ELEC 873
Assignment 2

Anthony Sicoie (20214793)

October 22, 2025

Preamble

For each question code was developed locally and tested on both my machine (Ryzen 9 7950x, 64 GB Ram) as well as the KNL server. This is the Makefile template that all questions also followed, with appropriate changes based on the question requirements for each one; where questions only required a single implementation (only OpenMP etc.), only that implementation was accounted for in. Furthermore, the Makefile was also used to run the code in all cases other than those utilizing bash scripts; in those cases it is mentioned explicitly. Comments are provided throughout the Code Blocks for any noteworthy or special conditions I felt were worth highlighting. As such discussion of code within the report is minimal, but rather the comments are intended to describe how the code functions.

Q1:

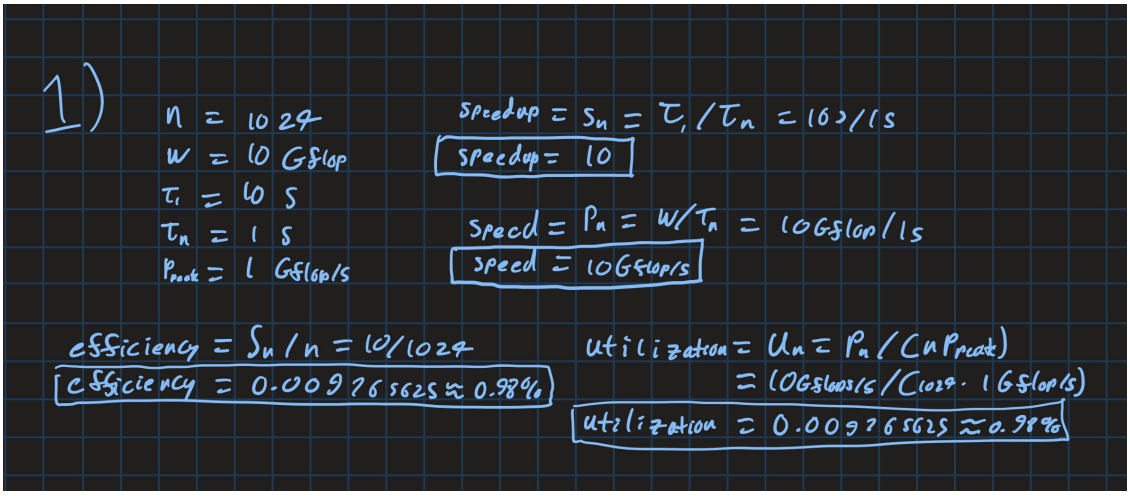

Figure 1: Math for Question 1

Q2:

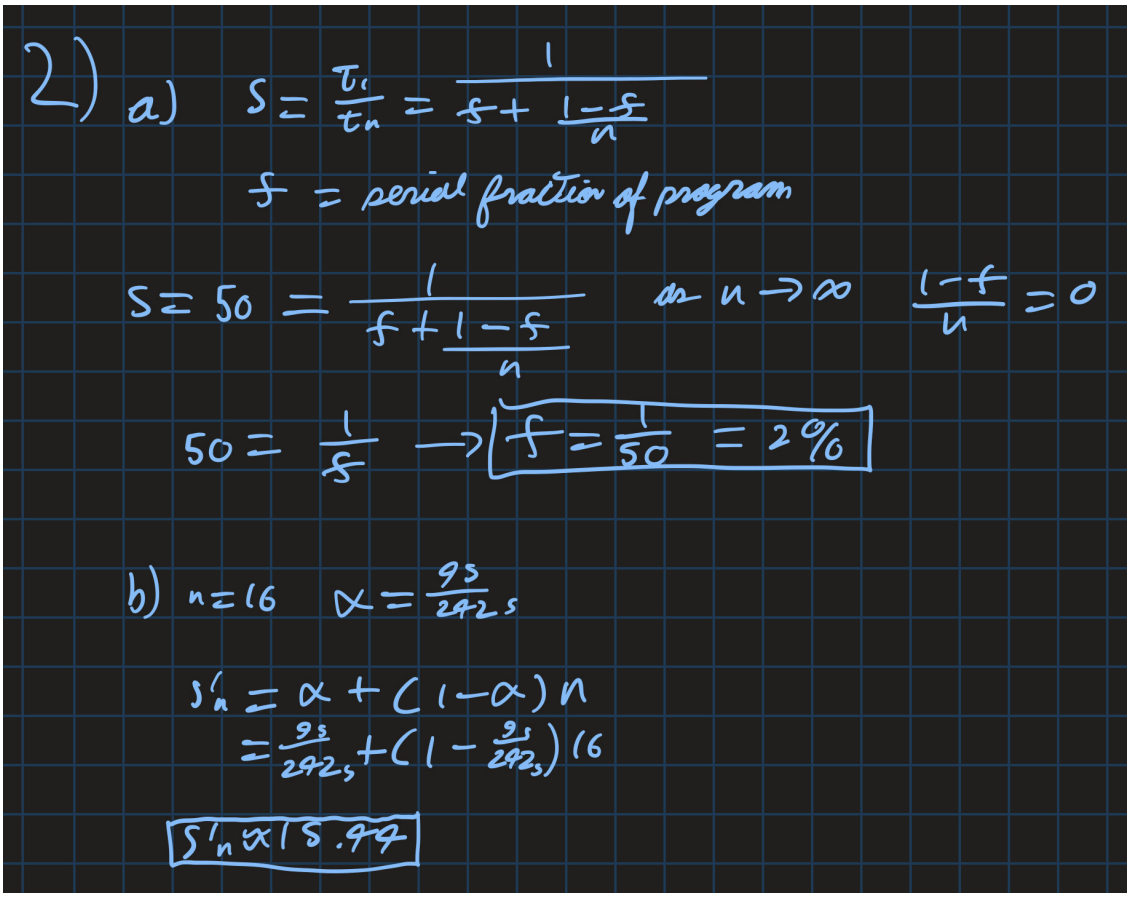

Figure 2: Math for Question 2

Q3:

Q4:

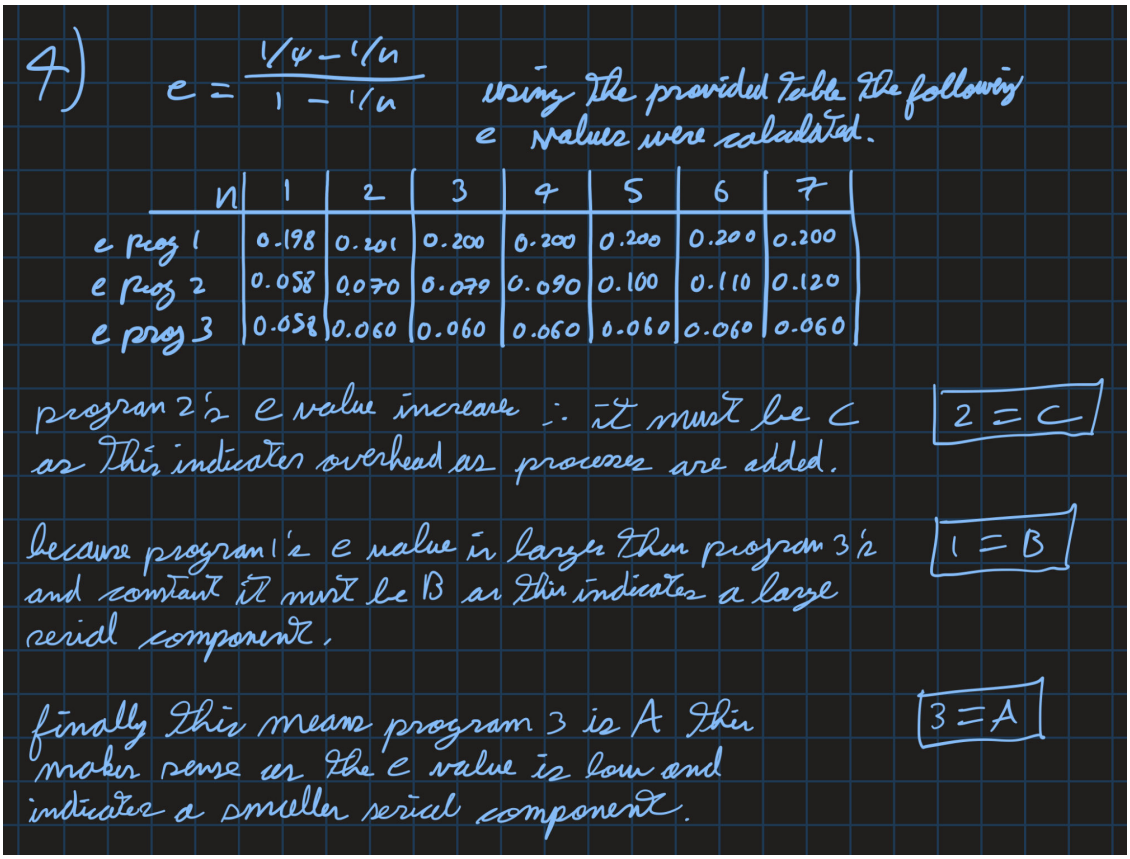

Figure 3: Math for Question 4

Q5:

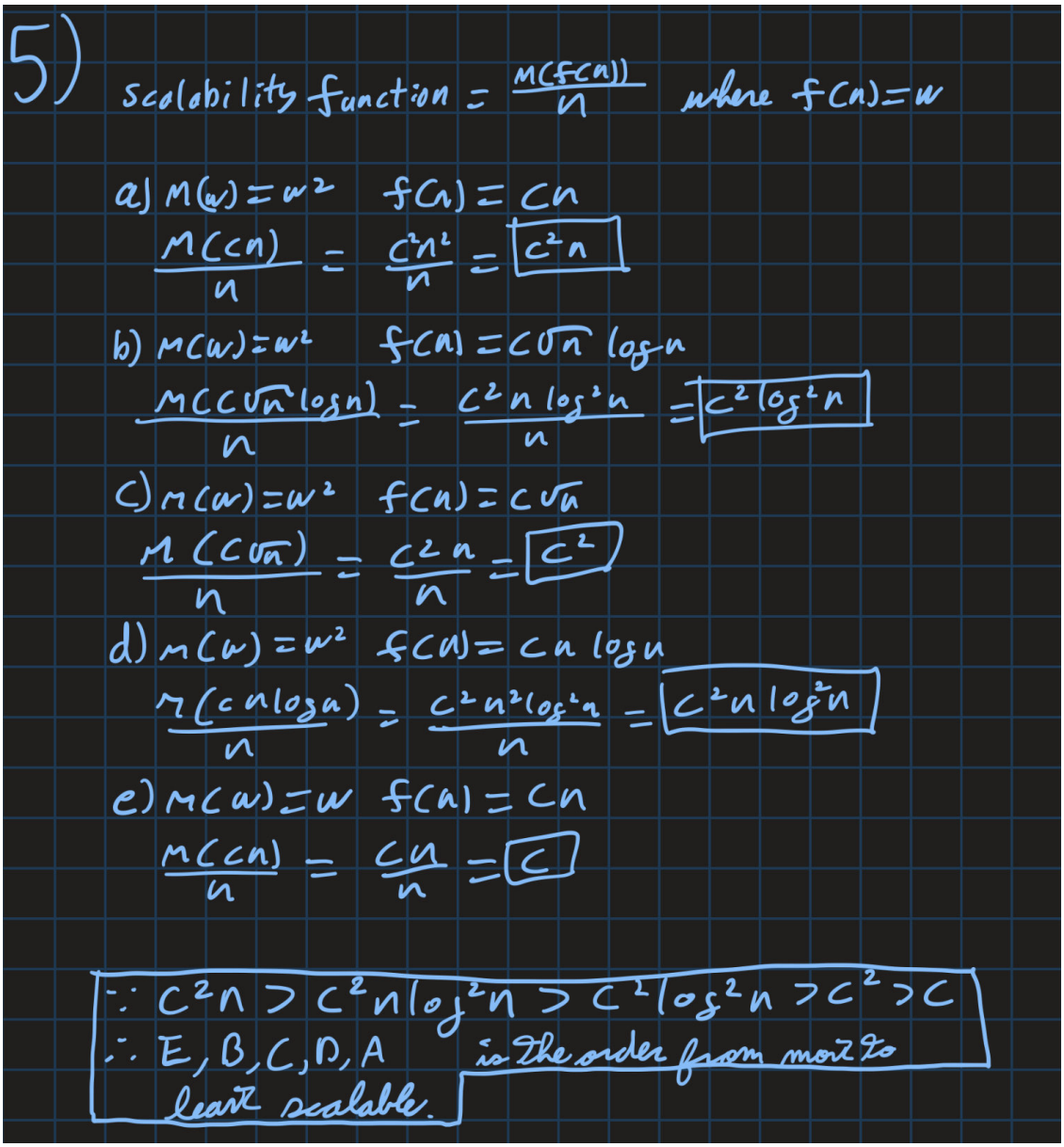

Figure 4: Math for Question 5

Q6:

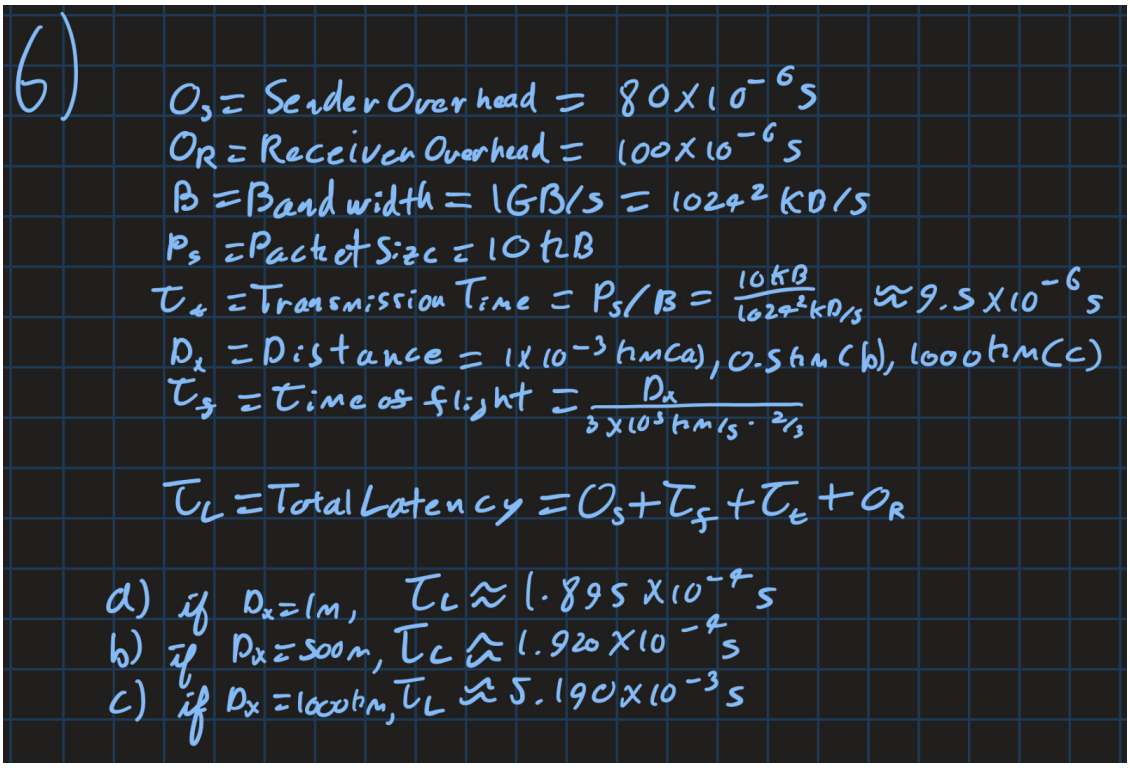

Figure 5: Math for Question 6

Q7:

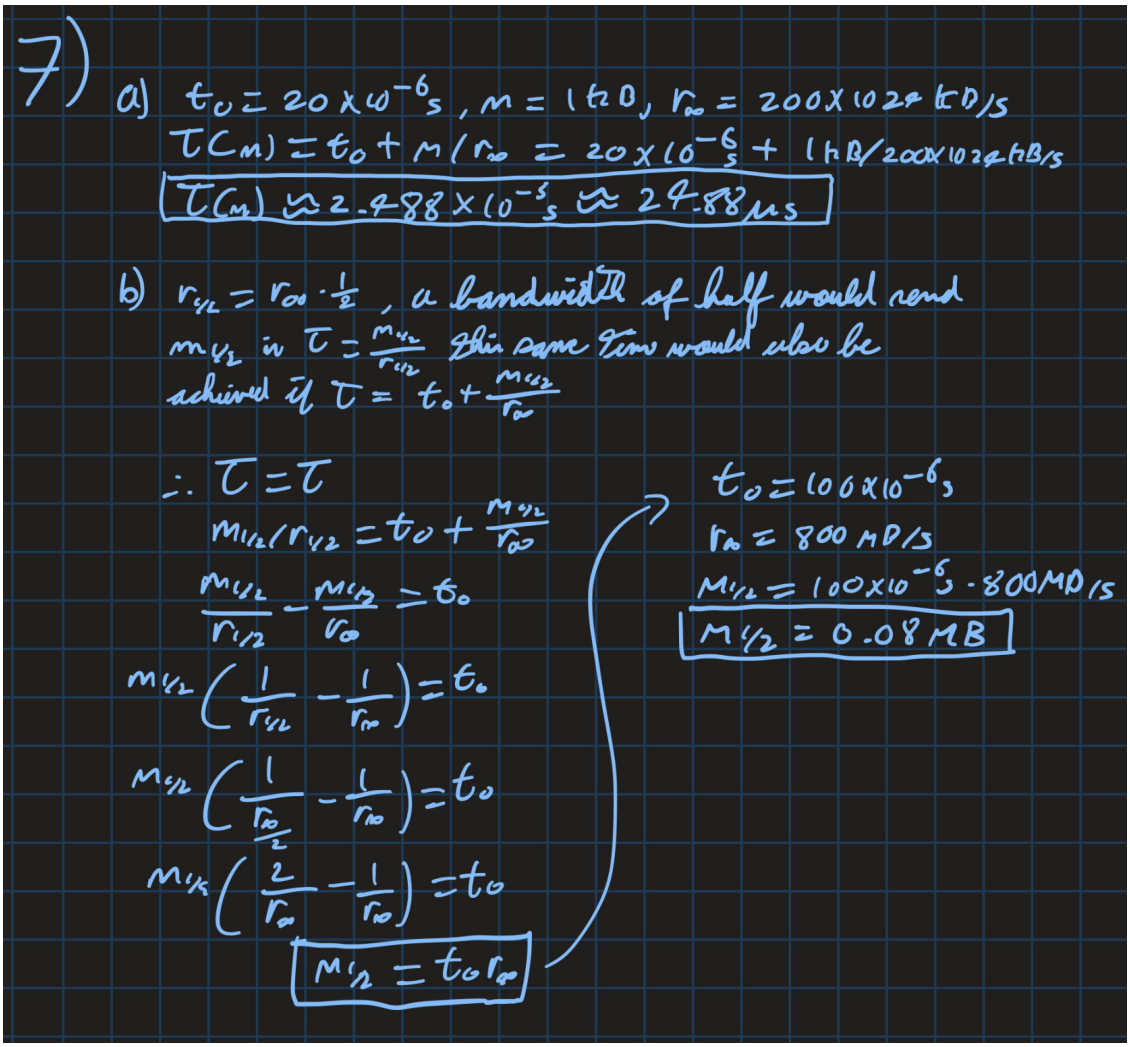

Figure 6: Math for Question 7

Q8:

Q9:

Code Block 1: MPI code for question 9

```c
// Anthony Sicoie (20214793)

#include <mpi.h>
#include <stdio.h>
#include <stdlib.h>

#define N_RUNS 1000

int main(int argc, char *argv[]) {
  int rank, size;
  MPI_Status status;
  double total_time = 0.0;

  MPI_Init(&argc, &argv);
  MPI_Comm_rank(MPI_COMM_WORLD, &rank);
  MPI_Comm_size(MPI_COMM_WORLD, &size);

  // Allow user to specify data coutn (for use in performance testing script)
  if (argc < 2) {
    if (rank == 0) {
      fprintf(stderr, "Usage: %s <array_size>\n", argv[0]);
    }
    MPI_Finalize();
    return 1;
  }

  int DATA_COUNT = atoi(argv[1]);

  char *data;
  data = malloc(DATA_COUNT * sizeof(char));

  if (!rank) {
    for (int i = 0; i < DATA_COUNT; i++) {
      data[i] = (char)i;
    }
  }

  // Loop for benchmarking
  for (int run = 0; run < N_RUNS; run++) {

    double offset = MPI_Wtime();
    double start = MPI_Wtime();

    if (!rank) {
      MPI_Send(data, DATA_COUNT, MPI_CHAR, 1, 0, MPI_COMM_WORLD);
      MPI_Recv(data, DATA_COUNT, MPI_CHAR, 1, 0, MPI_COMM_WORLD, &status);
    }

    if (rank) {
      MPI_Recv(data, DATA_COUNT, MPI_CHAR, 0, 0, MPI_COMM_WORLD, &status);
      MPI_Send(data, DATA_COUNT, MPI_CHAR, 0, 0, MPI_COMM_WORLD);
    }

    double end = MPI_Wtime();
    double timer_overhead = start - offset;
    total_time += (end - start - timer_overhead);
  }

  free(data);

```

```
61    if (!rank) {
62
63      double avg_time = total_time / N_RUNS;
64      double avg_latency = avg_time / 2.0;
65
66      double bandwidth = (DATA_COUNT / avg_latency) / (1024.0 * 1024.0);
67
68      FILE *f = fopen("results.csv", "a");
69      if (f == NULL) {
70        fprintf(stderr, "Error opening results.csv\n");
71        MPI_Abort(MPI_COMM_WORLD, 1);
72      }
73      fprintf(f, "%d,%d,%.9f,%.9f,%.6f\n", size, DATA_COUNT, avg_time,
74              avg_latency, bandwidth);
75      fclose(f);
76
77      printf("Processes=%d Arraysize=%d AvgTime=%.9f sec Latency=%.9f sec "
78             "Latency=%.6f MB/s\n",
79             size, DATA_COUNT, avg_time, avg_latency, bandwidth);
80    }
81
82    MPI_Finalize();
83
84    return 0;
85  }
```

Q10:

Code Block 2: MPI code for question 10

```c
// Anthony Sicoie (20214793)

#include <mpi.h>
#include <stdio.h>
#include <stdlib.h>

#define N_RUNS 1000

int main(int argc, char *argv[]) {
  int rank, size;
  MPI_Status status;
  int DATA_COUNT;
  double total_time = 0.0;

  MPI_Init(&argc, &argv);
  MPI_Comm_rank(MPI_COMM_WORLD, &rank);
  MPI_Comm_size(MPI_COMM_WORLD, &size);

  // Allow user to specify data coutn (for use in performance testing script)
  if (argc < 2) {
    if (rank == 0) {
      fprintf(stderr, "Usage: %s <array_size>\n", argv[0]);
    }
    MPI_Finalize();
    return 1;
  }

  DATA_COUNT = atoi(argv[1]);

  char *data;

  data = malloc(DATA_COUNT * sizeof(char));

  if (!rank) {
    for (int i = 0; i < DATA_COUNT; i++) {
      data[i] = (char)i;
    }
  }

  for (int run = 0; run < N_RUNS; run++) {
    double offset = MPI_Wtime();
    double start = MPI_Wtime();

    MPI_Bcast(data, DATA_COUNT, MPI_CHAR, 0, MPI_COMM_WORLD);

    double end = MPI_Wtime();

    double timer_overhead = start - offset;
    total_time += (end - start - timer_overhead);
  }

  double avg_time = total_time / N_RUNS;

  if (!rank) {
    FILE *f = fopen("results.csv", "a");
    if (f == NULL) {
      fprintf(stderr, "Error opening results.csv\n");
      MPI_Abort(MPI_COMM_WORLD, 1);
    }
    fprintf(f, "%d,%d,%.9f\n", size, DATA_COUNT, avg_time);
```

```
61      fclose(f);
62
63      printf("Processes=%d Arraysize=%d AvgTime=%.9f sec\n", size, DATA_COUNT,
64              avg_time);
65      // printf("sizeof(int) %lu\n", sizeof(int));
66    }
67
68    MPI_Finalize();
69
70    return 0;
71  }
```

Appendix