Lab 2

Introduction to Programming Laboratory

Goals

- MPI hello world
- Job submission
- Time measurement methods
- Task: π approximation
- Tutorial: MPI-IO

MPI hello world

Structure of an MPI program

```
#include <mpi.h>
int main(int argc, char** argv) {
        MPI_Init(&argc, &argv);
        // put your code here
        MPI_Finalize();
}
```

Which process am I?

- `int MPI_Comm_size( MPI_Comm comm, int *size );` tells the total number of process
- `int MPI_Comm_rank( MPI_Comm comm, int *rank );` tells the rank (id) of the calling process

Compilation

- To compile a MPI program, we use the MPI compiler wrapper.
- `mpicc -O3 -std=c99 source.c -o executable`
- `mpicxx -O3 -std=c++11 source.cc -o executable`
- In short, use `mpicc` for C and `mpicxx` for C++

What is a compiler wrapper? (informational)

- `mpicc` and `mpicxx` actually calls `gcc` and `g++`
- They just add some flags to the underlying compiler
- To use a different compiler, you can use `mpicc -cc=clang` or `mpicxx -cxx=clang++`

Running the code

- To run the code interactively, we use `srun`
- For example: `srun -n12 ./mpi-hello` means running `./mpi-hello` with 12 MPI processes.

MPI: walkthrough

Job submission

SLURM workload scheduler

On a cluster system, there are multiple users and multiple nodes. SLURM schedules jobs submitted by users across different nodes, so that the same resource is not used by two jobs at the same time (to ensure accuracy of performance-critical experiments), and also ensure the utilization of the cluseter.

srun: run a parallel job

Usage: `srun *[SLURM-OPTIONS] executable [EXECUTABLE-OPTIONS]*`

- `-n#` number of (MPI) processes.
- `-t*XX:YY*` job time limit, XX minutes YY seconds.
- `-J*name*` the name of the job.

sinfo: show cluster status

```
[ipl19-00@apollo31 ~]$ sinfo
PARTITION AVAIL  TIMELIMIT  NODES  STATE NODELIST
ipl*         up      15:00     16   idle apollo[33-47,50]
```

squeue: show queued jobs

```
[ipl19-00@apollo31 ~]$ squeue
  JOBID PARTITION     NAME     USER ST       TIME  NODES NODELIST(REASON)
1081413       ipl    job15 ipl19-00 PD       0:00      4 (QOSMaxJobsPerUserLimit)
1081414       ipl    job16 ipl19-00 PD       0:00      4 (QOSMaxJobsPerUserLimit)
1081415       ipl    job17 ipl19-00 PD       0:00      4 (QOSMaxJobsPerUserLimit)
1081416       ipl    job18 ipl19-00 PD       0:00      4 (QOSMaxJobsPerUserLimit)
1081412       ipl    job14 ipl19-00  R       0:07      4 apollo[37-40]
1081411       ipl    job13 ipl19-00  R       0:10      4 apollo[33-36]
```

sbatch: run jobs in the background (informational)

- Usage: `sbatch [SLURM-OPTIONS] job-script.sh`
- Job script looks like:

```
#!/bin/bash
srun ./mpi-hello
```

- Job output goes to `slurm-JOBID.out`
- Search online or ask TA if you're interested

scancel: cancel jobs

- `scancel JOBID` cancels a job with the associated JOBID
- `scancel -u ipl19-XX` cancels all job of the given username

Time measurement methods

What's wrong with this?

```
time srun -n12 ./my_program
```

The timing may include time used for resource allocation. Use `srun time ...` or `sbatch` with `time srun` in the job script instead.

What's wrong with this?

```
double start = time(NULL);
// run some stuff
double end = time(NULL);
```

`time()` has 1-second resolution.

What's wrong with this?

```
double start = clock();
// run some stuff
double end = clock();
```

`clock()` measures cpu clock time for the process, which means it will count 2x time when using 2 threads and will not include I/O time

What's wrong with this?

```
clock_gettime(CLOCK_REALTIME, &start);
// run some stuff
clock_gettime(CLOCK_REALTIME, &end);
```

`CLOCK_REALTIME` will be affected by NTP adjustments and DST changes. Use `CLOCK_MONOTONIC` instead.

What's wrong with this?

```
auto start = std::high_resolution_clock::now();
// run some stuff
auto end = std::high_resolution_clock::now();
```

`std::high_resolution_clock` may be affected by NTP adjustments and DST changes. Use `std::steady_clock` instead.

Summary: correct measurement methods

- `srun time ...`
- `sbatch` + `time srun`
- `MPI_Wtime()`
- `omp_get_wtime()`
- `clock_gettime(CLOCK_MONOTONIC, ...)`
- `std::steady_clock`

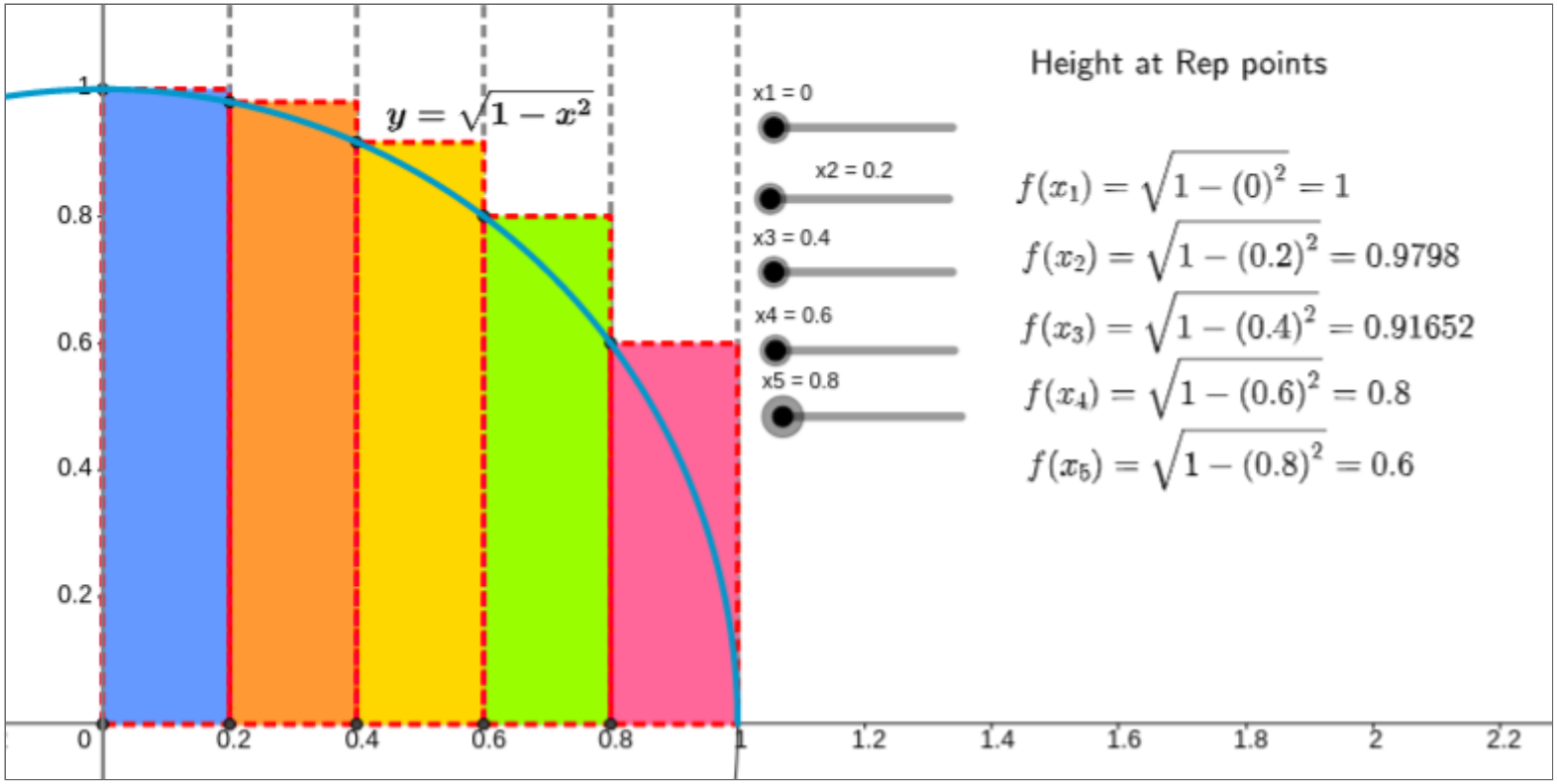
Task: π approximation

*You are required to demo to TA before leaving

Task description

Use the Left Riemann Sum to approximate the value of π.

$$4\sum_{i=0}^{k-1}\frac{\sqrt{1-(\frac{i}{k})^2}}{k}$$


$y = \sqrt{1 - x^2}$
x1 = 0
x2 = 0.2
x3 = 0.4
x4 = 0.6
x5 = 0.8
Height at Rep points
$f(x_1) = \sqrt{1 - (0)^2} = 1$
$f(x_2) = \sqrt{1 - (0.2)^2} = 0.9798$
$f(x_3) = \sqrt{1 - (0.4)^2} = 0.91652$
$f(x_4) = \sqrt{1 - (0.6)^2} = 0.8$
$f(x_5) = \sqrt{1 - (0.8)^2} = 0.6$
1
0.8
0.6
0.4
0.2
0
0.2
0.4
0.6
0.8
1
1.2
1.4
1.6
1.8
2
2.2

Requirements

- `srun -n# ./lab2 SAMPLES`
- # = number of processes, SAMPLES = number of slices
- Output your result with at least 6 digits in 1 line
- Name your executable `lab2`
- Demo with TA

Tutorial: MPI-IO

Functions

- `MPI_File_open`, `MPI_File_close`
- `MPI_File_read_at`, `MPI_File_write_at`

You can use `man` *`function-name`* such as `man` *`MPI_File_open`* to see their usage

Live coding