**CSE-443/543: High Performance Computing**

**Homework #03**

Max Points: 64

**Objective**: The objective of this exercise is to:

* Gain experience developing a non-trivial parallel code using OpenMP
* Gain experience using the CSE C++ style guidelines
* Build experience with benchmarking parallel programs
* Gain experience using spreadsheets for scientific data analysis

Fill in answers to all of the questions in this document and provide the requested data in the associated spreadsheet. You may discuss the questions with your instructor and the TA. Note that there is a question in the Discussion section only for CSE543 students.

## Background

In class, you have learned about OpenMP and have done a homework assignment and several labs to gain experience compiling and running programs in OpenMP. For this homework, you will now write a non-trivial program from scratch and then implement it in OpenMP in two different ways.

The program you are writing will implement the [Discrete Fourier Transform](https://en.wikipedia.org/wiki/Discrete_Fourier_transform) (DFT). If you are not familiar with it, please review the linked Wikipedia article, section 37.1 in your textbook, or find other descriptions online or in a textbook. **Note:** You do **not** need to be an expert in Fourier transforms to do this lab, but rather you must understand it well enough to implement it correctly. Though this may feel strange, in the future you will very likely be developing programs for clients who have a deep understanding of their business that you will not be able to match. Your job will be to understand the requirements and write a program that implements them correctly. Consider this practice for the future.

You should also know that for many situations, the DFT is not the most efficient way to compute the Fourier transform. It is an algorithm and a different approach, the Fast Fourier Transform (FFT), is . The FFT is, however, more complex to implement and, except for large values of N, can actually be too fast to make it easy to time. Make sure you understand this: I want you to implement the **DFT**, **not** the **FFT** for this assignment.

In this homework, you will be writing a program to calculate the DFT of a sine function, parallelizing the code using OpenMP. The code below can be used to initialize the original function. The Fourier transform is designed to handle complex functions, i.e. functions that have both a real and an imaginary component. The original function I am giving you only has a real component but you should write your code so that it could act on a complex function. Note that although C++ does have a complex data type, our version of OpenMP cannot parallelize code that uses it. So you will implement complex numbers via two real numbers.

void initialize(int N, double oR[], double oI[]) {

for (int n = 0; n < N; n++) {

double t = 2.0\*M\_PI\*(n - N/2);

oR[n] = sin(100.00\*t/N);

oI[n] = 0.0;

}

}

Some comments and hints:

1. Your code should take N as a command line argument and you should use N = 40000 for the final version you submit for this homework.
   1. This means that you should run it as, for example, ./fourier\_omp\_inner 40000
2. If your computation is correct, the real component of the DFT should essentially be zero, with just numerical noise, while the imaginary component should be zero (with noise) except for spikes at k = 100 and k = 39900. The height of the spikes should be -20000 and 20000 respectively.
   1. While you are debugging your code you should run it for a smaller value of N to save you time. If you pick N = 1000 you will get spikes at k = 100 and k = 900 with spike values of -500 and 500 respectively.
3. You should develop the code as a serial program first, then make the modifications for running in parallel. This will ensure that your code is implementing the DFT correctly. There are only minor changes required to make the code parallel.
   1. Writing this code from scratch will be the most time consuming part of the homework assignment. And, of course, the code you submit must be your own work, though you may refer to the Wikipedia article linked above or other sources to get information on the algorithm.
4. The heart of the Fourier transformation code will have a set of nested loops, an outer one with k ranging from 0 to N-1 and an inner one with n ranging from 0 to N-1. For this homework you will be ultimately be developing two versions of the code, one that parallelizes the inner loop and one that parallelizes the outer loop.
5. Your code must adhere to [CSE Programming Guidelines](https://miamioh.edu/cec/academics/departments/cse/academics/programming-style/index.html) for C++. You can test this by running your program through the [cpplint.py](http://cec.miamioh.edu/files/cpplint.py) style checker. **To get full credit for this assignment your code must pass the following checks:**
   1. No errors when checked with cpplint.py
   2. No errors or warning messages when compiled with these options

–O3 –std=c++ -Wall -fopenmp

* 1. It must give the correct answer.

1. In order to pass the cpplint.py check you will need to do your DFT computation in its own method, not the main method.
2. As with previous labs and homework assignments, you may do code development and testing on any Linux system, but your final timing runs must be performed on the BEN002 systems.
3. When your code is complete, your timing runs should take between 25-30 minutes to complete in total. You should coordinate with the person sharing your lab computer to make sure you are not doing timing runs at the same time. I would also suggest doing the timing runs in steps 4 and 8 below in two different sessions so each session is only taking about 15 minutes.

## Experiment

1. If you are doing development work on a system outside of the BEN002 lab, make sure that you’ve downloaded the [cpplint.py](http://cec.miamioh.edu/files/cpplint.py) style checker so you have it available to verify your adherence to CSE coding standards. It is already installed on all BEN002 lab machines.
2. Write a serial C++ program that implements the Discrete Fourier Transform as described above.
   1. Run your code through the cpplint.py program to check for style errors
   2. You should compile your code with our standard options:

–O3 –std=c++14 -Wall

* 1. Test your program for N = 1000 and ensure that you are getting the correct answer before moving to step 3.

1. Make a copy of your serial program called fourier\_omp\_inner.cpp and modify it to use OpenMP to parallelize the inner loop of the DFT.
   1. Run your code through the cpplint.py program to check for style errors
   2. You should compile your code with these options:

–O3 –std=c++14 -Wall -fopenmp

* 1. Test your program for N = 1000 and ensure that you are getting the correct answer before moving to step 4.

1. You should run your parallel code for 1 through 8 threads for N = 40000. For each value of the number of threads you should perform 8 runs, timing the program with /usr/bin/time and then enter your raw data for elapsed time and CPU time (user + system) in the appropriate table in the “Parallelizing Inner Loop” tab of the Homework03 spreadsheet.
2. You should compute the average elapsed times their standard errors, the average CPU times and their standard errors entering the results in the appropriate table.
3. You should then compute the parallel speedup (Sn), parallel efficiency (En) and parallel cost (Pn) using the average values you computed.
4. Make a second copy of your serial program called fourier\_omp\_outer.cpp and modify it to use OpenMP to parallelize the outer loop of the DFT.
   1. Run your code through the cpplint.py program to check for style errors
   2. You should compile your code with these options:

–O3 –std=c++14 -Wall -fopenmp

* 1. Test your program for N = 1000 and ensure that you are getting the correct answer before moving to step 8.

1. You should run your parallel code for 1 through 8 threads for N = 40000. For each value of the number of threads you should perform 8 runs, timing the program with /usr/bin/time and then enter your raw data for elapsed time and CPU time (user + system) in the appropriate table in the “Parallelizing Outer Loop” tab of the Homework03 spreadsheet.
2. You should compute the average elapsed times their standard errors, the average CPU time and their standard errors entering the results in the appropriate table.
   1. You will probably find it convenient to use spreadsheet formulas to compute these directly from the raw data tables above.
3. You should then compute the parallel speedup (Sn), parallel efficiency (En) and parallel cost (Pn) using the average values you computed.
4. When you are done, answer the question in the Discussion section below. Note that there is an additional question only for CSE543 students.

# Apparatus

The experiment documented in this report was conducted on the following platform (fill in the two lines of the Details column using information determined in your shell script):

|  |  |
| --- | --- |
| Component | Details |
| CPU Model | Intel(R) Core(TM) i7-4790 CPU @ 3.60GHz |
| Main Memory (RAM) size | 8071532 kB |

# Observations

Enter the raw timing data from your runs into the Google spreadsheet.

# Analysis

Compute the requested quantities and enter them into the Google spreadsheet.

# Discussion

You should have observed a statistically significant difference in run times when comparing your programs that parallelized the inner and outer loop of the DFT. Demonstrate that using data from your experiment and appropriate statistical methods.

|  |
| --- |
| Using Student’s T-Test, I performed a statistical significance test on the average CPU consumption and elapsed time per thread between parallelizing the inner and outer loop of the original code block.  This is saved on the Statistical Inference page of my excel sheet.  For threads 1 through 3, my results suggest there is no significant, or an indeterminate significance between the outer and inner approaches with respect to speed. On the side of CPU consumption, there is no/an indeterminate amount of statistical difference for threads 1 and 2.  The remainder of these data points are significant, with the majority of the data indicating that the outer loop is a much faster approach, as well as consuming much less CPU in the process.  The most interesting data point occurs at thread 5, where the outer approach gains a > 2.5 second approach over the inner approach, at close to a 30% reduction in CPU consumption simultaneously.  The outer approach seems like a clear winner for this experiment |

### CSE 543 students only:

Using data from your experiment, and applying Amdahl’s law, estimate the fraction of your code that is not parallelized (f). In addition to providing the estimate, explain how you determined it.

|  |
| --- |
|  |

# Submit files to Canvas

When you complete the lab, download this document from Google Drive as a Microsoft Word (.docx) file with the naming convention Homework03 - MUid.docx (example: Homework03 - ferrenam.docx).  When you complete the homework, download your Google spreadsheet file as a Microsoft Excel (.xlsx) file with the naming convention Homework03 – Results - MUid.xlsx (example: Homework03 - Results - ferrenam.xlsx)

Then, submit the following files to Canvas:

1. The Microsoft Word file you downloaded from Google Drive.
2. The Microsoft Excel file you downloaded from Google Drive.
3. Your fourier\_omp\_inner.cpp program.
4. Your fourier\_omp\_outer.cpp program.