Department of Computational and Data Sciences
DS226: ICAIML
October 10, 2022

Instructions

1. Writing your code

1) Download the zip file for the template code from Moodle and extract it. Inside, you will find two sub-folders `include` and `src`.
   Inside the `include` folder, you will find two header files:
   (a) `vectHandling.h`, which contains declarations for functions to handle vectors. You may inspect this file, but do not make any changes here.
   (b) `vectOps.h`, which contains declarations for the routines that you will be building. You can use this file to understand the return type of your routines and the inputs you will need to pass to these routines. Do not make any changes in this file.
   Inside the `src` folder, you will find several files with the definitions of the functions we wish to implement. As an example, `sumArray.c` contains the parallel implementation for computing the sum of elements of an array.
2) To begin with, open `dotProduct.c` and fill in your code in the section marked

```
/*
 *
 * Your code here
 *
 */
```

   Note that the function defined returns a double value, and takes pointers to the two vectors and their length as arguments. You cannot change the return type or arguments. This will be true for all the other function definitions, and you'll have to fill in your codes in the respective files in the same way.
3) Next, you'll write the code for Daxpy defined in `daxpy.c`. This function takes pointers to the first and second vector, the multiplier α, length of the vectors and a filename as the input. It performs the Daxpy operation on the first vector, and prints it to a file with the given file name.
4) Once you have your dot product and daxpy routines running, implement the L_2 norm error in `l2Norm.c`. This function takes pointers to the two vectors and their length as arguments. You can only use the dot product and daxpy routines (along with the `sqrt()` operator).
5) Finally, implement a matrix-vector multiplication in `matvectmult.c` using the dot product routine you have built. The function `MatVectMult` takes pointers to the matrix and vector, and prints the output vector to a file with the given filename.

2. Testing your code

1) The main function can be found in `assignment2.c`. The function `readParams` reads the data file `param.txt` to obtain names of the input files containing the vectors and matrix, and their dimensions. This way, you can execute your code for vectors of different sizes without having to compile again. Just copy the input files to your parent folder and fill in the details in `param.txt`.
2) The main function handles all the declarations and makes calls to `l2Norm` and `MatVectMult`.
3) Once you've written all parts of your code, you can simply go to the `src` directory and run the `makefile` to compile your project. The executable, `assignment2.out` can then be run in the parent folder.

```
cd src/
make
```

```
cd ../
./assignment2.out
```

When you run the main program, you should get `daxpy.txt` (from when Daxpy is called in the L2 error routine),`l2error.txt` and `matvectmult.txt` as outputs.

4) Repeat the same procedure for different size of vectors ($N = 10, 100, 10^4, 10^6, \cdots$) with varying number of threads (1,2,4,8). We have provided sample inputs and outputs for $N = 10, 10^4 and 10^6$ for you to test your code against.

3. Presenting your code

1) Run your code for varying number of threads (1,2,4,8) and measure the time required to execute the operative part of the code (without `printf`). Repeat this process a few times to get an average value of execution time, and plot this versus the number of threads. Assume your vectors to be of length N each and your matrix to be of dimension $N \times N$. Present the aforementioned results for $N = 10, 10^2, 10^4, 10^6$. Add your comments to these plots and upload this PDF to Moodle.
2) Your codes should be well documented. Use the code in `vectorHandling.c` as a template and comments to all functions in the same way.

4. Evaluating your code

1) We will use some test cases (similar to the sample inputs) on the code you've built to test the accuracy and speed of your code.
2) Codes will be judged on documentation, accuracy and speed of execution. Any deviation from the instructions given above will result in penalties.