MPI Practice Lab

Parallel Programming
2025/10/03

MPI

- Message Passing Interface
- Processes communicate via MPI
- Compared to multithreading program (e.g. pthread, OpenMP), MPI allows launching multiple processes.
- MPI libraries:
  - Intel MPI
  - Open MPI
- https://mpitutorial.com/tutorials/mpi-hello-world/

MPI Structure

1. Initialization
2. Get process rank
3. Compute
4. Communicate
5. Finalize

MPI Structure

1. Initialization
2. Get process rank
3. Compute
4. Communicate
5. Finalize

```
#include <mpi.h>
#include <iostream>
int main(int argc, char **argv) {
    MPI_Init(&argc, &argv);
    return 0;
}
```

MPI Structure

1. Initialization
2. Get process rank

   World size: # of total processes

   Rank: [0,world_size)

3. Compute
4. Communicate
5. Finalize

```
#include <mpi.h>
#include <iostream>
int main(int argc, char **argv) {
    MPI_Init(&argc, &argv);
    // Get world size and rank
    int rank, world_size;
    MPI_Comm_rank(MPI_COMM_WORLD, &rank);
    MPI_Comm_size(MPI_COMM_WORLD, &world_size);
    return 0;
}
```

MPI Structure

1. Initialization
2. Get process rank
3. Compute
4. Communicate
5. Finalize

```cpp
#include <mpi.h>
#include <iostream>
int main(int argc, char **argv) {
    MPI_Init(&argc, &argv);
    // Get world size and rank
    int rank, world_size;
    MPI_Comm_rank(MPI_COMM_WORLD, &rank);
    MPI_Comm_size(MPI_COMM_WORLD, &world_size);
    if (rank < world_size / 2)
        std::cout << "1\n";
    else
        std::cout << "2\n";
    return 0;
}
```

MPI Structure

1. Initialization
2. Get process rank
3. Compute
4. Communicate

   Barrier

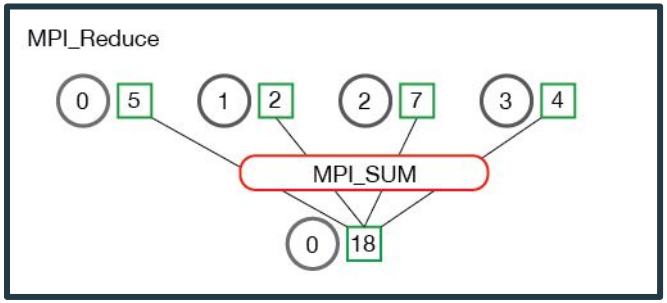
   Send, Recv, Reduce

5. Finalize

```
int MPI_Send(const void *buf, int count, MPI_Datatype datatype,
    int dest,int tag, MPI_Comm comm)
int MPI_Recv(void *buf, int count, MPI_Datatype datatype,
    int source, int tag, MPI_Comm comm, MPI_Status *status)
```

```
#include <mpi.h>
#include <iostream>
#define SEND_INT 0
int main(int argc, char **argv) {
    MPI_Init(&argc, &argv);
    int rank;
    int data[100];
    if (rank == 1)
        MPI_Send(data, 100, MPI_INT, 0,
            SEND_INT, MPI_COMM_WORLD);
    else if (rank == 0)
        MPI_Recv(data, 100, MPI_INT, 1,
            SEND_INT, MPI_COMM_WORLD, MPI_STATUS_IGNORE);
    return 0;
}
```

MPI Structure



4. Communicate

   Barrier

   Send, Recv, Reduce

5. Finalize

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

```
#include <mpi.h>
#include <iostream>
int main(int argc, char **argv) {
    MPI_Init(&argc, &argv);
    int rank;
    int count = rand();
    int sum; // rank 0
    MPI_Reduce(&count, &sum, 1,
           MPI_INT, MPI_SUM, 0,
           MPI_COMM_WORLD);
    return 0;
}
```

MPI Structure

1. Initialization
2. Get process rank
3. Compute
4. Communicate
5. Finalize

```
#include <mpi.h>
#include <iostream>
int main(int argc, char **argv) {
    MPI_Init(&argc, &argv);
    // Get world size and rank
    int rank, world_size;
    MPI_Comm_rank(MPI_COMM_WORLD, &rank);
    MPI_Comm_size(MPI_COMM_WORLD, &world_size);
    MPI_Finalize();
    return 0;
}
```

System Environment

- module load gcc/13
- module load openmpi

Compile

- Compile a C program with MPI

```
mpicc path/to/source.c
```

- Compile a C++ program with MPI

```
mpicxx path/to/source.cpp
```

- It accepts compiler flags too, such as `-O3` `-g`

Running MPI Programs with Slurm

Run 5 MPI processes:

```
srun -n5 -A ACD114118 path/to/program
```

Run 5 MPI processes, giving each process 4 CPUs:

```
srun -n5 -c4 -A ACD114118 path/to/program
```

Run 5 MPI processes, prefixing the output with the process rank:

```
srun -n5 -l -A ACD114118 path/to/program
```

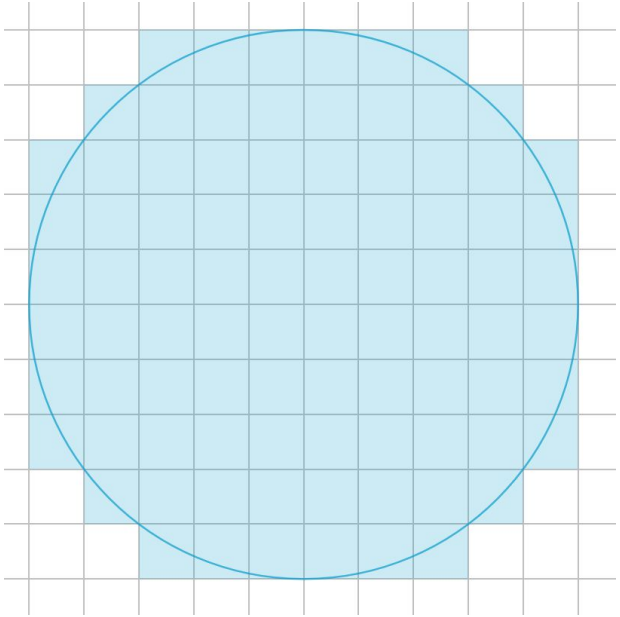
(Useful for debugging)

Practice Lab (No submission required):
Calculate number of pixels of a circle on a 2D monitor

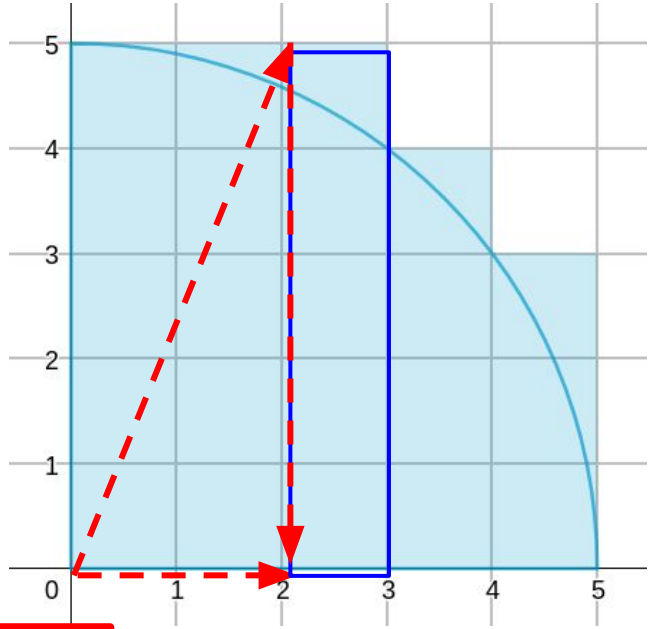
Suppose we want to draw a filled circle of radius r on a 2D monitor, how many pixels will be filled?

We fill a pixel when any part of the circle overlaps with the pixel. We also assume that the circle center is at the boundary of 4 pixels.

For example 88 pixels are filled when r=5.



Example: radius = 5



To calculate, we count the number of pixels of a quarter circle, and multiply the result by 4.

$$\text{pixels}(r) = 4 \times \sum_{x=0}^{r-1} \left\lceil \sqrt{r^2 - x^2} \right\rceil$$

$$\begin{aligned}
\text{pixels}(5) &= 4\left(\left\lceil \sqrt{25-0} \right\rceil + \left\lceil \sqrt{25-1} \right\rceil + \left\lceil \sqrt{25-4} \right\rceil + \left\lceil \sqrt{25-9} \right\rceil + \left\lceil \sqrt{25-16} \right\rceil \right) \\
&= 4(5+5+5+4+3) \\
&= 88
\end{aligned}$$

I/O Format

Input format (command line):

```
srun -n $nproc ./lab2 r k
```

Output format (print to stdout):

```
pixels % k
```

- nproc: # of MPI processes
- r: the radius of the circle, integer
- k: integer
- pixels: # of pixels needed to draw the circle

Requirements

- Parallelize the calculation using MPI
- Sequential code is located at `/work/b10502076/pp25/mpi_pr_lab`
- Your program should be at least **(n/2) times faster** than the sequential version when running with **n processes**. For example, when running with 12 processes, your execution time should not exceed 1/6 of the sequential code.

Thank you