Project - MATH 458: SCIENTIFIC COMPUTING II
Instructor: Dr. Elliot S. Menkah, esmenkah@knust.edu.gh

July 9, 2024

All major data buffers should be stored in the **heap** memory.
This project is in two (2) parts and you would be submitting two (2) sets of files even though they may be similar. First set for the first part of the project and Second set for the second set of the project.

Add a script that will compile each parts of project this, run the code and make the necessary plot from the data generated, if necessary, either with **gnuplot** or python(**matplotlib**).
Adding a **Makefile** in instead of a script is a **bonus**.

NOTE: This project, requires you to submit seven files (7) for the 1^{st} part and eight (8) files for the 2^{nd} part. **Six (6)** source files, including the script/Makefile to compile, run and generate or show plots, and one (1) **project report(.pdf)** file. Please precede **EVERY FILE NAME** with your Exam number and an underscore.
DO NOT submit more files than have mentioned in this instructional document
DO NOT put spaces in file names: Any such file won't be looked at for assessment
PLEASE DO NOT submit a binary file

Eg. If your exam number is 9326519, then your main.c file would be **9326519_main.c**

Project - Part I

1. Write a serial code, in the C programming language, to have the following defined function prototypes that perform the implied mathematical operations;

   a. Vector Vector Multiplication:

   Listing 1: Vector-Vector multiplication Function prototype

```
double vector_vector(int m, int n, double * vecA, double * vecB)
```

   b. Matrix Vector Multiplication:

   Listing 2: Matrix-Vector multiplication Function prototype

```
void matrix_vector(int m, int n, double * matA, double * vecA, double * matC)
```

   c. Matrix Matrix Multiplication:

Listing 3: Matrix-Matrix Multiplication Function prototype

```
void matrix_matrix(int m, int n, double * matA, double * matB, double * matC)
```

Your matrix should be a square-matrix of dimension m x n where the value of both m and n is attained from the **command line**. You could use a single marco variable, **N**, to catch and set them.

2. Include a time function that would be used to time your operations.
3. Implement your code such that the data buffers are all populate internally.
4. Organize your project into appropriately and respective source files(.c and .h) so to categorize your code such that you have a **main function**, *where functions are being called and used*, functions implementations(.c) and function prototypes (.h). Please group like functions. vector-vector relation function should be in separate library files and matrix-matrix related functions. Essentially, you should have
   - main.c
   - vector.c , vector.h
   - matrix.c , matrix.h
5. Use your code to test for the time-to-solution of all the operations whiles doubling the values of **N**: From 5000, 10000, 20000, 40000 and write the results in a data file, **examNumber**_serial.txt . Add a ploting script that would be used to generate the plot of the performance. Tabulat your output just as has been demonstrated in table 1.

Submit the files for this part and move on to the next part.

Project - Part II

1. Write a **parallel** version, using MPI, of the vector-vector multiplication function, from Part I, in your **vector.c** library file and modify your main.c to use it there for a vector x vector operation.
2. Your code should be designed such that there is no duplication of data, since it's being generated internally, but rather a *root* rank is designated to generated all the data needed and apportion and send data-to-be-computed on to the respective ranks.
3. Your code should write out the size of N, number of cores, and computing time for the relevant mathematical operation function into a file **examNumber**_scale.txt .
4. Perform some scalability test on an N value of **2.5 x** 10^8 on varying the number of cores being used. Write a report on the performance whilst indicating **speed-Up**. You report should have scalability plots that include **strong** scaling charts which shows performance from a serial run, parallel runs from 2 cores to 8 cores.

   Scalabilty should be done on the HP desktops in the Laboratory that is designated as SCB-SF26. They have following specification:

   - Cores: 8 at 3.40 GHz per core.
   - RAM: 12 GB.

N	Core	V.V/s	M.V/s	M.M/s
2.5 x 10^8	1	8960	90000	50000
2.5 x 10^8	2	4480	-	-
2.5 x 10^8	3	3240	-	-
2.5 x 10^8	4	2240	-	-
2.5 x 10^8	5	1120	-	-
2.5 x 10^8	6	140	-	-
2.5 x 10^8	7	70	-	-
2.5 x 10^8	8	10	-	-

Table 1: 9326519_scale.txt

If you verify the specification of the machine and the data of dimension size N = **2.5 x** 10^8, cumulatively, does not fit into memory, let me known for any possible adjustment.

For any clarity you need and discussion you have on this task, you can bring it up in the class for deliberation.

BONUS

You can also challenge yourself by writing a distributed-memory parallel version of the matrix-vector and matrix-matrix multiplication operations functions.