Lecture 19 - MPI Part 2

DSE 512

Drew Schmidt
2022-04-05

From Last Time

- Homework 3 due Saturday
- Open floor to talk about the homework
- Questions?

Today

- More basic MPI Examples
- I/O
- Task parallelism with MPI
- Some common problems with MPI programs

Recall: SPMD

- One program is written
- All processes execute the single program
- Operates primarily on *collectives*

More MPI Examples

RNG

R

```
suppressMessages(library(pbdMPI))
comm.set.seed(1234, diff=TRUE)

x_local = sample(1:10, size=1)
comm.print(x_local, all.rank=TRUE)

finalize()
```

```
COMM.RANK = 0
[1] 4
COMM.RANK = 1
[1] 6
```

Python

```
from mpi4py import MPI
import random
comm = MPI.COMM_WORLD

random.seed(1234 + comm.rank)
x_local = random.randint(1, 10)
print(x_local)

MPI.Finalize()
```

```
7
8
```

Hello World: mpi4py

```
from mpi4py import MPI

comm = MPI.COMM_WORLD
rank = comm.Get_rank()
size = comm.Get_size()

comm_local = MPI.Comm.Split_type(comm, MPI.COMM_TYPE_SHARED, 0)
rank_local = comm_local.Get_rank()
size_local = comm_local.Get_size()

for p in range(0, size):
    if p == rank:
        print("Hello from rank ", end="")
        print(str(rank) + "/" + str(size) + " global ", end="")
        print(str(rank_local) + "/" + str(size_local) + " local")

    comm.Barrier()

MPI.Finalize()
```

Hello World: mpi4py

```
mpirun -np 2 python p.py
```

```
Hello from rank 0/2 global 0/2 local
Hello from rank 1/2 global 1/2 local
```

Hello World: pbdR

```
suppressMessages(library(pbdMPI))

rank = comm.rank()
size = comm.size()
rank_local = comm.localrank()

hostname = system("uname -n", intern=TRUE)
hostnames = allgather(hostname) |> unlist() |> table()
size_local = hostnames[hostname] |> unname()

msg = paste0("Hello from rank ", rank, "/", size, " global ", rank_local, " local\n")
comm.cat(msg, all.rank=TRUE, quiet=TRUE)

finalize()
```

Hello World: pbdR

```
$ mpirun -np 2 Rscript x.r
```

```
Hello from rank 0/2 global 0 local
Hello from rank 1/2 global 1 local
```

I/O

I/O With MPI

- No, not MPI-IO
- Old lessons still true (much more important)
  - Binary files are better than text files
  - I/O scales horribly
- A Few Strategies
  - Read on every rank
  - Read on rank 0 only
  - Reading from node-local storage
  - Read on a small set of ranks

Reading from Every Rank (Pseudocode)

R

```
suppressMessages(library(pbdMPI))

x = read_my_data("my_file")

finalize()
```

Python

```
from mpi4py import MPI
comm = MPI.COMM_WORLD

x = read_my_data("my_file")

MPI.Finalize()
```

Reading on Rank 0 (Pseudocode)

R

```r
suppressMessages(library(pbdMPI))

if (comm.rank() == 0){
  x_local = read_my_data("my_file")
} else {
  x_local = NULL
}

x = bcast(x_local)

finalize()
```

Python

```python
from mpi4py import MPI
comm = MPI.COMM_WORLD

if comm.rank == 0:
  x = read_my_data("my_file")
else:
  x_local = None

x = comm.bcast(x_local)

MPI.Finalize()
```

Reading from Node-Local Storage (Pseudocode)

R

```r
suppressMessages(library(pbdMPI))

file = "/shared_fs_storage/myfile"
local_file = "/node_local_storage/myfile"

if (comm.localrank() == 0){
  file.copy(from=file, to=local_file)
}

x = read_my_data(local_file)

finalize()
```

Python

```python
from mpi4py import MPI
import shutil
comm = MPI.COMM_WORLD
comm_local = MPI.Comm.Split_type(comm, MPI
rank_local = comm_local.Get_rank()

file = '/shared_fs_storage/myfile'
local_file = '/node_local_storage/myfile'

if rank_local == 0:
  shutil.copyfile(file, local_file)

x = read_my_data(local_file)

MPI.Finalize()
```

Reading from Subset of Ranks

- Useful, but fairly advanced technique
- Questions
  - How many readers?
  - *Which* ranks receive *which* data from *which* other ranks
- Very application dependent
- Honestly, try to avoid if possible

Task Parallelism with MPI

Task Parallelism

- SPMD vs MPSD
- We can fit MPSD inside SPMD

Distributing Tasks

- Pre-scheduling
  - All tasks chopped up evenly
  - Every worker knows which tasks to take based on rank
  - Good for homogeneous tasks
  - Easy to do
- Without pre-scheduling
  - Manager/worker by another name
  - Harder to implement
  - Need lots of heterogeneity and expensive tasks to scale at all

MPI Implementations

- Python
  - ... ?
- R
  - `pbdMPI::get.jid()`
  - `pbdMPI::pbdLapply()`
  - tasktools https://hpcran.org/packages/tasktools/index.html
- `get_my_tasks()` from lecture 17
  - For the sake of portability, we will use this one

Recall: R Implementation

```
get_my_tasks = function(num_tasks, num_workers, my_id){
  if (num_tasks == num_workers)
    my_id + 1
  else if (num_tasks > num_workers){
    local = as.integer(num_tasks / num_workers)
    rem = num_tasks %% num_workers
    if (rem == 0 || (my_id < (num_workers - rem))){
      start = my_id * local
      start:(start + local - 1) + 1
    } else {
      start = my_id*(local + 1) - (num_workers - rem)
      start:(start + local) + 1
    }
  } else {
    if (num_tasks > my_id)
      my_id + 1
    else
      integer(0)
  }
}
```

Recall: Python Implementation

```
def get_my_tasks(num_tasks, num_workers, my_id):
    if num_tasks == num_workers:
        return range(my_id, my_id+1)
    elif num_tasks > num_workers:
        local = int(num_tasks / num_workers)
        rem = num_tasks % num_workers

        if rem == 0 or (my_id < (num_workers - rem)):
            start = my_id*local
            return range(start, start + local)
        else:
            start = my_id*(local + 1) - (num_workers - rem)
            return range(start, start + local + 1)
    else:
        if num_tasks > my_id:
            return [my_id]
        else:
            return range(0)
```

Example: Square Roots

R

```
suppressMessages(library(pbdMPI))

# define get_my_tasks() here

data = 1:5
my_tasks = get_my_tasks(length(data), comm
results_local = sqrt(data[my_tasks])
results = gather(results_local)
comm.print(unlist(results))

finalize()
```

```
mpirun -np 2 Rscript example.r
```

```
[1] 1.000000 1.414214 1.732051 2.000000 2.236068
```

Python

```
import numpy as np
from mpi4py import MPI
comm = MPI.COMM_WORLD

# define get_my_tasks() here

data = np.linspace(1, 5, 5)
my_tasks = get_my_tasks(data.size, comm.si
results_local = np.sqrt(data[my_tasks])
results = comm.gather(results_local)

if comm.rank == 0:
  print(np.concatenate(results))

MPI.Finalize()
```

```
mpirun -np 2 python example.py
```

Task Parallelism with MPI

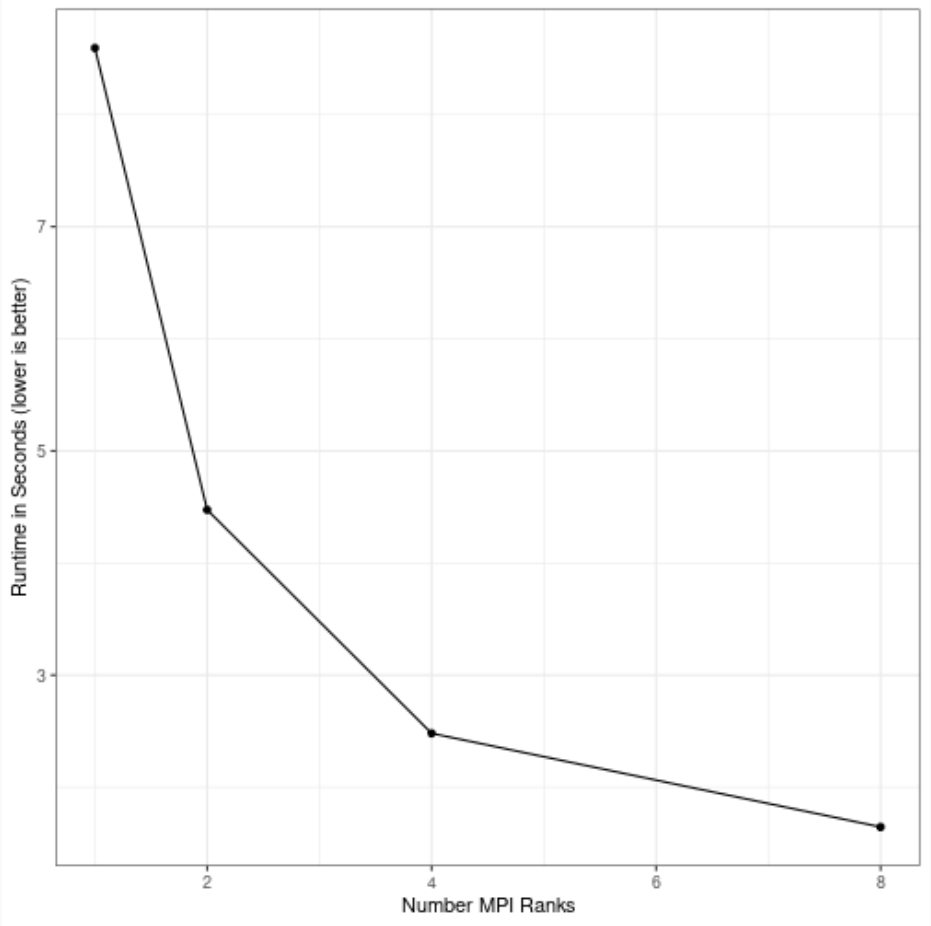
- General structure
  - Every rank gets its tasks
  - Every rank gets its data
  - Every rank performs its operation(s) locally
  - Collect data on rank 0 and/or write file(s) to lustre

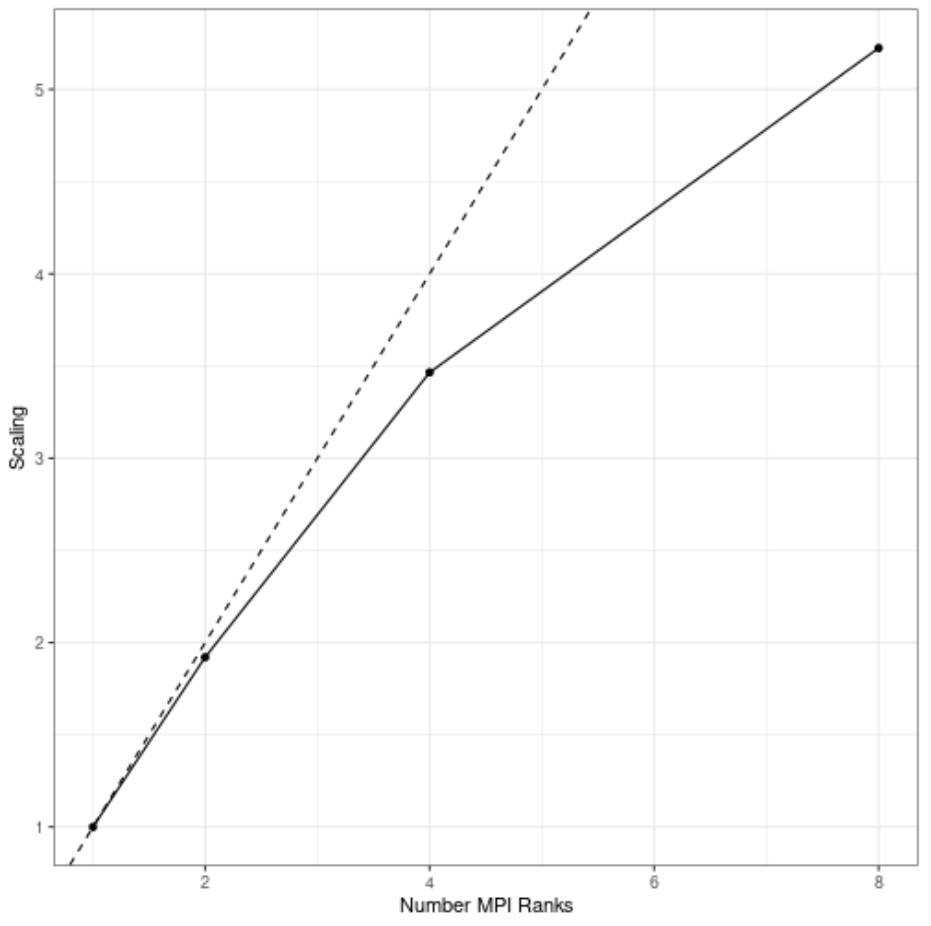
Monte Carlo π Simulation

```
suppressMessages(library(pbdMPI))
comm.set.seed(1234, diff=TRUE)

pi_sim_count = function(n){
  x = runif(n)
  y = runif(n)
  sum(x*x + y*y < 1)
}

n = 1e8
t = comm.timer({
  n_local = n / comm.size()
  count_local = pi_sim_count(n_local)
  count = reduce(count_local)
  comm.print(4 * count / n)
})

finalize()
```

```
mpirun -np 2 Rscript mcsim.r

[1] 3.141166
```

Monte Carlo π Simulation





Could You Translate This to Python?

As in our previous example, we can use "good enough" seed-setting:

```
np.random.seed(1234 + comm.rank)
```

More Complicated Task Parallelism

- Useful technique: create a "matrix" of parameters
- Each row is a task!
- Write this to disk (in a reasonable format)
- Start MPI program:
  - Every rank gets param matrix
    - Every rank reads it; or
    - One rank reads then distributes
  - Get tasks
  - etc.

```
expand.grid(a=1:3, b=0:1, c=0:1)
```

```
##    a b c
## 1  1 0 0
## 2  2 0 0
## 3  3 0 0
## 4  1 1 0
## 5  2 1 0
## 6  3 1 0
## 7  1 0 1
## 8  2 0 1
## 9  3 0 1
## 10 1 1 1
## 11 2 1 1
## 12 3 1 1
```

Problems with MPI Programs

Common Problems in MPI Programs

- The usual
  - Wrong file paths
  - Bad logic
  - ...
- Deadlock
  - Problem with blocking communication
  - A sender never finished until

Deadlock Example

```
suppressMessages(library(pbdMPI))

if (comm.rank() > 0){
  reduce(comm.rank())
}

finalize()
```

Deadlock Example

```
suppressMessages(library(pbdMPI))

if (comm.rank() == 0){
  barrier()
}

finalize()
```

Deadlock Example

```
suppressMessages(library(pbdMPI))

if (comm.rank() == 0){
  comm.print(1)
}

finalize()
```

Deadlocks Behind the Scenes

Won't work

```
if (comm.rank() == 0){
  send(x, rank.dest=1)
  y = recv(rank.source=1)
} else if (comm.rank() == 1){
  send(x, rank.dest=0)
  y = recv(rank.source=0)
}
```

Works fine

```
if (comm.rank() == 0){
  send(x, rank.dest=1)
  y = recv(rank.source=1)
} else if (comm.rank() == 1){
  y = recv(rank.source=0)
  send(x, rank.dest=0)
}
```

Wrapup

Wrapup

- Task parallelism with MPI is straightforward
  - Assign tasks
  - Get data
  - Perform operation
  - Collect/write result(s)
- As always I/O is application dependent, hard
- Next time: data parallelism with MPI

Questions?