# Solution to Homework 2

Meshach Aruoriwo Ogbor,
Md Rasel

October 25, 2023

## 1 Exercise 2.1 - Collective Communication

```c
#include <stdio.h>
#include <stdlib.h>
#include <math.h>
#include <float.h>
#include <mpi.h>

int main(int argc, char* argv[]) {
    int rank, numprocs;
    double start_time, end_time; // Variables to store start and end times

    double local_sum;
    double local_sum_sq; // the sum of squares of the numbers
    double local_min;
    double local_max;
    int num_elements_per_file = 100;
    int num_files = 64;


    MPI_Init(&argc, &argv);

    MPI_Comm_rank(MPI_COMM_WORLD, &rank);
    MPI_Comm_size(MPI_COMM_WORLD, &numprocs);


    start_time = MPI_Wtime(); // Record the start time

    // Process data files based on rank
    for (int i = rank; i < num_files; i += numprocs) {
        char filename[20];
        snprintf(filename, sizeof(filename), "data_%d.dat", i);
        FILE *file = fopen(filename, "r");
        if (file == NULL) {
            fprintf(stderr, "Error: Could not open file %s\n", filename);
            MPI_Abort(MPI_COMM_WORLD, -1);
        }

        // Process data from the file
        for (int j = 0; j < num_elements_per_file; ++j) {
            double num;
            if (fscanf(file, "%lf", &num) != 1) {
                fprintf(stderr, "Error: Invalid data format in file %s\n", filename);
                fclose(file);
                MPI_Abort(MPI_COMM_WORLD, -1);
            }
            // Update local calculations
            local_sum += num;
            local_sum_sq += num * num;
            if (num < local_min) local_min = num;
            if (num > local_max) local_max = num;
        }
        fclose(file);
    }

    // Perform reduction to calculate global statistics
    double global_sum, global_sum_sq, global_min, global_max;
    MPI_Reduce(&local_sum, &global_sum, 1, MPI_DOUBLE, MPI_SUM, 0, MPI_COMM_WORLD);
    MPI_Reduce(&local_sum_sq, &global_sum_sq, 1,
        MPI_DOUBLE, MPI_SUM, 0, MPI_COMM_WORLD);
    MPI_Reduce(&local_min, &global_min, 1, MPI_DOUBLE, MPI_MIN, 0, MPI_COMM_WORLD);
    MPI_Reduce(&local_max, &global_max, 1, MPI_DOUBLE, MPI_MAX, 0, MPI_COMM_WORLD);

    // Calculate mean and variance
    double mean = global_sum / (num_elements_per_file * num_files);
    double variance = (global_sum_sq / (num_elements_per_file * num_files)) - (mean * mean);
    // Print results
    if (rank == 0) {
        printf("Mean: %.2lf\n", mean);
        printf("Variance: %.2lf\n", variance);
        printf("Minimum: %.2lf\n", global_min);
        printf("Maximum: %.2lf\n", global_max);
    }
    MPI_Barrier(MPI_COMM_WORLD);
    end_time = MPI_Wtime(); // Record the end time
    // Print the execution time on the root process (rank 0)
    if (rank == 0) {
        double elapsed_time = end_time - start_time;
        printf("Execution Time: %.6f seconds\n", elapsed_time);
    }

    MPI_Finalize();
    return 0;
}
```

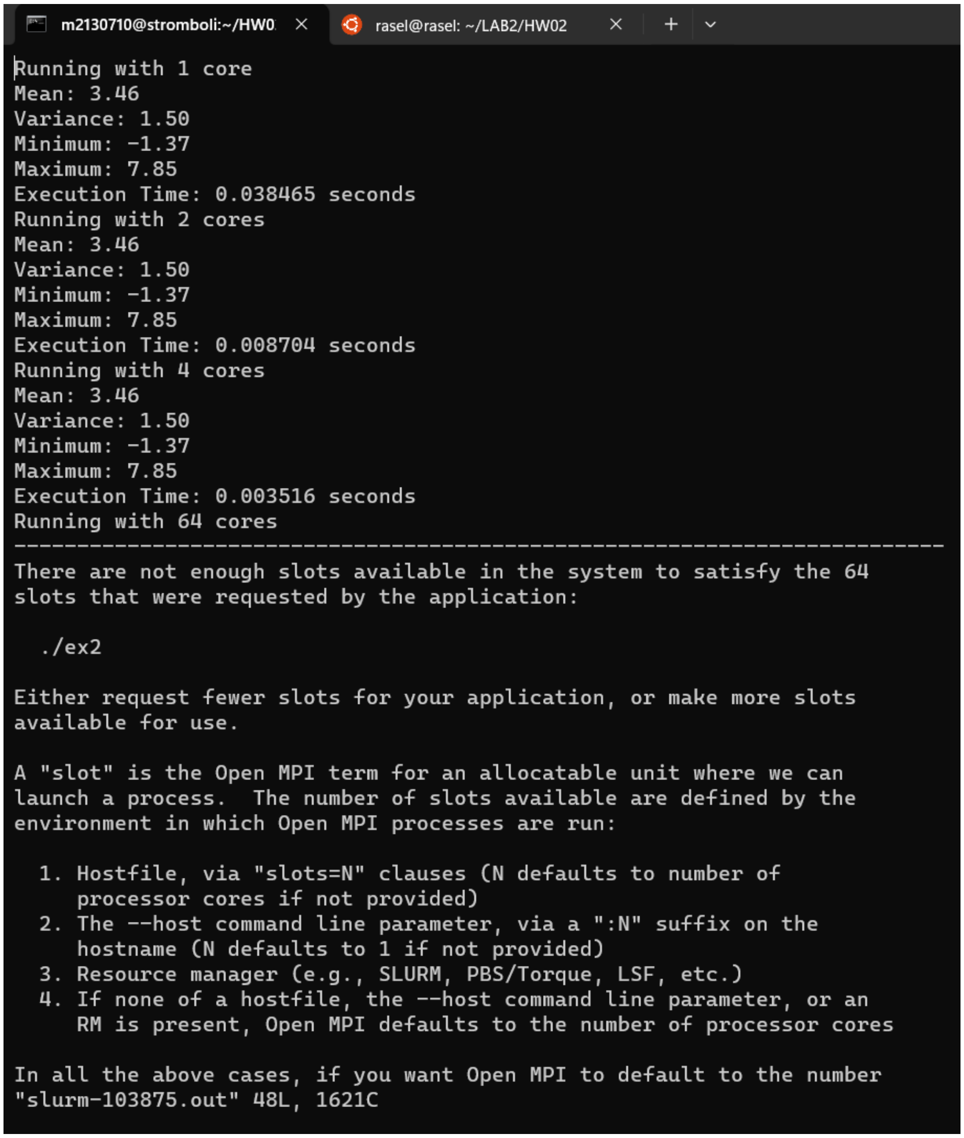
## 2 Submit Script

```bash
#!/bin/bash
#SBATCH --partition=compute2011
#SBATCH --exclusive

module load mpi/openmpi/4.1.0
```

```

# Compile the MPI program
mpicc –o ex2 ex2.c –lm

# Run the MPI program with 1 core
echo ”Running with 1 core”
mpirun –np 1 ./ex2

# Run the MPI program with 2 cores
echo ”Running with 2 cores”
mpirun –np 2 ./ex2

# Run the MPI program with 4 cores
echo ”Running with 4 cores”
mpirun –np 4 ./ex2

# Run the MPI program with 64 cores
echo ”Running with 64 cores”
mpirun –np 64 ./ex2
```

## 3 Results

The following results obtained are on the next page:



```
Running with 1 core
Mean: 3.46
Variance: 1.50
Minimum: -1.37
Maximum: 7.85
Execution Time: 0.038465 seconds
Running with 2 cores
Mean: 3.46
Variance: 1.50
Minimum: -1.37
Maximum: 7.85
Execution Time: 0.008704 seconds
Running with 4 cores
Mean: 3.46
Variance: 1.50
Minimum: -1.37
Maximum: 7.85
Execution Time: 0.003516 seconds
Running with 64 cores
--------------------------------------------------------------------------
There are not enough slots available in the system to satisfy the 64
slots that were requested by the application:

  ./ex2

Either request fewer slots for your application, or make more slots
available for use.

A "slot" is the Open MPI term for an allocatable unit where we can
launch a process.  The number of slots available are defined by the
environment in which Open MPI processes are run:

  1. Hostfile, via "slots=N" clauses (N defaults to number of
     processor cores if not provided)
  2. The --host command line parameter, via a ":N" suffix on the
     hostname (N defaults to 1 if not provided)
  3. Resource manager (e.g., SLURM, PBS/Torque, LSF, etc.)
  4. If none of a hostfile, the --host command line parameter, or an
     RM is present, Open MPI defaults to the number of processor cores

In all the above cases, if you want Open MPI to default to the number
"slurm-103875.out" 48L, 1621C
```

Figure 1