Contents

1. Introduction to pbdMPI
   - Managing a Communicator
   - Reduce, Gather, Broadcast, and Barrier
   - Other

Message Passing Interface (MPI)

- *MPI*: Standard for managing communications (data and instructions) between different nodes/computers.
- *Implementations*: OpenMPI, MPICH2, Cray MPT, ...
- Enables parallelism on distributed machines.
- *Communicator*: manages communications between processors.

MPI Operations (1 of 2)

- **Managing a Communicator**: Create and destroy communicators.
  `init()` — initialize communicator
  `finalize()` — shut down communicator(s)

- **Rank query**: determine the processor's position in the communicator.
  `comm.rank()` — "who am I?"
  `comm.size()` — "how many of us are there?"

- **Printing**: Printing output from various ranks.
  `comm.print(x)`
  `comm.cat(x)`
  WARNING: only use these functions on *results*, never on yet-to-be-computed things.

Quick Example 1

Rank Query: 1_rank.r

```
1 library(pbdMPI, quiet = TRUE)
2 init()
3
4 my.rank <- comm.rank()
5 comm.print(my.rank, all.rank=TRUE)
6
7 finalize()
```

Execute this script via:

```
1 mpirun -np 2 Rscript 1_rank.r
```

Sample Output:

```
1 COMM.RANK = 0
2 [1] 0
3 COMM.RANK = 1
4 [1] 1
```

Quick Example 2

Hello World: 2_hello.r

```
1 library(pbdMPI, quiet=TRUE)
2 init()
3
4 comm.print("Hello, world")
5
6 comm.print("Hello again", all.rank=TRUE, quiet=TRUE)
7
8 finalize()
```

Execute this script via:

```
1 mpirun -np 2 Rscript 2_hello.r
```

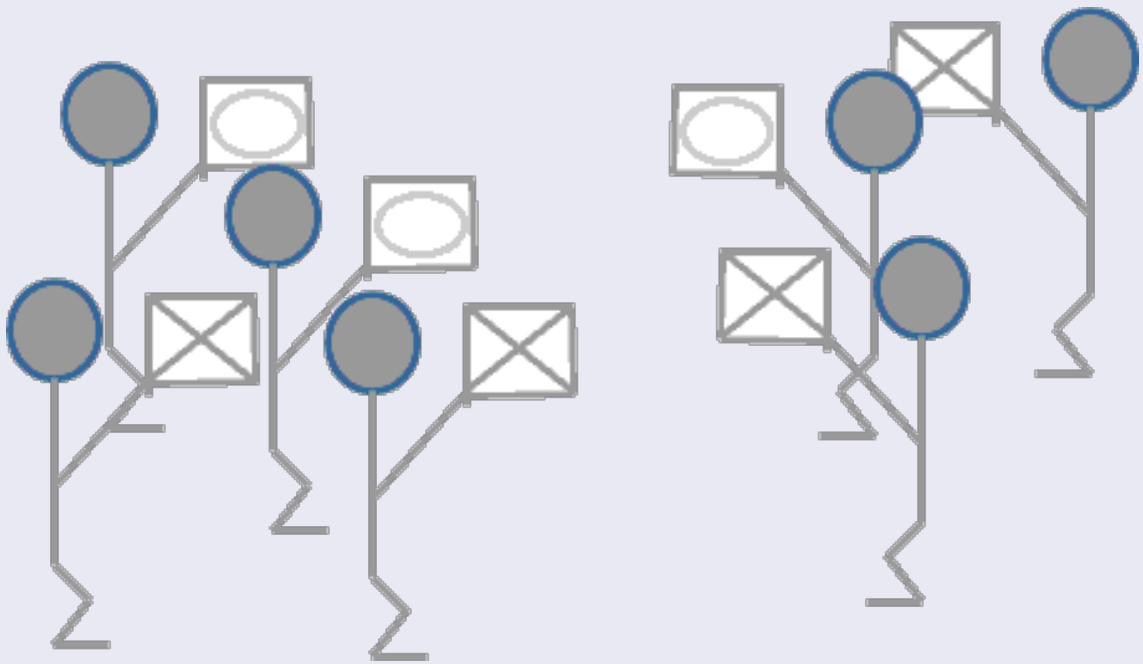
Sample Output:

```
1 COMM.RANK = 0
2 [1] "Hello, world"
3 [1] "Hello again"
4 [1] "Hello again"
```

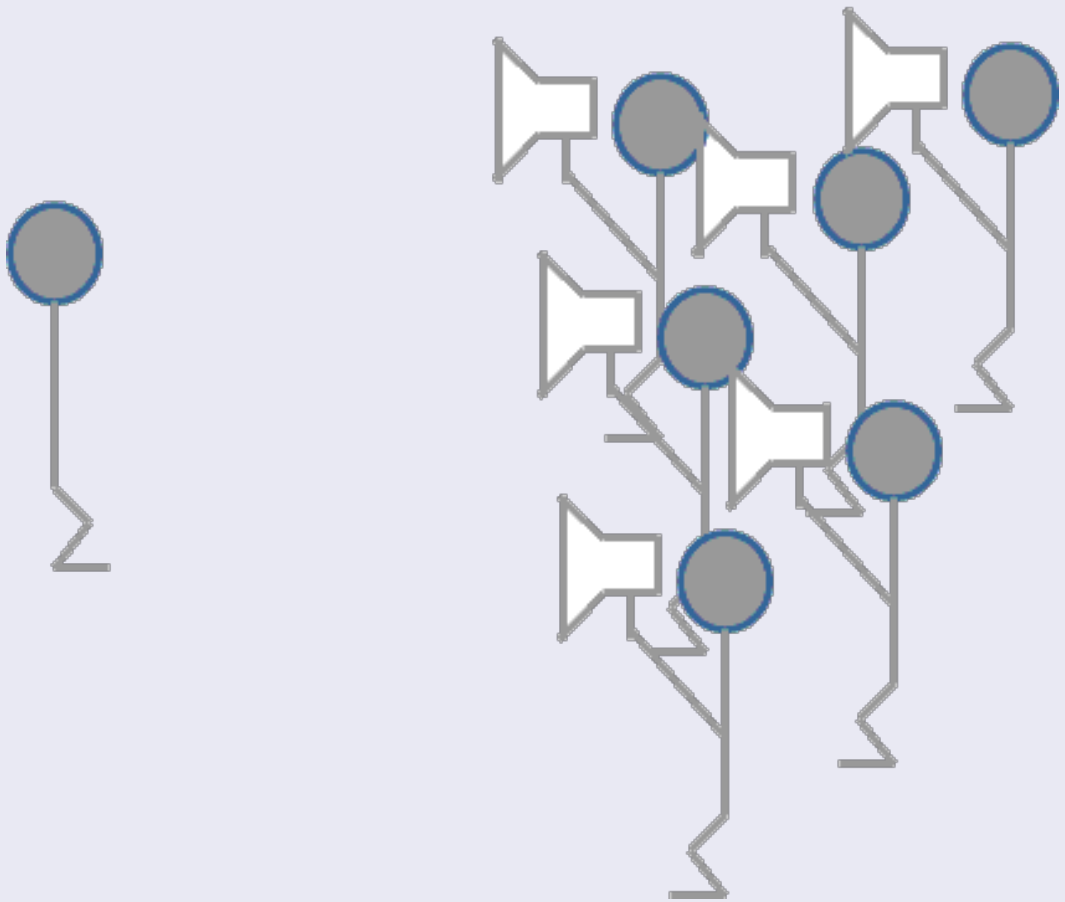
MPI Operations

1. Reduce
2. Gather
3. Broadcast
4. Barrier

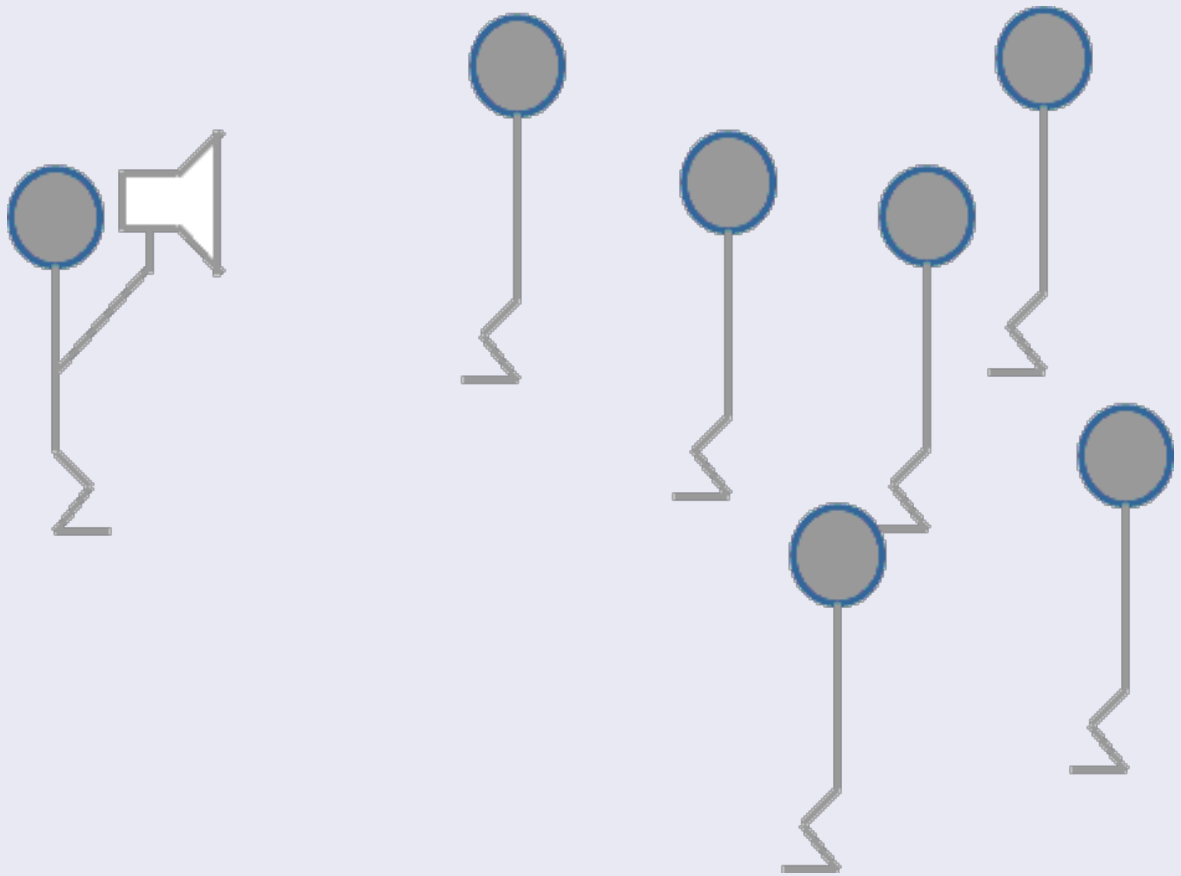
Reductions — Combine results into single result



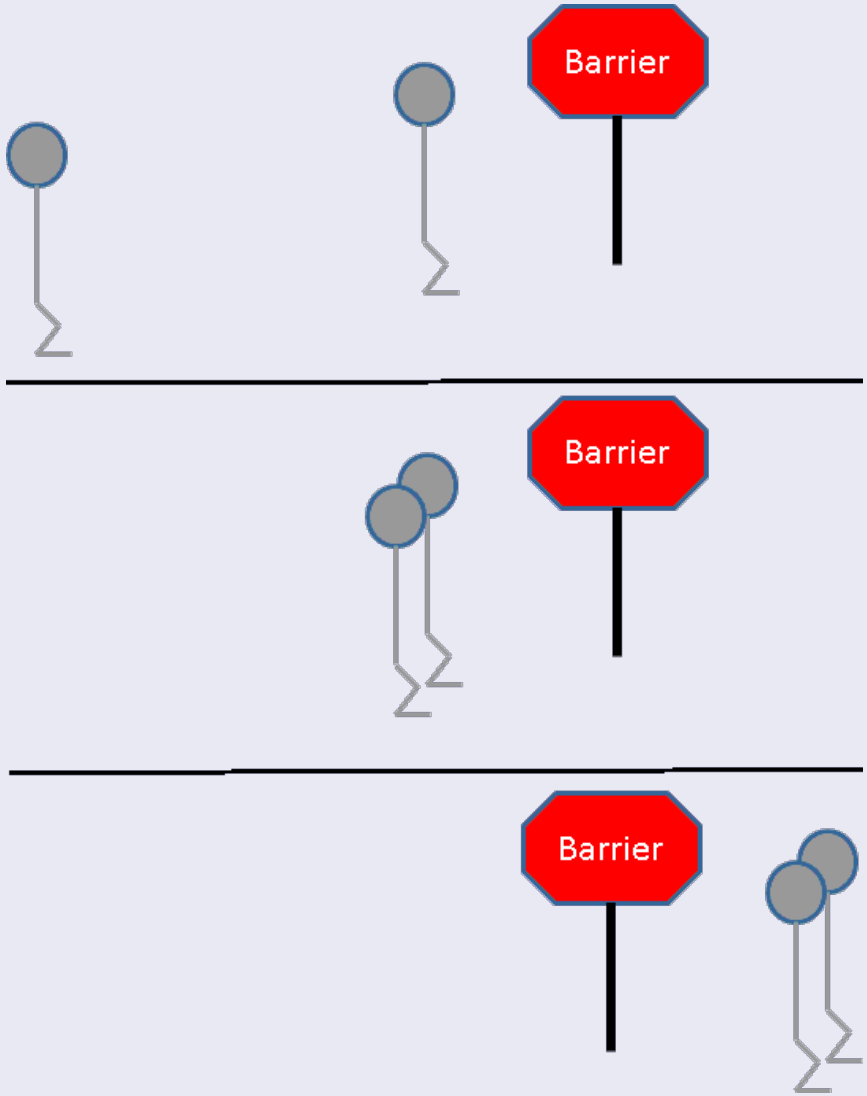
Gather — Many-to-one



Broadcast — One-to-many



Barrier — Synchronization



MPI Operations (2 of 2)

- **Reduction**: each processor has a number `x`; add all of them up, find the largest/smallest, ...
  `reduce(x, op='sum')` — reduce to one
  `allreduce(x, op='sum')` — reduce to all

- **Gather**: each processor has a number; create a new object on some processor containing all of those numbers.
  `gather(x)` — gather to one
  `allgather(x)` — gather to all

- **Broadcast**: one processor has a number `x` that every other processor should also have.
  `bcast(x)`

- **Barrier**: "computation wall"; no processor can proceed until *all* processors can proceed.
  `barrier()`

Quick Example 3

Reduce and Gather: 3_gt.r

```
1  library(pbdMPI, quiet = TRUE)
2  init()
3
4  comm.set.seed(diff=TRUE)
5
6  n <- sample(1:10, size=1)
7
8  gt <- gather(n)
9  comm.print(unlist(gt))
10
11 sm <- allreduce(n, op='sum')
12 comm.print(sm, all.rank=T)
13
14 finalize()
```

Execute this script via:

```
1  mpirun -np 2 Rscript 3_gt.r
```

Sample Output:

```
1  COMM.RANK = 0
2  [1] 2 8
3  COMM.RANK = 0
4  [1] 10
5  COMM.RANK = 1
6  [1] 10
```

Quick Example 4

Broadcast: 4_bcast.r

```
1  library(pbdMPI, quiet=T)
2  init()
3
4  if (comm.rank()==0){
5    x <- matrix(1:4, nrow=2)
6  } else {
7    x <- NULL
8  }
9
10 y <- bcast(x, rank.source=0)
11
12 comm.print(y, rank=1)
13
14 finalize()
```

Execute this script via:

```
1  mpirun -np 2 Rscript 4_bcast.r
```

Sample Output:

```
1  COMM.RANK = 1
2       [,1] [,2]
3  [1,]    1    3
4  [2,]    2    4
```

MPI Package Controls

The `.SPMD.CT` object allows for setting different package options with **pbdMPI**. See the entry *SPMD Control* of the **pbdMPI** manual for information about the `.SPMD.CT` object:

`http://cran.r-project.org/web/packages/pbdMPI/pbdMPI.pdf`

Quick Example 5

Barrier: 5_barrier.r

```
1  library(pbdMPI, quiet = TRUE)
2  init()
3
4  .SPMD.CT$msg.barrier <- TRUE
5  .SPMD.CT$print.quiet <- TRUE
6
7  for (rank in 1:comm.size()-1){
8    if (comm.rank() == rank){
9      cat(paste("Hello", rank+1, "of", comm.size(), "\n"))
10   }
11   barrier()
12 }
13
14 comm.cat("\n")
15
16 comm.cat(paste("Hello", comm.rank()+1, "of",
       comm.size(), "\n"), all.rank=TRUE)
17
18 finalize()
```

Execute this script via:

```
1  mpirun -np 2 Rscript 5_barrier.r
```

Sample Output:

```
1  Hello 1 of 2
2  Hello 2 of 2
```

Random Seeds

pbdMPI offers a simple interface for managing random seeds:

- `comm.set.seed(diff=TRUE)` — Independent streams via the **rlecuyer** package.
- `comm.set.seed(seed=1234, diff=FALSE)` — All processors use the same seed `seed=1234`
- `comm.set.seed(diff=FALSE)` — All processors use the same seed, determined by processor 0 (using the system clock and PID of processor 0).

Quick Example 6

Timing: 6_timer.r

```
1  library(pbdMPI, quiet=TRUE)
2  init()
3
4  comm.set.seed(diff=T)
5
6  test <- function(timed)
7  {
8    ltime <- system.time(timed)[3]
9
10   mintime <- allreduce(ltime, op='min')
11   maxtime <- allreduce(ltime, op='max')
12   meantime <- allreduce(ltime, op='sum')/comm.size()
13
14   return(data.frame(min=mintime, mean=meantime,
         max=maxtime))
15 }
16
17 times <- test(rnorm(1e6)) # ~7.6MiB of data
18 comm.print(times)
19
20 finalize()
```

Execute this script via:

```
1  mpirun -np 2 Rscript 6_timer.r
```

Sample Output:

```
1    min  mean   max
2  1 0.17 0.173 0.176
```

Other Helper Tools

pbdMPI Also contains useful tools for Manager/Worker and task parallelism codes:

- **Task Subsetting**: Distributing a list of jobs/tasks
  `get.jid(n)`
- ***ply**: Functions in the *ply family.
  `pbdApply(X, MARGIN, FUN, ...)` — analogue of `apply()`
  `pbdLapply(X, FUN, ...)` — analogue of `lapply()`
  `pbdSapply(X, FUN, ...)` — analogue of `sapply()`

Quick Comments for Using pbdMPI

1. Start by loading the package:

```
1 library(pbdMPI, quiet = TRUE)
```

2. Always initialize before starting and finalize when finished:

```
1 init()
2
3 # ...
4
5 finalize()
```