Iowa State University
Aerospace Engineering

AERE 361 Lab 10 Report

Rohan Gupta

May 3, 2021

Contents

Chapter 1

Pre-Lab

1.1. Objectives

In this lab, we will gain additional experience with using the HPC-Classroom cluster on campus. We aill also learn how to parallelize tasks to multiple processors. Finally we will learn how processors can communicate between each other using MPI.

1.2. Methodology

For this lab, I used VSCode and Windows Subsystem for Linux (WSL) through VSCode, along with Github Desktop for the entire lab. I accessed the HPC through WSL as opposed to PowerShell, which I used last time to access the HPC.

Chapter 2

Lab Work

2.1. Results

2.1.1. Exercise 1 Code

```c
#include <stdio.h>
#include <string.h>
#include <mpi.h>

int  main(int argc , char* argv []) {
    int  rank;  // rank of processors
    int p;      // number of processors

    // Start up MPI
    MPI_Init(&argc , &argv);

    // Find out processor rank
    MPI_Comm_rank(MPI_COMM_WORLD , &rank);

    printf("Hello world from processor %d!\n", rank);

    // Shut down MPI
    MPI_Finalize();
    return 0;
}
```

2.1.2. Exercise 1 Outputs

Outputs for 4 processors

```
Hello world from processor 0!
Hello world from processor 1!
Hello world from processor 2!
Hello world from processor 3!
```

Outputs for 8 processors

```
Hello world from processor 0!
Hello world from processor 1!
Hello world from processor 2!
Hello world from processor 3!
Hello world from processor 4!
Hello world from processor 5!
Hello world from processor 6!
```

```
Hello world from processor 7!
```

Outputs for 16 processors

```
Hello world from processor 0!
Hello world from processor 1!
Hello world from processor 2!
Hello world from processor 3!
Hello world from processor 4!
Hello world from processor 5!
Hello world from processor 6!
Hello world from processor 7!
Hello world from processor 8!
Hello world from processor 9!
Hello world from processor 10!
Hello world from processor 11!
Hello world from processor 12!
Hello world from processor 13!
Hello world from processor 14!
Hello world from processor 15!
```

2.1.3. Exercise 2 Code

```c
/*
AER E 361
Lab 10
Rohan Gupta
*/

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

#define TRIALS 100;

int main(int argc, char *argv[]) {
  int i, j, tag = 1;
  int length = 10;
  int trial = 10;
  int *A, *B;
  double *time;
  double t1, t2;
  int n = 1;

  // MPI Initialization
  MPI_Init(&argc, &argv);

  int size;
  MPI_Comm_size(MPI_COMM_WORLD, &size);

  int rank;
  MPI_Comm_rank(MPI_COMM_WORLD, &rank);

  MPI_Status stat;

  // Allocating memory for A,B,time vector
  // INSERT YOUR CODE HERE FOR ALLOCATING A,B, and time
  A = malloc(length * sizeof(int));
  B = malloc(length * sizeof(int));
  time = malloc(trial * sizeof(double));

  for (j = 0; j < trial; j++) {
    if (rank == 0) {
      for (i = 0; i < length; i++) {
```

```
        A[i] = i;
      }
    }
    for (i = 0; i < length; i++) {
      B[i] = rank;
    }
  }

  // Calculating A = A + B on different processor and send value of B back to the root
      processor.
  int iter;
  for (iter = 0; iter < trial; iter++) {
    MPI_Barrier(MPI_COMM_WORLD);

    if (rank == 0) {
      t1 = MPI_Wtime();

      for (j = 1; j < size; j++) {
        MPI_Send(A, n, MPI_INT, j, tag, MPI_COMM_WORLD);
      };

      // INSERT YOUR MPI_Recv below
      MPI_Recv(B, 1, MPI_INT, 1, tag, MPI_COMM_WORLD, &stat);

      t2 = MPI_Wtime();
      time[iter] = (t2 - t1) * 0.5 * (1e+6);
    }

    if (rank > 0) {
      //INSERT to receive A
      MPI_Recv(A, n, MPI_INT, 0, tag, MPI_COMM_WORLD, &stat);

      for (i = 0; i < length; ++i) {
        A[i] += B[i];
      }

      // Insert code to send A
      MPI_Send(A, 1, MPI_INT, 0, tag, MPI_COMM_WORLD);
    }

    if (rank == 0) {
      for (i = 0; i < length; ++i) {
        printf("A[%d] = %d\n", i, A[i]);
      }
    }

    // Print out average time
    if (rank == 0) {
      printf("number of trial = %d, number of processor used = %d\n", trial, size);
      double sum;
      for (i = 0; i < trial; i++) {
        sum += time[i];
      }
      double average = sum / trial; // in ms
      printf("average time = %lf microsecond\n", average);
    }

    //INSERT code to free up what you did for malloc
    free(A);
    free(B);
    free(time);

    MPI_Finalize();
    return 0;
  }
}
```

2.1.4. Exercise 2 Outputs

Outputs for 4 processors

```
A[0] = 0
A[1] = 1
A[2] = 2
A[3] = 3
A[4] = 4
A[5] = 5
A[6] = 6
A[7] = 7
A[8] = 8
A[9] = 9
number of trial = 10, number of processor used = 4
average time = 20.754337 microsecond
```

Outputs for 8 processors

```
A[0] = 0
A[1] = 1
A[2] = 2
A[3] = 3
A[4] = 4
A[5] = 5
A[6] = 6
A[7] = 7
A[8] = 8
A[9] = 9
number of trial = 10, number of processor used = 8
average time = 1.096725 microsecond
```

Outputs for 16 processors

```
A[0] = 0
A[1] = 1
A[2] = 2
A[3] = 3
A[4] = 4
A[5] = 5
A[6] = 6
A[7] = 7
A[8] = 8
A[9] = 9
number of trial = 10, number of processor used = 16
average time = 2.050400 microsecond
```

2.1.5. Exercise 3 Code

```
/*-------------------------
AerE 361
Lab 10
Rohan Gupta
-------------------------*/

/*-----------------------------
Original code information:
Licensing:
This code is distributed under the GNU LGPL license.
```

```
  Modified: 14 December 2011
  Author: John Burkardt

  Modification by Matthew Nelson
  Modified on April 19th, 2021
  ---------------------------*/
#include <math.h>
#include <stdlib.h>
#include <stdio.h>
#include <time.h>
#include <omp.h>

int main(int argc, char *argv[]);
double eval_f(double x);
double cpu_time();
void timestamp();
/*
  Purpose:
    MAIN is the main program for QUAD_OPENMP.
*/

int main(int argc, char *argv[]) {

  // Variables that we will use.
  double a = 0.0;
  double b = 10.0;
  double error;
  double exact = 0.49936338107645674464;
  int i;
  int n = 10000000;
  double total;
  double wtime;
  double x;

  timestamp();
  printf("\n");
  printf("QUAD_OPENMP:\n");
  printf("  Use OpenMP for parallel execution.\n");
  printf("  Estimate the integral of f(x) from A to B.\n");
  printf("  f(x) = 50 / ( pi * ( 2500 * x * x + 1 ) ).\n");
  printf("\n");
  printf("  A        = %f\n", a);
  printf("  B        = %f\n", b);
  printf("  N        = %d\n", n);
  printf("  Exact    = %24.16f\n", exact);

  wtime = omp_get_wtime();

  total = 0.0;
  // INSERT your code for the pragma for OpemMP
  // Setup the shared and private
  #pragma omp parallel shared(a, b, n) private(i, x)
  #pragma omp for reduction(+ : total)

  // Setup the reduction

  for (i = 0; i < n; i++) {
    x = ((double)(n - i - 1) * a + (double)(i)*b) / (double)(n - 1);
    total = total + eval_f(x);
  }

  wtime = omp_get_wtime() - wtime;

  total = (b - a) * total / (double)n;
  error = fabs(total - exact);

  printf("\n");
```

```
  printf("  Estimate = %24.16f\n", total);
  printf("  Error    = %e\n", error);
  printf("  Time     = %f\n", wtime);

  // Terminate
  printf("\n");
  printf("QUAD_OPENMP:\n");
  printf("  Normal end of execution.\n");
  printf("\n");
  timestamp();

  return 0;
}

// Functions
/*
  Function: f
  Purpose: F evaluates the function.

  Parameters:
  Input, double X, the argument.
  Output, double F, the value of the function.
*/

double eval_f(double x) {
  double r8_pi = 3.141592653589793;
  double value;
  // Insert the code based on the formula in the writeup
  value = 50.0/(r8_pi*(2500.0*x*x + 1.0));

  return value;
}

  /*
    Function: cpu_time
    Purpose: CPU_TIME reports the total CPU time for a program.

    Parameters:
    Output, double CPU_TIME, the current total elapsed CPU time in second.
  */
  double cpu_time() {
    double value;

    value = (double)clock() / (double)CLOCKS_PER_SEC;

    return value;
  }
  /*
    Function: timestamp
    Purpose:
    TIMESTAMP prints the current YMDHMS date as a time stamp.

    Example:
    31 May 2001 09:45:54 AM

    Parameters:
    None
  */

  void timestamp() {
  #define TIME_SIZE 40
  static char time_buffer[TIME_SIZE];
  const struct tm *tm;
  time_t now;

  now = time(NULL);
  tm = localtime(&now);
```

```
  strftime(time_buffer, TIME_SIZE, "%d %B %Y %I:%M:%S %p", tm);

  printf("%s\n", time_buffer);

  return;
  #undef TIME_SIZE
}
```

2.1.6. Exercise 3 Outputs

Outputs for 4 processors

```
03 May 2021 02:26:15 AM

QUAD_OPENMP:
  Use OpenMP for parallel execution.
  Estimate the integral of f(x) from A to B.
  f(x) = 50 / ( pi * ( 2500 * x * x + 1 ) ).

  A        = 0.000000
  B        = 10.000000
  N        = 10000000
  Exact    =       0.4993633810764567

  Estimate =       0.4993712889190901
  Error    = 7.907843e-06
  Time     = 0.025390

QUAD_OPENMP:
  Normal end of execution.

03 May 2021 02:26:15 AM
03 May 2021 02:26:15 AM

QUAD_OPENMP:
  Use OpenMP for parallel execution.
  Estimate the integral of f(x) from A to B.
  f(x) = 50 / ( pi * ( 2500 * x * x + 1 ) ).

  A        = 0.000000
  B        = 10.000000
  N        = 10000000
  Exact    =       0.4993633810764567

  Estimate =       0.4993712889190901
  Error    = 7.907843e-06
  Time     = 0.025193

QUAD_OPENMP:
  Normal end of execution.

03 May 2021 02:26:15 AM
03 May 2021 02:26:15 AM

QUAD_OPENMP:
  Use OpenMP for parallel execution.
```

```
  Estimate the integral of f(x) from A to B.
  f(x) = 50 / ( pi * ( 2500 * x * x + 1 ) ).

  A        = 0.000000
  B        = 10.000000
  N        = 10000000
  Exact    =       0.4993633810764567

  Estimate =       0.4993712889190901
  Error    = 7.907843e-06
  Time     = 0.025277

QUAD_OPENMP:
  Normal end of execution.

03 May 2021 02:26:15 AM
03 May 2021 02:26:15 AM

QUAD_OPENMP:
  Use OpenMP for parallel execution.
  Estimate the integral of f(x) from A to B.
  f(x) = 50 / ( pi * ( 2500 * x * x + 1 ) ).

  A        = 0.000000
  B        = 10.000000
  N        = 10000000
  Exact    =       0.4993633810764567

  Estimate =       0.4993712889190901
  Error    = 7.907843e-06
  Time     = 0.025770

QUAD_OPENMP:
  Normal end of execution.

03 May 2021 02:26:15 AM
```

Outputs for 8 processors

```
03 May 2021 02:26:24 AM

QUAD_OPENMP:
  Use OpenMP for parallel execution.
  Estimate the integral of f(x) from A to B.
  f(x) = 50 / ( pi * ( 2500 * x * x + 1 ) ).

  A        = 0.000000
  B        = 10.000000
  N        = 10000000
  Exact    =       0.4993633810764567

  Estimate =       0.4993712889190981
  Error    = 7.907843e-06
  Time     = 0.048410
```

```
QUAD_OPENMP:
  Normal end of execution.

03 May 2021 02:26:24 AM
03 May 2021 02:26:24 AM

QUAD_OPENMP:
  Use OpenMP for parallel execution.
  Estimate the integral of f(x) from A to B.
  f(x) = 50 / ( pi * ( 2500 * x * x + 1 ) ).

  A        = 0.000000
  B        = 10.000000
  N        = 10000000
  Exact    =       0.4993633810764567

  Estimate =       0.4993712889190981
  Error    = 7.907843e-06
  Time     = 0.048468

QUAD_OPENMP:
  Normal end of execution.

03 May 2021 02:26:24 AM
03 May 2021 02:26:24 AM

QUAD_OPENMP:
  Use OpenMP for parallel execution.
  Estimate the integral of f(x) from A to B.
  f(x) = 50 / ( pi * ( 2500 * x * x + 1 ) ).

  A        = 0.000000
  B        = 10.000000
  N        = 10000000
  Exact    =       0.4993633810764567

  Estimate =       0.4993712889190981
  Error    = 7.907843e-06
  Time     = 0.048559

QUAD_OPENMP:
  Normal end of execution.

03 May 2021 02:26:24 AM
03 May 2021 02:26:24 AM

QUAD_OPENMP:
  Use OpenMP for parallel execution.
  Estimate the integral of f(x) from A to B.
  f(x) = 50 / ( pi * ( 2500 * x * x + 1 ) ).

  A        = 0.000000
  B        = 10.000000
  N        = 10000000
```

```
  Exact    =       0.4993633810764567

  Estimate =       0.4993712889190981
  Error    = 7.907843e-06
  Time     = 0.048646

QUAD_OPENMP:
  Normal end of execution.

03 May 2021 02:26:24 AM
03 May 2021 02:26:24 AM

QUAD_OPENMP:
  Use OpenMP for parallel execution.
  Estimate the integral of f(x) from A to B.
  f(x) = 50 / ( pi * ( 2500 * x * x + 1 ) ).

  A        = 0.000000
  B        = 10.000000
  N        = 10000000
  Exact    =       0.4993633810764567

  Estimate =       0.4993712889190981
  Error    = 7.907843e-06
  Time     = 0.048636

QUAD_OPENMP:
  Normal end of execution.

03 May 2021 02:26:24 AM
03 May 2021 02:26:24 AM

QUAD_OPENMP:
  Use OpenMP for parallel execution.
  Estimate the integral of f(x) from A to B.
  f(x) = 50 / ( pi * ( 2500 * x * x + 1 ) ).

  A        = 0.000000
  B        = 10.000000
  N        = 10000000
  Exact    =       0.4993633810764567

  Estimate =       0.4993712889190981
  Error    = 7.907843e-06
  Time     = 0.048723

QUAD_OPENMP:
  Normal end of execution.

03 May 2021 02:26:24 AM
03 May 2021 02:26:24 AM

QUAD_OPENMP:
  Use OpenMP for parallel execution.
```

```
  Estimate the integral of f(x) from A to B.
  f(x) = 50 / ( pi * ( 2500 * x * x + 1 ) ).

  A        = 0.000000
  B        = 10.000000
  N        = 10000000
  Exact    =       0.4993633810764567

  Estimate =       0.4993712889190981
  Error    = 7.907843e-06
  Time     = 0.048754

QUAD_OPENMP:
  Normal end of execution.

03 May 2021 02:26:24 AM
03 May 2021 02:26:24 AM

QUAD_OPENMP:
  Use OpenMP for parallel execution.
  Estimate the integral of f(x) from A to B.
  f(x) = 50 / ( pi * ( 2500 * x * x + 1 ) ).

  A        = 0.000000
  B        = 10.000000
  N        = 10000000
  Exact    =       0.4993633810764567

  Estimate =       0.4993712889190981
  Error    = 7.907843e-06
  Time     = 0.048619

QUAD_OPENMP:
  Normal end of execution.

03 May 2021 02:26:24 AM
```

Outputs for 16 processors

```
03 May 2021 02:26:37 AM

QUAD_OPENMP:
  Use OpenMP for parallel execution.
  Estimate the integral of f(x) from A to B.
  f(x) = 50 / ( pi * ( 2500 * x * x + 1 ) ).

  A        = 0.000000
  B        = 10.000000
  N        = 10000000
  Exact    =       0.4993633810764567

  Estimate =       0.4993712889190963
  Error    = 7.907843e-06
  Time     = 0.094677
```

```
QUAD_OPENMP:
  Normal end of execution.

03 May 2021 02:26:37 AM
03 May 2021 02:26:37 AM

QUAD_OPENMP:
  Use OpenMP for parallel execution.
  Estimate the integral of f(x) from A to B.
  f(x) = 50 / ( pi * ( 2500 * x * x + 1 ) ).

  A        = 0.000000
  B        = 10.000000
  N        = 10000000
  Exact    =       0.4993633810764567

  Estimate =       0.4993712889190963
  Error    = 7.907843e-06
  Time     = 0.094656

QUAD_OPENMP:
  Normal end of execution.

03 May 2021 02:26:37 AM
03 May 2021 02:26:37 AM

QUAD_OPENMP:
  Use OpenMP for parallel execution.
  Estimate the integral of f(x) from A to B.
  f(x) = 50 / ( pi * ( 2500 * x * x + 1 ) ).

  A        = 0.000000
  B        = 10.000000
  N        = 10000000
  Exact    =       0.4993633810764567

  Estimate =       0.4993712889190963
  Error    = 7.907843e-06
  Time     = 0.094529

QUAD_OPENMP:
  Normal end of execution.

03 May 2021 02:26:37 AM
03 May 2021 02:26:37 AM

QUAD_OPENMP:
  Use OpenMP for parallel execution.
  Estimate the integral of f(x) from A to B.
  f(x) = 50 / ( pi * ( 2500 * x * x + 1 ) ).

  A        = 0.000000
  B        = 10.000000
  N        = 10000000
```

```
  Exact    =       0.4993633810764567

  Estimate =       0.4993712889190963
  Error    = 7.907843e-06
  Time     = 0.094524

QUAD_OPENMP:
  Normal end of execution.

03 May 2021 02:26:37 AM
03 May 2021 02:26:37 AM

QUAD_OPENMP:
  Use OpenMP for parallel execution.
  Estimate the integral of f(x) from A to B.
  f(x) = 50 / ( pi * ( 2500 * x * x + 1 ) ).

  A        = 0.000000
  B        = 10.000000
  N        = 10000000
  Exact    =       0.4993633810764567

  Estimate =       0.4993712889190963
  Error    = 7.907843e-06
  Time     = 0.094625

QUAD_OPENMP:
  Normal end of execution.

03 May 2021 02:26:37 AM
03 May 2021 02:26:37 AM

QUAD_OPENMP:
  Use OpenMP for parallel execution.
  Estimate the integral of f(x) from A to B.
  f(x) = 50 / ( pi * ( 2500 * x * x + 1 ) ).

  A        = 0.000000
  B        = 10.000000
  N        = 10000000
  Exact    =       0.4993633810764567

  Estimate =       0.4993712889190963
  Error    = 7.907843e-06
  Time     = 0.094638

QUAD_OPENMP:
  Normal end of execution.

03 May 2021 02:26:37 AM
03 May 2021 02:26:37 AM

QUAD_OPENMP:
  Use OpenMP for parallel execution.
```

```
  Estimate the integral of f(x) from A to B.
  f(x) = 50 / ( pi * ( 2500 * x * x + 1 ) ).

  A        = 0.000000
  B        = 10.000000
  N        = 10000000
  Exact    =       0.4993633810764567

  Estimate =       0.4993712889190963
  Error    = 7.907843e-06
  Time     = 0.094674

QUAD_OPENMP:
  Normal end of execution.

03 May 2021 02:26:37 AM
03 May 2021 02:26:37 AM

QUAD_OPENMP:
  Use OpenMP for parallel execution.
  Estimate the integral of f(x) from A to B.
  f(x) = 50 / ( pi * ( 2500 * x * x + 1 ) ).

  A        = 0.000000
  B        = 10.000000
  N        = 10000000
  Exact    =       0.4993633810764567

  Estimate =       0.4993712889190963
  Error    = 7.907843e-06
  Time     = 0.094521

QUAD_OPENMP:
  Normal end of execution.

03 May 2021 02:26:37 AM
03 May 2021 02:26:37 AM

QUAD_OPENMP:
  Use OpenMP for parallel execution.
  Estimate the integral of f(x) from A to B.
  f(x) = 50 / ( pi * ( 2500 * x * x + 1 ) ).

  A        = 0.000000
  B        = 10.000000
  N        = 10000000
  Exact    =       0.4993633810764567

  Estimate =       0.4993712889190963
  Error    = 7.907843e-06
  Time     = 0.094613

QUAD_OPENMP:
  Normal end of execution.
```

```
03 May 2021 02:26:37 AM
03 May 2021 02:26:37 AM

QUAD_OPENMP:
  Use OpenMP for parallel execution.
  Estimate the integral of f(x) from A to B.
  f(x) = 50 / ( pi * ( 2500 * x * x + 1 ) ).

  A        = 0.000000
  B        = 10.000000
  N        = 10000000
  Exact    =       0.4993633810764567

  Estimate =       0.4993712889190963
  Error    = 7.907843e-06
  Time     = 0.094694

QUAD_OPENMP:
  Normal end of execution.

03 May 2021 02:26:37 AM
03 May 2021 02:26:37 AM

QUAD_OPENMP:
  Use OpenMP for parallel execution.
  Estimate the integral of f(x) from A to B.
  f(x) = 50 / ( pi * ( 2500 * x * x + 1 ) ).

  A        = 0.000000
  B        = 10.000000
  N        = 10000000
  Exact    =       0.4993633810764567

  Estimate =       0.4993712889190963
  Error    = 7.907843e-06
  Time     = 0.094577

QUAD_OPENMP:
  Normal end of execution.

03 May 2021 02:26:37 AM
03 May 2021 02:26:37 AM

QUAD_OPENMP:
  Use OpenMP for parallel execution.
  Estimate the integral of f(x) from A to B.
  f(x) = 50 / ( pi * ( 2500 * x * x + 1 ) ).

  A        = 0.000000
  B        = 10.000000
  N        = 10000000
  Exact    =       0.4993633810764567
```

```
  Estimate =       0.4993712889190963
  Error    = 7.907843e-06
  Time     = 0.094559

QUAD_OPENMP:
  Normal end of execution.

03 May 2021 02:26:37 AM
03 May 2021 02:26:37 AM

QUAD_OPENMP:
  Use OpenMP for parallel execution.
  Estimate the integral of f(x) from A to B.
  f(x) = 50 / ( pi * ( 2500 * x * x + 1 ) ).

  A        = 0.000000
  B        = 10.000000
  N        = 10000000
  Exact    =       0.4993633810764567

  Estimate =       0.4993712889190963
  Error    = 7.907843e-06
  Time     = 0.094518

QUAD_OPENMP:
  Normal end of execution.

03 May 2021 02:26:37 AM
03 May 2021 02:26:37 AM

QUAD_OPENMP:
  Use OpenMP for parallel execution.
  Estimate the integral of f(x) from A to B.
  f(x) = 50 / ( pi * ( 2500 * x * x + 1 ) ).

  A        = 0.000000
  B        = 10.000000
  N        = 10000000
  Exact    =       0.4993633810764567

  Estimate =       0.4993712889190963
  Error    = 7.907843e-06
  Time     = 0.094560

QUAD_OPENMP:
  Normal end of execution.

03 May 2021 02:26:37 AM
03 May 2021 02:26:37 AM

QUAD_OPENMP:
  Use OpenMP for parallel execution.
  Estimate the integral of f(x) from A to B.
  f(x) = 50 / ( pi * ( 2500 * x * x + 1 ) ).
```

```
  A        = 0.000000
  B        = 10.000000
  N        = 10000000
  Exact    =       0.4993633810764567

  Estimate =       0.4993712889190963
  Error    = 7.907843e-06
  Time     = 0.094395

QUAD_OPENMP:
  Normal end of execution.

03 May 2021 02:26:37 AM
03 May 2021 02:26:37 AM

QUAD_OPENMP:
  Use OpenMP for parallel execution.
  Estimate the integral of f(x) from A to B.
  f(x) = 50 / ( pi * ( 2500 * x * x + 1 ) ).

  A        = 0.000000
  B        = 10.000000
  N        = 10000000
  Exact    =       0.4993633810764567

  Estimate =       0.4993712889190963
  Error    = 7.907843e-06
  Time     = 0.094438

QUAD_OPENMP:
  Normal end of execution.

03 May 2021 02:26:37 AM
```

2.1.7. Exercise 4 Code

```c
/*-------------------------
AerE 361
Lab 10
Rohan Gupta
-------------------------*/
#include <stdlib.h>
#include <stdio.h>
#include <math.h>
#include <time.h>
#include <omp.h>

int main();
void ccopy(int n, double x[], double y[]);
void cfft2(int n, double x[], double y[], double w[], double sgn);
void cffti(int n, double w[]);
double ggl(double *ds);
void step(int n, int mj, double a[], double b[], double c[], double d[], double w[], double
    sgn);
void timestamp();

/*
  Purpose:
```

```
    MAIN is the main program for fft.c.

  Discussion:

    The "complex" vector A is actually stored as a double vector B.

    The "complex" vector entry A[I] is stored as:

      B[I*2+0], the real part,
      B[I*2+1], the imaginary part.
  Author:

    Original C version by Wesley Petersen.
    This C version by John Burkardt.
    Additional modifications by Matthew Nelson

  Reference:

    Wesley Petersen, Peter Arbenz,
    Introduction to Parallel Computing - A practical guide with examples in C,
    Oxford University Press,
    ISBN: 0-19-851576-6,
    LC: QA76.58.P47.
*/

int main() {
  double error;
  int first;
  double flops;
  double fnm1;
  int i;
  int icase;
  int it;
  int ln2;
  int ln2_max = 20;
  double mflops;
  int n;
  int nits = 1000;
  static double seed;
  double sgn;
  double *w;
  double wtime;
  double *x;
  double *y;
  double *z;
  double z0;
  double z1;

  timestamp();
  printf("\n");
  printf("FFT_OPENMP\n");
  printf("  C/OpenMP version\n");
  printf("\n");
  printf("  Demonstrate an implementation of the Fast Fourier Transform\n");
  printf("  of a complex data vector, using OpenMP for parallel execution.\n");

  printf("\n");
  printf("  Number of processors available = %d\n", omp_get_num_procs());
  printf("  Number of threads =              %d\n", omp_get_max_threads());


  printf("\n");
  printf("  Accuracy check:\n");
  printf("\n");
  printf("    FFT ( FFT ( X(1:N) ) ) == N * X(1:N)\n");
  printf("\n");
  printf("             N      NITS    Error         Time          Time/Call     MFLOPS\n");
```

```
  printf("\n");

  seed = 331.0;
  n = 1;

  //LN2 is the log base 2 of N.  Each increase of LN2 doubles N.

  for (ln2 = 1; ln2 <= ln2_max; ln2++) {
    n = 2 * n;
    /*
      Allocate storage for the complex arrays W, X, Y, Z.

      We handle the complex arithmetic,
      and store a complex number as a pair of doubles, a complex vector as a doubly
      dimensioned array whose second dimension is 2.
    */
    w = (double *)malloc(n * sizeof(double));
    x = (double *)malloc(2 * n * sizeof(double));
    y = (double *)malloc(2 * n * sizeof(double));
    z = (double *)malloc(2 * n * sizeof(double));

    first = 1;

    for (icase = 0; icase < 2; icase++) {
      if (first) {
        for (i = 0; i < 2 * n; i = i + 2) {
          z0 = ggl(&seed);
          z1 = ggl(&seed);
          x[i] = z0;
          z[i] = z0;
          x[i + 1] = z1;
          z[i + 1] = z1;
        }

      } else {
        // Insert the Pragma for OpenMP
        // Insert the Pragma for the nowait option
        #pragma omp for nowait

        for (i = 0; i < 2 * n; i = i + 2) {
          z0 = 0.0; // real part of array
          z1 = 0.0; // imaginary part of array
          x[i] = z0;
          z[i] = z0; // copy of initial real data
          x[i + 1] = z1;
          z[i + 1] = z1; // copy of initial imag. data
        }
      }

      // Initialize the sine and cosine tables.
      cffti(n, w);
      // Transform forward, back
      if (first) {
        sgn = +1.0;
        cfft2(n, x, y, w, sgn);
        sgn = -1.0;
        cfft2(n, y, x, w, sgn);

        // Results should be same as the initial data multiplied by N
        fnm1 = 1.0 / (double)n;
        error = 0.0;

        for (i = 0; i < 2 * n; i = i + 2) {
          error = error + pow(z[i] - fnm1 * x[i], 2) + pow(z[i + 1] - fnm1 * x[i + 1], 2);
        }

        error = sqrt(fnm1 * error);
        printf("  %12d  %8d  %12e", n, nits, error);
```

```
        first = 0;
      }
      else {
        wtime = omp_get_wtime();

        for (it = 0; it < nits; it++) {
          sgn = +1.0;
          cfft2(n, x, y, w, sgn);
          sgn = -1.0;
          cfft2(n, y, x, w, sgn);

        }
        wtime = omp_get_wtime() - wtime;

        flops = 2.0 * (double)nits * (5.0 * (double)n * (double)ln2);

        mflops = flops / 1.0E+06 / wtime;

        printf("  %12e  %12e  %12f\n", wtime, wtime / (double)(2 * nits), mflops);
      }
    }

    if ((ln2 % 4) == 0) {
      nits = nits / 10;
    }

    if (nits < 1) {
      nits = 1;
    }

    free(w);
    free(x);
    free(y);
    free(z);
  }

  // Terminate

  printf("\n");
  printf("FFT_OPENMP:\n");
  printf("  Normal end of execution.\n");
  printf("\n");
  timestamp();

  return 0;
}

/*
  Purpose: CCOPY copies a complex vector.

  Discussion:
    The "complex" vector A[N] is actually stored as a double vector B[2*N].

    The "complex" vector entry A[I] is stored as:

      B[I*2+0], the real part,
      B[I*2+1], the imaginary part.

  Parameters:
    Input, int N, the length of the vector.

    Input, double X[2*N], the vector to be copied.

    Output, double Y[2*N], a copy of X.
*/

void ccopy(int n, double x[], double y[]) {
  int i;
```

```
  for (i = 0; i < n; i++) {
    y[i * 2 + 0] = x[i * 2 + 0];
    y[i * 2 + 1] = x[i * 2 + 1];
  }
  return;
}

/*
  Purpose:
    CFFT2 performs a complex Fast Fourier Transform.

  Parameters:
    Input, int N, the size of the array to be transformed.

    Input/output, double X[2*N], the data to be transformed.
    On output, the contents of X have been overwritten by work information.

    Output, double Y[2*N], the forward or backward FFT of X.

    Input, double W[N], a table of sines and cosines.

    Input, double SGN, is +1 for a "forward" FFT and -1 for a "backward" FFT.
*/

void cfft2(int n, double x[], double y[], double w[], double sgn) {
  int j;
  int m;
  int mj;
  int tgle;

  m = (int)(log((double)n) / log(1.99));
  mj = 1;

  // Toggling switch for work array.

  tgle = 1;
  step(n, mj, &x[0 * 2 + 0], &x[(n / 2) * 2 + 0], &y[0 * 2 + 0], &y[mj * 2 + 0], w, sgn);

  if (n == 2) {
    return;
  }

  for (j = 0; j < m - 2; j++) {
    mj = mj * 2;
    if (tgle) {
      step(n, mj, &y[0 * 2 + 0], &y[(n / 2) * 2 + 0], &x[0 * 2 + 0], &x[mj * 2 + 0], w, sgn)
          ;
      tgle = 0;
    }
    else {
      step(n, mj, &x[0 * 2 + 0], &x[(n / 2) * 2 + 0], &y[0 * 2 + 0], &y[mj * 2 + 0], w, sgn)
          ;
      tgle = 1;
    }
  }

  // Last pass through data: move Y to X if needed.

  if (tgle) {
    ccopy(n, y, x);
  }

  mj = n / 2;
  step(n, mj, &x[0 * 2 + 0], &x[(n / 2) * 2 + 0], &y[0 * 2 + 0], &y[mj * 2 + 0], w, sgn);

  return;
}
```

```c
/*
  Purpose:
    CFFTI sets up sine and cosine tables needed for the FFT calculation.
  Parameters:
    Input, int N, the size of the array to be transformed.

    Output, double W[N], a table of sines and cosines.
*/

void cffti(int n, double w[]) {
  #pragma omp parallel shared(aw, n, w) private(arg, i)
  #pragma omp for nowait

  double arg;
  double aw;
  int i;
  int n2;
  const double pi = 3.141592653589793;

  n2 = n / 2;
  aw = 2.0 * pi / ((double)n);

  for (i = 0; i < n2; i++) {
    arg = aw * ((double)i);
    w[i * 2 + 0] = cos(arg);
    w[i * 2 + 1] = sin(arg);

  }
  return;
}

/*
  Purpose:
    GGL generates uniformly distributed pseudorandom real numbers in [0,1].
  Parameters:

    Input/output, double *SEED, used as a seed for the sequence.

    Output, double GGL, the next pseudorandom value.
*/


double ggl(double *seed) {
  double d2 = 0.2147483647e10;
  double t;
  double value;

  t = (double)*seed;
  t = fmod(16807.0 * t, d2);
  *seed = (double)t;
  value = (double)((t - 1.0) / (d2 - 1.0));

  return value;
}

/*
  Purpose:
    STEP carries out one step of the workspace version of CFFT2.
  Parameters:
*/

void step(int n, int mj, double a[], double b[], double c[], double d[], double w[], double
    sgn) {

  #pragma omp parallel shared(a, b, c, d, lj, mj, mj2, sgn, w) private(ambr, ambu, j, ja, jb
      , jc, jd, jw, k, wjw)
  #pragma omp for nowait
```

```
  double ambr;
  double ambu;
  int j;
  int ja;
  int jb;
  int jc;
  int jd;
  int jw;
  int k;
  int lj;
  int mj2;
  double wjw[2];

  mj2 = 2 * mj;
  lj = n / mj2;

  for (j = 0; j < lj; j++) {
    jw = j * mj;
    ja = jw;
    jb = ja;
    jc = j * mj2;
    jd = jc;

    wjw[0] = w[jw * 2 + 0];
    wjw[1] = w[jw * 2 + 1];

    if (sgn < 0.0) {
      wjw[1] = -wjw[1];
    }

    for (k = 0; k < mj; k++) {
      c[(jc + k) * 2 + 0] = a[(ja + k) * 2 + 0] + b[(jb + k) * 2 + 0];
      c[(jc + k) * 2 + 1] = a[(ja + k) * 2 + 1] + b[(jb + k) * 2 + 1];

      ambr = a[(ja + k) * 2 + 0] - b[(jb + k) * 2 + 0];
      ambu = a[(ja + k) * 2 + 1] - b[(jb + k) * 2 + 1];

      d[(jd + k) * 2 + 0] = wjw[0] * ambr - wjw[1] * ambu;
      d[(jd + k) * 2 + 1] = wjw[1] * ambr + wjw[0] * ambu;
    }
  }
  return;
}

/*
  Purpose:
    TIMESTAMP prints the current YMDHMS date as a time stamp.
  Example:
    31 May 2001 09:45:54 AM
  Parameters:

    None
*/

void timestamp(void) {
  #define TIME_SIZE 40

  static char time_buffer[TIME_SIZE];
  const struct tm *tm;
  time_t now;

  now = time(NULL);
  tm = localtime(&now);

  strftime(time_buffer, TIME_SIZE, "%d %B %Y %I:%M:%S %p", tm);

  printf("%s\n", time_buffer);
```

```
  return;
  #undef TIME_SIZE
}
```

2.1.8. Exercise 4 Outputs

Outputs for 4 processors

```
03 May 2021 01:37:29 AM

FFT_OPENMP
  C/OpenMP version

  Demonstrate an implementation of the Fast Fourier Transform
  of a complex data vector, using OpenMP for parallel execution.

  Number of processors available = 4
  Number of threads =              4

  Accuracy check:

    FFT ( FFT ( X(1:N) ) ) == N * X(1:N)

             N      NITS    Error         Time          Time/Call     MFLOPS

             2      1000  7.859082e-17  1.899958e-03  9.499788e-07     10.526550
             4      1000  1.209837e-16  3.973961e-03  1.986980e-06     20.131049
             8      1000  6.820795e-17  6.060123e-03  3.030062e-06     39.603154
            16      1000  1.438671e-16  8.073092e-03  4.036546e-06     79.275702
            32       100  1.331210e-16  1.094103e-03  5.470514e-06    146.238536
            64       100  1.776545e-16  1.463890e-03  7.319450e-06    262.314778
           128       100  1.929043e-16  1.854181e-03  9.270906e-06    483.232144
           256       100  2.092319e-16  2.786875e-03  1.393437e-05    734.873350
           512        10  1.927488e-16  3.640652e-04  1.820326e-05   1265.707455
          1024        10  2.308607e-16  5.970001e-04  2.985001e-05   1715.242530
          2048        10  2.444507e-16  9.379387e-04  4.689693e-05   2401.862748
          4096        10  2.483274e-16  1.927853e-03  9.639263e-05   2549.572474
          8192         1  2.575054e-16  3.318787e-04  1.659393e-04   3208.883612
         16384         1  2.729354e-16  7.259846e-04  3.629923e-04   3159.516172
         32768         1  2.911847e-16  1.367807e-03  6.839037e-04   3593.488412
         65536         1  2.827259e-16  3.037214e-03  1.518607e-03   3452.426808
        131072         1  3.148261e-16  5.892992e-03  2.946496e-03   3781.142063
        262144         1  3.221995e-16  1.348281e-02  6.741405e-03   3499.709872
        524288         1  3.278767e-16  5.142903e-02  2.571452e-02   1936.935494
       1048576         1  3.286406e-16  1.228330e-01  6.141651e-02   1707.319506

FFT_OPENMP:
  Normal end of execution.

03 May 2021 01:37:29 AM
03 May 2021 01:37:29 AM

FFT_OPENMP
  C/OpenMP version
```

```
  Demonstrate an implementation of the Fast Fourier Transform
  of a complex data vector, using OpenMP for parallel execution.

  Number of processors available = 4
  Number of threads =              4

  Accuracy check:

    FFT ( FFT ( X(1:N) ) ) == N * X(1:N)

           N      NITS    Error         Time          Time/Call     MFLOPS

           2      1000  7.859082e-17  1.774073e-03  8.870363e-07     11.273495
           4      1000  1.209837e-16  3.762007e-03  1.881003e-06     21.265246
           8      1000  6.820795e-17  5.796909e-03  2.898455e-06     41.401372
          16      1000  1.438671e-16  7.781982e-03  3.890991e-06     82.241255
          32       100  1.331210e-16  1.030922e-03  5.154610e-06    155.200888
          64       100  1.776545e-16  1.434088e-03  7.170439e-06    267.766041
         128       100  1.929043e-16  1.812935e-03  9.064674e-06    494.226247
         256       100  2.092319e-16  2.681017e-03  1.340508e-05    763.889248
         512        10  1.927488e-16  3.619194e-04  1.809597e-05   1273.211649
        1024        10  2.308607e-16  6.039143e-04  3.019571e-05   1695.604933
        2048        10  2.444507e-16  9.407997e-04  4.703999e-05   2394.558553
        4096        10  2.483274e-16  1.924992e-03  9.624958e-05   2553.361781
        8192         1  2.575054e-16  3.368855e-04  1.684427e-04   3161.193197
       16384         1  2.729354e-16  7.338524e-04  3.669262e-04   3125.642217
       32768         1  2.911847e-16  1.397848e-03  6.989241e-04   3516.261815
       65536         1  2.827259e-16  3.063917e-03  1.531959e-03   3422.337959
      131072         1  3.148261e-16  5.964041e-03  2.982020e-03   3736.097876
      262144         1  3.221995e-16  1.388097e-02  6.940484e-03   3399.324866
      524288         1  3.278767e-16  5.192208e-02  2.596104e-02   1918.542447
     1048576         1  3.286406e-16  1.240852e-01  6.204259e-02   1690.090522

FFT_OPENMP:
  Normal end of execution.

03 May 2021 01:37:29 AM
03 May 2021 01:37:29 AM

FFT_OPENMP
  C/OpenMP version

  Demonstrate an implementation of the Fast Fourier Transform
  of a complex data vector, using OpenMP for parallel execution.

  Number of processors available = 4
  Number of threads =              4

  Accuracy check:

    FFT ( FFT ( X(1:N) ) ) == N * X(1:N)

           N      NITS    Error         Time          Time/Call     MFLOPS
```

```
             2      1000  7.859082e-17  1.796961e-03  8.984804e-07     11.129903
             4      1000  1.209837e-16  3.728151e-03  1.864076e-06     21.458356
             8      1000  6.820795e-17  5.767107e-03  2.883554e-06     41.615319
            16      1000  1.438671e-16  7.801056e-03  3.900528e-06     82.040176
            32       100  1.331210e-16  1.033068e-03  5.165339e-06    154.878523
            64       100  1.776545e-16  1.405954e-03  7.029772e-06    273.124086
           128       100  1.929043e-16  1.791000e-03  8.955002e-06    500.279071
           256       100  2.092319e-16  2.676964e-03  1.338482e-05    765.045831
           512        10  1.927488e-16  3.519058e-04  1.759529e-05   1309.441249
          1024        10  2.308607e-16  5.950928e-04  2.975464e-05   1720.740103
          2048        10  2.444507e-16  9.951591e-04  4.975796e-05   2263.758517
          4096        10  2.483274e-16  1.940966e-03  9.704828e-05   2532.347749
          8192         1  2.575054e-16  3.330708e-04  1.665354e-04   3197.398703
         16384         1  2.729354e-16  7.269382e-04  3.634691e-04   3155.371185
         32768         1  2.911847e-16  1.386881e-03  6.934404e-04   3544.067908
         65536         1  2.827259e-16  3.041029e-03  1.520514e-03   3448.096049
        131072         1  3.148261e-16  5.953074e-03  2.976537e-03   3742.980831
        262144         1  3.221995e-16  1.360106e-02  6.800532e-03   3469.281347
        524288         1  3.278767e-16  5.341697e-02  2.670848e-02   1864.851654
       1048576         1  3.286406e-16  1.260171e-01  6.300855e-02   1664.180580

FFT_OPENMP:
  Normal end of execution.

03 May 2021 01:37:29 AM
03 May 2021 01:37:29 AM

FFT_OPENMP
  C/OpenMP version

  Demonstrate an implementation of the Fast Fourier Transform
  of a complex data vector, using OpenMP for parallel execution.

  Number of processors available = 4
  Number of threads =              4

  Accuracy check:

    FFT ( FFT ( X(1:N) ) ) == N * X(1:N)

             N      NITS    Error         Time          Time/Call     MFLOPS

             2      1000  7.859082e-17  1.756907e-03  8.784533e-07     11.383645
             4      1000  1.209837e-16  3.669977e-03  1.834989e-06     21.798501
             8      1000  6.820795e-17  5.661011e-03  2.830505e-06     42.395256
            16      1000  1.438671e-16  7.628918e-03  3.814459e-06     83.891323
            32       100  1.331210e-16  1.004219e-03  5.021095e-06    159.327787
            64       100  1.776545e-16  1.384020e-03  6.920099e-06    277.452668
           128       100  1.929043e-16  1.764059e-03  8.820295e-06    507.919500
           256       100  2.092319e-16  2.609015e-03  1.304507e-05    784.970720
           512        10  1.927488e-16  3.480911e-04  1.740456e-05   1323.791290
          1024        10  2.308607e-16  5.910397e-04  2.955198e-05   1732.540257
          2048        10  2.444507e-16  9.360313e-04  4.680157e-05   2406.757018
```

```
          4096        10  2.483274e-16  1.927137e-03  9.635687e-05   2550.518746
          8192         1  2.575054e-16  3.528595e-04  1.764297e-04   3018.085127
         16384         1  2.729354e-16  7.460117e-04  3.730059e-04   3074.696946
         32768         1  2.911847e-16  1.379013e-03  6.895065e-04   3564.288212
         65536         1  2.827259e-16  3.042936e-03  1.521468e-03   3445.934742
        131072         1  3.148261e-16  5.923986e-03  2.961993e-03   3761.359052
        262144         1  3.221995e-16  1.379704e-02  6.898522e-03   3420.001953
        524288         1  3.278767e-16  5.353212e-02  2.676606e-02   1860.840059
       1048576         1  3.286406e-16  1.264608e-01  6.323040e-02   1658.341680

FFT_OPENMP:
  Normal end of execution.

03 May 2021 01:37:29 AM
```

Outputs for 8 processors

```
03 May 2021 01:37:43 AM

FFT_OPENMP
  C/OpenMP version

  Demonstrate an implementation of the Fast Fourier Transform
  of a complex data vector, using OpenMP for parallel execution.

  Number of processors available = 2
  Number of threads =              2

  Accuracy check:

    FFT ( FFT ( X(1:N) ) ) == N * X(1:N)

           N      NITS    Error         Time          Time/Call     MFLOPS

           2      1000  7.859082e-17  1.652956e-03  8.264780e-07     12.099536
           4      1000  1.209837e-16  3.558874e-03  1.779437e-06     22.479019
           8      1000  6.820795e-17  5.294085e-03  2.647042e-06     45.333617
          16      1000  1.438671e-16  7.490873e-03  3.745437e-06     85.437301
          32       100  1.331210e-16  9.350777e-04  4.675388e-06    171.108781
          64       100  1.776545e-16  1.255989e-03  6.279945e-06    305.735144
         128       100  1.929043e-16  1.694918e-03  8.474588e-06    528.639244
         256       100  2.092319e-16  2.496958e-03  1.248479e-05    820.198090
         512        10  1.927488e-16  3.709793e-04  1.854897e-05   1242.117791
        1024        10  2.308607e-16  6.568432e-04  3.284216e-05   1558.971795
        2048        10  2.444507e-16  1.131773e-03  5.658865e-05   1990.505172
        4096        10  2.483274e-16  2.442122e-03  1.221061e-04   2012.676269
        8192         1  2.575054e-16  4.811287e-04  2.405643e-04   2213.461837
       16384         1  2.729354e-16  1.075029e-03  5.375147e-04   2133.671932
       32768         1  2.911847e-16  2.110004e-03  1.055002e-03   2329.473788
       65536         1  2.827259e-16  4.686117e-03  2.343059e-03   2237.622239
      131072         1  3.148261e-16  1.064992e-02  5.324960e-03   2092.244921
      262144         1  3.221995e-16  4.903388e-02  2.451694e-02    962.312584
      524288         1  3.278767e-16  1.051190e-01  5.255949e-02    947.637720
     1048576         1  3.286406e-16  2.349470e-01  1.174735e-01    892.606546
```

```
FFT_OPENMP:
  Normal end of execution.

03 May 2021 01:37:44 AM
03 May 2021 01:37:43 AM

FFT_OPENMP
  C/OpenMP version

  Demonstrate an implementation of the Fast Fourier Transform
  of a complex data vector, using OpenMP for parallel execution.

  Number of processors available = 2
  Number of threads =              2

  Accuracy check:

    FFT ( FFT ( X(1:N) ) ) == N * X(1:N)

           N      NITS    Error         Time          Time/Call     MFLOPS

           2      1000  7.859082e-17  1.631975e-03  8.159876e-07     12.255088
           4      1000  1.209837e-16  3.484011e-03  1.742005e-06     22.962042
           8      1000  6.820795e-17  5.180120e-03  2.590060e-06     46.330969
          16      1000  1.438671e-16  7.364035e-03  3.682017e-06     86.908879
          32       100  1.331210e-16  9.400845e-04  4.700422e-06    170.197474
          64       100  1.776545e-16  1.230955e-03  6.154776e-06    311.952883
         128       100  1.929043e-16  1.657963e-03  8.289814e-06    540.422258
         256       100  2.092319e-16  2.410889e-03  1.205444e-05    849.479291
         512        10  1.927488e-16  3.678799e-04  1.839399e-05   1252.582815
        1024        10  2.308607e-16  6.639957e-04  3.319979e-05   1542.178562
        2048        10  2.444507e-16  1.132011e-03  5.660057e-05   1990.085942
        4096        10  2.483274e-16  2.467871e-03  1.233935e-04   1991.676458
        8192         1  2.575054e-16  4.799366e-04  2.399683e-04   2218.959756
       16384         1  2.729354e-16  1.068115e-03  5.340576e-04   2147.483648
       32768         1  2.911847e-16  2.130985e-03  1.065493e-03   2306.538713
       65536         1  2.827259e-16  4.688978e-03  2.344489e-03   2236.256933
      131072         1  3.148261e-16  1.071906e-02  5.359530e-03   2078.749268
      262144         1  3.221995e-16  4.962492e-02  2.481246e-02    950.851308
      524288         1  3.278767e-16  1.065500e-01  5.327499e-02    934.910749
     1048576         1  3.286406e-16  2.352560e-01  1.176280e-01    891.434177

FFT_OPENMP:
  Normal end of execution.

03 May 2021 01:37:44 AM
03 May 2021 01:37:43 AM

FFT_OPENMP
  C/OpenMP version

  Demonstrate an implementation of the Fast Fourier Transform
  of a complex data vector, using OpenMP for parallel execution.
```

```
  Number of processors available = 2
  Number of threads =              2

  Accuracy check:

    FFT ( FFT ( X(1:N) ) ) == N * X(1:N)

           N      NITS    Error         Time          Time/Call     MFLOPS

           2      1000  7.859082e-17  1.557827e-03  7.789135e-07     12.838396
           4      1000  1.209837e-16  3.376961e-03  1.688480e-06     23.689941
           8      1000  6.820795e-17  4.982948e-03  2.491474e-06     48.164256
          16      1000  1.438671e-16  7.207870e-03  3.603935e-06     88.791829
          32       100  1.331210e-16  9.009838e-04  4.504919e-06    177.583657
          64       100  1.776545e-16  1.295090e-03  6.475449e-06    296.504554
         128       100  1.929043e-16  1.652002e-03  8.260012e-06    542.372115
         256       100  2.092319e-16  2.511024e-03  1.255512e-05    815.603360
         512        10  1.927488e-16  3.550053e-04  1.775026e-05   1298.008921
        1024        10  2.308607e-16  6.430149e-04  3.215075e-05   1592.498070
        2048        10  2.444507e-16  1.127005e-03  5.635023e-05   1998.927026
        4096        10  2.483274e-16  2.437115e-03  1.218557e-04   2016.811096
        8192         1  2.575054e-16  4.680157e-04  2.340078e-04   2275.479362
       16384         1  2.729354e-16  1.078129e-03  5.390644e-04   2127.537979
       32768         1  2.911847e-16  2.108097e-03  1.054049e-03   2331.581432
       65536         1  2.827259e-16  4.719973e-03  2.359986e-03   2221.572214
      131072         1  3.148261e-16  1.071286e-02  5.356431e-03   2079.952115
      262144         1  3.221995e-16  4.912400e-02  2.456200e-02    960.547139
      524288         1  3.278767e-16  1.060731e-01  5.303657e-02    939.113511
     1048576         1  3.286406e-16  2.354889e-01  1.177444e-01    890.552410

FFT_OPENMP:
  Normal end of execution.

03 May 2021 01:37:44 AM
03 May 2021 01:37:43 AM

FFT_OPENMP
  C/OpenMP version

  Demonstrate an implementation of the Fast Fourier Transform
  of a complex data vector, using OpenMP for parallel execution.

  Number of processors available = 2
  Number of threads =              2

  Accuracy check:

    FFT ( FFT ( X(1:N) ) ) == N * X(1:N)

           N      NITS    Error         Time          Time/Call     MFLOPS

           2      1000  7.859082e-17  1.591921e-03  7.959604e-07     12.563439
           4      1000  1.209837e-16  3.486872e-03  1.743436e-06     22.943201
           8      1000  6.820795e-17  5.189896e-03  2.594948e-06     46.243705
```

```
           16      1000  1.438671e-16  7.329941e-03  3.664970e-06     87.313120
           32       100  1.331210e-16  9.269714e-04  4.634857e-06    172.605103
           64       100  1.776545e-16  1.232862e-03  6.164312e-06    311.470264
          128       100  1.929043e-16  1.636028e-03  8.180141e-06    547.667791
          256       100  2.092319e-16  2.418995e-03  1.209497e-05    846.632623
          512        10  1.927488e-16  3.669262e-04  1.834631e-05   1255.838391
         1024        10  2.308607e-16  6.899834e-04  3.449917e-05   1484.093744
         2048        10  2.444507e-16  1.150131e-03  5.750656e-05   1958.733012
         4096        10  2.483274e-16  2.460003e-03  1.230001e-04   1998.046426
         8192         1  2.575054e-16  5.040169e-04  2.520084e-04   2112.945122
        16384         1  2.729354e-16  1.070023e-03  5.350113e-04   2143.655691
        32768         1  2.911847e-16  2.151966e-03  1.075983e-03   2284.050855
        65536         1  2.827259e-16  4.737139e-03  2.368569e-03   2213.521823
       131072         1  3.148261e-16  1.065516e-02  5.327582e-03   2091.214973
       262144         1  3.221995e-16  4.947186e-02  2.473593e-02    953.793219
       524288         1  3.278767e-16  1.060669e-01  5.303347e-02    939.168396
      1048576         1  3.286406e-16  2.350190e-01  1.175095e-01    892.333080

FFT_OPENMP:
  Normal end of execution.

03 May 2021 01:37:44 AM
03 May 2021 01:37:43 AM

FFT_OPENMP
  C/OpenMP version

  Demonstrate an implementation of the Fast Fourier Transform
  of a complex data vector, using OpenMP for parallel execution.

  Number of processors available = 2
  Number of threads =              2

  Accuracy check:

    FFT ( FFT ( X(1:N) ) ) == N * X(1:N)

           N      NITS    Error         Time          Time/Call     MFLOPS

           2      1000  7.859082e-17  1.609087e-03  8.045435e-07     12.429409
           4      1000  1.209837e-16  3.396988e-03  1.698494e-06     23.550275
           8      1000  6.820795e-17  5.121946e-03  2.560973e-06     46.857188
          16      1000  1.438671e-16  7.274866e-03  3.637433e-06     87.974128
          32       100  1.331210e-16  9.119511e-04  4.559755e-06    175.448010
          64       100  1.776545e-16  1.225948e-03  6.129742e-06    313.226903
         128       100  1.929043e-16  1.629829e-03  8.149147e-06    549.750788
         256       100  2.092319e-16  2.448082e-03  1.224041e-05    836.573295
         512        10  1.927488e-16  3.519058e-04  1.759529e-05   1309.441249
        1024        10  2.308607e-16  6.599426e-04  3.299713e-05   1551.650035
        2048        10  2.444507e-16  1.117945e-03  5.589724e-05   2015.126477
        4096        10  2.483274e-16  2.453089e-03  1.226544e-04   2003.678008
        8192         1  2.575054e-16  4.649162e-04  2.324581e-04   2290.649225
       16384         1  2.729354e-16  1.062870e-03  5.314350e-04   2158.081369
       32768         1  2.911847e-16  2.117872e-03  1.058936e-03   2320.819883
```

```
         65536         1  2.827259e-16  4.670858e-03  2.335429e-03   2244.932117
        131072         1  3.148261e-16  1.029992e-02  5.149961e-03   2163.340857
        262144         1  3.221995e-16  5.016613e-02  2.508307e-02    940.593184
        524288         1  3.278767e-16  1.089141e-01  5.445707e-02    914.616990
       1048576         1  3.286406e-16  2.360680e-01  1.180340e-01    888.367720

FFT_OPENMP:
  Normal end of execution.

03 May 2021 01:37:44 AM
03 May 2021 01:37:43 AM

FFT_OPENMP
  C/OpenMP version

  Demonstrate an implementation of the Fast Fourier Transform
  of a complex data vector, using OpenMP for parallel execution.

  Number of processors available = 2
  Number of threads =              2

  Accuracy check:

    FFT ( FFT ( X(1:N) ) ) == N * X(1:N)

             N      NITS    Error         Time          Time/Call     MFLOPS

             2      1000  7.859082e-17  1.663923e-03  8.319616e-07     12.019785
             4      1000  1.209837e-16  3.525972e-03  1.762986e-06     22.688777
             8      1000  6.820795e-17  5.301952e-03  2.650976e-06     45.266344
            16      1000  1.438671e-16  7.541180e-03  3.770590e-06     84.867359
            32       100  1.331210e-16  9.591579e-04  4.795790e-06    166.812985
            64       100  1.776545e-16  1.319885e-03  6.599426e-06    290.934382
           128       100  1.929043e-16  1.725912e-03  8.629560e-06    519.145791
           256       100  2.092319e-16  2.536774e-03  1.268387e-05    807.324680
           512        10  1.927488e-16  3.669262e-04  1.834631e-05   1255.838391
          1024        10  2.308607e-16  7.100105e-04  3.550053e-05   1442.232134
          2048        10  2.444507e-16  1.132965e-03  5.664825e-05   1988.410785
          4096        10  2.483274e-16  2.501965e-03  1.250982e-04   1964.536213
          8192         1  2.575054e-16  4.730225e-04  2.365112e-04   2251.394147
         16384         1  2.729354e-16  1.052141e-03  5.260706e-04   2180.087637
         32768         1  2.911847e-16  2.124786e-03  1.062393e-03   2313.267843
         65536         1  2.827259e-16  4.734993e-03  2.367496e-03   2214.524930
        131072         1  3.148261e-16  1.032305e-02  5.161524e-03   2158.494350
        262144         1  3.221995e-16  5.014515e-02  2.507257e-02    940.986730
        524288         1  3.278767e-16  1.092560e-01  5.462801e-02    911.754901
       1048576         1  3.286406e-16  2.383981e-01  1.191990e-01    879.684955

FFT_OPENMP:
  Normal end of execution.

03 May 2021 01:37:44 AM
03 May 2021 01:37:43 AM
```

```
FFT_OPENMP
  C/OpenMP version

  Demonstrate an implementation of the Fast Fourier Transform
  of a complex data vector, using OpenMP for parallel execution.

  Number of processors available = 2
  Number of threads =              2

  Accuracy check:

    FFT ( FFT ( X(1:N) ) ) == N * X(1:N)

             N      NITS    Error         Time          Time/Call     MFLOPS

             2      1000  7.859082e-17  1.646996e-03  8.234978e-07     12.143324
             4      1000  1.209837e-16  3.525972e-03  1.762986e-06     22.688777
             8      1000  6.820795e-17  5.299091e-03  2.649546e-06     45.290784
            16      1000  1.438671e-16  7.557154e-03  3.778577e-06     84.687969
            32       100  1.331210e-16  9.667873e-04  4.833937e-06    165.496582
            64       100  1.776545e-16  1.270056e-03  6.350279e-06    302.348927
           128       100  1.929043e-16  1.679897e-03  8.399487e-06    533.365936
           256       100  2.092319e-16  2.536058e-03  1.268029e-05    807.552373
           512        10  1.927488e-16  3.650188e-04  1.825094e-05   1262.400577
          1024        10  2.308607e-16  6.558895e-04  3.279448e-05   1561.238566
          2048        10  2.444507e-16  1.121998e-03  5.609989e-05   2007.847015
          4096        10  2.483274e-16  2.430916e-03  1.215458e-04   2021.954004
          8192         1  2.575054e-16  4.601479e-04  2.300739e-04   2314.386522
         16384         1  2.729354e-16  1.051903e-03  5.259514e-04   2180.581764
         32768         1  2.911847e-16  2.100945e-03  1.050472e-03   2339.519181
         65536         1  2.827259e-16  4.652023e-03  2.326012e-03   2254.021377
        131072         1  3.148261e-16  1.023388e-02  5.116940e-03   2177.301471
        262144         1  3.221995e-16  4.993892e-02  2.496946e-02    944.872710
        524288         1  3.278767e-16  1.089821e-01  5.449104e-02    914.046735
       1048576         1  3.286406e-16  2.382231e-01  1.191115e-01    880.331174

FFT_OPENMP:
  Normal end of execution.

03 May 2021 01:37:44 AM
03 May 2021 01:37:43 AM

FFT_OPENMP
  C/OpenMP version

  Demonstrate an implementation of the Fast Fourier Transform
  of a complex data vector, using OpenMP for parallel execution.

  Number of processors available = 2
  Number of threads =              2

  Accuracy check:

    FFT ( FFT ( X(1:N) ) ) == N * X(1:N)
```

```
         N      NITS    Error         Time          Time/Call     MFLOPS

         2      1000  7.859082e-17  1.575947e-03  7.879734e-07     12.690784
         4      1000  1.209837e-16  3.341913e-03  1.670957e-06     23.938383
         8      1000  6.820795e-17  5.102873e-03  2.551436e-06     47.032330
        16      1000  1.438671e-16  7.205009e-03  3.602505e-06     88.827087
        32       100  1.331210e-16  9.169579e-04  4.584789e-06    174.490026
        64       100  1.776545e-16  1.221895e-03  6.109476e-06    314.265900
       128       100  1.929043e-16  1.611948e-03  8.059740e-06    555.849192
       256       100  2.092319e-16  2.394915e-03  1.197457e-05    855.145305
       512        10  1.927488e-16  3.650188e-04  1.825094e-05   1262.400577
      1024        10  2.308607e-16  6.501675e-04  3.250837e-05   1574.978840
      2048        10  2.444507e-16  1.127005e-03  5.635023e-05   1998.927026
      4096        10  2.483274e-16  2.471924e-03  1.235962e-04   1988.410785
      8192         1  2.575054e-16  4.708767e-04  2.354383e-04   2261.653665
     16384         1  2.729354e-16  1.083136e-03  5.415678e-04   2117.703443
     32768         1  2.911847e-16  2.137899e-03  1.068950e-03   2299.079182
     65536         1  2.827259e-16  4.796982e-03  2.398491e-03   2185.907809
    131072         1  3.148261e-16  1.045704e-02  5.228519e-03   2130.836488
    262144         1  3.221995e-16  5.067897e-02  2.533948e-02    931.074990
    524288         1  3.278767e-16  1.102531e-01  5.512655e-02    903.509506
   1048576         1  3.286406e-16  2.370181e-01  1.185091e-01    884.806657

FFT_OPENMP:
  Normal end of execution.

03 May 2021 01:37:44 AM
```

Outputs for 16 processors

2.2. Analysis

2.2.1. Exercise 1

For this exercise, we got practice in working with multiple processors. We assigned the program to execute on both 4, 8 and 16 processors and saw that command executed on each processor and got the "Hello World from processor n" message. This was quite simple and was incresting to learn how compiling with the Intel C compiler worked.

2.2.2. Exercise 2

Similar to the last exercise, we added a couple lines to the communicate.c file and used the Intel C compiler to have our program communicate between the processors and output the results of running with 4, 8 and 16 processors.

2.2.3. Exercise 3

This exercise was a little more different, we used a flag to compile our program. We simply had to write our equation that was given and compile using the -fopenmp flag.

2.2.4. Exercise 4

Similar to the last exercise, we used the -fopenmp flag while compiling the code.

Chapter 3

Conclusion

3.1. Summary

In this lab, we continued to stregnthen our skills with UNIX commands via the HPC, as well as learning about parallelization and utilizing multiple processors for a task.

3.2. Reflection

This lab was simpler than other ones. I wish we had fewer labs where we were just given the code, because there was less learning and more "copy-paste till it works" involved. While I understand that to fully learn these tasks takes time, it's still something I wish I had the time to learn.