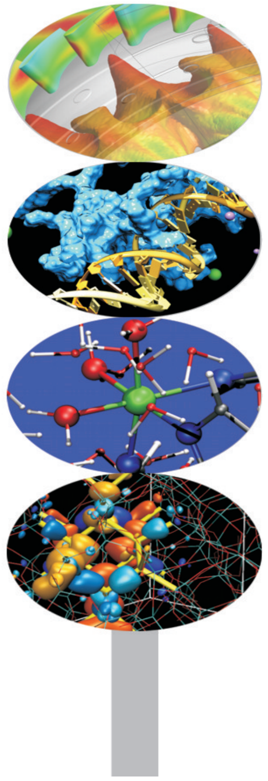
Exercitations on use of communicators to couple different modules in a single code



Giorgio Amati - CINECA
Giorgio Bolzon - OGS
Paolo Lazzari - OGS

Caveat

- ✓ All examples are written in Fortran, but C translation is straightforward
- ✓ In this lesson only a very little subset of MPI function are used: take a look to MPI manual for a complete description
- ✓ This problem can be "solved" in different ways, we present the one we found more simple (according to us)
- ✓ Remember that the MPI library is a very big one:
  - ✓ Still evolving: now we are at 3.0
  - ✓ About 400 different MPI functions (https://www.open-mpi.org/doc/v1.8/)
  - ✓ The Standard accounts for more than 600 pages, (ver. 2.2 http://www.mpi-forum.org/docs/mpi-2.2/mpi22-report.pdf)

Model coupling

- Important issue in Earth Science
  - Different complex models (Atmospheric, Ocean, …) developed by many researchers in many years
  - years of testing/validation
  - Almost impossible to rewrite them from scratch according to your problem

- How couple them in an efficient way?
  - Hi-level approach:
    - http://www.earthsystemmodeling.org/
    - http://www.messy-interface.org/
    - https://verc.enes.org/oasis
  - Low-level approach: using some MPI feature

MPMD

- What we need
  1. To define new communicators to allow different pools of workers
  2. To define a suitable synchronization between the different pools of workers

- Step to address:
  1. Build new communicators
  2. Include different programs (serial)
  3. Include different programs (parallel)
  4. Sync between different communicators (overlapping)
  5. Sync between different communicators (no overlapping)

MPI Group

- Group definitions:
  - ✓ A group is an ordered set of processes. Each process in a group is associated with a unique integer rank. Rank values start at zero and go to N-1, where N is the number of processes in the group.
  - ✓ One process can belong to two or more groups.
  - ✓ A group is used within a communicator to describe the participants in a communication "universe" and to rank such participants.
  - ✓ Group is a dynamic object in MPI and can be created and destroyed during program execution.

MPI Communicator

- Communicator definitions:
  - ✓ The communicator determines the scope and the "communication universe" in which a point-to-point or collective operation is to operate.
  - ✓ Each communicator contains a group of valid participants. The source and destination of a message is identified by process rank within that group.
  - ✓ Communicators are dynamic, i.e., they can be created and destroyed during program execution.

MPI_COMM_GROUP

This function accesses to the group associated with given communicator

- C:

```
MPI_Comm_group(MPI_Comm, group)
```

- Fortran:

```
MPI_Comm_group(MPI_Comm, group, ierr)
```

✓ **(IN) MPI_comm:** existing MPI communicator

✓ **(OUT) group:** group of processes of **MPI_comm**

MPI_GROUP_INCL

This function produces a group by reordering an existing group and taking only listed members:

- C:

```
MPI_GROUP_INCL(base_grp,n,list,new_grp)
```

- Fortran:

```
MPI_GROUP_INCL(base_grp,n,list,new_grp,ierr)
```

✓ `(IN)` **`base_grp`**`:` existing group

✓ `(IN)` **`n`**`:` elements of the array list

✓ `(IN)` **`list`**`:` rank of processes in **`base_grp`** to belong to **`new_group`**

✓ `(OUT)` **`new_grp`**`:` new group of processes

MPI_COMM_CREATE

This function create a new mpi communicator from a group of processes:

- C:

```
MPI_COMM_CREATE(MPI_COMM,new_grp,NEW_COMM)
```

- Fortran:

```
MPI_COMM_CREATE(MPI_COMM,new_grp,NEW_COMM,ierr)
```

✓ `(IN)` **`MPI_COMM`**: old communicator

✓ `(IN)` **`new_grp`**: new group (a subset of **`MPI_COMM`**)

✓ `(OUT)` **`NEW_COMM`**: new communicator with processor defined in **`new_grp`**

EXERCISES

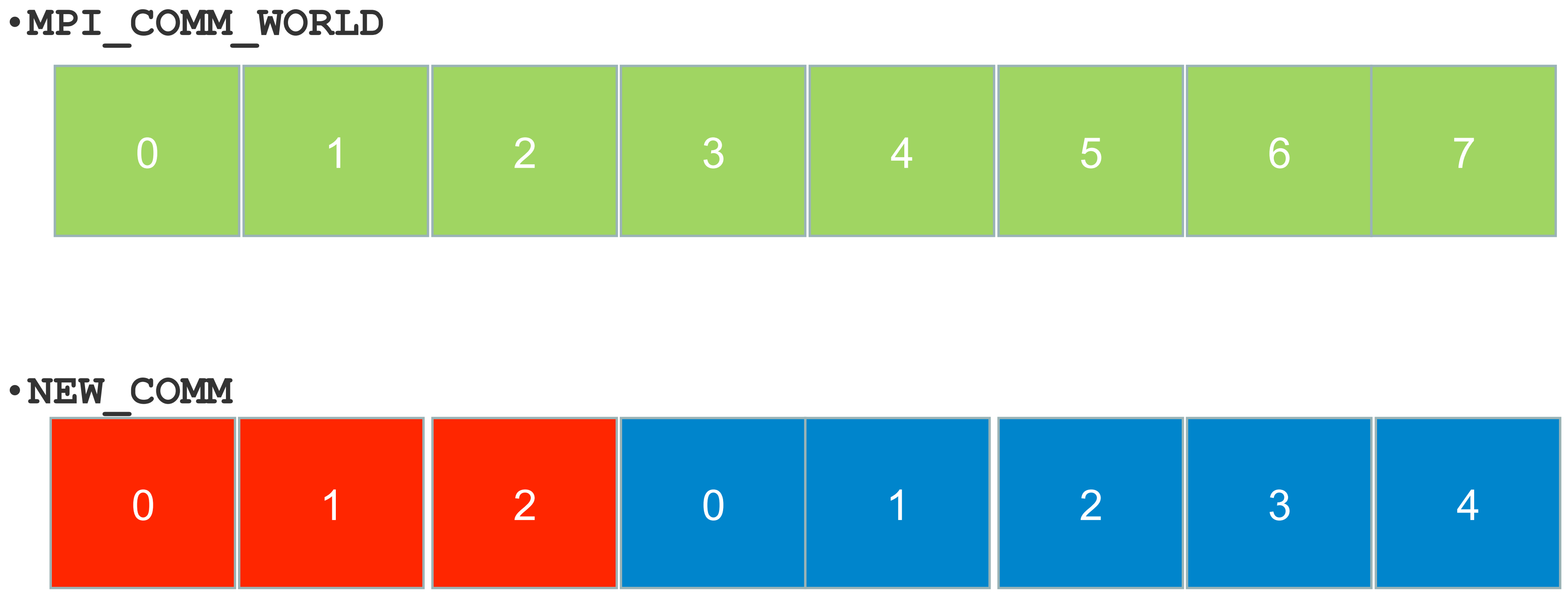
Directory structure:

```
LAB_SESSION
|-- EXE_1:how to create a new communicator
|-- EXE_2:embedding two (serial) programs
|-- EXE_3:embedding two (parallel) programs
|-- EXE_4:"simple" synchronization
`-- EXE_5:"complex" synchronization
```

EXE_1

Let's divide **nprocs** mpi task in two pools:

- Red: from task **0** to task **nproc-5**
- Blue: from task **nproc-5** to **nproc**



EXE_1/two_work_mpi.F90

```
!mpi stuff
    call MPI_INIT(ierr)
    call MPI_COMM_SIZE(MPI_COMM_WORLD,nproc,ierr)
    call MPI_COMM_RANK(MPI_COMM_WORLD,myrank,ierr)
    call MPI_COMM_GROUP(MPI_COMM_WORLD,base_grp,ierr)
…
create new group
    if (myrank.lt.nRed) then
       call MPI_GROUP_INCL(base_grp,nRed,red_list,new_grp,ierr)
       imred=.True.
    else
       call MPI_GROUP_INCL(base_grp,nBlue,blue_list,new_grp,ierr)
       imblue=.True.
    endif

    call MPI_COMM_CREATE(MPI_COMM_WORLD,new_grp,MPI_NEW_COMM,ierr)
    call MPI_COMM_RANK(MPI_NEW_COMM,new_rank,ierr)
```

`EXE_1`: to do

Write the correct `red_list` and `blu_list` so that:

- ✓The first `nproc-5` are red
- ✓The last `5 tasks` are blue

Compile and check the result varying the number of MPI task.

EXE_1: (a) solution

```
! some setting
    nBlue       = 5                    ! number of blue tasks
    nRed        = nproc - nBlue        ! number of red tasks
!
    do i = 1, nRed
       red_list(i) = i-1
    enddo
!
    do i = 1, nBlue
       blue_list(i) = (nRed -1) + i
    enddo
!
```

EXE_1: right output

```
---------------------------------------
Red vs. Blue           3            5
---------------------------------------
Red  --> I'm task 1 in COMM_WORLD and in 1 NEW_COMM…
Blue --> I'm task 5 in COMM_WORLD and in 2 NEW_COMM…
Blue --> I'm task 6 in COMM_WORLD and in 3 NEW_COMM…
Red  --> I'm task 2 in COMM_WORLD and in 2 NEW_COMM…
Blue --> I'm task 3 in COMM_WORLD and in 0 NEW_COMM…
Blue --> I'm task 4 in COMM_WORLD and in 1 NEW_COMM…
Blue --> I'm task 7 in COMM_WORLD and in 4 NEW_COMM…
Red  --> I'm task 0 in COMM_WORLD and in 0 NEW_COMM…
```

Homework – I

If you have enough time (☺) you can:

1. Build a new communicator in which
   - ✓ All processes in the red pool have even rank (according to **`MPI_COMMON_WORLD`**)
   - ✓ All processes in the blue pool have odd rank (according to **`MPI_COMMON_WORLD`**)
2. Build a communicator with three pools of processors
   - ✓ Red, Blue & Green

EXE_2: embedding (serial) programs

- With this communicator we can, starting from a 2 task simulation
  - ✓ Give to red process program A
  - ✓ Give to blue process program B
  - ✓ No communication and/or synchronization
  - ✓ Typical problems:
    - ✓ Allocation/deallocation
    - ✓ Deadlocks
    - ✓ Load balancing
    - ✓ I/O

`EXE_2`:to do

- Just a simple exercise:
  - ✓ Program A: Pi-computation (blue pool)
    - using the integral pi = 4 ∑ 1/(1+x*x) x in [0-1]
  - ✓ Program B: matrix-matrix computation (red pool)

- Original code A & B must be modified
  - ✓ Take care of data allocation/de-allocation
  - ✓ Pass to program A&B the new communicator
  - ✓ Pass to program A&B the right values (if any)
  - ✓ Only 2 task: one for Program A and one for Program B

EXE_2: pi.F90

✓ http://www.hpc.cineca.it/content/pi-fortran-openmp

```
program pigreco
    implicit none
    integer(selected_int_kind(18)) :: i
    integer(selected_int_kind(18)), parameter :: intervals=1e7
    integer:: nthreads, threadid
    real(kind(1.d0)) :: dx,sum,x
    real(kind(1.d0)) :: f,pi
    real(kind(1.d0)), parameter :: PI25DT = acos(-1.d0)
    real :: time1, time2
    write(*,*) "Serial version "
    sum=0.d0
    dx=1.d0/intervals
    do i=1, intervals
        x=dx*(i-0.5d0)
        f=4.d0/(1.d0+x*x)
        sum=sum+f
    end do
    pi=dx*sum
    …
```

EXE_2: mm.F90

✓ http://www.hpc.cineca.it/content/mm-fortran-openmp

```
...
   implicit none
   integer :: n
   real*8, dimension(:,:), allocatable   :: a, b, c
…
   allocate(a(n,n),b(n,n),c(n,n),stat=ierr)
…
   call random_number(a)
   call random_number(b)
   c = 0.d0
   call cpu_time(time1)
   do j=1, n
      do k=1, n
         do i=1, n
            c(i,j) = c(i,j) + a(i,k)*b(k,j)
         end do
      end do
   end do
   call cpu_time(time2)
…
```

EXE_2: two_work_mpi.F90

```
…
!mpi stuff  (the same as example 1)
…
!create new group (Only two task!!!!!)
…
!create new communicator & rank (the same as example I)
…
! Do something…
    if(imred) then
        write(6,*) “Red  → I’m task in COMMON WORLD….
        call do_red(MPI_NEW_COMM,myrank,new_rank,nproc,nRed,nBlue,nsize)
    endif

    if(imblue) then
        write(6,*) “Blue → I’m task in COMMON WORLD….
        call do_blue(MPI_NEW_COMM,myrank,new_rank,nproc,nRed,nBlue)
    endif
```

EXE_2: do_red

```
subroutine do_red(MPI_NEW_COMM,myrank,new_rank,nproc,nred,nblue,n)

   integer :: nproc, myrank, n
   integer :: MPI_NEW_COMM,new_rank,nred,nblue
…
   deallocate(a,b,c)

end subroutine do_red
```

- Removed the reading from the standard input of the size of the matrix
- Deallocate vectors
- Modified output
  - `write(*,*) "Red` → ….

EXE_2: do_blue

```
subroutine do_blue(MPI_NEW_COMM,myrank,new_rank,nproc,nred,nblue)
…
   integer :: nproc, myrank
   integer :: MPI_NEW_COMM,new_rank,nred,nblue
…
end subroutine do_blue
```

- Modified output
  - `write(*,*) "Blue →  ….`

EXE_2: output

```
…
BLUE: Serial version
BLUE: Number of intervals:              10000000
BLUE:  Computed PI =     3.14159265358943606827324404
BLUE:  The True PI =     3.14159265358979311599796356
BLUE:  Error             0.00000000000035704772471956
BLUE: Elapsed time   0.259961009      s
BLUE: all done
RED: Elapsed time    1.4567790000000000       s
RED: Gflops         0.18426642339023283
RED: all done....
…
```

Homework - II

If you have enough time (☺) you can:

1. Parallelize with OMP only program B (matrix product)
2. Parallelize with OMP only program A (compute pi)
3. Parallelize with OMP programs A+B

Step 3: embedding (parallel) programs

- Just a simple example
  - ✓ Program A: Pi-computation (blue pool, MPI)
  - ✓ Program B: matrix-matrix computation (red pool, MPI)

- Again, no synchronization
  - ✓ Take care of data allocation/de-allocation
  - ✓ Pass to program A&B the new communicator
  - ✓ Pass to program A&B the right values (if any)
  - ✓ I/O

EXE_3: pi_mpi.F90

- http://www.hpc.cineca.it/content/pi-fortran-mpi

```
program pigreco
    use mpi
    implicit none
…
!mpi stuff
    call MPI_INIT(ierr)
    call MPI_COMM_SIZE(MPI_COMM_WORLD,nproc,ierr)
    call MPI_COMM_RANK(MPI_COMM_WORLD,myrank,ierr)

    if(myrank == 0) then
       write(*,*) "MPI version with tastks = ", nproc
       write(*,*) "Number of intervals     = ", intervals
    endif

    if(mod(intervals,nproc) /= 0) then
       if(myrank == 0) then
          write(*,*) "The number of process must divide", intervals, "exactly."
       endif
       call MPI_BARRIER(MPI_COMM_WORLD,ierr)
       call MPI_FINALIZE(ierr)
       stop
    endif
…
```

EXE_3: pi_mpi.F90 (II)

- http://www.hpc.cineca.it/content/pi-fortran-mpi

```
…
      sum=0.d0
     dx=1.d0/intervals
     time1 = MPI_WTIME()
     istart = (intervals/nproc)*myrank + 1
     iend   = (intervals/nproc)*(myrank+1)
     sum = 0.0;
     total_sum = 0.0;
     do i=iend, istart, -1
         x=dx*(i-0.5d0)
         f=4.d0/(1.d0+x*x)
         sum=sum+f
     end do
     call MPI_Reduce(sum,total_sum,1,MPI_DOUBLE,MPI_SUM,0,MPI_COMM_WORLD,ierr)
     pi=dx*total_sum
     call MPI_BARRIER(MPI_COMM_WORLD,ierr)
     if(myrank == 0) then
        PRINT '(a13,2x,f30.25)',' Computed PI =', pi
        PRINT '(a13,2x,f30.25)',' The True PI =', PI25DT
        PRINT '(a13,2x,f30.25)',' Error        ', PI25DT-pi
     endif
     call MPI_FINALIZE(ierr)
end program
…
```

EXE_3: mm_mpi.F90

- http://www.hpc.cineca.it/content/mm-fortran-mpi

```
program matrix_matrix_prod
   use mpi
   implicit none
…
!mpi stuff
   call MPI_INIT(ierr)
   call MPI_COMM_SIZE(MPI_COMM_WORLD,nprocs,ierr)
   call MPI_COMM_RANK(MPI_COMM_WORLD,myrank,ierr)
!reading matrix size
…
   call MPI_BCAST(n,1,MPI_INTEGER,0,MPI_COMM_WORLD,ierr)
   call MPI_BARRIER(MPI_COMM_WORLD,ierr)
!a check
   if(nprocs /= 2) then
     if(myrank == 0) then
       write(*,*) "Error, nprocs=", nprocs, "is not 2!!!!"
       write(*,*) "This (stupid) code works only with 2 task!!!"
     endif
     call MPI_BARRIER(MPI_COMM_WORLD,ierr)
     call MPI_FINALIZE(ierr)
endif
```

SuperComputing Applications and Innovation

EXE_3: mm_mpi.F90 (II)

- http://www.hpc.cineca.it/content/mm-fortran-mpi

```
!allocation/inizializations
…
   if(myrank == 0) then
     call random_number(a)
     call random_number(b)
   endif
!
!sending a and b elements
   if(myrank.eq.1) then
     call mpi_recv(a(1,1), n*n, MPI_DOUBLE,0,1,MPI_COMM_WORLD, status,ierr)
     call mpi_recv(b(1,1), n*n, MPI_DOUBLE,0,2,MPI_COMM_WORLD, status,ierr)
   endif
!
   if(myrank.eq.0) then
     call mpi_send(a(1,1), n*n, MPI_DOUBLE,1,1,MPI_COMM_WORLD, ierr)
     call mpi_send(b(1,1), n*n, MPI_DOUBLE,1,2,MPI_COMM_WORLD, ierr)
   endif
   call mpi_barrier(MPI_COMM_WORLD,ierr)
!
```

SuperComputing Applications and Innovation

EXE_3: mm_mpi.F90 (III)

- http://www.hpc.cineca.it/content/mm-fortran-mpi

```
    if (myrank == 1) then
      jstart = n/2+1
      jend = n
   endif
!
   do j=jstart, jend
      do k=1, n
         do i=1, n
            c(i,j) = c(i,j) + a(i,k)*b(k,j)
         end do
      end do
   end do
!collecting elements of c
   if(myrank == 0) then
     call mpi_recv(c(1,n/2+1), n*n/2, MPI_DOUBLE,1,4,MPI_COMM_WORLD, status,ierr)
   endif
   if(myrank == 1) then
     call mpi_send(c(1,n/2+1), n*n/2, MPI_DOUBLE,0,4,MPI_COMM_WORLD, ierr)
   endif
```

SuperComputing Applications and Innovation

EXE_3: two_work_mpi.F90

```
…
!mpi stuff  (the same as example II)
…
!create new group (similar to example II, nRed must be equal to 2)
…
!create new communicator & rank (the same as example II)
…
! do something...
    if(imred) then
        write(6,*) "Red  --> I'm task in COMMON WORLD….
        call do_red(MPI_NEW_COMM,myrank,new_rank,nproc,nRed,nBlue,nsize)
    endif

    if(imblue) then
        write(6,*) "Blue --> I'm task in COMMON WORLD….
        call do_blue(MPI_NEW_COMM,myrank,new_rank,nproc,nRed,nBlue)
    endif
```

EXE_3: do_red

```
subroutine do_red(MPI_NEW_COMM,myrank,new_rank,nproc,nred,nblue,n)
   integer :: nproc, myrank
   integer :: MPI_NEW_COMM,new_rank,nred,nblue
…
end subroutine do_red
```

- Removed the reading from the standard input of the size of the matrix
- Removed the **MPI_Initialize** & **MPI_Finalize**
- Changed **MPI_COMM_WORLD → MPI_NEW_COMM**
- Changed **Myrank → new_rank**
- Changed **Nproc → nred**
- Modified output
  - **write(*,*) "Red →** ….

EXE_3: do_blue

```
subroutine do_blue(MPI_NEW_COMM,myrank,new_rank,nproc,nred,nblue)
   integer :: nproc, myrank
   integer :: MPI_NEW_COMM,new_rank,nred,nblue
…

end subroutine do_blue
```

- Removed the **`MPI_Initialize`** & **`MPI_Finalize`**
- Changed **`MPI_COMM_WORLD`** → **`MPI_NEW_COMM`**
- Changed **`Myrank`** → **`new_rank`**
- Changed **`Nproc`** → **`nred`**
- Modified output
  - **`write(*,*) "Blue → ….`**

Exercise 3: output

```
---------------------------------------
Red vs. Blue           2           5
---------------------------------------
Red: Matrix-Matrix MPI version with task =            2
Red: Matrix size is            10
Blue: Computing PI: MPI version with tasks =            5
Red: Error on a random element:  0.000000000000000E+000
Red: Elapsed time   4.315376281738281E-005  s
Red: Tot Gflops     4.634590055248619E-002
Red: All done...
Blue: Number of intervals     =              10000000
Blue:  Computed PI     3.1415926535898051064066294
Blue:  The True PI     3.1415926535897931159979635
Blue:  Error          -0.0000000000000119904086660
Blue: Elapsed time   1.434707641601562E-002  s
Blue: all done....
Blue: all done....
Blue: all done....
Blue: all done....
Blue: all done....
```

Homework - III

If you have enough time (☺) you can:

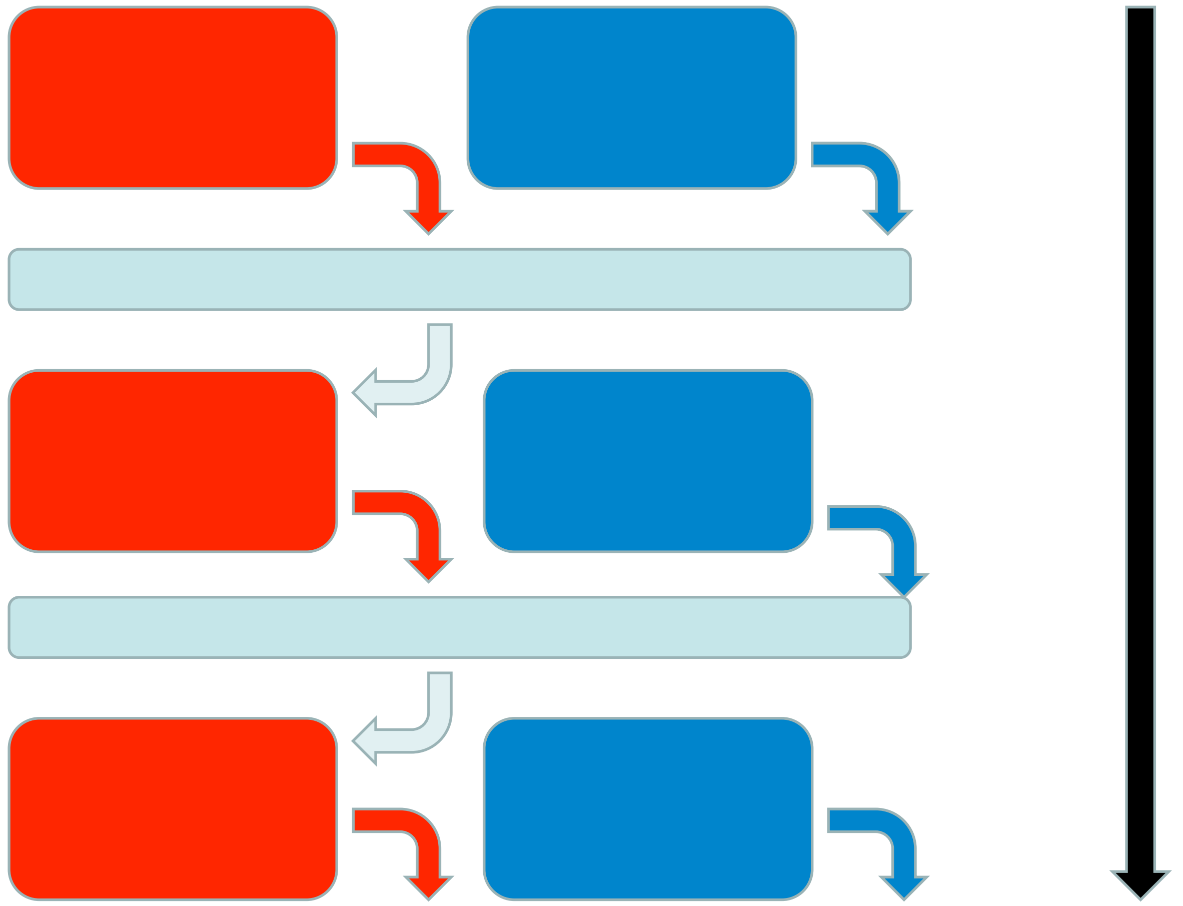
1. Try to make a hybrid parallelization (MPI-OpenMP) only for program B (matrix-products)
2. Try to make an hybrid parallelization (MPI-OpenMP) only program A (compute pi)
3. Hybrid parallelization (MPI-OpenMP) of both programs

`EXE_4`: “simple” synchronization

- Just a simple synchronization between programs
  - Compare the time for program A and program B
  - If time to complete B (matrix-matrix multiplication) is bigger than the time to complete A then reduce matrix’s size of 5 elements, else increase by 5
- Synchronization between the two pools
  - ✓ Pass between different pools the time
  - ✓ Iterate 200 times…..
  - ✓ If possible an output for `gnuplot` can be fine!

EXE_4

- "Simple" synchronization



EXE_4: two_work_mpi.F90

Modifications to do respect **EXE_3**

- ✓ Loop for iteration
- ✓ Synchronization (blocking send/recv)
- ✓ Check of different time (from red & blue pool)
- ✓ Propagation of information between red pool

EXE_4: do_blue

```
subroutine do_blue(MPI_NEW_COMM,myrank,new_rank,nproc,nred,nblue,Tblue)
…
    Real*8 :: Tblue
…
end subroutine do_blue
```

- The same as exercise 3, unless:
  - Define and pass as argument the time needed to complete the operation (e.g. **Tblue**)

EXE_4: do_red

```
subroutine do_red(MPI_NEW_COMM,myrank,new_rank,nproc,nred,nblue,n,Tred)
…
    real*8 :: Tred
…
end subroutine do_red
```

- The same as exercise 3, unless:
  - Define and pass as argument the time needed to complete the operation (e.g. `Tred`)

EXE_4: two_work_mpi.F90

```
! iterate!!!
do k = 1, kmax
   if(imred) then
      call do_red(MPI_NEW_COMM,myrank,new_rank,nproc,nRed,nBlue,nsize,Tred)
      if(new_rank == 0) then
         call mpi_send(Tred,1,MPI_REAL8,first_blue,1,MPI_COMM_WORLD,ierr)
         call mpi_recv(Tblue,1,MPI_REAL8,first_blue,2,MPI_COMM_WORLD,status,ierr)
      endif
      call MPI_BARRIER(MPI_NEW_COMM,ierr)
   endif

   if(imblue) then
      call do_blue(MPI_NEW_COMM,myrank,new_rank,nproc,nRed,nBlue,Tblue)
      if(new_rank == 0) then
         call mpi_send(Tblue,1,MPI_REAL8,first_red,2,MPI_COMM_WORLD,ierr)
         call mpi_recv(Tred,1,MPI_REAL8,first_red,1,MPI_COMM_WORLD,status,ierr)
      endif
      call MPI_BARRIER(MPI_NEW_COMM,ierr)
   endif
….
```

SuperComputing Applications and Innovation

EXE_4: two_work_mpi.F90

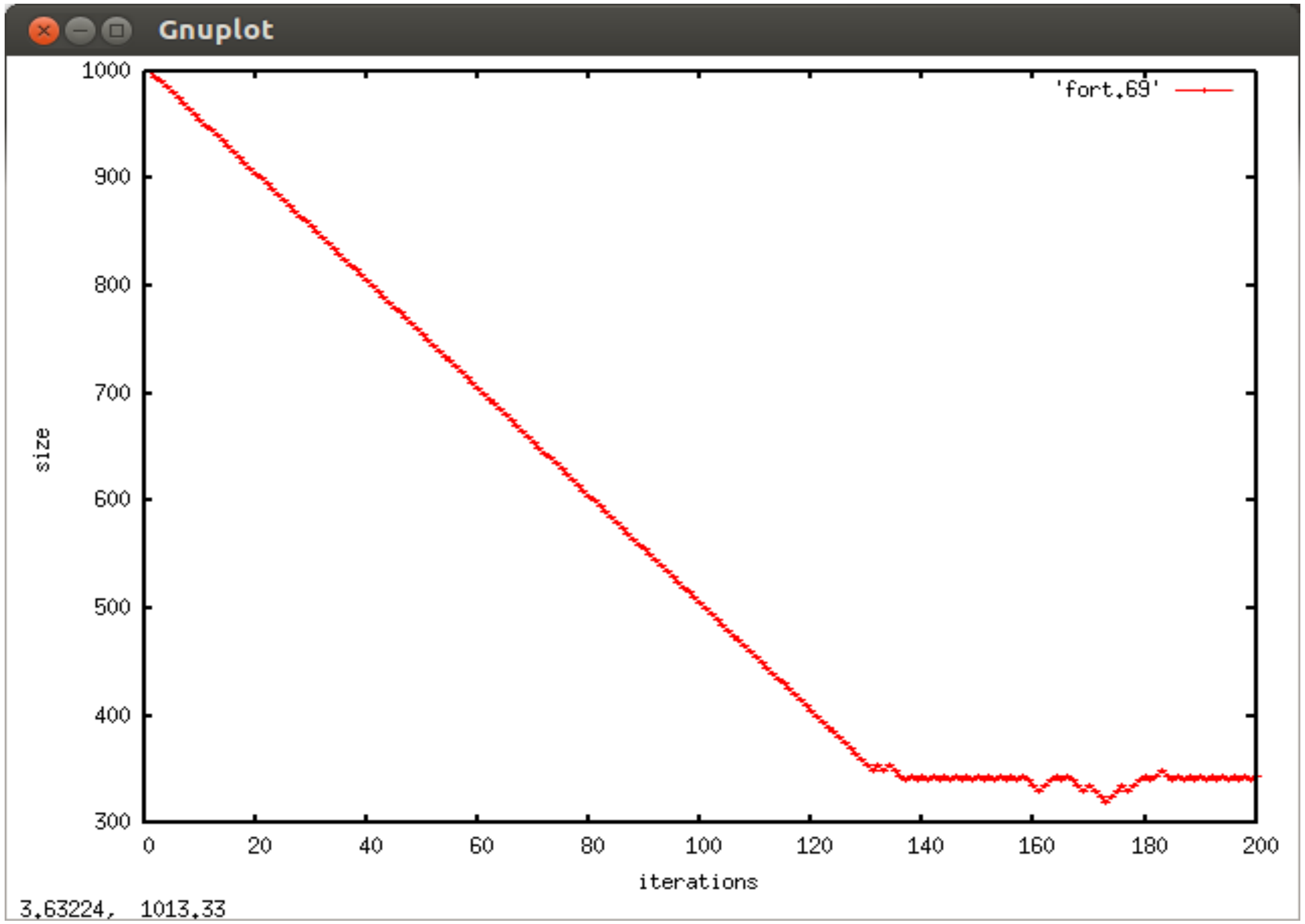
….

```
   if(myrank == 0) then
      write(*,*) "All ---> ", k, nsize, Tred, Tblue
      write(69,*) k, nsize, Tred, Tblue
      if(Tred > Tblue) then
         nsize = nsize - 5
      else
         nsize = nsize + 5
      endif
   endif
   call MPI_BCAST(nsize,1,MPI_INTEGER,0,MPI_COMM_WORLD,ierr)
   call MPI_BARRIER(MPI_COMM_WORLD,ierr)
enddo
```

EXE_4: output

```
......
All --->    33          840  0.367211103439331     1.43010163024902E-002
All --->    34          835  0.381738901138306     1.43909454345703lE-002
All --->    35          830  0.356688976287842     1.44506645202637E-002
All --->    36          825  0.364209890365601     1.50289535522460 9E-002
All --->    37          820  0.335249900817871     1.65269374847412lE-002
All --->    38          815  0.349348068237305     1.41291618347168 0E-002
All --->    39          810  0.338562965393066     1.39789581298828lE-002
All --->    40          805  0.319899082183838     1.20420455932617 2E-002 All
--->    41          800  0.299830913543701     1.35338306427002 0E-002
All --->    42          795  0.318428993225098     1.44219398498535 2E-002
All --->    43          790  0.286552906036377     1.35309696197509 8E-002
All --->    44          785  0.294275999069214     1.21181011199951 2E-002
All --->    45          780  0.260970115661621     1.43480300903320 3E-002
All --->    46          775  0.266450166702271     1.31750106811523 4E-002
All --->    47          770  0.231342077255249     1.43220424652099 6E-002
All --->    48          765  0.242427110671997     1.66788101196289lE-002
…
```

EXE_4: output

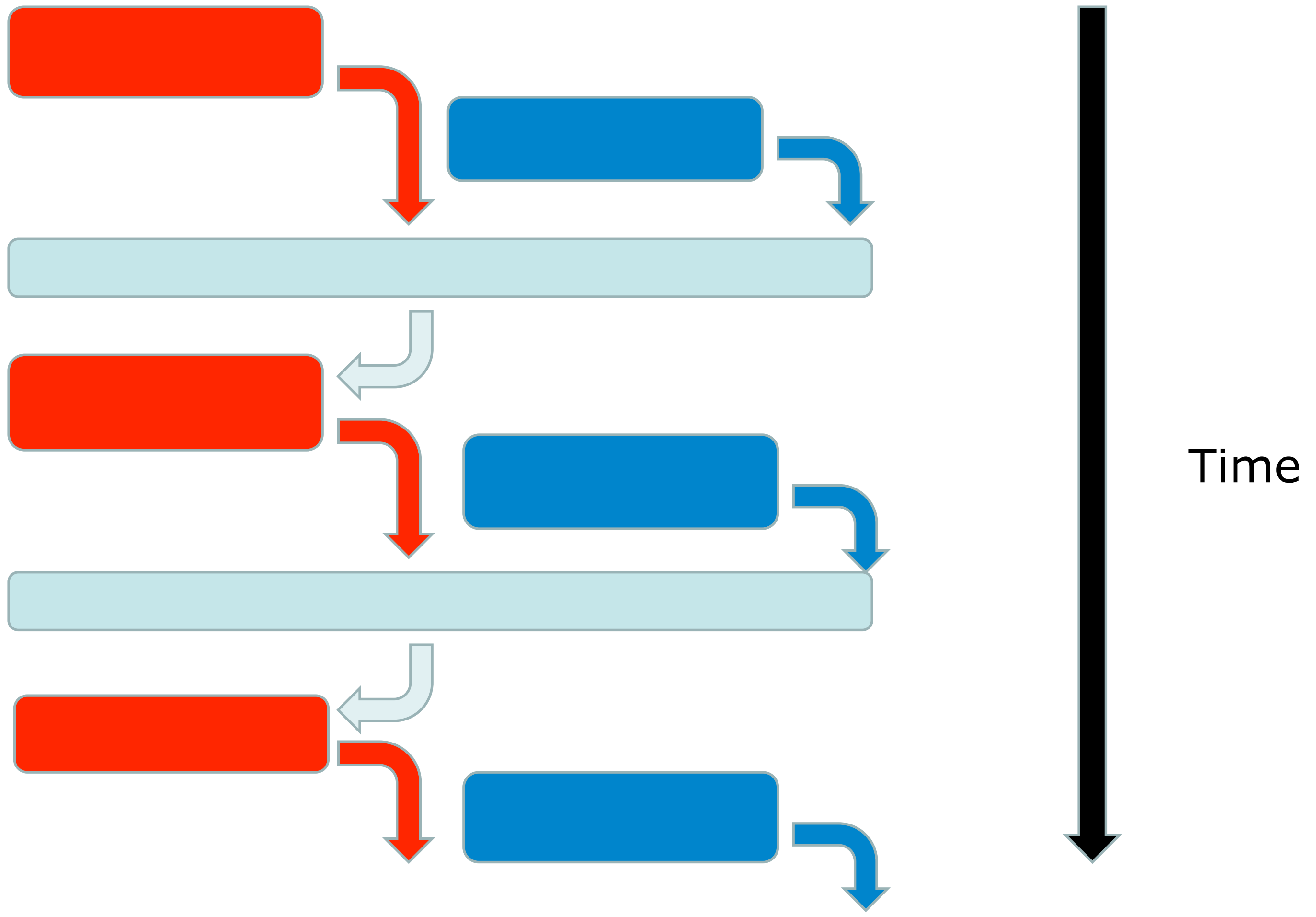

Gnuplot
'fort.69'
size
iterations
1000
900
800
700
600
500
400
300
0
20
40
60
80
100
120
140
160
180
200
3.63224, 1013.33

`EXE_5`: “complex” synchronization

- An other synchronization between programs
  - Compare the time for program A and program B
  - If time to complete B (matrix-matrix multiplication) is bigger than the time to complete A then reduce matrix’s size of 5 elements, else increase by 5
  - First run program A, then Program B
- Synchronization between the two pools
  - ✓ Pass between different pools the time
  - ✓ Iterate 200 times…..
  - ✓ If possible an output for `gnuplot` can be fine!

EXE_5

- “complex” synchronization



SuperComputing Applications and Innovation

EXE_5: two_work_mpi.F90

```fortran
! iterate!!!
do k = 1, kmax
   if(imred) then
      call do_red(MPI_NEW_COMM,myrank,new_rank,nproc,nRed,nBlue,nsize,Tred)
      if(new_rank == 0) then
         call mpi_send(Tred,1,MPI_REAL8,first_blue,1,MPI_COMM_WORLD,ierr)
         call mpi_recv(Tblue,1,MPI_REAL8,first_blue,2,MPI_COMM_WORLD,status,ierr)
      endif
      call MPI_BARRIER(MPI_NEW_COMM,ierr)
   endif

   if(imblue) then
      if(new_rank == 0) then
         call mpi_recv(Tred,1,MPI_REAL8,first_red,1,MPI_COMM_WORLD,status,ierr)
      endif
      call MPI_BARRIER(MPI_NEW_COMM,ierr)
      call do_blue(MPI_NEW_COMM,myrank,new_rank,nproc,nRed,nBlue,Tblue)
      if(new_rank == 0) then
         call mpi_send(Tblue,1,MPI_REAL8,first_red,2,MPI_COMM_WORLD,ierr)
      endif
      call MPI_BARRIER(MPI_NEW_COMM,ierr)
   endif
```

EXE_5: two_work_mpi.F90

….

```
   if((new_rank == 0).AND.(imred)) then
      write(*,*) "All ---> ", k, nsize, Tred, Tblue
      write(69,*) k, nsize, Tred, Tblue
      if(Tred > Tblue) then
         nsize = nsize - 5
      else
         nsize = nsize + 5
      endif
      call MPI_BCAST(nsize,1,MPI_INTEGER,0,MPI_NEW_COMM,ierr)
      call MPI_BARRIER(MPI_NEW_COMM,ierr)
   endif
   call MPI_BARRIER(MPI_COMM_WORLD,ierr)
enddo
```

EXE_5: output

```
......
All --->    187       335  1.21691226959228E-002 1.59778594970703E-002
All --->    188       330  1.47740840911865E-002 1.19991302490234E-002
All --->    189       325  1.21450424194335E-002 1.17712020874023E-002
All --->    190       320  1.34289264678955E-002 1.18620395660400E-002
All --->    191       325  1.13210678100585E-002 1.16548538208007E-002
All --->    192       320  1.34179592132568E-002 1.18658542633056E-002
All --->    193       325  1.13639831542968E-002 1.19640827178955E-002
All --->    194       320  1.33950710296630E-002 1.16448402404785E-002
All --->    195       325  1.13859176635742E-002 1.16131305694580E-002
All --->    196       320  1.33609771728515E-002 1.16820335388183E-002
All --->    197       325  1.13849639892578E-002 1.24800205230712E-002
All --->    198       330  1.34670734405517E-002 1.35459899902343E-002
All --->    199       335  1.21378898620605E-002 1.24409198760986E-002
All --->    200       330  1.47740840911865E-002 1.23291015625000E-002
...
```

Any question?

- Contact:
  - g.amati@cineca.it
  - gbolzon@ogs.trieste.it
  - plazzari@ogs.trieste.it