# CS5300 - Parallel & Concurrent Programming: Autumn 2024

## Programming Assignment 1: Measuring Matrix Sparsity

### Submission Deadline: 14th August 2024, 9:00 pm

**Goal:-** This assignment aims to count the number of zero-valued elements in a square matrix through static and dynamic mechanisms in C++.

**Details:-** Consider a square matrix A. The goal of the problem is to count the number of zero-valued elements in the matrix. Assume that the matrices are given in row-major order.

The number of zero-valued elements divided by the total number of elements is called the sparsity of the matrix.

We can divide the work by creating one thread to compute the number of zero-valued elements of a row or a collection of rows in the A matrix and then executing those threads on different processors in parallel.

One has to design three different techniques for computing the sparsity in matrix A in parallel:

- Chunk: The matrix A is divided into chunks of size *p = N/K*. Then thread1 will compute the values of the rows 1 to p of A; thread2 will compute the values of the rows p + 1 to 2*p; thread3 will compute the values of the rows 2*p + 1 to 3*p and so on. Thus, the thread $th_i$ will be responsible for computing the rows corresponding to chunk i.

- Mixed: Here, the rows of the A matrix are evenly distributed among the threads. Thread1 will be responsible for the following rows of the A matrix: 1, k+1, 2*k+1 ….,. Similarly, thread2 will be responsible for the subsequent rows of the A matrix: 2, k+2, 2*k+2 ….,. This pattern continues for all the threads.

- Dynamic: Here, the sequence of rows of matrix A will be allocated dynamically to the threads. Each thread increments a shared counter by a value rowInc (row Increment) to claim the set of rows of the A matrix it is responsible for. The value rowInc is similar to the chunk size (described above), where the chunk size was fixed *(p)*.

The main program will read **N, S, K, rowInc and the matrix A** from an input file. Here, the variable N represents the number of rows of A; S is the sparsity of the matrix A; K represents the number of threads; rowInc is the row Increment explained above.

**Input File:-** As mentioned above, the input will consist of four parameters: **N, S, K, rowInc and the matrix A**. Please name the input file as *inp.txt*

Please note various files of matrix A for all these experiments are provided. You may refer to the files to create inp.txt on this:- **Link**

*In this link, there are multiple sub-folders for each experiment. The filename A-B.txt represents the square matrix with A rows and B as the sparsity.*

**Output File:-** Your program should output the following in an outfile file named *out.txt: (a) Time taken to compute the sparsity (b) Total Number of zeros in the matrix (c) Number of zeros counted by each thread*

Sample Output:-

Time taken to count the number of zeros: 123 milliseconds
Total Number of zero-valued elements in the matrix: 592
Number of zero-valued elements counted by thread1: 21
Number of zero-valued elements counted by thread2: 13
..
..
Number of zero-valued elements counted by threadK: X
..
..

**Report Details:-** As part of this assignment, you have to prepare a report describing your program's low-level design. Further, you have to measure the performance of the algorithms. You will have to create two plots to measure the time taken:

**1. Time vs. Size, N:** In this graph, the y-axis will show the time taken to compute the sparsity using all the approaches mentioned above: chunks, mixed and dynamic. The x-axis will be the values of N (size on input matrix) varying from 1000 to 5000 (size of the matrix will vary as 1000*1000, 2000*2000, …., 5000*5000). Please fix S at 40% and K at 16 for all these experiments. Please have the rowInc fixed at 50 for dynamic.
**Also, create a table showing the total number of zero-valued elements for each matrix size.**

**2. Time vs. Number of threads, K:** Like the previous graph, the y-axis will show the time taken to compute the sparsity using all the approaches mentioned above: chunks, mixed and dynamic (similar to experiment 1). The x-axis will be the values of K, the number of threads varying from 2 to 32 (in powers of 2, *i.e.*, 1,2,4,8,16,32). All experiments must have N fixed at 5000, S at 40%, and rowInc at 50 for dynamic.

**3. Time vs. Sparsity, S:** Like the previous graph, the y-axis will show the time to compute the zero-valued elements. The x-axis will be the sparsity of the graph varying from 20% to 80% (20, 40, 60, 80). Please keep N=5000, K=16 and rowInc=50 for this experiment.
**Also, create a table showing the total number of zero-valued elements for each sparsity value.**

**4. Time vs. row Increment, rowInc:** Like the previous graph, the y-axis will show the time to compute the sparsity. The x-axis will be the rowInc varying from 10 to 50 (10, 20, 30, 40, 50). Please keep N=5000, S=40% and K=16 for this experiment. **Note: This experiment is relevant**

**only for the dynamic case**.

*To handle temporary outliers, ensure that each point in the above plots is averaged over 5 times. Thus, you will have to run your experiment 5 times for each point on the x-axis.*

Note that all the plots will have three curves as mentioned above: chunks, mixed and dynamic (except plot 4, which will have only the dynamic curve).

You must write the observation you gained based on the plots described above and mention any anomalies in the report. The matrix data for performing these experiments are in the link shared above: **Link**.

**Extra Credit:-** As mentioned above, one can develop another approach for the parallel computation of the zero-valued elements in A matrix based on mixed, chunks, dynamic or some other technique. Please work on developing such a different technique.

To show the effectiveness of this technique, you will have to perform the experiments and generate the graph plots 1, 2, 3 and 4 mentioned above. Each plot will contain three curves (except plot 4): chunks, mixed and dynamic (and any other appropriate technique developed by you).

This extra-credit option will be evaluated only if you have implemented it. You will not be given extra credit if you describe this idea without implementing it.

**Deliverables:-**

You will have to submit the following as a part of this assignment:

1. A report describing the following:
(a) Low-level design of your program. You must explain the implementation of the techniques in chunks, mixed and dynamic. If you implemented the extra credit option, you will have to explain the details of how that technique works.

(b) The graph plots described above and their analysis. Please name this report file as Assgn1-Report-<Roll No>.pdf

2. Prepare a README file that contains the instructions on how to execute your submitted file. The file should be named Assgn1-ReadMe-<Roll No>.txt

3. Name the source code file in the following format: Assgn1-Src-<Roll No>.cpp
In case of multiple source files, name the source code files accordingly, like Assgn1-Chunk-<Roll No>.cpp, Assgn1-Mixed-<Roll No>.cpp, Assgn1-Dynamic-<Roll No>.cpp, etc.

4. Combine the source code, report, input file, and README as a zip archive file and name it Assgn1-<Roll No>.zip. Upload this zip file.

Please see the instructions given above before uploading your file. **Please follow this naming convention.** Your assignment will **NOT be evaluated** if there is any deviation from the instructions posted there, especially with regard to the naming convention.

**Marking Scheme:**

The evaluation scheme for this assignment will be as follows:

| S. No | Section | Marks |
|---|---|---|
| 1. | Program Design & Report | 50 |
| 2. | Program Execution | 40 |
| 3. | Code Indentation and Documentation | 10 |
| 4. | Extra Credit | 30 |
| **Total** | | **130/100** |

It can be seen that with the extra credit, one can get 130 marks out of 100. Please note that, as mentioned above, the extra credit portion will be evaluated only if the results are also shown. Otherwise, no marks for extra credit will be awarded.

**Please keep in mind that all submissions are subject to plagiarism checks.**