**Project 1: Introduction (100 points)**

**Problem 0. [10 points for CS4473 and CS5473]**

Students must comply with the code specifications and submission guidelines in order to receive all the points for this section. Up to 10 points will be deducted for non-compliance with the code specifications or submission guidelines no matter whether the solution is correct or not. A rubric is available in Canvas under the assignment.

**Submission guidelines:**

Please submit **2 files** in Canvas.

* a PDF report file (filename: **Lastname\_Firstname\_Project\_1\_Report.pdf**)
* a ZIP file (filename: **Lastname\_Firstname\_Project\_1.zip**) with two sub-directories containing the required files in their specified filenames:
  + Problem\_1/
    - my\_number\_#######\_stdout.txt
    - my\_number\_#######\_stdout.txt (After editing my\_number.c)
  + Problem\_2/
    - serial\_mult\_mat\_vec.c
    - Makefile

**Note:** We will use auto-grading scripts to test your code and check the outputs from your code. So, it is important for you to follow the instructions, including the names of the directories and files. An example on how to create the zip file is available on the Canvas assignment page.

This directory will contain test data for Problem 2 and an auto-grading script.

**How to Get the Autograder for Problem 2:**

* The autograder and the test data is located on Schooner at:  
  /home/oucspdnta/current/test\_data/Project\_1\_Tests/
* To copy the autograder and the test data to your own home directory, run the command:  
  cp -r /home/oucspdnta/current/test\_data/Project\_1\_Tests/ .
* This will copy over the test data, in the Problem\_2/ folder, and the autograder from the Autograder/ folder
* The Autograder/ folder contains three files:
* autograder\_individual.sbatch  
  This is the file that you use in order to run a job on Schooner. (NOTE: Do NOT run programs on the command-line on Schooner, always submit jobs. Otherwise, this can crash the Schooner login node.). Please don’t forget to change the directory and the email address in this file.
* autograding\_base.py  
  This is the actual script that has the base autograder code. This script will call your program, and then test with the various test inputs
* autograding\_project\_1.py  
  This is the script that uses the base autograder specifically for project 1 problem 2.

We will run the auto-grader on Schooner to test your code. Please do not only test your code on your own computer. Even if the code may run fine on your computer, it may not run correctly on another computer system. Please make sure your programs can pass the auto-grader on Schooner before you turn it in.

**Problem 1. [30 points for CS4473 and CS5473]**

In this course, you will need to use the Schooner supercomputer at OU to work on your projects. Please reset the password on your assigned Schooner user account (**oucspdn###**).

After you have access to the Schooner supercomputer, please complete the following exercises created by Dr. Henry Neeman, the Director of OSCER:

**oscer\_hpcexercise\_learningbatch\_schooner\_20190827.docx**, which is in the **Problem\_1** folder in the **Project\_1\_Problems.zip** file. Please compile and run “Guess the number” program on Schooner. Inside the docx file, you will find the details.

For further information, read the instructions about how to run jobs on Schooner <https://ou.edu/oscer/support>. Below are a list of useful tutorials:

* “Logging in to OSCER Computers via Secure Shell” (<https://ou.edu/oscer/support/machine_access>).
* “Uploading & Downloading Files via Secure Copy” (<https://www.ou.edu/oscer/support/file_transfer>).
* “Running jobs in Schooner:

(<https://www.ou.edu/oscer/support/running_jobs_schooner>)

The Schooner user account assigned to you has access to the following two special queues that may have higher priorities and lower wait time than the normal queue and the debug queue:

* #SBATCH ––partition=oucspdn\_cpu

This queue contains CPU-only nodes, which should be used in the OpenMP module and the MPI module of this course. Your job can request only 1 or 2 nodes and can run only for 15 minutes

* #SBATCH ––partition=oucspdn\_gpu

This queue contains GPU nodes, which should be used in the GPU module of this course. Your job can request only 1 node and can run only for 15 minutes:

There are other queues available on Schooner for you to use. The debug queue has a time limit of 30 minutes. The normal queue has a time limit of 48 hours. You may try these two queues if your job requires more time. For more details about the queues on Schooner, please read:

<https://www.ou.edu/oscer/support/partitions>.

Learning outcomes:

This exercise will help you get familiar with the Schooner supercomputer.

What to submit:

In the PDF report, please answer the following questions about SLURM based on the information provided at <https://www.ou.edu/oscer/support/running_jobs_schooner>:

* How do you request 2 compute nodes to run a job? (5 points)
* How do you find the status of a job after it’s submitted? (5 points)
* Why should you NOT run a job on the login node? (5 points)

In the ZIP file, please include the output files of your jobs in the Problem\_1 directory as proof of work (15 points):

* Problem\_1/
  + my\_number\_#######\_stdout.txt
  + my\_number\_#######\_stdout.txt (After editing my\_number.c)

**Problem 2. [40 points for CS4473 and CS5473]**

In this course, we will use C for the OpenMP, GPU and MPI programming. This exercise will refresh your memory about C programming. Please write a serial C program to do the matrix-vector multiplication.

Implementation requirements:

* Because the input matrix and vector can be very large, you must use dynamic memory allocation (i.e., malloc and free) to store them in the heap. If your implementation stores the input matrix, the input vector, or the output vector in the stack, we will deduct 10 points.
* There are many ways to store the matrix in a dynamically allocated 2-dimensional array in C, including (a) using the row-major order or column-major order in a 1-dimentional array or (b) using an array of pointers to a set of 1-dimensional arrays. In this exercise, please store the matrix in a dynamically allocated 1-dimensional array in the row-major order (5 points).

To help you meeting the two implementation requirements, we have provided a toy C program that dynamically allocates a 2D array stored by row-major order. This C program is called array\_allocation.c under the Problem\_2 sub-directory.

Your matrix-vector multiplication program should take two input files, one for the input matrix and the other one for the input vector. All the numbers are integers. The two input files are provided in the comma-separated values (CSV) format. To prevent the numbers from running out of range, please declare the type of all variables as long int. The output vector should be saved in a CSV file.

Your program should be run using the following command line:

serial\_mult\_mat\_vec file\_1.csv n\_row\_1 n\_col\_1 file\_2.csv n\_row\_2 result\_vector.csv

The input parameters of your programs include:

* file\_1.csv: input CSV file for the input matrix
* n\_row\_1: number of rows in the input matrix
* n\_col\_1: number of columns in the input matrix
* file\_2.csv: input CSV file for the input vector
* n\_row\_2: number of rows in the input vector
* result\_vector.csv: output CSV file for the result vector

The starter code and the make file are provided for your reference in the Problem\_2 folder. Please keep the names of the C code and executable as serial\_mult\_mat\_vec.c and serial\_mult\_mat\_vec, respectively.

To use the Make command, you may need to first load GCC using the following command:

module load GCC

Our auto-grading script will unzip the zip files submitted by every student, compile the code using the Makefile, execute the program using the command specified above, and validate the output. Therefore, it is important for you to name the directories and files as instructed.

Please test your program on Schooner. Three test cases are provided on Schooner for you to test your program. As described on Page 1, the test data can be copied from /home/oucspdnta/current/test\_data/Project\_1\_Problems/Problem\_2. Grading will be conducted on our test of your program on Schooner using additional cases. Your program should be able to handle a matrix with up to ~20,000 rows and ~20,000 columns.

**Required File Structure for the autograder only:**

* As noted within the python script, the file structure of the project is necessary. The required file structure is:
  + test\_data/
    - Problem\_2/
      * [All test input and output files]
  + LastName\_FirstName\_Project\_1/
    - Problem\_1/
    - Problem\_2/
      * Makefile
      * serial\_mult\_mac\_vec.c
    - autograder\_project\_1.sbatch
    - autograding\_project\_1.py
  + autograder\_base.py

**How to Run the Autograder for problem 2:**

* **Running on Schooner:**
  + First, make sure to edit the sbatch file where indicated. You will have to edit your username, and the directories of where the script is located, and where to output the stdout and stderr files.
  + To submit the job, run the command:  
    sbatch autograder\_individual.sbatch
  + To see the status of the job, run the command:  
    squeue –u oucspdn###
  + Once the job completes, it will return the output and error messages in two files:  
    autograding\_########\_stderr.txt  
    autograding\_########\_stdout.txt  
    (Where the #s are replaced with the job number)
* **Running Locally:**
  + For Windows users, we recommend using Windows Subsystem for Linux (WSL) with Ubuntu to run code from the command-line.
  + Make sure that your file structure is correct, otherwise the autograder will not be able to find your files.
  + Navigate to the location of autograding\_individual\_1b.py, and then run either:  
    python autograding\_individual\_1b.py  
    or  
    python3 autograding\_individual\_1b.py
  + This will then test your program against the provided test data, and will output your score for each test, where a 1.0 for each test indicates that the test succeeded, and a 0.0 means the test failed.

Learning outcomes:

This is a warm-up exercise for the C programming. You will practice how to use dynamic memory allocation to create and use 1-dimensional and 2-dimensional arrays in the heap. Please learn to use pointers in a C program. You should pay attention to memory leak.

What to submit:

In the PDF report, please briefly describe the key differences between the stack and the heap. Your C program should have some variables in the stack and some other variables in the heap. Please use a few of these variables as examples and explain why they are stored in the stack or the heap. (10 points)

In the ZIP file for the Canvas submission, please submit your code and makefile in the Problem\_2 folder: (30 points)

* Problem\_2/
  + serial\_mult\_mat\_vec.c
  + Makefile

**Problem 3. [20 points for CS4473 and CS5473]**

After taking this course, you will practice parallelizing computer programs. Please review Module 0 - Chapter 1 - Section 3 and answer the following two questions about the performance of your future parallel algorithms.

**Problem 3A. (10 points)**

Based on theoretical analysis of the complexity of the serial algorithm and the parallel algorithm for matrix-vector multiplication, you have derived the following expected serial runtime () and parallel runtime () for the multiplication on a matrix using processors:

The parallel runtime is composed of the parallelized computation (), the unparallelized computation (), and the parallel overhead ().

Please compute the expected speedup and efficiency in the two tables below:

|  |  |  |  |
| --- | --- | --- | --- |
| Speedup | n = 100 | n = 200 | n = 400 |
| p = 8 |  |  |  |
| p = 16 |  |  |  |
| p = 32 |  |  |  |

|  |  |  |  |
| --- | --- | --- | --- |
| Efficiency | n = 100 | n = 200 | n = 400 |
| p = 8 |  |  |  |
| p = 16 |  |  |  |
| p = 32 |  |  |  |

Please discuss the changes in the expected speedup and efficiency as the matrix size and/or the processor count are increased. Please explain the changes based on the theoretic analysis of the parallel runtime.

What to submit:

In the PDF report, please provide the two completed tables and your discussion of the tables (10 points)

**Problem 3B. (10 points)**

The actual parallel runtime of your algorithm was benchmarked using real-world datasets. The benchmarking results are provided below:

|  |  |  |  |
| --- | --- | --- | --- |
| Runtime (seconds) | n = 100 | n = 200 | n = 400 |
| serial | 113 | 428 | 1638 |
| p = 8 | 19 | 62 | 223 |
| p = 16 | 14 | 36 | 116 |
| p = 32 | 11 | 23 | 63 |

Please compute the observed speedup and efficiency in the two tables below:

|  |  |  |  |
| --- | --- | --- | --- |
| Speedup | n = 100 | n = 200 | n = 400 |
| p = 8 |  |  |  |
| p = 16 |  |  |  |
| p = 32 |  |  |  |

|  |  |  |  |
| --- | --- | --- | --- |
| Efficiency | n = 100 | n = 200 | n = 400 |
| p = 8 |  |  |  |
| p = 16 |  |  |  |
| p = 32 |  |  |  |

Please discuss the changes in the measured speedup and efficiency as the matrix size and/or the processor count are increased.

What to submit:

In the PDF report, please provide the two completed tables and your discussion of the tables (10 points).