EEL6763 Parallel Computer Architecture, Spring 2017
Homework #2 Assignment
Due: February 10th (Friday) by 11AM

Summary

This homework contains three sections with various requirements for each section. Please read all material carefully.

Submission Guidelines

- Submit a single PDF containing copies of all code and answers to the discussion questions.
- Sub-divide the document clearly delineating and labeling the three parts of assignment.
- Include source code by copy-pasting the complete source code from your .c files.
- Please ensure source code (in .c files and in the PDF) is consistently indented and commented.
- All code and answers MUST be your own original work and NOT copied from the Internet or elsewhere.

Part 1: 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*]. 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. Write a short MPI program that will use many samples to calculate:

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

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

```
int main(int argc, char **argv)
{
 MPI_Init(&argc, &argv);

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

```
 initialize_data();

 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);

Provide 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* through command-line arguments.

Every MPI node will generate its own random numbers. Ensure that each node uses a different starting seed for its random number generator. This should be the only thing that occurs in initialize_data(…).

Each node should return a single value, which should be combined on the root node 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

Your code should be submitted as a text file named hw2_part1_yourlastname.c (replacing <yourlastname> as appropriate) and copied appropriately into the PDF submission.

Part 2: Monte-Carlo Integration and MPI (using reduce)

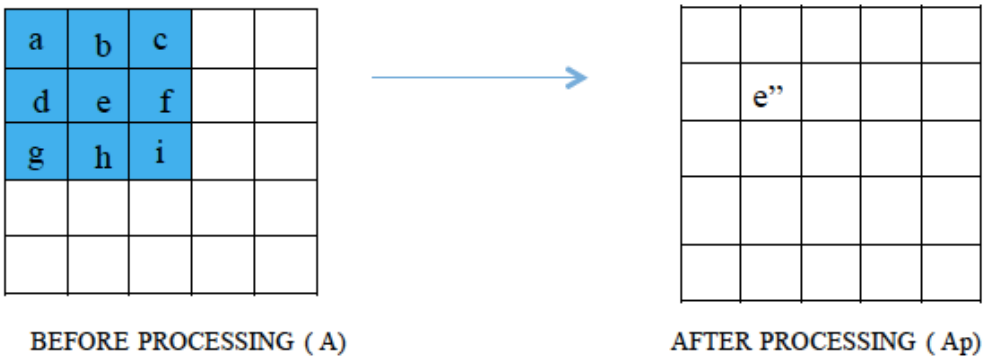
Rewrite the previous functions used in Part 1 to now use MPI_Reduce instead of MPI_Send and MPI_Recv. Submit this version of your code as a text file named hw2_part2_yourlastname.c (replacing <yourlastname> as appropriate) and copied appropriately into the PDF submission.

Comment on the performance of both programs from Parts 1 and 2 in the PDF file. Execute each version of the code with several different numbers of samples, *N*, and comment on the changes observed (e.g. answer received, runtime, etc.). Additionally, vary the system size (number of MPI nodes, or cores) from 1 to 32 and comment on any changes observed. This type of code is sometimes called embarrassingly parallel. Why is that?

Part 3: Matrix Mask Operation

A Mask operation (neighborhood weighted-averaging filter) is commonly used for image processing. The basic concept is to recalculate the value of each pixel on the basis of the adjacent

pixel values and value of the current pixel. For our purposes consider a grayscale image as a matrix with each pixel having a value of 0 to 10000. A mask operator is defined over a matrix as shown in the figure. The operator takes the average of adjacent matrix values defined as:
e" = (a+b+c+d+2e+f+g+h+i)/10.



BEFORE PROCESSING (A)

AFTER PROCESSING (Ap)

Perform the following:

- Implement the mask operation on a matrix using MPI.
- The operation is performed on a square matrix of size NxN.
- The matrix must be initialized in one of the nodes using the rand() function.
- Use Scatterv | Gatherv along with any other MPI functions required.
- To keep the program simple, it is not required to process the first and last rows and columns.
- All MPI nodes must perform the mask operation.

You will provide code for three functions, namely, scatter_data, mask_operation, and gather_results with the following prototypes:

- void scatter_data(int *A, int N);
- void mask_operation(int *A, int N, int *Ap);
- void gather_results(int *Ap, int N);
- Note: Ap is the processed matrix

Use the following pseudocode as a starting point for developing your main function. N is the size of the matrix NxN

```
int main(int argc, char **argv)
{
 MPI_Init(&argc, &argv);
 int N = atof(argv[1]);
 initialize_data(&A, &Ap);
 scatter_data(A, N);
 mask_operation(A, N, Ap);
 gather_results(Ap, N);
 MPI_Finalize();
 return 0;
}
```

Note: Include any other features required to initialize MPI, etc.

Submit your code as a text file named hw2_part3_yourlastname.c (replacing <yourlastname> as appropriate) and copied appropriately into the PDF submission.

Quantitatively analyze the performance and scalability of your program. How does performance scale with system size (i.e. number of MPI nodes, or cores) from 1 to 32? How does the performance scale with problem size? Provide your comments in the PDF file.