CSC 1300 Intro to Problem Solving and Computer Programming

LAB 9, FALL 2024
DATA PARALLELISM
LAB ASSIGNMENT DIRECTIONS

REQUIRED SKILLS & KNOWLEDGE

1. Lambda functions
2. C++ STL Vectors library
3. C++ Algorithm library (for_each() function and sort() function)
4. Data parallelism

ASSIGNMENT OBJECTIVE

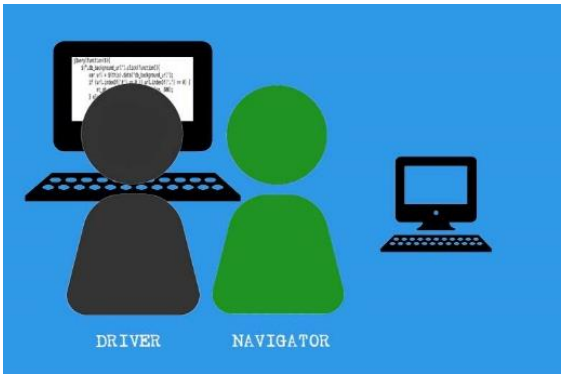
You will be able to write a C++ program that processes data in parallel.

DESCRIPTION

- You are given a program that creates a vector, fills the vector elements with strings read in from a text file (words.txt), changes the vector elements, and sorts the vector elements alphabetically in ascending order.
- The given program also calculates the number of seconds (runtime) the program takes to process and sort the vector elements.
- The given program is written with sequential instructions.
- Your job in this assignment is to write a C++ program that performs the same activities with parallel algorithms instead of sequential.
- You will then run both programs, record the execution run times of each, and then document which version ran faster on your machine.

PAIRED PROGRAMMING OPTION

You may complete this lab assignment alone or you have an OPTION to complete this lab with a lab partner using paired programming techniques. If you choose to pair program, follow the directions in this section. Your first step is to **exchange preferred contact information** just in case you are unable to complete the lab during lab and need to meet outside of lab class to finish.



SUBMISSION IN ILEARN

You will both upload the same exact zip file to your Lab 3 assignment in ilearn. Each source file should have both of your names in the comment block at the top. Both students will receive the same feedback and grade.

HOW TO PAIR PROGRAM

One of you can start writing (or debugging) the initial code (DRIVER) while the other reviews and suggests improvements (NAVIGATOR). **Take turns regularly (every 10 to 15 minutes)** to ensure both of you are actively involved.

WARNING ABOUT ACADEMIC MISCONDUCT

Programming assignments in this course are your exams and should be only YOUR work. **You are not allowed to get help from friends, family, or other students**. **You are also not allowed to use websites other then ZyBooks or generative AI such as ChatGPT or CoPilot to assist in writing your code.** If I suspect generative AI was used in your code, I will fill out a charging document to charge you with academic misconduct. If the charge stands after review from the Academic Misconduct Committee, you will receive a zero for your grade and this charge will be on your permanent student record. Being charged with academic misconduct more than once can result in expulsion from Tennessee Tech. This is a very serious matter, which is why I am stressing it in every assignment in addition to the syllabus. In more advanced CSC courses, using generative AI as a tool is encouraged, but in this course where you are learning foundational aspects of coding and problem solving, it can't be used.

INSTRUCTIONS

NEW C++ LIBRARIES YOU WILL USE

```
#include <vector>
```

This library is required to create vectors, use vectors, and call vector functions in your program. Vector is not a built-in data structure like an array and therefore, requires an additional library.

```
#include <algorithm>
```

This assignment uses two functions that come from the algorithm library: **`for_each()`** https://en.cppreference.com/w/cpp/algorithm/for_each and **`sort()`** https://cplusplus.com/reference/algorithm/sort/.

```
#include <parallel/algorithm>
```

The parallel version of for_each() and sort() are available in the libstdc++ parallel mode library. When using this library, you must use the -fopenmp and -D_GLIBCXX_PARALLEL compiler flags. More information can be found here: https://gcc.gnu.org/onlinedocs/libstdc++/manual/parallel_mode_using.html

```
#include <omp.h>
```

This library is required to use OpenMP, which is a way we can perform parallel computation for C, C++, and Fortran languages. More information on this library can be found on the OpenMP website at https://www.openmp.org/

PROGRAM SPECIFICATIONS

1. Download **sequential.cpp** and **words.txt** from the lab 9 assignment in ilearn and save it in your **Lab 9** folder. **If you are on a Mac**, you will also need to download **run.sh**.

2. **Review the sequential.cpp program to see what is happening in this program.**
   This program does the following:
   a. Reads 466,537 words from the **words.txt** text file.
   b. Start a timer (needed to calculate runtime of the for_each and sort functions).

c. Uses a for_each loop to iterate through the string vector and does the following for each element in the vector.
    i. Uses a for loop to count the number of characters in the string.
    ii. Converts the number of characters to a string.
    iii. Prepends the number of characters to the front of the string.
    Example: if the string is "apricot" then the resulting string after running this code would be "7_apricot" and "cat" would be changed to "3_cat"
d. Uses a sort function to sort the vector in ascending order.
e. Stops the timer.
f. Calculates the runtime.
g. [optional] – you can uncomment out line 61 that is a for_each loop to print all the words read in from the file to the screen by calling the print lambda function on line 28.
h. Prints the computation time to the screen.

3. Open your **Lab 9** folder in VS Code and create a new file in this same folder and name it **parallel.cpp**.

4. Create a comment block at the top of the source file that contains the filename, author, date, and purpose of the program.

5. Put your #includes and using namespace std after the comment block. Include the following libraries:

```
#include <iostream> //used for input & output
#include <fstream> //used to read from text file
#include <vector> //used to create & use a vector data structure
#include <ctime> //used for computing runtime
#include <parallel/algorithm> //used for the for_each and sort functions
#include <omp.h> //used for __gnu_parallel
```

6. Copy the functions from **sequential.cpp** and paste them in **parallel.cpp**. You will be using the same code but making some parts of the code run in parallel (at the same time) instead of serially (one at a time).

7. Explicitly set the number of **threads**. A thread is a logical branch of the program which can be executed in parallel with other threads of the same program. You will add the code below to set the number of threads. In this code, you are setting the number of threads to 30. If you use too many threads (try 1,000), then you will see a slower runtime of the program due to the overhead of separating the tasks into threads and then combining the solution back together. If you put too few threads (try 2), then you will not see much speedup at all. For this program, about 30 threads will give you the best speedup.

```
const int threads_wanted = 30;
omp_set_dynamic(false);
omp_set_num_threads(threads_wanted);
```

8. Modify the **for_each** loop to accomplish data parallelism. The only difference in the sequential.cpp version and the parallel.cpp version is that you will add __gnu_parallel:: before for_each so that the first line of the for_each loop will be as follows.

```
__gnu_parallel::for_each(str.begin(), str.end(), [](string &n) {
```

9. Modify the **sort** function to accomplish data parallelism. The only difference in the sequential.cpp version and this version is that you will add __gnu_parallel:: before sort.

```
__gnu_parallel::sort(str.begin(), str.end());
```

10. Your output should look like the sequential.cpp output but should print "Computation time for parallel = X.XXX seconds." Where X.XXX is replaced by the actual computation time in seconds.

11. To **compile** your code:
    a. **Windows**:
        i. Navigate to your lab 9 folder in your command prompt
        ii. Once there, type
        `g++ parallel.cpp -std=c++20 -fopenmp -D_GLIBCXX_PARALLEL`
    b. **Mac:**
        i. Navigate to your lab 9 folder in a terminal
        ii. Once there, type `sudo chmod 701 run.sh` (note: you may have to enter in your password)
        iii. Type `./run.sh` (note: you may have to enter in your password here, too)

12. **Run** your program. Fix any issues you see. Remember to recompile your code (step 10) before you run it again if you make changes to the code.
    MAC command to run program- `./a`
    WINDOWS command to run program- `a`

13. Once your program is compiling and working as expected, create a text file named **results.txt** and answer the following questions in this text file:
    a. How many cores/processors are on your computer?
        i. **Windows**: Press Ctrl + Shift + Esc to open Task Manager, then select the Performance tab. You can also press Ctrl+Alt+Delete, select Task Manager, click More details, then click Performance and CPU. Look for the number of cores on this screen.
        ii. **Mac**: Click the Apple icon, select About This Mac, then select System Report. In the System Information window, look for the Total Number of Cores.
    b. What was the computation time when you run the sequential.cpp program?
    c. What was the computation time when you run your parallel.cpp program?
    d. Now, change the number of threads to **1000** in **parallel.cpp** by changing the 30 to 1000 for the **threads_wanted** variable. Then, compile & run the program again. What was the computation time?
    e. Now, change the number of threads to **2** in **parallel.cpp** by changing the 1000 to 2 for the **threads_wanted** variable. Then, compile & run the program again. What was the computation time?
    f. The sample output was created with a computer that has 14 cores. Did you get the same numbers as the sample output? Why or why not?

SAMPLE OUTPUT

SERIAL.CPP

Note: You will most likely not get the same compilation time as we did in this sample output because runtime is computer-dependent and is based on many factors. When running your executable files, ensure all other software is closed to get the best runtime possible for both programs.

```
********************************************************************************
Computation time for sequential = 0.156 seconds.
```

PARALLEL.CPP

Note: this is the computation time we got when running the program with 30 threads.

```
********************************************************************************
Computation time for parallel = 0.06 seconds.
```

FILL OUT THE LAB REPORT

You will fill out this lab report for every lab and it is part of your grade. To get credit, you must upload a screenshot of the confirmation page to this lab assignment. Name your screenshot **lab9ReportProof**.
Lab Report Link: https://tntech.co1.qualtrics.com/jfe/form/SV_d6BGc6kzQdSvBmS

WHAT TO TURN IN

Create a zip file named **YOUR-TTU-USERNAME_lab9** containing the following .cpp files and upload it to ilearn. Replace YOUR-TTU-USERNAME with your TTU username . Example: **acrockett43_lab9.zip**

- sequential.cpp
- parallel.cpp
- words.txt
- results.txt
- Lab9ReportProof