GWDG – Kurs
Parallel Programming with MPI

Collective Operations
Exercises

Oswald Haan

ohaan@gwdg.de

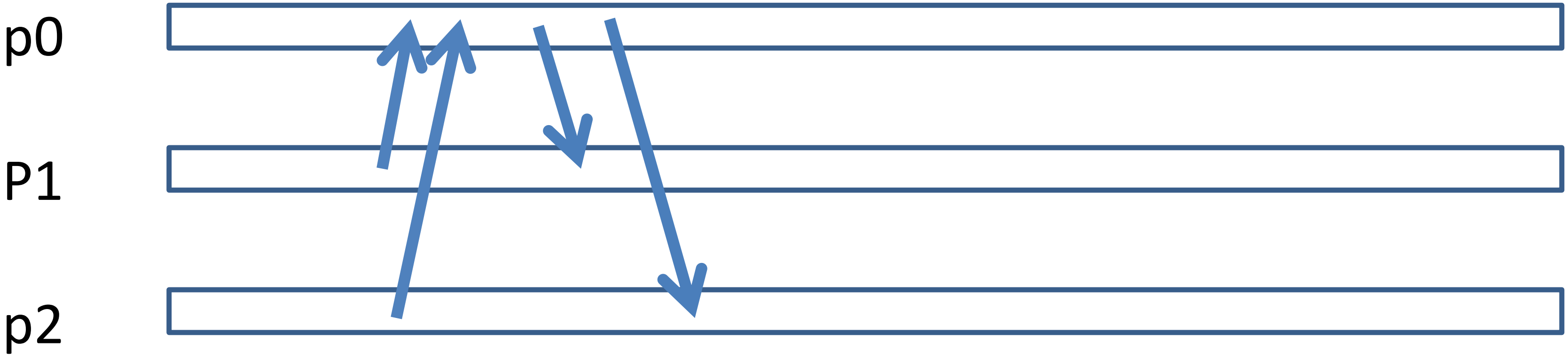
Exercise 1: Synchronization

(Source code in directory `Uebungen_*/MPI-coll`)

```
call MPI_BARRIER(comm,ierr)
MPI_Barrier(comm)
Comm.Barrier
```

Determine the time needed for synchronization for different number of processes (use **synch.f** **(make synch)**, **synch.py**)

Program your own barrier using point-to-point communication (complete the program **synch_s.f** | **synch_s.c** | **synch_s.py**):



Solution for Exercises

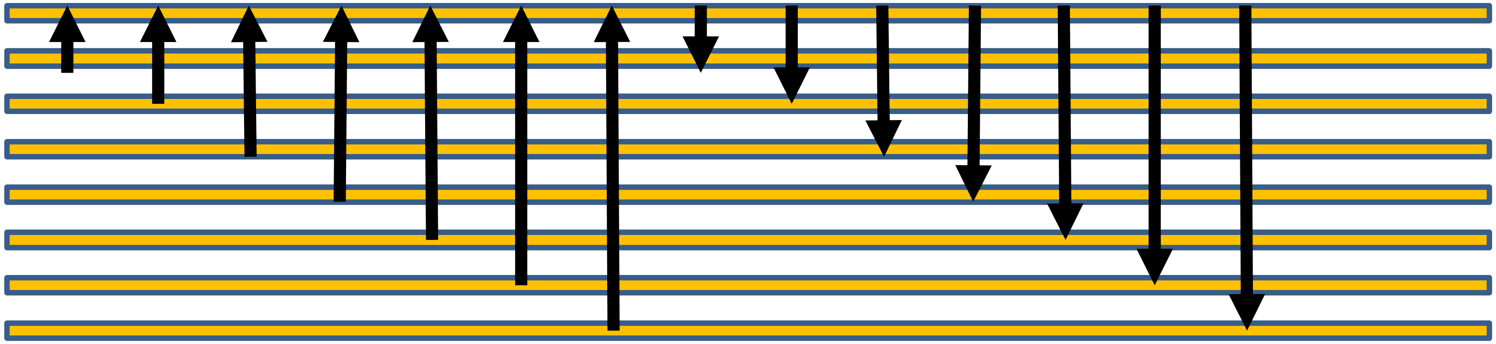
If you have tried hard to perform the required exercises and the programs still don‘t work, you are allowed to look into the directories

```
~ohaan/mpikurs_solutions/f
~ohaan/mpikurs_solutions/c
~ohaan/mpikurs_solutions/py
```

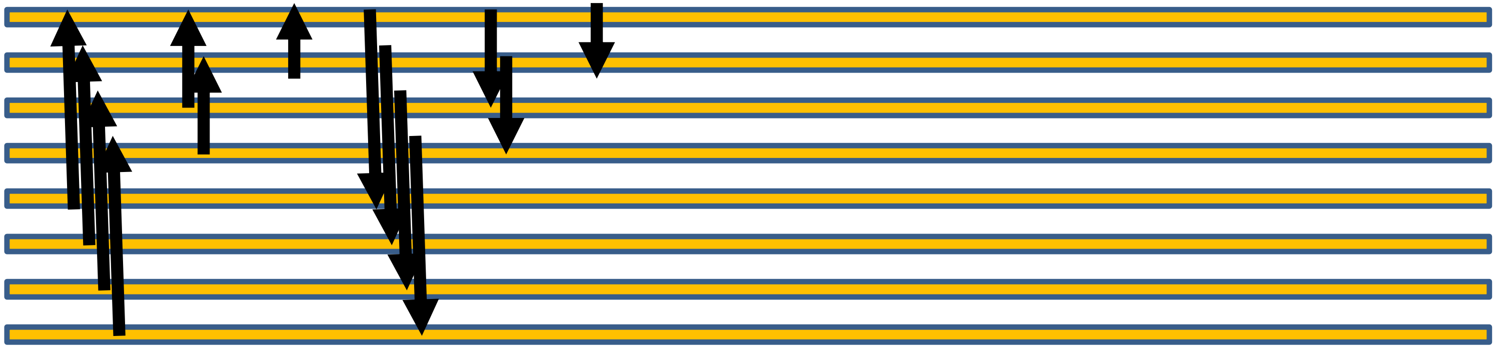
where you will find the completed programs for some exercises

Exercise 1: Synchronization

Sequential Synchronization (t~ np)



Cascade Synchronization (t~ln (np))



Example implementation in program **`synch_casc.f`** (valid only for np =2^k)

Exercise 2: Broadcast

Modify program bcast (*distribution of input value n from process 0 to all other processes*) by using the broadcast function instead of the sequential send and receive operations

C:

```
   MPI_Bcast( void *buf, int count,
                 MPI_Type datatype, int root, MPI_Comm comm )
```

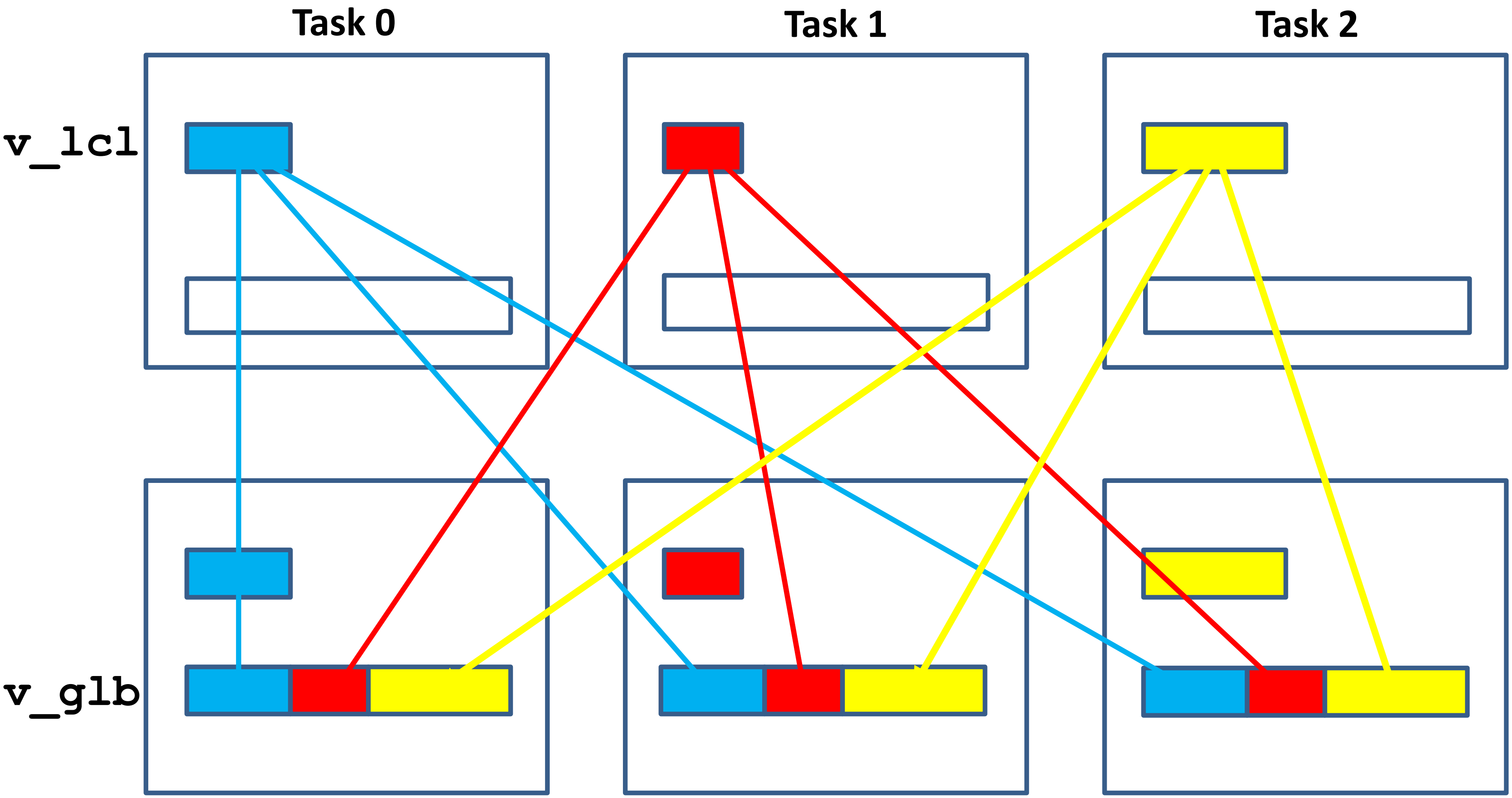
Fortran:

```
   MPI_BCAST( buf, count, datatype, root, comm, ierror )
   <type>buf(*), INTEGER count, datatype, root, comm,
   ierror
```

mpi4py:

```
    robj = comm.bcast(sobj, root= 0)
    comm.Bcast(ar, root= 0)
```

Exercise 3: Gather Data(1)

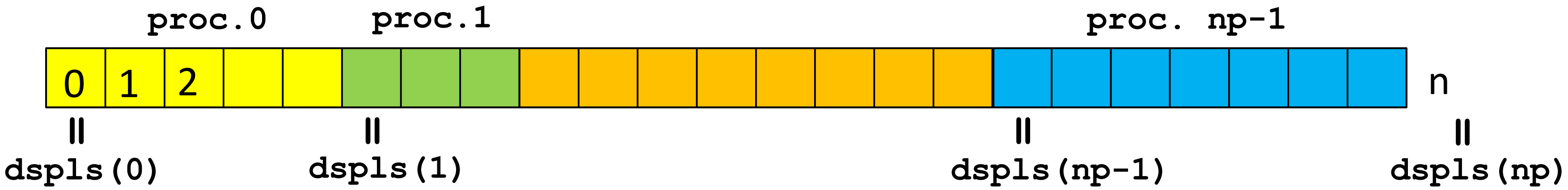


Exercise 3: Gather Data(2)

`v_glb` is a vector with **`n`** elements in **`np`** intervals

vector of interval sizes: **`counts(0),...,counts(np-1)`**

vector of start indices : **`dspls(0),...,dspls(np)`**



Length of local vectors on process ip :

`counts(ip) = dspls(ip+1)-dspls(ip), ip=0,np-1`

Length of global vector :

`n = dspls(np)`

Exercise 3: Gather Data(3)

Example:

Length of local vector on process ip is ip+3:

```
dspls(0)  = 0
do ip = 1 , np
   dspls(ip)  = dspls(ip-1)+ 3 +(ip-1)
   counts(ip-1)  = dspls(ip)  - dspls(ip-1)
end do
nglb = dspls(np)
```

Initialize the local vectors (such that v_glb(i) = i):

```
do i = 0 , counts(myid)-1
   v_lcl(i)  = dspls(myid)+i
end do
```

Exercise 3: Gather Data(4)

Solution 1: gather with BSEND / RECV
(`program collect_sendrecv`)

Every process sends its local vector to all other processes:

```
nlcl = counts(myid)
do ip = 0,np-1
  call MPI_BSEND(v_lcl,nlcl,type,ip ...
```

Every process stores local vectors from other processes at the appropriate location in the global vector:

```
do ip = 0,np-1
    nrecv = counts(ip)
    call MPI_RECV(v_glb(dspls(ip)),nrecv,type,ip ...
```

Exercise 3: Gather Data(5)

Solution 2: with BCAST
(complete **program collect_bcast**)

Every process copies its **v_lcl** to its **v_glb**:

```
   nlcl = counts(myid)
    do i = 1 , nlcl
        v_glb(dspls(myid)+i) = v_lcl(i)
```

Every process broadcasts this part of v_glb

Syntax for broadcast:
MPI_BCAST(buffer, count, datatype, root, comm)
comm.Bcast(buf, root = root)

Exercise 3: Gather Data(6)

Solution 3: with GATHERV
(complete **program collect_gather**)

Gather local Data v_lcl of all processes in v_glb in process 0:

```
   call MPI_GATHERV( v_lcl, counts(myid), sendtype,
      v_glb, counts, dspls, recvtype, 0, comm, ierr )
```

BCAST v_glb from process 0 to all processes

```
   call MPI_BCAST(v_glb,nglb,type,0,comm, ierr )
```

Combine the two steps with: **MPI_ALLGATHERV**

Exercise 3: Gather Data(7)

Solution 3: with GATHERV (mpi4py)
(complete **program collect_gather**)

Gather local Data v_lcl of all processes in v_glb in process 0:

```
comm.Gatherv( sendbuf, recvbuf, root=0)
```

where:

```
sendbuf = v_lcl
recvbuf =[v_glb,counts,dspls[0:nproc],MPI.DOUBLE]
```

BCAST v_glb from process 0 to all processes

```
comm.Bcast(v_glb, root=0)
```

Alternatively :Combine the two steps with: **MPI_ALLGATHERV**

Exercise 4: Monitoring Program Execution

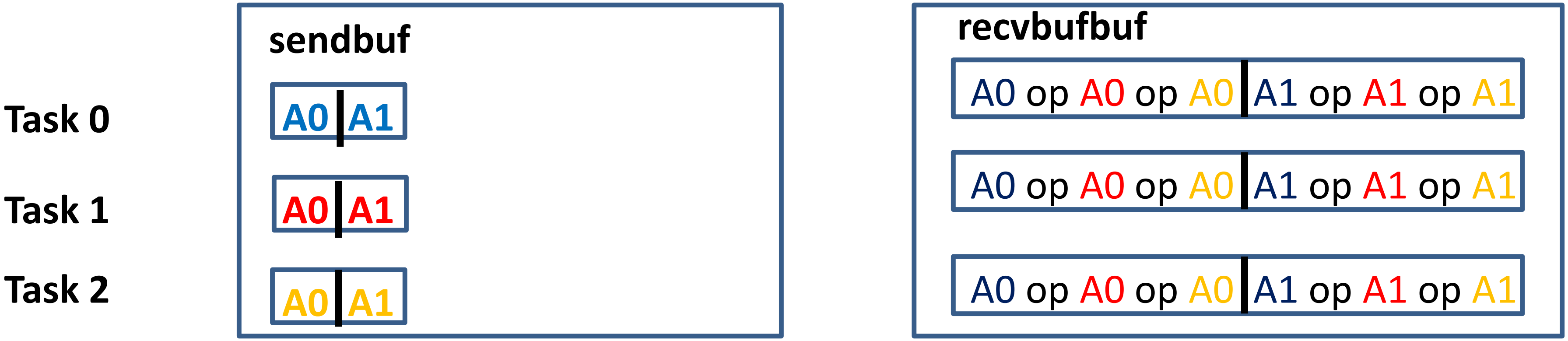
Signaling an error in one process to all other processes

- Look at the program errexit.f, errexit.py ,

  find out its behaviour
- Combine MPI_REDUCE + MPI_BCAST to MPI_ALLREDUCE

Syntax:

MPI_ALLREDUCE(sendbuf, recvbuf, count, datatype, op, comm)
recvbuf= comm.reduce(*sendobj* = sendbuf, *recvobj*=None, *op*=op)



	sendbuf	recvbufbuf
Task 0	A0 | A1	A0 op A0 op A0 | A1 op A1 op A1
Task 1	A0 | A1	A0 op A0 op A0 | A1 op A1 op A1
Task 2	A0 | A1	A0 op A0 op A0 | A1 op A1 op A1

Exercise 5: Reduce: MPI_SUM

- Generate a program to distribute the summation of integers from 1 to N.
- Hint: Calculate partial sums on every process and combine them to the total result with MPI_REDUCE using the operation MPI_SUM
- Modify the sequential code in
- `intsum.f, intsum.c, intsum.py`

Syntax of MPI_REDUCE

```
call MPI_REDUCE(suml,sum,1,MPI_INTEGER, MPI_SUM,
   :                0,MPI_COMM_WORLD, ierr )
sum = comm.reduce(suml,op=MPI.SUM,root=0)
comm.Reduce(suml,sum,op=MPI.SUM,root=0)
```

Exercise 6: Reduce

Modify step 7 in program piapp_mpi (*add up all local **res** to the total results* ***pia*** *on process **0***) by using the reduce function instead of the sequential send and receive operations

C:

```
MPI_Reduce( void *sendbuf, void *recvbuf, int count,
            MPI_Type datatype, MPI_Op op, int root,
            MPI_Comm comm )
```

Fortran:

```
MPI_REDUCE( sendbuf, recvbuf, count, datatype, op,
root, comm, ierror )
<type>sendbuf(*), recvbuf
INTEGER count, datatype, op, root, comm, ierror
```

mpi4py:

```
comm.Reduce(sbuf, rbuf,op=oper root= 0)
```