Lab 1: Introduction to MPI

EEL 6763 – Spring 2023

Part 1: Matrix Multiply

Run the given mat_mult.c as is (without changing anything) **with 8 ranks** to obtain the results (as a baseline).

Then modify the mat_mult.c to do the following; and again run it **using 8 ranks**:

Each worker rank will:

- Determine two times: (using the now() and tdiff() functions)
  - wcalcTime: the actual time spent on performing its calculation of matrix C.
- Send its wcalTime to the master rank.

The master rank will:

- Determine time_spent: (time from sending the data to workers until the master have all the results). By the way, this is already in the original mat_mult.c.
- Print (at the end):
  - wcalcTime for each worker rank
  - maximum wcalcTime
  - sum of the wcalcTime for all worker ranks
  - time_spent (in the master rank, again already in the original code)

Part 2: Monte-Carlo Integration and MPI

One method of numerically estimating integrals is by using Monte-Carlo simulation. Consider the following integral g(a,b) and its estimate h(x):

$$g(a,b) = \int_a^b f(x)dx \qquad h(x) = \frac{(b-a)}{N}\sum_{i=1}^{N} f(x_i)$$

The numerical solution to g(*a*,*b*) can be estimated using a uniform random variable *x* that is evenly distributed over [*a*, *b*] (i.e., between a and b). The estimate h(*x*) will converge to the correct solution as the number of samples *N* grows. Since each sample is independent, the calculation can be easily parallelized (embarrassingly parallel). Using this method, write a short MPI program that will use N samples to calculate:

$$\int_a^b \frac{8\sqrt{2\pi}}{e^{(2x)^2}}\,dx$$

For this problem you must provide code for three functions, init_rand_seed (), estimate_g(…) and collect_results(…), which will be used by the following main(…) function:

Important note: Use the main function as it is defined below. It cannot be changed (e.g., you should not be doing any calculation, send/recv, or printing in the main function).

```
int main(int argc, char **argv)
{
  double result = 0.0;
  MPI_Init(&argc, &argv);
```

```
  float lower_bound  = atof(argv[1]);
  float upper_bound  = atof(argv[2]);
  long long int N = atof(argv[3]);

  init_rand_seed(); // using srand()
  result = estimate_g(lower_bound, upper_bound, N);
  collect_results(&result);

  MPI_Finalize();
  return 0;
}
```

The following function prototypes must be used for your functions:

- **`double estimate_g(double lower_bound, double upper_bound, long long int N);`**
- **`void collect_results(double *result);`**

Other specifications:

- The total number of samples to calculate **N** (to be split among all MPI nodes), as well as the bounds of the integral **a** and **b**, will be provided through command-line arguments.
- Every MPI rank will generate its own random numbers. Ensure that each rank uses a different starting seed for its random number generator - this should be the only thing that occurs in , init_rand_seed ().

  Then in the estimate_g function, you should use the rand function to generate the random numbers. Hint: While debugging print out the random numbers to verify that they are indeed random.
- Each rank should return a single value, which should be combined (i.e., sum) in the master rank in order to compute the final integral.
- Use only the following functions:
  - `MPI_Init`
  - `MPI_Comm_rank`
  - `MPI_Comm_Size`
  - `MPI_Send`
  - `MPI_Recv`
  - `MPI_Finalize`

Part 3: MPI_Reduce

Rewrite the previous function (from Part 2) to use MPI_Reduce instead of MPI_Send and MPI_Recv.

Lab 1: Introduction to MPI

EEL 6763 – Spring 2023

SUBMISSION INSTRUCTIONS

- Make sure your name is at the top of every file.

You are to submit 2 files on Canvas:

(1) Create a directory named **Lab1**. Use the following structure and zip the entire directory and submit the zip file on Canvas using the following name: lastNamefirstNameLab1.zip

Lab1/

partA/

.c file

batch script

output file

partB/

.c file

batch script

7 output files: a = -10 ; b = 10 (keep a and b constant);

- Keep N = 100000, vary R (number of ranks) = 4, 8, 16, 32
- Keep R = 32, vary N= 100, 1000, (10000), 100000
- Name each output file appropriately: e.g., SendRecvR32N100, SendRecvR8N100000)
- Also, to maximize your credit (especially partial credit), use printf statements like they were used in the mat_mult code (e.g., number of elements per rank, partial results sent back by each rank, etc.)

partC/

.c file

batch script

7 output files: same specifications as Part B.

- Name each output file appropriately: e.g., ReduceR32N100, ReduceR8N10000).
- Also, to maximize your credit (especially partial credit), use printf statements like they were used in the mat_mult code (e.g., number of elements per rank, partial results sent back by each rank, etc.)

(2) A pdf file using the following name: lastNamefirstNameLab1.pdf

- Any information that you want to give me (e.g., what worked, what didn't, etc.)
- **Part 1:** What the master rank printed (at the end).: wcalcTime for each worker rank, the maximum wcalcTime, the sum of the wcalcTime for all worker ranks, and the total time_spent (time from sending the data to workers until the master have all the results).
- **Part 2:** Two tables
  - Table 1 (for N = 100000) - Row labels: R (# of ranks) = 4, 8, 16, 32; Column labels: estimated result, time taken
  - Table 2 (for R = 32) - Row labels: N = 100, 1000, … , 100000 ; Column labels: estimated result, time taken
- **Part 3:** Two tables (same as Part 2)