Programming with MPI
Collectives

Jan Thorbecke

Collectives

- Classes
- Communication types
  - exercise: BroadcastBarrier
- Gather Scatter
  - exercise: GatherScatter
- Reduction
  - exercise: Reduce
- Optional
  - exercises: Collectives
  - exercises: MPIpi

Collective Communications in MPI

- Communication is co-ordinated among a group of processes, as specified by communicator.
- All processes in the communicator group must call the collective operation
- All collective operations are blocking and no message tags are used (in MPI-1).
- In MPI-3 non-blocking collectives are introduced.

Collective Classes

- Synchronization
  - Barrier
- Data movement
  - Broadcast
  - Gather Scatter
  - All Gather
- Collective Computation
  - Reduce
  - All Reduce

Collective Communication pattern

- One to Many
  - broadcast
  - scatter
- Many to One
  - allreduce
- Many to Many
  - alltoall

Collective communication

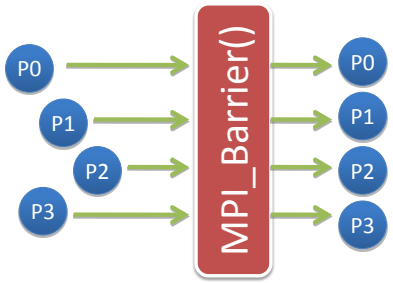
- Collective communication may outperform point-to-point communication
  - Depends on implementation and case
- Code is more compact and easier to read:

```
if (my_id == 0) then
   do i = 1, ntasks-1
      call mpi_send(a, 1048576, &
        MPI_REAL, i, tag, &
        MPI_COMM_WORLD, rc)
   end do
else
   call mpi_recv(a, 1048576, &
        MPI_REAL, 0, tag, &
        MPI_COMM_WORLD, status, rc)
end if
```

```
call mpi_bcast(a, 1048576, &
        MPI_REAL, 0, &
        MPI_COMM_WORLD, rc)
```

Communicating a vector a consisting of 1M float elements from the task 0 to all other tasks with point-to-point and collective communication

MPI_Barrier()

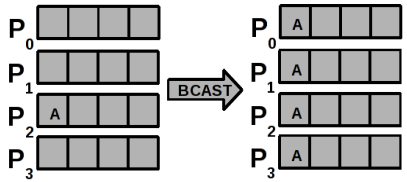


Creates barrier synchronization in a communicator group comm. Each process, when reaching the MPI_Barrier call, blocks until all the processes in the group reach the same MPI_Barrier call.

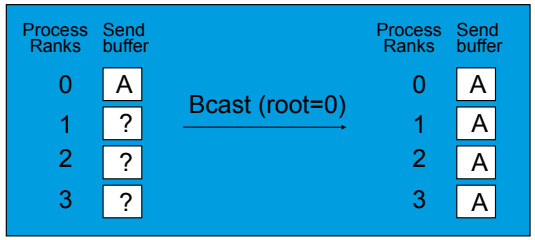
Broadcast

- With **MPI_Bcast**, the task *root* sends a *buffer* of data to all other tasks

```
MPI_Bcast(buffer, count, datatype, root, comm)
buffer      data to be distributed
count       number of entries in buffer
datatype    data type of buffer
root        rank of broadcast root
comm        communicator
```

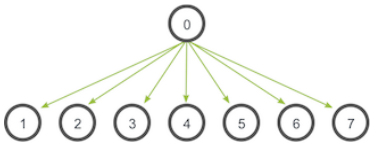


Broadcast

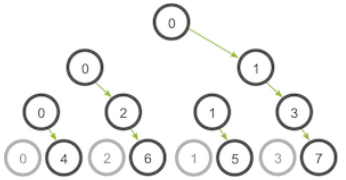


Exercise: SendRecvCollectives/ BroadcastBarrier

'Write' and compare a simple broadcast program,



with a more advanced algorithm




```
Data size = 4000000, Trials = 10
```

```
mpi=56 1-node
Avg my_bcast time = 0.043814
Avg MPI_Bcast time = 0.003524
mpi=112 2-nodes
Avg my_bcast time = 0.083837
Avg MPI_Bcast time = 0.005280
mpi=224 4-nodes
Avg my_bcast time = 0.163445
Avg MPI_Bcast time = 0.005970
mpi=448 8-nodes
Avg my_bcast time = 0.317528
Avg MPI_Bcast time = 0.006469
mpi=896 16-nodes
Data size = 4000000, Trials = 10
Avg my_bcast time = 0.633746
Avg MPI_Bcast time = 0.007451
```

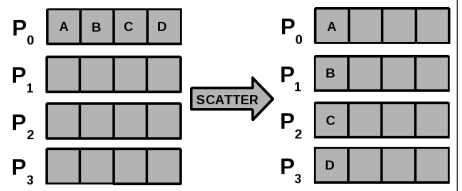

Scatter

- **MPI_Scatter**: Task *root* sends an equal share of data (*sendbuf*) to all other processes

MPI_Scatter(sendbuf, sendcount, sendtype, recvbuf, recvcount, recvtype, root, comm)

sendbuf	send buffer (data to be scattered)
sendcount	number of elements sent to each process
sendtype	data type of send buffer elements
recvbuf	receive buffer
recvcount	number of elements in receive buffer
recvtype	data type of receive buffer elements
root	rank of sending process
comm	communi-cator

Not the size of the whole sendbuf, but the chunk of data to be sent



Bcast vs Scatter

```
if (my_id==0) then
  do i = 1, 16
    a(i) = i
  end do
end if

call mpi_bcast(a, 16,&
    MPI_INTEGER, 0, &
    MPI_COMM_WORLD, rc)

if (my_id==3) print *, a(:)
```

```
if (my_id==0) then
  do i = 1, 16
    a(i) = i
  end do
end if
call mpi_scatter(a, 4, &
    MPI_INTEGER,&
    aloc, 4, MPI_INTEGER, &
    MPI_COMM_WORLD, rc)

if (my_id==3) print *, aloc(:)
```

Assume 4 MPI tasks. What would the (full) program print?

```
> aprun -n 4 ./a.out

 1    2    3    4
 5    6    7    8
 9   10   11   12
13   14   15   16
```

```
> aprun -n 4 ./a.out

13      14      15        16
```

Variable width Scatter

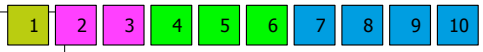
- C & Fortran bindings

```
int MPI_Scatterv(void* sendbuf, int *sendcounts, int *displs,
      MPI_datatype sendtype, void* recvbuf, int recvcount,
      MPI_datatype recvtype, int root, MPI_Comm comm)


MPI_SCATTERV(sendbuf, sendcounts, displs, sendtype, recvbuf,
      recvcount, recvtype, root, comm, ierror)
  type sendbuf(*), recvbuf(*)
  integer sendcounts(*), displs(*), recvcount, sendtype, recvtype,
  root, comm, ierror
```

sendbuf - address of sending buffer (choice, significant only at root)
sendcounts - integer array (of length group size) specifying the number of elements to send to each processor
displs - integer array (of length group size). Entry i specifies the displacement (relative to sendbuf from which to take the outgoing data to process i

Scatterv example

```
if (my_id==0) then
  do i = 1, 10
    a(i) = i
  end do
  sendcounts = (/ 1, 2, 3, 4 /)
  displs = (/ 0, 1, 3, 6 /)
end if
call mpi_scatterv(a, sendcnts, &
    displs, MPI_INTEGER,&
    aloc, 4, MPI_INTEGER, &
    0, MPI_COMM_WORLD, rc)
```



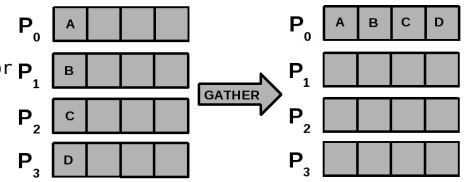

```
1   0   0   0   TASK 0

2   3   0   0   TASK 1

4   5   6   0   TASK 2

7   8   9   10  TASK 3
```

Assume 4 MPI tasks. What are the values in **aloc** in different tasks?

Gather

- **MPI_Gather** collects individual data from each task to the *root* task

```
MPI_Gather(sendbuf, sendcount, sendtype, recvbuf, recvcount,
     recvtype, root, comm)
sendbuf   send buffer
sendcount number of elements in send buffer
sendtype  data type of send buffer elements
recvbuf   receive buffer
recvcount number of elements for any single receive
recvtype  data type of recv buffer elements
root      rank of
          receiving
          process
comm      communicator
```



Variable width Gather

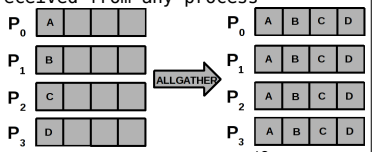
- MPI_Gatherv is similar to MPI_Gather, but allows for varying amounts of data

```
MPI_Gatherv(sendbuf, sendcount, sendtype, recvbuf,
     recvcounts, displs, recvtype, root, comm)

sendbuf   send buffer
sendcount number of elements in send buffer
sendtype  data type of send buffer elements
recvbuf   receive buffer
recvcnts  array of number of elements to be receive from
          each task
displs    integer array (of length group size). Entry i
          specifies the displacement relative to recvbuf at
          which to place the incoming data from process i
recvtype  data type of recv buffer elements
root      rank of receiving process
comm      communicator
```

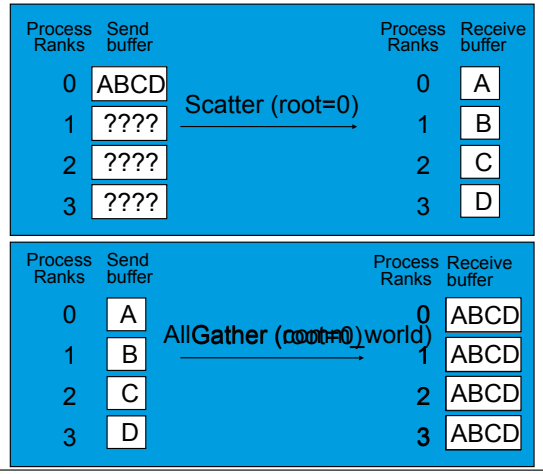

Allgather

- MPI_Allgather gathers data from each task and distributes the resulting data to each task
  - Compare: MPI_Gather + MPI_Bcast

```
MPI_Allgather(sendbuf, sendcount, sendtype, recvbuf,
     recvcount, recvtype, comm)

sendbuf   send buffer
sendcount number of elements in send buffer
sendtype  data type of send buffer elements
sendbuf   receive buffer
recvcount number of elements received from any process
recvtype  data type of receive
          buffer elements
comm      communicator
```



Scatter and Gather



Exercise: SendRecvCollectives/ GatherScatter

1. Generate a random array of numbers on the root process (process 0).

2. Scatter the numbers to all processes, giving each process an equal amount of numbers.

3. Each process computes the average of their subset of the numbers.

4. Gather all averages to the root process. The root process then computes the average of these numbers to get the final average.

Exercise: SendRecvCollectives/ GatherScatter

Directory SendRecvCollectives: Implement

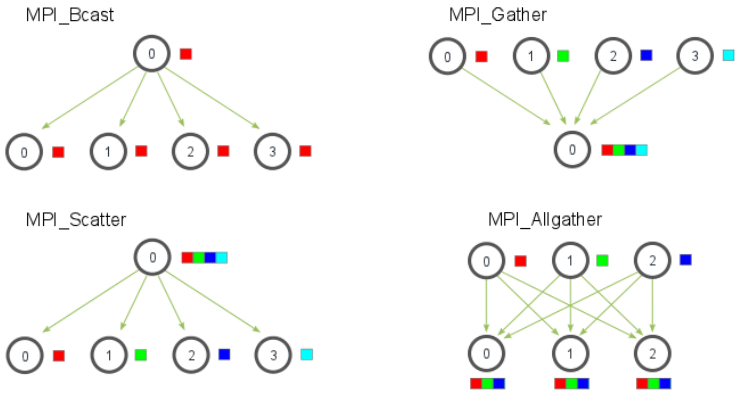
- call to MPI_Scatter(….)
- call to MPI_Gather(….)

into template avg.c (avg_sol.c contains a solution)

- Extend avg.c to use MPI_Allgather(), so all tasks receive the data (all_avg.c)

Follow the guidelines in the README

Exercise: GatherScatter implemented in MPI

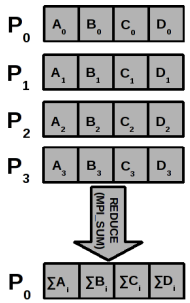


Reduce operation

- Applies a reduction operation *op* to *sendbuf* over the set of tasks and places the result in *recvbuf* on *root*

MPI_Reduce(sendbuf, recvbuf, count, datatype, op, root, comm)

sendbuf	send buffer
recvbuf	receive buffer
count	number of elements in send buffer
datatype	data type of elements of send buffer
op	reduce operation
root	rank of root process
comm	communicator

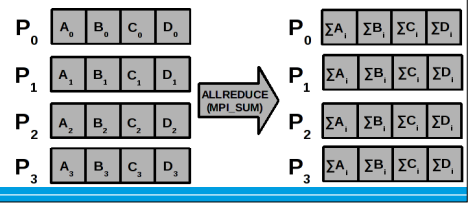


Global reduce operation

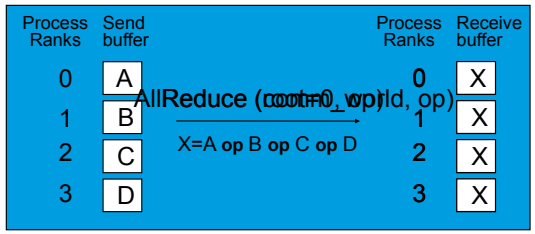
- **MPI_Allreduce** Combines values from all processes and distribute the result back to all processes
  - Compare: MPI_Reduce + MPI_Bcast

MPI_Allreduce(sendbuf, recvbuf, count, datatype, op, comm)

sendbuf	starting address of send buffer
recvbuf	starting address of receive buffer
count	number of elements in send buffer
datatype	data type of elements of send buffer
op	operation
comm	communicator



Reduce



Allreduce example

```
real :: a(1024), aloc(128)
real :: rloc, r
integer :: i, my_id, ntasks, rc

call mpi_init(rc)
call mpi_comm_rank(MPI_COMM_WORLD,my_id,rc)
if (my_id==0) then
   call random_number(a)
end if

call mpi_scatter(a, 128, MPI_INTEGER, &
     aloc, 128, MPI_INTEGER, &
      0, MPI_COMM_WORLD, rc)
rloc = dot_product(aloc,aloc)
call mpi_allreduce(rloc, r, 1, MPI_REAL, &
     MPI_SUM, MPI_COMM_WORLD, rc)

print *, 'id=', my_id, 'local=', rloc, &
     'global=', r
call mpi_finalize(rc)
```

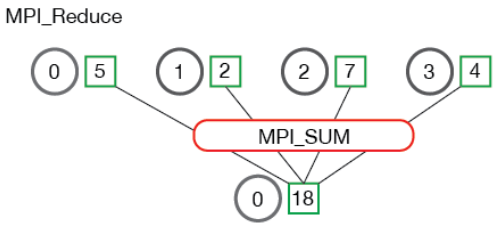
```
> mpirun -np 8 a.out
 id= 6 local= 39.68326  global= 338.8004
 id= 7 local= 39.34439  global= 338.8004
 id= 1 local= 42.86630  global= 338.8004
 id= 3 local= 44.16300  global= 338.8004
 id= 5 local= 39.76367  global= 338.8004
 id= 0 local= 42.85532  global= 338.8004
 id= 2 local= 40.67361  global= 338.8004
 id= 4 local= 49.45086  global= 338.8004
```

Parallel $r = \mathbf{a} \cdot \mathbf{a}$

Predefined Reduction Operation Handles

Predefined operation handle	Function
MPI_MAX	Maximum
MPI_MIN	Minimum
MPI_SUM	Sum
MPI_PROD	Product
MPI_LAND	Logical AND
MPI_BAND	Bitwise AND
MPI_LOR	Logical OR
MPI_BOR	Bitwise OR
MPI_LXOR	Logical exclusive OR
MPI_BXOR	Bitwise exclusive OR
MPI_MAXLOC	Maximum and location of the maximum
MPI_MINLOC	Minimum and location of the minimum

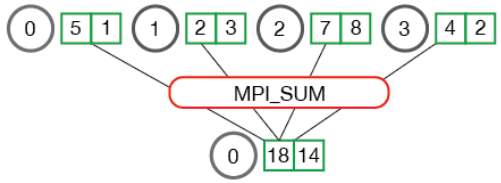
Exercise: Reduce



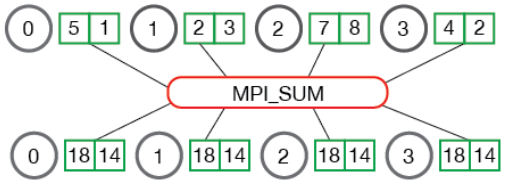
Follow the guidelines in the README

Exercise: Reduce

MPI_Reduce



MPI_Allreduce

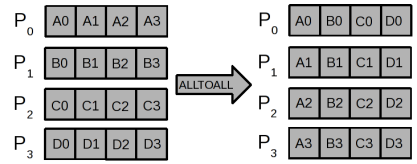


All happy together: from each to everyone

- **MPI_Alltoall** sends a distinct message from each task to every task
  - Compare: "All scatter"

MPI_Alltoall(sendbuf, sendcount, sendtype, recvbuf, recvcount, recvtype, comm)

sendbuf	send buffer
sendcount	number of elements to send to each process
sendtype	data type of send buffer elements
recvbuf	receive buffer
recvcount	number of elements received from any process
recvtype	data type of receive buffer elements
comm	communicator



Alltoall example

Suppose there are four processes including the root, each with arrays as shown below on the left. After the all-to-all operation

```
MPI_Alltoall(u, 2, MPI_INT, v, 2, MPI_INT, MPI_COMM_WORLD);
```

the data will be distributed as shown below on the right:

array u	Rank	array v
10 11 12 13 14 15 16 17	0	10 11 20 21 30 31 40 41
20 21 22 23 24 25 26 27	1	12 13 22 23 32 33 42 43
30 31 32 33 34 35 36 37	2	14 15 24 25 34 35 44 45
40 41 42 43 44 45 46 47	3	16 17 26 27 36 37 46 47

MPI Collective Routines

- Several routines:

MPI_ALLGATHER	**MPI_ALLGATHERV**
MPI_ALLTOALL	**MPI_ALLTOALLV**
MPI_GATHER	**MPI_GATHERV**
MPI_REDUCE_SCATTER	**MPI_REDUCE**
MPI_ALLREDUCE	**MPI_BCAST**
MPI_SCATTERV	**MPI_SCATTER**

- **All** versions deliver results to all participating processes
- **"V"** versions allow the chunks to have different sizes

- **MPI_ALLREDUCE, MPI_REDUCE, MPI_REDUCE_SCATTER,** and **MPI_REDUCE_SCAN** take both built-in and user-defined combination functions (**MPI_OP_CREATE**)

Common mistakes with collectives

- Using a collective operation within one branch of an if-test of the rank

```
IF (my_id == 0) CALL MPI_BCAST(...
```

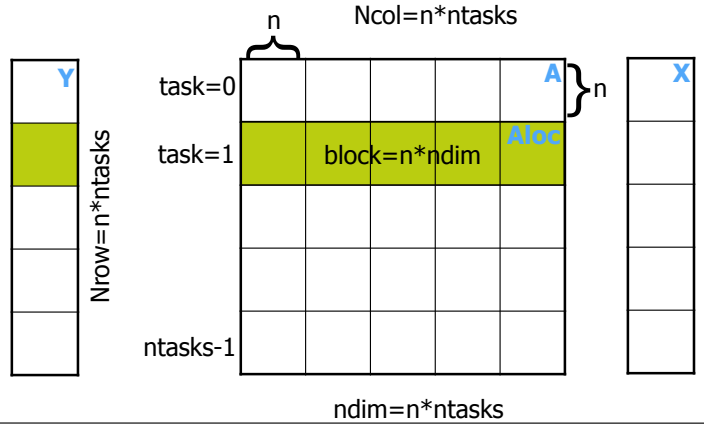
  - All processes, both the root (the sender or the gatherer) and the rest (receivers or senders), must call the collective routine
- Assuming that all processes making a collective call would complete at the same time
- Using the input buffer as the output buffer

```
CALL MPI_ALLREDUCE(a, a, n, MPI_REAL, MPI_SUM, ...
```


Exercises: Collectives (homework?)

- Directory MatrixVector part 1.
  - **Matrix vector multiplication (mvx.c / f90)**
  - solution in mvx_mpi.c / f90
- optional: Collectives/computePi:
  compute the value of PI parallel
- optional: Collectives/average:
  same as reduce exercise, but now have to write it yourself.

Collectives: Matrix-Vector



Exercise: PI with MPI and OpenMP