0.0 task 0: serial: write to screen

write the numbers 1 to 10 on the screen, one number per line.

```
 write(*,*) 1
 write(*,*) 2
 write(*,*) 3
```

0.1 task 1: serial: one loop

write the numbers 1 to 10 on the screen, one number per line. Use a do loop.

```
aend=10
do a=1,aend
  write(*,*) a
enddo
```

Graphically, we can picture a one-dimensional count like

a=1	a=2	a=3	a=4	a=5	a=6
1	2	3	4	5	6

0.2 task 2: serial: nested do loops

write the numbers 1 to 12 on the screen in sequential order, one number per line. Use two nested do loops.

```
jend=3 ! outer loop
iend=4 ! inner loop
do j=1,jend
  do i=1,iend
    write(*,*) i+iend*(j-1)
  enddo
enddo
```

	i=1	i=2	i=3	i=4
j=1	1	2	3	4
j=2	5	6	7	8
j=3	9	10	11	12

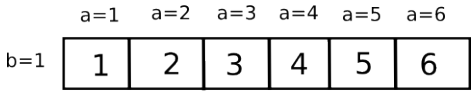
The inner loop i initially goes from 1 to 4 while $j = 1$. Then $j = 2$ and i goes from 5 to 8.

0.2.1 common problems

For some reason, the first response to "count 1 to 6" for 4 out of 4 students is

```
bend=1 ! outer loop
aend=6 ! inner loop
do b=1,bend
  do a=1,aend
    write(*,*) a
  enddo
enddo
```

which is graphically like



This "solution" does not follow the task 2 requirement "Use two nested do loops" since the outer loop is not actually being used.

Alternative non-solutions:

Use two loops, but specific to 1..10

```
do a=1,2
  do b=1,5
    if a==1 then
      number=b
    else
      number=b+5
    endif
    write(*,*) number
  enddo
enddo
```

Use a counter

```
z=1
do a=1,2
  do b=1,5
    write(*,*) z
    z=z+1
  enddo
enddo
```

This fulfills the requirements, but doesn't depend on a and b

0.3 task 3: serial: three nested loops

write the numbers 1 to N on the screen in sequential order, one number per line, where N is the product of three integers. Use a three nested do loops.

```
aend=5
bend=2
cend=3
do a=1,aend
  do b=1,bend
    do c=1,cend
      write(*,*) c+(b-1)*cend+(a-1)*bend*cend
    enddo
  enddo
enddo
```

0.4 task 4: serial: four nested loops

write the numbers 1 to N on the screen in sequential order, one number per line, where N is the product of four integers. Use a four nested do loops.

```
aend=5
bend=2
cend=3
dend=4
do a=1,aend
  do b=1,bend
    do c=1,cend
      do d=1,dend
        write(*,*) d+(c-1)*dend+(b-1)*cend*dend+(a-1)*bend*cend*dend
      enddo
    enddo
  enddo
enddo
```

By this point, students should be able to see the pattern for an arbitrary number of loops

0.5 task 5: MPI: hello from CPU

print the statement "hello from rank I of N CPUs" to the screen, where I is the rank of the CPU and N is the number of CPUs

What are the essentials of a Fortran 90 MPI program?

```
program do_nothing
  implicit none
  include 'mpif.h'
  integer :: ierr
  call MPI_INIT(ierr) ! initialized the MPI environment
  call MPI_Finalize(ierr)
end program do_nothing
```

However, this does not do anything.

We need to figure out how many CPUs there are, and which CPU the executable is running on.

```
program to_print_numbers
  implicit none
  include 'mpif.h'
  integer :: ierr, num_proc, my_rank
  call MPI_INIT(ierr) ! initialized the MPI environment
  call MPI_COMM_SIZE(MPI_COMM_WORLD, num_proc, ierr)  ! assign total number of CPUs to n
  call MPI_COMM_RANK(MPI_COMM_WORLD, my_rank, ierr)   ! which CPU is this (between 0 and
  write(*,*) "my rank is", my_rank, "of", num_proc, "CPUs"
  call MPI_Finalize(ierr)
end program to_print_numbers
```

The vital reason to do this is to understand that the write statement is being executed on each process independently. Between MPI_INIT and MPI_FINALIZE, there are N copies of the program running simultaneously.

0.5.1 common problems

Students are not clear on the difference between a compiler (ifort, gfortran, pgi) and an MPI library (mvapich, openmpi, mvapich2). The MPI library is required when `include 'mpif.h'` is used. The MPI library is a wrapper for a serial compiler.

0.6 task 6: MPI: serial loops

print a do loop from 1 to N on each CPU

```
program to_print_numbers
  implicit none
  include 'mpif.h'
  integer :: ierr, num_proc, my_rank
  integer :: jend, j
  call MPI_INIT(ierr) ! initialized the MPI environment
  call MPI_COMM_SIZE(MPI_COMM_WORLD, num_proc, ierr)  ! assign total number of CPUs to n
  call MPI_COMM_RANK(MPI_COMM_WORLD, my_rank, ierr)   ! which CPU is this (between 0 and
  jend=5
```

```
  do j=1,jend
    write(*,*) "j=",j,"on CPU",my_rank
  enddo
  call MPI_Finalize(ierr)
end program to_print_numbers
```

This demonstrates what happens when serial code is executed in parallel. No dependence on rank or number of CPUs yet.

0.7 task 7: MPI:

Parallelize the loop

```
program to_print_numbers
  implicit none
  include 'mpif.h'
  integer :: ierr, num_proc, my_rank
  integer :: j
  call MPI_INIT(ierr) ! initialized the MPI environment
  call MPI_COMM_SIZE(MPI_COMM_WORLD, num_proc, ierr)  ! assign total number of CPUs to n
  call MPI_COMM_RANK(MPI_COMM_WORLD, my_rank, ierr)   ! which CPU is this (between 0 and
  j=my_rank+1
  write(*,*) "j=",j,"on CPU",my_rank
  call MPI_Finalize(ierr)
end program to_print_numbers
```

Instead of iterating though the loop, just assign each CPU one number.

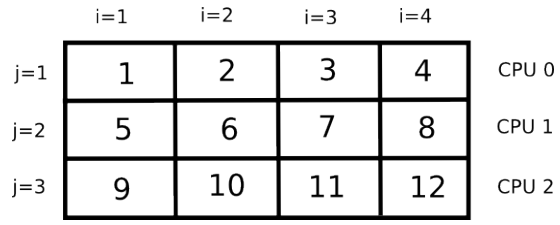
0.8 task 8: MPI: parallelize outer loop of two nested loops

Parallelize outer loop of two nested loops

```
program to_print_numbers
  implicit none
  include 'mpif.h'
  integer :: ierr, num_proc, my_rank
  integer :: j,i,iend ! jend is set by number of processors (mpirun -np 4 ./a.out)
  call MPI_INIT(ierr) ! initialized the MPI environment
  call MPI_COMM_SIZE(MPI_COMM_WORLD, num_proc, ierr)  ! assign total number of CPUs to n
  call MPI_COMM_RANK(MPI_COMM_WORLD, my_rank, ierr)   ! which CPU is this (between 0 and
  j=my_rank+1
  do i=1,iend
    write(*,*) i+iend*(j-1),"on CPU",my_rank
  enddo
```

```
  call MPI_Finalize(ierr)
end program to_print_numbers
```

Recall task 2, double nested loops. Here we have replaced the outer loop with the rank+1.



0.9 task 9: MPI: parallelize two nested loops

```
program to_print_numbers
  implicit none
  include 'mpif.h'
  integer :: ierr, num_proc, my_rank
  integer :: iend,jend,i,j
  call MPI_INIT(ierr) ! initialized the MPI environment
  call MPI_COMM_SIZE(MPI_COMM_WORLD, num_proc, ierr)  ! assign total number of CPUs to n
  call MPI_COMM_RANK(MPI_COMM_WORLD, my_rank, ierr)   ! which CPU is this (between 0 and
  j=
  i=
  write(*,*) i+iend*(j-1)
  call MPI_Finalize(ierr)
end program to_print_numbers
```

Recall task 2, double nested loops. When

$$\text{number} = i + i_{end} * (j - 1) = \text{rank} + 1 \tag{1}$$

where i is the inner loop index. Thus if rank is known, then we want to find i and j. This seems mathematically unsolvable (one known, two unknowns).

We want to assign $i = 1$, $j = 1$ to CPU 0. The next iteration, $i = 2$, $j = 1$, belongs on CPU 1.

CPU rank	inner loop i	outer loop j
0	1	1
1	2	1
2	3	1
$\vdots$	$\vdots$	$\vdots$
$i_{end} - 1$	i_{end}	1
i_{end}	1	2
$i_{end} + 1$	2	2
$\vdots$	$\vdots$	$\vdots$

Find the pattern: if rank is known, what is i and j?

0.10 task 10: MPI: parallelize two of three nested loops

```
program to_print_numbers
  implicit none
  include 'mpif.h'
  integer :: ierr, num_proc, my_rank
  integer :: iend,jend,i,j,k,kend
  call MPI_INIT(ierr) ! initialized the MPI environment
  call MPI_COMM_SIZE(MPI_COMM_WORLD, num_proc, ierr)  ! assign total number of CPUs to n
  call MPI_COMM_RANK(MPI_COMM_WORLD, my_rank, ierr)   ! which CPU is this (between 0 and
  kend=
  j=
  i=
  do k=1,kend
    write(*,*) i+iend*(j-1)
  enddo
  call MPI_Finalize(ierr)
end program to_print_numbers
```

0.11 task 11: MPI: parallelize three nested loops

```
program to_print_numbers
  implicit none
  include 'mpif.h'
  integer :: ierr, num_proc, my_rank
  integer :: i,j,k
  call MPI_INIT(ierr) ! initialized the MPI environment
  call MPI_COMM_SIZE(MPI_COMM_WORLD, num_proc, ierr)  ! assign total number of CPUs to n
  call MPI_COMM_RANK(MPI_COMM_WORLD, my_rank, ierr)   ! which CPU is this (between 0 and
  k=
  j=
  i=
  write(*,*) i+iend*(j-1)
  call MPI_Finalize(ierr)
end program to_print_numbers
```

0.12 task 12: MPI: parallelize three of five nested loops

```
program to_print_numbers
  implicit none
  include 'mpif.h'
  integer :: ierr, num_proc, my_rank
  integer :: i,j,k,l,m
  call MPI_INIT(ierr) ! initialized the MPI environment
```

```
  call MPI_COMM_SIZE(MPI_COMM_WORLD, num_proc, ierr)  ! assign total number of CPUs to n
  call MPI_COMM_RANK(MPI_COMM_WORLD, my_rank, ierr)   ! which CPU is this (between 0 and
  k=
  j=
  i=
  do l=
    do m=
      write(*,*) i+iend*(j-1)
    enddo
  enddo
  call MPI_Finalize(ierr)
end program to_print_numbers
```

0.13 task 13: MPI Groups: hello from group

0.14 task 14: MPI Groups: