

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:

1)

$$\begin{aligned} n &= 1024 \\ W &= 10 \text{ Gflop} \\ T_i &= 10 \text{ s} \\ T_n &= 1 \text{ s} \\ P_{\text{peak}} &= 1 \text{ Gflop/s} \end{aligned}$$
$$\begin{aligned} \text{Speedup} &= S_n = T_i / T_n = 10 / 1 \text{ s} \\ \boxed{\text{Speedup} &= 10} \end{aligned}$$
$$\begin{aligned} \text{Speed} &= P_n = W / T_n = 10 \text{ Gflop} / 1 \text{ s} \\ \boxed{\text{Speed} &= 10 \text{ Gflops}} \end{aligned}$$
$$\begin{aligned} \text{efficiency} &= S_n / n = 10 / 1024 \\ \boxed{\text{efficiency} &= 0.009765625 \approx 0.98\%} \end{aligned}$$
$$\begin{aligned} \text{utilization} &= U_n = P_n / (n P_{\text{peak}}) \\ &= 10 \text{ Gflops} / (1024 \cdot 1 \text{ Gflop/s}) \\ \boxed{\text{utilization} &= 0.009765625 \approx 0.98\%} \end{aligned}$$

Figure 1: Math for Question 1

Q2:

2) a) $S = \frac{\tau_i}{\tau_n} = \frac{1}{f + \frac{1-f}{n}}$
 $f = \text{serial fraction of program}$

$S = 50 = \frac{1}{f + \frac{1-f}{n}}$ as $n \rightarrow \infty$ $\frac{1-f}{n} = 0$

$50 = \frac{1}{f} \rightarrow \boxed{f = \frac{1}{50} = 2\%}$

b) $n = 16$ $\alpha = \frac{95}{292.5}$

$S'_n = \alpha + (1-\alpha)n$
 $= \frac{95}{292.5} + (1 - \frac{95}{292.5})16$

$\boxed{S'_n \approx 15.99}$

Figure 2: Math for Question 2

Q3:

Q4:

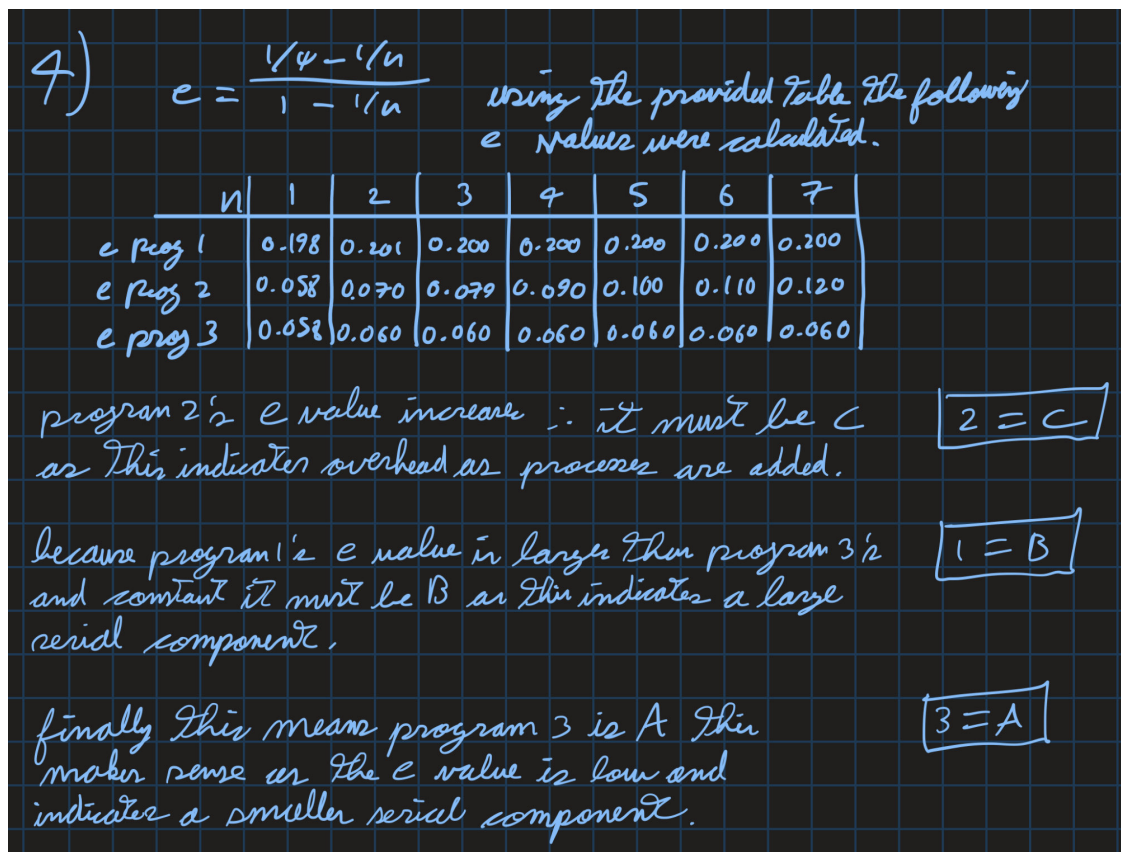


Figure 3: Math for Question 4

Q5:

5) scalability function = $\frac{M(f(n))}{n}$ where $f(n)=w$

a) $M(w)=w^2$ $f(n)=cn$
 $\frac{M(cn)}{n} = \frac{c^2n^2}{n} = \boxed{c^2n}$

b) $M(w)=w^2$ $f(n)=c\sqrt{n} \log n$
 $\frac{M(c\sqrt{n} \log n)}{n} = \frac{c^2 n \log^2 n}{n} = \boxed{c^2 \log^2 n}$

c) $M(w)=w^2$ $f(n)=c\sqrt{n}$
 $\frac{M(c\sqrt{n})}{n} = \frac{c^2 n}{n} = \boxed{c^2}$

d) $M(w)=w^2$ $f(n)=cn \log n$
 $\frac{M(cn \log n)}{n} = \frac{c^2 n^2 \log^2 n}{n} = \boxed{c^2 n \log^2 n}$

e) $M(w)=w$ $f(n)=cn$
 $\frac{M(cn)}{n} = \frac{cn}{n} = \boxed{c}$

$\therefore c^2n > c^2n \log^2 n > c^2 \log^2 n > c^2 > c$
 $\therefore E, B, C, D, A$ is the order from most to least scalable.

Figure 4: Math for Question 5

Q6:

6)

$$O_s = \text{Sender Overhead} = 80 \times 10^{-6} \text{ s}$$
$$O_R = \text{Receiver Overhead} = 100 \times 10^{-6} \text{ s}$$
$$B = \text{Bandwidth} = 1 \text{ GB/s} = 1024^2 \text{ KB/s}$$
$$P_s = \text{Packet Size} = 10 \text{ KB}$$
$$T_t = \text{Transmission Time} = P_s / B = \frac{10 \text{ KB}}{1024^2 \text{ KB/s}} \approx 9.5 \times 10^{-6} \text{ s}$$
$$D_x = \text{Distance} = 1 \times 10^{-3} \text{ km (a)}, 0.5 \text{ km (b)}, 1000 \text{ km (c)}$$
$$T_f = \text{Time of flight} = \frac{D_x}{3 \times 10^8 \text{ km/s} \cdot 2/3}$$
$$T_L = \text{Total Latency} = O_s + T_t + T_f + O_R$$

a) if $D_x = 1 \text{ m}$, $T_L \approx 1.895 \times 10^{-4} \text{ s}$

b) if $D_x = 500 \text{ m}$, $T_L \approx 1.920 \times 10^{-4} \text{ s}$

c) if $D_x = 1000 \text{ km}$, $T_L \approx 5.190 \times 10^{-3} \text{ s}$

Figure 5: Math for Question 6

Q7:

7)

a) $t_0 = 20 \times 10^{-6} \text{ s}$, $m = 1 \text{ kB}$, $r_0 = 200 \times 10^2 \text{ kB/s}$
 $T(m) = t_0 + m/r_0 = 20 \times 10^{-6} \text{ s} + 1 \text{ kB} / 200 \times 10^2 \text{ kB/s}$
 $T(m) \approx 2.488 \times 10^{-5} \text{ s} \approx 24.88 \mu\text{s}$

b) $r_{1/2} = r_0 \cdot \frac{1}{2}$, a bandwidth of half would read $m_{1/2}$ in $T = \frac{m_{1/2}}{r_{1/2}}$ this same time would also be achieved if $T = t_0 + \frac{m_{1/2}}{r_0}$

$\therefore T = T$
 $m_{1/2}/r_{1/2} = t_0 + \frac{m_{1/2}}{r_0}$
 $\frac{m_{1/2}}{r_{1/2}} - \frac{m_{1/2}}{r_0} = t_0$
 $m_{1/2} \left(\frac{1}{r_{1/2}} - \frac{1}{r_0} \right) = t_0$
 $m_{1/2} \left(\frac{1}{\frac{r_0}{2}} - \frac{1}{r_0} \right) = t_0$
 $m_{1/2} \left(\frac{2}{r_0} - \frac{1}{r_0} \right) = t_0$
 $m_{1/2} = t_0 r_0$

$t_0 = 100 \times 10^{-6} \text{ s}$
 $r_0 = 800 \text{ MB/s}$
 $m_{1/2} = 100 \times 10^{-6} \text{ s} \cdot 800 \text{ MB/s}$
 $m_{1/2} = 0.08 \text{ MB}$

Figure 6: Math for Question 7

Q8:

Q9:

Code Block 1: MPI code for question 9

```
1 // Anthony Sicoie (20214793)
2
3 #include <mpi.h>
4 #include <stdio.h>
5 #include <stdlib.h>
6
7 #define N_RUNS 1000
8
9 int main(int argc, char *argv[]) {
10     int rank, size;
11     MPI_Status status;
12     double total_time = 0.0;
13
14     MPI_Init(&argc, &argv);
15     MPI_Comm_rank(MPI_COMM_WORLD, &rank);
16     MPI_Comm_size(MPI_COMM_WORLD, &size);
17
18     // Allow user to specify data count (for use in performance testing script)
19     if (argc < 2) {
20         if (rank == 0) {
21             fprintf(stderr, "Usage: %s <array_size>\n", argv[0]);
22         }
23         MPI_Finalize();
24         return 1;
25     }
26
27     int DATA_COUNT = atoi(argv[1]);
28
29     char *data;
30     data = malloc(DATA_COUNT * sizeof(char));
31
32     if (!rank) {
33         for (int i = 0; i < DATA_COUNT; i++) {
34             data[i] = (char)i;
35         }
36     }
37
38     // Loop for benchmarking
39     for (int run = 0; run < N_RUNS; run++) {
40
41         double offset = MPI_Wtime();
42         double start = MPI_Wtime();
43
44         if (!rank) {
45             MPI_Send(data, DATA_COUNT, MPI_CHAR, 1, 0, MPI_COMM_WORLD);
46             MPI_Recv(data, DATA_COUNT, MPI_CHAR, 1, 0, MPI_COMM_WORLD, &status);
47         }
48
49         if (rank) {
50             MPI_Recv(data, DATA_COUNT, MPI_CHAR, 0, 0, MPI_COMM_WORLD, &status);
51             MPI_Send(data, DATA_COUNT, MPI_CHAR, 0, 0, MPI_COMM_WORLD);
52         }
53
54         double end = MPI_Wtime();
55         double timer_overhead = start - offset;
56         total_time += (end - start - timer_overhead);
57     }
58
59     free(data);
60 }
```

```

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

```
1 // Anthony Sicoie (20214793)
2
3 #include <mpi.h>
4 #include <stdio.h>
5 #include <stdlib.h>
6
7 #define N_RUNS 1000
8
9 int main(int argc, char *argv[]) {
10     int rank, size;
11     MPI_Status status;
12     int DATA_COUNT;
13     double total_time = 0.0;
14
15     MPI_Init(&argc, &argv);
16     MPI_Comm_rank(MPI_COMM_WORLD, &rank);
17     MPI_Comm_size(MPI_COMM_WORLD, &size);
18
19     // Allow user to specify data count (for use in performance testing script)
20     if (argc < 2) {
21         if (rank == 0) {
22             fprintf(stderr, "Usage: %s <array_size>\n", argv[0]);
23         }
24         MPI_Finalize();
25         return 1;
26     }
27
28     DATA_COUNT = atoi(argv[1]);
29
30     char *data;
31
32     data = malloc(DATA_COUNT * sizeof(char));
33
34     if (!rank) {
35         for (int i = 0; i < DATA_COUNT; i++) {
36             data[i] = (char)i;
37         }
38     }
39
40     for (int run = 0; run < N_RUNS; run++) {
41         double offset = MPI_Wtime();
42         double start = MPI_Wtime();
43
44         MPI_Bcast(data, DATA_COUNT, MPI_CHAR, 0, MPI_COMM_WORLD);
45
46         double end = MPI_Wtime();
47
48         double timer_overhead = start - offset;
49         total_time += (end - start - timer_overhead);
50     }
51
52     double avg_time = total_time / N_RUNS;
53
54     if (!rank) {
55         FILE *f = fopen("results.csv", "a");
56         if (f == NULL) {
57             fprintf(stderr, "Error opening results.csv\n");
58             MPI_Abort(MPI_COMM_WORLD, 1);
59         }
60         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