Assignment 3 - Sparse matrices in CSR and their solution using Jacobi Method

Yajat Sharma

3 December 2023

1 Introduction

This assignment requires the creation of a sparse matrix solver that reads a Matrix Market file and converts the data into Compressed Sparse Row (CSR) format. This CSR data is then used to solve the matrix using a sparse matrix solving algorithm. The accuracy of the algorithm is measured through the norm of the residual vector that is found by comparing the value that the matrix produces for the solved values of x and the given result it should produce.

Additionally, tools like Vtune have been used to breakdown the performance of the code into separate sections. The matrix sparsity pattern was plotted using python and some help from Chatgpt.

2 Mathematical Algorithm behind Jacobi method

Jacobi method is an iterative method that uses the result of each previous iteration to find a better value of x. The algorithm stops when either the maximum number of iterations is reached or the difference in the current and previous values of x is below a set value (which is usually very small) which would imply that the algorithm has converged to the correct answer.

The two main assumptions made in this method is that the given system of equations has a unique solution and that none of the diagonal entries are zero. The latter assumption can be a source of error for some of the matrices because it is never mentioned that all of them have non-zero diagonal entries. Therefore, to prevent division by zero, 10^{-16} is added to all diagonal entries when they are used in this algorithm.

The algorithm sets up the expressions for the solutions in the following format:

$$x_1 = \frac{1}{a_{11}}(b_1 - a_{12}x_2 - a_{13}x_3 - \cdots a_{1n}x_n)$$
$$x_2 = \frac{1}{a_{22}}(b_2 - a_{21}x_1 - a_{23}x_3 - \cdots a_{2n}x_n)$$
$$\vdots$$
$$x_n = \frac{1}{a_{nn}}(b_n - a_{n1}x_1 - a_{n2}x_2 - \cdots a_{n,n-1}x_{n-1})$$

Figure 1: Enter Caption

The algorithm begins with an initial guess for all the x-values and uses them to create the next set of values which are used in the next iteration. The initial values for all x-values in the program are 1.0. Additionally, the maximum difference between two corresponding x-values from the set of all x-values is compared with the tolerance which is used to check if the algorithm has converged to a solution or not.

NOTE: This information is obtained from University of Notre Dame's lecture on Jacobi and Gauss Siedel Iterative methods. (The lecture slides can be accessed here: https://www3.nd.edu/~zxu2/acms40390F12/Lec-7.3.pdf)

3 questions

1. The solver is not able to get a result for b1 ss.mtx because this matrix is not symmetrical and Jacobi method mainly works for symmetric matrices.

2. My solver is not able to get the correct result for the last 4 matrices in the results section because the ReadMMtoCSR function takes an unreasonable amount of time to read the files and therefore, no calculation can be done on the matrix. The ACTIVSg70K.mtx file produces a result but with a residual that is nan. Additionally, Jacobi method is not the most optimal way of solving very large sparse matrices which would render it unable to find the solution to the required accuracy even if the files were read by the ReadMMtoCSR function.

4 Results

This section contains the outputs from all the files that were used with the program. The program took an unreasonably long time to process and therefore, did not produce a result for the last three matrices. It can be noticed that the solving algorithm does not work for very large matrices.

Table 1: Results

Problem	Size		CPU time (Sec)	Residual
	row	column		
LFAT5.mtx	14	14	0.000306	3.40e-14
LF10.mtx	18	18	0.004301	8.55e-13
ex3.mtx	1821	1821	54.221911	3.15e+01
jnlbrng1.mtx	40000	40000	0.603473	4.59e-12
ACTIVSg70K.mtx	69999	69999	0.107621	nan
2cubes sphere.mtx	-	-	-	-
tmt sym.mtx	-	-	-	-
StocF-1465.mtx	-	-	-	-

5 Sparsity Pattern

The sparsity pattern was plotted using Python (the code is in the next section) and it produced the following plots.

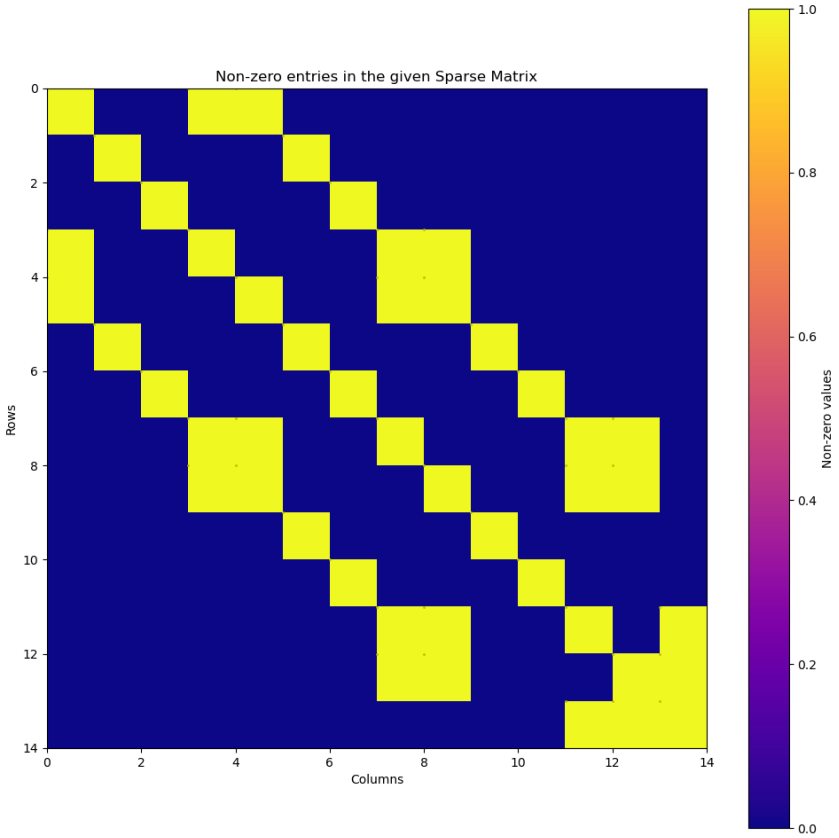

Figure 2: Sparsity pattern of LFAT5.mtx

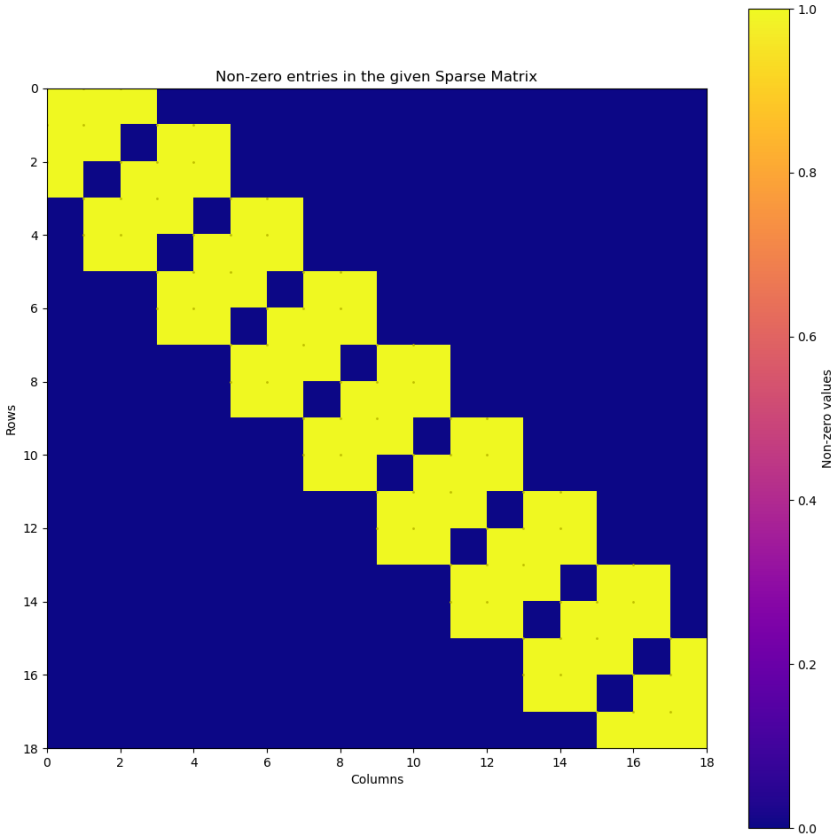

Figure 3: Sparsity pattern of LF10.mtx

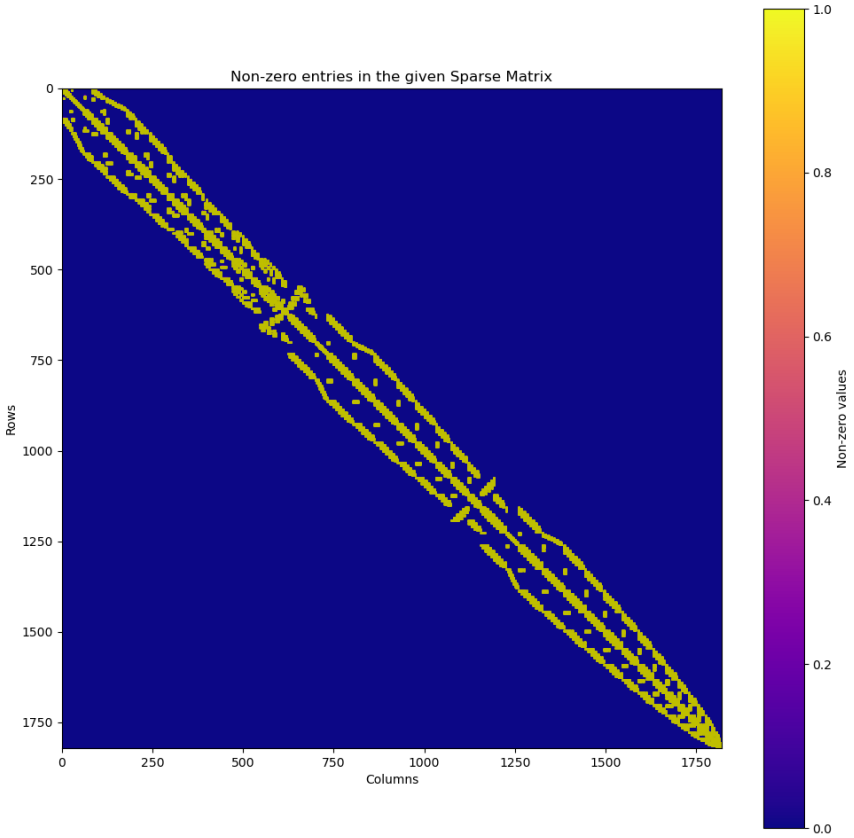

Figure 4: Sparsity pattern of ex3.mtx

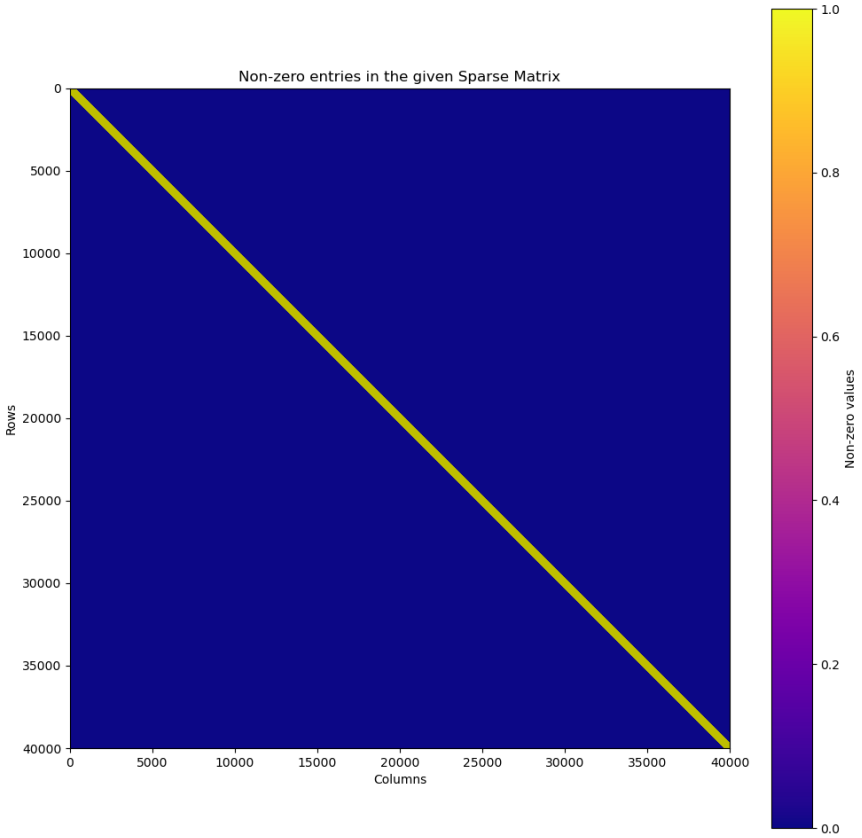

Figure 5: sparsity pattern of jnlbrng1.mtx

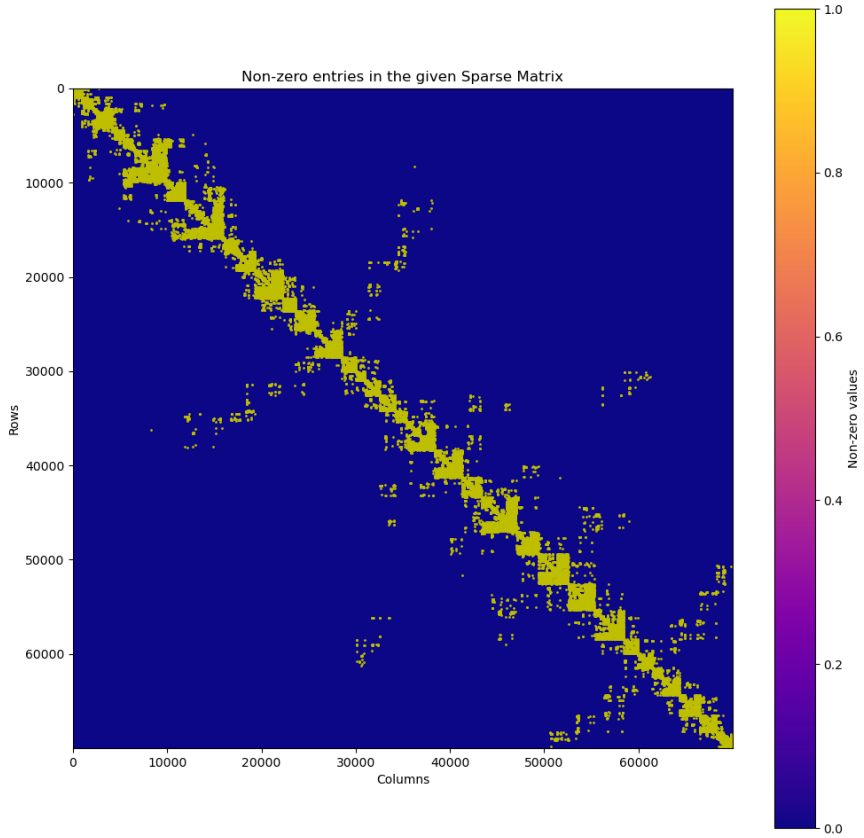

Figure 6: sparsity pattern of ACTIVSg70K.mtx

6 Python code for sparsity pattern plotting

```
import matplotlib.pyplot as plt
import numpy as np
from scipy.sparse import lil_matrix, csr_matrix

file_name = "Matrix_data.txt"

with open(file_name, 'r') as file:
    num_rows, num_cols = map(int, file.readline().split())

    #This line is from ChatGPT
```

```
    # Initialize a lil_matrix for efficient construction
    matrix = lil_matrix((num_rows, num_cols), dtype=np.int8)

    # Fill the lil_matrix
    for line in file:
        r, c = map(int, line.split())
        if r < num_rows and c < num_cols:
            matrix[r, c] = 1

#This section is from ChatGPT
matrix = csr_matrix(matrix)
dense_matrix = matrix.toarray()

# Plot the dense matrix
plt.figure(figsize=(12, 12))
plt.imshow(dense_matrix, cmap='plasma', extent=[0, num_cols,
   num_rows, 0])
plt.plot(np.nonzero(dense_matrix)[1], np.nonzero(dense_matrix)[0],
   'yo', markersize=1) #only used for very large matrices
plt.xlabel('Columns')
plt.ylabel('Rows')
plt.title('Non-zero entries in the given Sparse Matrix')
plt.colorbar(label='Non-zero values')
plt.savefig('matrix_graph.png')
plt.show()
```

7 Vtune

The code was run with Vtune and produced the following result. Vtune required an SSH connection to a linux system on my Mac which is why I used my friend's Vtune with my code.

This result is not ideal and the primary reason might be the use of Bubble sort when the row ptr array is being converted into the Compressed Sparse Row (CSR) format. This can be seen in the Top Hotspots section where Bubblesort takes the most CPU time.

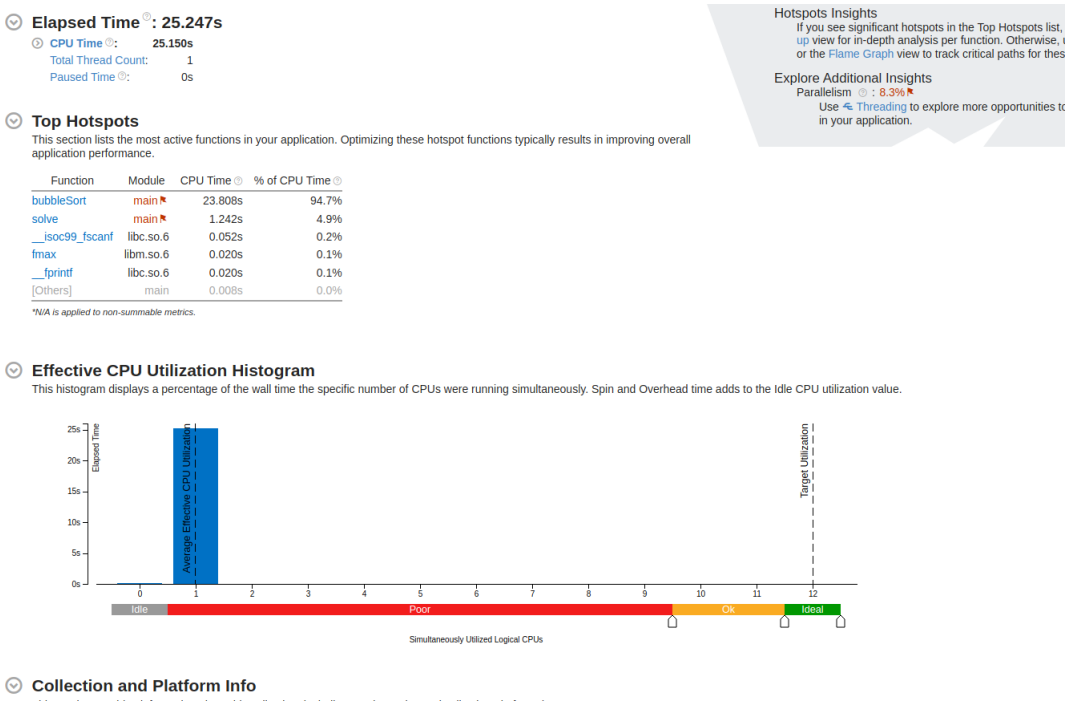

Figure 7: Vtune analysis of program

8 gcov

The clang compiler on MacOS 14.0 does not support the -pg flag. Therefore, I was unable to conduct a gprof analysis on my program.

9 Makefile

```
CC = gcc
CFLAGS = -Wextra -Wall -std=c99 -Ofast
EXECUTABLE = main

all: $(EXECUTABLE)

$(EXECUTABLE): main.o functions.o
 $(CC) $(CFLAGS) -o $(EXECUTABLE) main.o functions.o

main.o: main.c functions.h
 $(CC) $(CFLAGS) -c main.c

functions.o: functions.c functions.h
```

```
 $(CC) $(CFLAGS) -c functions.c

clean:
 rm -f $(EXECUTABLE)
```

This makefile compiles the code in a three step process. Firstly, it creates the object files which allow for the connection to functions.h to be made and these object files are then used to make the final executable "main".

To run the program, put all the C and Header files in the same directory with the Makefile and then type "make" in the terminal, after you have ensured that you are in the same directory.

To run the executable, ensure that the mtx file you wish to run is in the same directory as all the other files and type ./main (name of your file).