DS 221 - Introduction to Scalable Systems
Assignment 2

Shriram R.
M Tech (CDS)
06-02-01-10-51-18-1-15763

October 6, 2018

1 2D Matrices

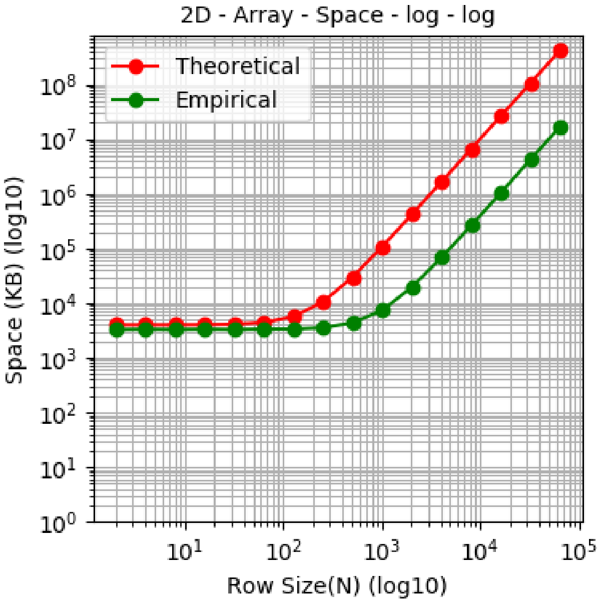
The 2D matrix abstract data structure has been implemented using 2D array and compressed sparse matrix representation (CSR) and the space and time complexities have been analyzed and empirically verified. The following sections will cover the analysis and empirical results in detail.

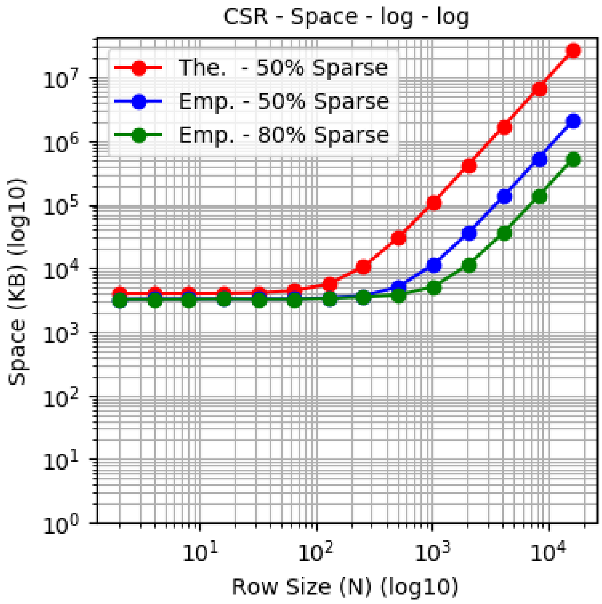
1.1 Storage Space

Let N, M and NNZ denote the number of rows, columns and non-zero elements of the matrix respectively. This notation is adopted for the remainder of this report. For 2D array implementation, space gets allocated to each element irrespective of its value and so the asymptotic space complexity is O(NM) (independent of NNZ).

For CSR implementation, we have three 1D arrays namely A for storing matrix values, IA for storing the cumulative number of non-zero values for each row and JA for storing the column index for each non-zero value. These arrays have NNZ, N+1 and NNZ elements respectively. Hence, the asymptotic space complexity is O(N + 2NNZ). In the worst case where NNZ = NM (all elements are non zero), CSR will take more space as its space complexity goes to O(N + 2NM).

The space complexities has been empirically determined for different matrix sizes and the results plotted below match with the analysis. Square matrices (N=M) were used for simplicity of analysis.





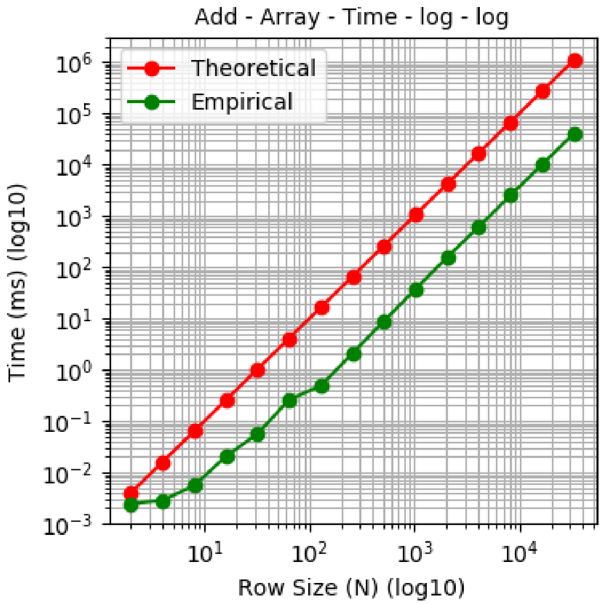
For CSR, the sparse percentage denotes the percentage of elements that are zero in the matrix.

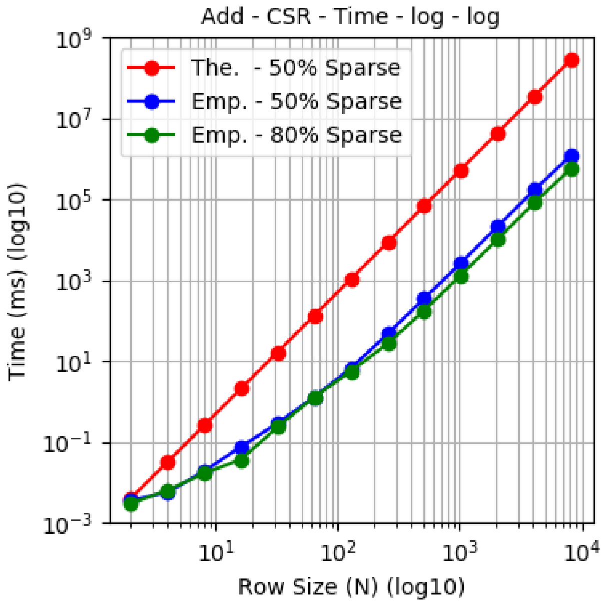
1.2 Matrix Addition

For matrix addition, all the elements of both input and output matrices have to be processed exactly once and there are NM elements in each matrix. The effects of locality and caching is ignored in this analysis. In the case of 2D array implementation, each element can be accessed in O(1) time and so the total time complexity for addition in this case is O(NM).

For the case of CSR implementation, each element can be accessed in $O(\frac{NNZ}{N})$ time (worst case) assuming that each row has $\frac{NNZ}{N}$ non-zero elements. This is because the array JA has to be scanned for matching column index in order to access an element. So, the worst case time complexity for addition in this case will be O(M*NNZ).

The time complexities has been empirically determined for different matrix sizes and the results plotted below match with the analysis. Square matrices (N=M) were used for simplicity of analysis,





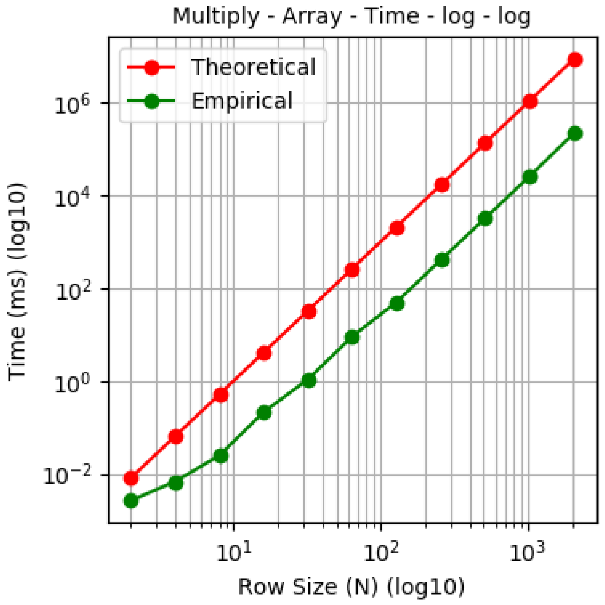
1.3 Matrix Multiplication

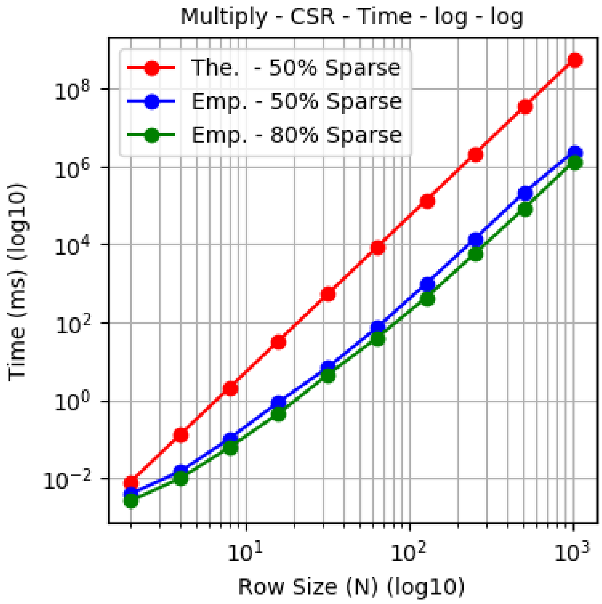
For matrix multiplication, analysis is done for multiplying two NxN square matrices without loss of generality. The naive or brute force implementation of the algorithm takes $O(N^3)$ multiplication and addition operations. Similar to addition, cache and locality effects are ignored in the analysis.

In the case of 2D array implementation, each element can be accessed in $O(1)$ time and so the total time complexity for multiplication in this case is $O(N^3)$.

For the case of CSR implementation, each element can be accessed in $O(\frac{NNZ}{N})$ time (worst case) assuming that each row has $\frac{NNZ}{N}$ non-zero elements. So, the time complexity for multiplication in this case will be $O(N^2*\text{NNZ})$.

The time complexities has been empirically determined for different matrix sizes and the results plotted below match with the analysis,





1.4 Breadth First Search

Let V be the number of nodes and E be the number of edges. The breadth first search algorithm has the following components,

* Inserting and removing nodes from the queue. Each node will be inserted and removed once. So, the time complexity for this component is O(V)

* Scanning the neighbors for each node to add to the queue. Each edge will be scanned once and so the time complexity is O(E)

* Inserting the visited nodes to the STL list. The time complexity is O(V)

* Sorting the list of nodes in each depth. The time complexity is O(V log V)

It can be inferred that the total time complexity is $O(V + E + VlogV)$. If sorting is not required, the time complexity will come down to $O(V + E)$.

The total space complexity of BFS is O(V + E) and can be determined by the size of following data structures,

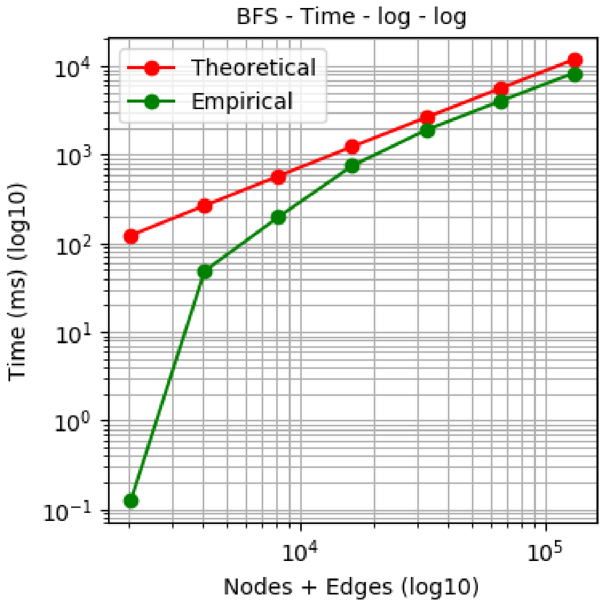
* The size of CSR adjacency matrix is O(V + 2E)

* The size of queue is O(V)

* The size of list to store depth wise nodes is O(V)

The empirical data is plotted below and matches with the analysis. Please note that the x axis in the plot is Vertex + Edges.

The limiting factor for testing is the size of input tsv file. Since, it has a 2D array structure with value for all the elements present, graphs with large vertex count could not be generated for testing.



2 Code and Testing Remarks

The makefile has been modified to use the C++ 11 standard during compilation. This is because the code uses some string parsing functions like std::stoi, std::stof etc. which are not available as part of C++ 98 standard.

The precision of the output is set by std::fixed and std::setprecision functions. The memory used by the program has been measured using the *time* shell program and the time is measured using clock_gettime() function.

Input files with random float numbers were generated for the purpose of empirical testing. The tests were run using custom shell scripts. [4] is the location of these scripts in the cluster.

3 References

1. https://en.cppreference.com/
2. https://www.geeksforgeeks.org/
3. DS 221 Course lecture notes
4. /home/shriramr/ds221/part_2/code/test