など）を、連続したアドレスの小部分に分割して、各プロセスに割り当てる方法



2. サイクリック分割

- ◆ 全体のデータを、要素1個ずつ、サイクリックに各プロセスに割り当てる方法



3. ブロックサイクリック分割

- ◆ 全体のデータを、複数の要素の塊にして、サイクリックに各プロセスに割り当てる方法

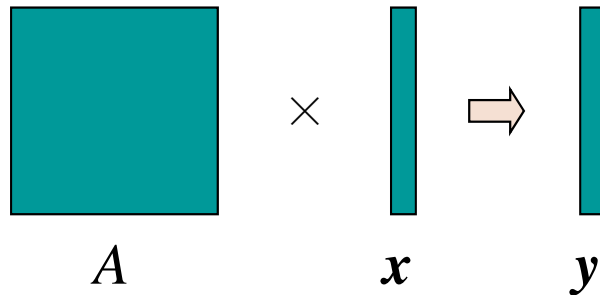


- 通信量，負荷分散などを考慮し，最適な分割を決める必要がある。

リダクション演算の応用：行列ベクトル積

■ 問題

- ◆ $n \times n$ 行列 A の第 (i, j) 要素を $i + j$ とする ($a_{i,j} = i + j$) .
- ◆ n 次元ベクトル x の第 i 要素を i とする ($x_i = i$) .
- ◆ このとき, $y = Ax$ を計算する.



■ プログラムのカーネル（主要な部分）

- ◆ ベクトル y の要素を1個ずつ計算する.
- ◆ y の第 i 要素は, A の第 i 行とベクトル x との内積

$$y_i = \sum_j a_{i,j} x_j$$

逐次プログラム M-5 (mv_s.f90)

```
program mv
implicit none
integer, parameter :: n=100
integer :: i, j
real(kind=8), dimension(n,n) :: a
real(kind=8), dimension(n) :: x, y
real(kind=8) :: error, ans
do i = 1, n
  x(i) = i
end do
do i = 1, n
  do j = 1, n
    a(i,j) = dble(i+j)
  end do
end do
do i = 1, n
  y(i) = 0.0d0
  do j = 1, n
    y(i) = y(i) + a(i,j)*x(j)
  end do
end do
error = 0.0d0
do i = 1, n
  ans = dble( i*n*(n+1)/2 + n*(n+1)*(2*n+1)/6)
  error = error + abs( y(i) - ans )
end do
print *, 'error =', error
end program mv
```

x(i), a(i,j) の初期設定

y=Axの計算

結果の確認

abs() は絶対値関数

演習9-3 : M-5プログラムの実行

- mv_s.f90 をコンパイルして, 実行せよ

```
$ cp /tmp/cpmpi/M-5/mv_s.f90 ./
$ f95 -o mv_s mv_s.f90
$ ./mv_s
```

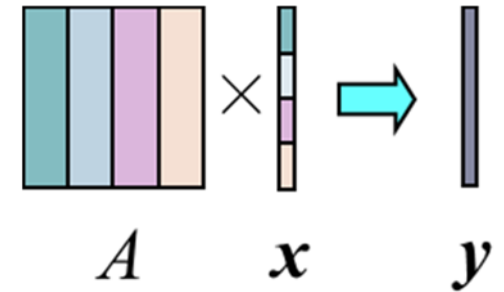
- 結果が正しいことを確認せよ.

```
$ ./mv_s
error = 0.000000000000000000
```

行列ベクトル積の並列化

■ 行列とベクトルの分割

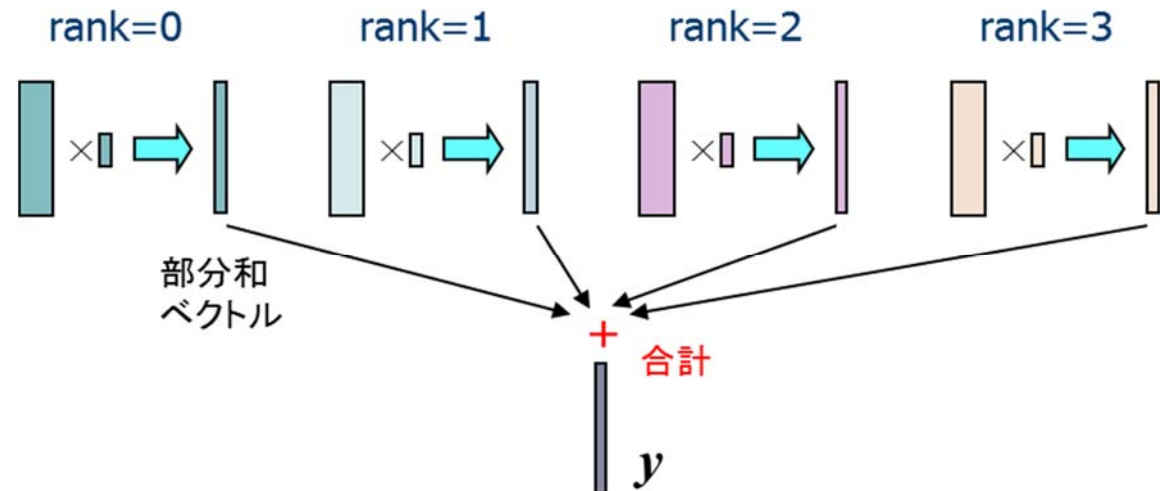
- ◆ A は列方向に短冊形に分割（ブロック列分割）する。
- ◆ x は行方向にブロック分割する。



■ プログラムの方針

- ◆ 各プロセスが持つ自分の要素のみを使って、部分和ベクトルを計算
- ◆ 部分和ベクトルを `mpi_reduce`関数によりプロセス0で集計する。

$$\begin{aligned} y_i &= \sum_{\text{rank}=0} a_{i,j} x_j + \sum_{\text{rank}=1} a_{i,j} x_j \\ &+ \sum_{\text{rank}=2} a_{i,j} x_j + \sum_{\text{rank}=3} a_{i,j} x_j \end{aligned}$$



演習9-4 : M-5 (mv_s.f90) を並列化せよ

■ プログラム書き換えの方針

◆ MPIの定義, 初期化, 終了処理を忘れないこと.

◆ 各プロセスの計算範囲を求める

- $istart = (n/nprocs) * myrank + 1$
- $iend = (n/nprocs) * (myrank + 1)$

◆ A , x について, 各プロセスが担当する部分のみ初期化する.

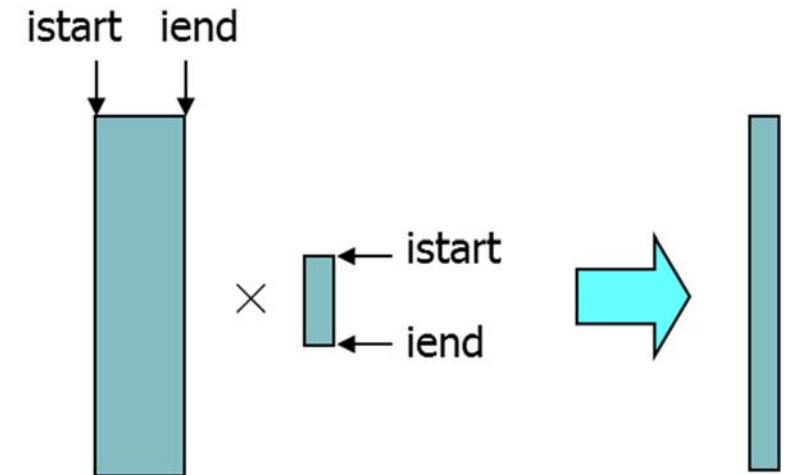
- A : 第 $istart$ 列 ~ 第 $iend$ 列
- x : 第 $istart$ 要素 ~ 第 $iend$ 要素

◆ 部分和ベクトルは, 各プロセスの持つ要素のみを使って計算

- 部分和ベクトルは, 別の配列 (例えば y_tmp) を用いる.

◆ 部分和ベクトルの合計

- `mpi_reduce` 関数により, ランク0のプロセスで, 配列 y_tmp の合計を配列 y に入れる.
- `mpi_reduce` 関数の第3引数 (`count`) に注意 (前回資料 29ページ)
- 結果は,



演習9-4：続き

- $n=1000$ として，プロセス数1，2，4，及び8と変化させて実行させ，結果が正しいことを確認せよ.
- そのときの計算時間の変化を調べよ.
 - ◆ 初期設定，結果の確認部分は，計測範囲に含めないこと.
 - ◆ プロセス数 (n) ， 計算時間 (T_n) ， 加速率 ($S_n=T_1/T_n$) をまとめる.

n	T_n	S_n
1	xxxxxxx	1.000
2	xxxxxxx	xxxxxxx
4	xxxxxxx	xxxxxxx
8	xxxxxxx	xxxxxxx

演習9-5（任意課題）

- プログラムM-4に対し，次の2つの方法で，全プロセスが総和を持つように修正し，8プロセスで実行したときの計算時間を比較せよ。
 - ① MPI_send() & MPI_recv() だけを用いる場合
 - ② MPI_allreduce() を用いる場合
- 総和がすべてのプロセスで等しいことを，出力して確認すること。
- 計測した計測時間を見て，考察せよ。

課題の提出方法と提出期限

- 演習9-2, 9-4 (必須), 9-5 (任意) の提出方法

① 課題ごとに修正したプログラムと実行結果を一つのファイルにまとめる.

```
$ cat program.f90 > report9-2.txt  
$ cat xxxxx.onnnnn >> report9-2.txt
```

② 以下の方法で, メールにより提出

```
$ nkf -Lu report9-2.txt | mail -s "9-2:アカウント" yokokawa@port.kobe-u.ac.jp
```

Note) アカウントは自分のログインID
番号 (9-2) は, 演習番号

※ プログラムがうまく動かない場合でも, 途中結果を提出せよ.

- 期限: 7月1日 (火) 午後5時

※ Wiki ページのアンケート (6/26) への協力をお願いします.