**Sabancı University**

Faculty of Engineering and Natural Sciences

**CS406-531 Parallel Computing / Parallel Processing and Algorithms**

**Spring 2018-2019**

**Homework 1**

**Due: 24/03/2019 - 23:00**

|  |
| --- |
| **PLEASE NOTE:**  **Your program should be a robust one such that you have to consider all relevant programmer mistakes and extreme cases; you are expected to take actions accordingly!**  **You HAVE TO write down the code on your own.**  **You CANNOT HELP any friend while coding.**  **Plagiarism will not be tolerated!** |

1. **Introduction**

In this assignment, you will implement a Matrix Permanent Calculation code. The definition of matrix permanent can be found on [Wikipedia](https://en.0wikipedia.org/wiki/Permanent_(mathematics)). The purpose of the homework is implementing a **parallel** algorithm with **OpenMP** **as efficient as possible.** Note that the grading will be done based on the end-to-end execution times of implementations.

1. **Program Flow**

You will be reading matrices from provided files and calculate the *permanent* of the given matrix. The result will be printed out to standard output. The filename of the matrix must be passed to your program by command line (use argc/argv). The format of the input files will be as follows:

* Each matrix will be a square matrix
* Each file starts with a line containing a single integer N, representing the size of the matrix (# of rows & #of columns)
* Following N lines will contain N numbers each, representing the matrix elements.

The input files will be given in correct format and you do not need to implement any input checks.

1. **Permanent computation**

The best algorithm we know to compute the permanent is proposed by **Ryser** and uses the inclusion-exclusion principle. However, the naïve version of this algorithm has O(2n-1n2) complexity. You are strongly recommended to use the **Nijenhuis and Wilf** variant with O(2n-1n) complexity. The Matlab code for this algorithm is provided [here](https://ww2.mathworks.cn/matlabcentral/fileexchange/53784-matrix-permanent-using-nijenhuis-wilf-in-cmex) (which is in the *Functions* tab, **perman\_mat(A)** function). For clarity, a more detailed pseudocode of this variant is also provided below:

* **Input:** M(matrix), n(number of rows)
* **Output:** Permanent of M
  + lastCol = M[:, n-1] // the last **column** in matrix (**1xn** vector)
  + sumCol = sum(M[:]) // column-wise sum of the matrix (**1xn** vector)
  + x = lastCol – (sumCol / 2)
  + p = // p is the product of elements in x
  + **for** i **in** (0, **in Gray code order** // 1,3,2,6,7 ...
    - z = FindChangingBit() // 0,1,0,2,0 …
    - s = FindSign() // 1,1,-1,1,1 ...
    - prodSign = (-1)i  // Either -1 or 1 depending on i
    - x = x + s \* M[:z] // Either add or subtract the zth column.
    - p = p + prodSign \* // Either add or subtract the prod. of x
  + **return** (4 \* (n % 2) – 2 ) \* p

The **n**-bit Gray code order is an order of numbers in between 0 and 2n-1-1 in which two consecutive numbers only have a single bit difference. Hence, the naïve Gray order proceeds as:

0 , 1 , 3 , 2 , 6 , 7 …

0000, 000**1**, 00**1**1, 001**0**, 0**1**10, 011**1** …

The **FindChangingBit** function in the pseudocode finds the index of the changing bit for the current number in Gray order. For example from 1 to 3, the changing bit is bit 1 (the bits are zero-indexed where the LSB is bit 0). Similarly, for 6 to 7 the changing bit is bit 0.

The **FindSign** function decides the sign of the operation based on the changing bit. If the changing bit switches from 1 to 0, sign is -1 and if the changing bit is switching from 0 to 1 the sign is +1. The whole purpose of this operation is to implement the inclusion/exclusion mechanism for either adding or deleting the next value.

The **prodSign** variable gets either 1 or -1 based on the current value of **i**.

Your report will have equal important as the implementation. A report of at least two pages should be submitted with the code, including:

* A general explanation of the implementation
* Execution times, speedups and efficiency etc.
* Tricks done for parallelization and efficiency (scheduling, preprocessing etc.)
* How you compile & run your program

1. **Some Remarks**

In the grading process three things will be checked:

* Correctness of your implementation
* Speedup and efficiency values
* How well the report is written

The body of the code is given to you. Please just use the commented section in the code and **DO NOT CHANGE THE REST OF IT.** Your code will be compiled with **–O0** and **-O3** optimization flags with **1, 2, 4, 8, 16 threads**, so you are expected to include all these results in your project.

Finally, to check the correctness of your code, you can use an n x n one-matrix in which all the elements are 1. The permanent of such a matrix should be **n!** (which will exceed the maximum value after some point). You can also use the test matrices accompanying the source code to check the correctness.

**4. What and where to submit (PLEASE READ, IMPORTANT):**

**Please don’t forget to submit your code and the report together.** Your **REPORT** must be a **pdf** file (preferable prepared by LaTeX but MS Word converted pdf’s are also OK). It must contain the description of the optimizations you implemented, i.e., it must explain how you improved the performance, what was the timings before and after. You must do this for both of the –O0 and –O3 optimization options. Please see above what else do you need to include in the report.

The grading process is not automatic. However, the students are expected to strictly follow the guidelines in order to have a smooth grading process. If you do not follow these guidelines, depending on the severity of the problem created during the grading process, 5 or more penalty points are to be deducted from the grade. The name you're the source code file that contains your program must be **permanent\_hw1.cpp.** Similarly the report must be named as **report\_hw1.pdf.**

Put both of these files into a folder named

***SUCourseUserName\_YourLastname\_YourName\_HWnumber***

Your SUCourse user name is actually your SUNet username that is used for checking sabanciuniv e-mails. Do NOT use any spaces, non-ASCII and Turkish characters in the file name. For example, if your SUCourse user name is cago, name is Çağlayan, and last name is Özbugsızkodyazaroğlu, then the folder name must be:

***cago \_Caglayan\_Ozbugsizkodyazaroglu\_hw1***

Do not add any other character or phrase to the folder name. Make sure that it contains the last version of the source code and the report. Compress this folder using a zip program. Please use "zip" compression. **"rar" or another compression mechanism is NOT allowed.** **Please make sure that you include both of the files in the compressed folder.**

You will receive no credits if your compressed folder does not expand or it does not contain the correct files. The name of the zip file should be as follows:

***SUCourseUserName\_YourLastname\_YourName\_HWnumber.zip***

For example zubzipler\_Zipleroglu\_Zubeyir\_hw1.zip is a valid name, but

***hw1\_hoz\_HasanOz.zip, HasanOzHoz.zip***

are **NOT** valid names. **Submit via SUCourse ONLY!** You will receive no credits if you submit by other means (e-mail, paper, etc.).

Successful submission is one of the requirements of the homework. If, for some reason, you cannot successfully submit your homework and we cannot grade it, your grade will be 0.

Good Luck!

CS406-531 Team (Mustafa Kemal Taş, Kamer Kaya)