Matrix Solver with the Jacobi Method

Developing a Matrix Solver in C

Alexis Fernandez

1 Implementation with Jacobi's Method

For this assignment the prompt was to create a program that reads a sparse matrix from an MM file. After reading the matrix data from the MM file and converting it to CSR format, the task was to use a matrix solving method to find the vector x given matrix A and solution b. Then the requirement was to compute the matrix multiplication A*x=b with the calculated x vector and compare accuracy of this result by computing the residual between this result and the given solution vector. The norm of this residual vector was then computed to measure the accuracy.

My implementation includes all of these steps required. I first read the matrix data from the MM file. Because CSR format only provides the lower triangular portion of a symmetric matrix, I implemented an algorithm to mirror the non-zero entries in the lower triangular portion to the upper triangular portion of the matrix. After this was completed, the CSR format was complete with the row pointer array, the column indices array, and the values array for the matrix. After this I used Jacobi's method to solve for vector x given matrix A and solution b. I used Jacobi's method because this method converges quickly for symmetric sparse matrices (mostly for those that are diagonally dominant) therefore it was good for the provided sets of matrices. This method is suitable for sparse matrices therefore it is efficient in terms of memory usage. The only disadvantage is that it converges slowly for matrices that are not diagonally dominant. After solving for vector x, the residual was computed by comparing the result of A*x=b to the provided b vector of all 1's. The norm of this residual vector was then calculated and outputted to show the accuracy.

The following is the formula I used to implement Jacobi's method:

$$x_i^{(k+1)} = \frac{1}{a_{ii}}\left(b_i - \sum_{j\neq i} a_{ij}x_j^{(k)}\right), \quad i = 1, 2, \ldots, n.$$

Figure 1: Jacobi's Method Formula

CSR format is much more efficient than full matrix format because it takes up less memory size. It is also faster to work with when performing operations like sparse matrix multiplication when compared to full matrix format. This is because due to the nature of CSR format only including non-zero entries of the matrix, it inherently allows for efficient iteration through the rows and columns without having to deal with zero values in the matrix. Because of this reason, it also has reduced storage requirements, meaning less values and operations to perform when doing matrix multiplication. The structure of CSR format also allows for easier parallelization of the operation when compared to full matrix format.

My solver is not able to get the results for `b1 ss.mtx`. My program is able to transfer the matrix into CSR format however it cannot get the results of A*x=b because this matrix is not a symmetric matrix, therefore it cannot be solved using Jacobi's method which is the method that I have in place to solve the given matrices.

My solver is not able to get the results for `2cubes sphere.mtx` `ACTIVSg70K.mtx` `tmt sym.mtx` `StocF-1465` because the matrices in these MM files are too large therefore Jacobi's method does not converge to a value for such large matrices with the initial conditions, maximum iterations and tolerance that I initially set. For my solver to get the results for these matrices, I would have to implement an adaptive Jacobi's method solver that includes relaxation factors to control the stability and the speed at which the solution converges.

2 Results

It is important to note that for LFAT5.mtx the maximum number of iterations used was 10 000 and the tolerance was 1e-16. For LF10.mtx the max iterations were 100 000 and the tolerance was 1e-16. For ex3.mtx the max iterations were 100 000 and the tolerance was 1e-2. For jnlbrng1.mtx the max iterations were 100 000 and the tolerance was 1e-16. For ACTIVSg70K.mtx the max iterations were 10 000 and the tolerance was 1e-16.

Table 1: Results For all of the Problems Provided

Problem	Size		Non-zeros	CPU time (Sec)	Norm-Residual
	row	*column*			
LFAT5.mtx	14	14	30	0.000547	3.59109e-14
LF10.mtx	18	18	50	0.01854	5.9746e-13
ex3.mtx	1821	1821	27253	0.0002	7.9469e+01
jnlbrng1.mtx	40000	40000	119600	2.4897	1.2535e-12
ACTIVSg70K.mtx	69999	69999	154313	18.6279	-nan
2cubes sphere.mtx	101492	101492	874378	-	-
tmt sym.mtx	726713	726713	2903837	-	-
StocF-1465.mtx	1465137	1465137	11235263	-	-

3 Makefile

Listing 1: Makefile

```
CC = gcc
CFLAGS = -Wall -Wextra -std=c99
SOURCES = functions.c main.c
OBJECTS = $(SOURCES:.c=.o)
EXECUTABLE = main
LIBRARIES = -lm

all: $(EXECUTABLE)

$(EXECUTABLE):$(SOURCES)
 $(CC) $(SOURCES) -o $(EXECUTABLE) $(LIBRARIES)

%.o: %.c
 $(CC) $(CFLAGS) -c $< -o $@

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

The Makefile was created in order to compile the program and link it to the header files and the files needed for the initialization of the functions used in the main function.

In order to run the makefile, first the all of the files need including the makefile need to be in the same directory. After, one can type in the terminal the command ”make”. Once the makefile is run, the code will be compiled with the specified flags and can then be executed like normal. The makefile can also be cleaned with the command ”make clean”.

The following is a breakdown of how the Makefile works.

’CC = gcc’ - Defines the compiler that will be used. GCC in this case.

’CFLAGS = -Wall -Wextra -std=c99 - Defines the flags to be used when compiling the code. These flags are used to provide warnings to the user for correct coding practice. -Wall is used to display all of the commonly used warnings. -Wextra is used to display the extra warnings and -stx=c99 is used to specify the C language standard.

’SOURCES = functions.c main.c’ - Defines the source files to be used when compiling the program. It is important to note that functions.h is not included because it is already included in the headers of the functions included.

’EXECUTABLE = main’ - Specifies the name of the executable to be used.

'LIBRARIES = -lm' - specifies that the math library will be linked to the files.

all:(EXECUTABLE): - Sets the target to be used.

(EXECUTABLE):(SOURCES) (CC) (SOURCES) -o (EXECUTABLE)(LIBRARIES) - Builds the executable using the compiler specified by CC, the source files in SOURCES, the output flag -o, the executable in EXECUTABLE and the libraries specified in LIBRARIES.

.o: .c (CC) (CFLAGS) -c <-o @ - Pattern rule that builds the object file. Uses the input file (<) and the output file (@)

clean: rm -f (SOURCES) (EXECUTABLE) - Clean rule. Uses rm to remove the object files specified and the executable. The f flag ensures that the terminal does not prompt for confirmation for this action.

4 Plots

The following are visualization aids in the form of plots that help visualize the sparsity pattern of the different matrices. In the following plots, each row and column represent a row and column in the matrix. Non-zero elements are marked and zero elements are left blank. The position and density of non-zero elements is displayed. From these plots, the symmetry of these matrices can be seen if a mirror line is placed on the diagonal of each of the matrices.

The following plots were created with python. The matplotlib library was used in order to plot the data read from the .mtx files and the mm reader was imported to be able to read the data from the MM files.

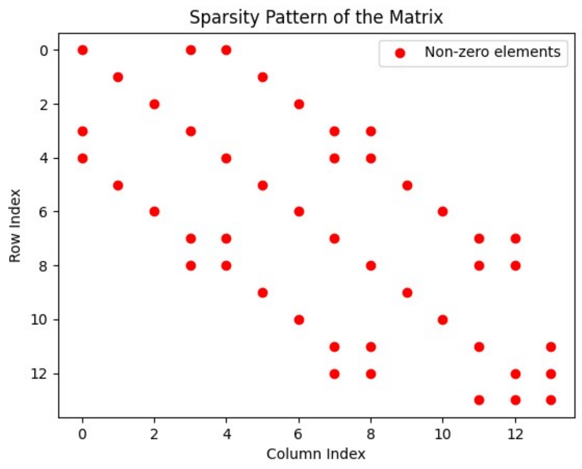


Figure 2: Visualize sparsity pattern of matrix LFAT5.mtx

The following is the code that was used to create the plots. The same code was used for all of

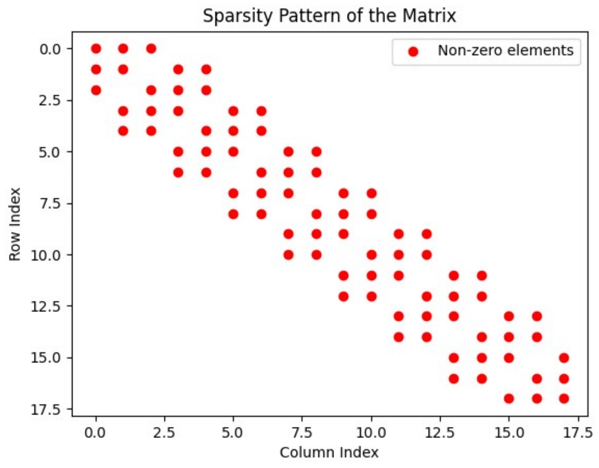


Figure 3: Visualize sparsity pattern of matrix LF10.mtx

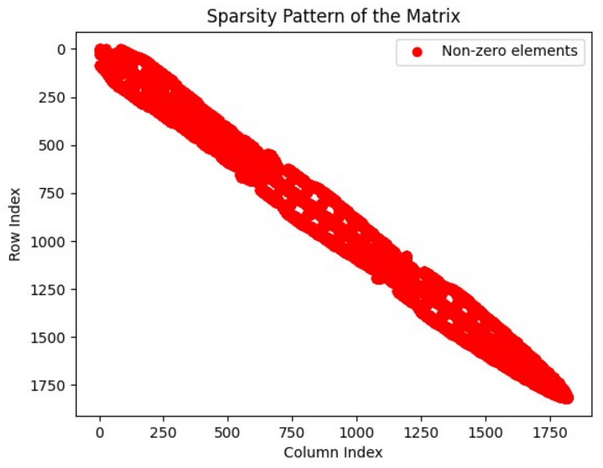


Figure 4: Visualize sparsity pattern of matrix ex3.mtx

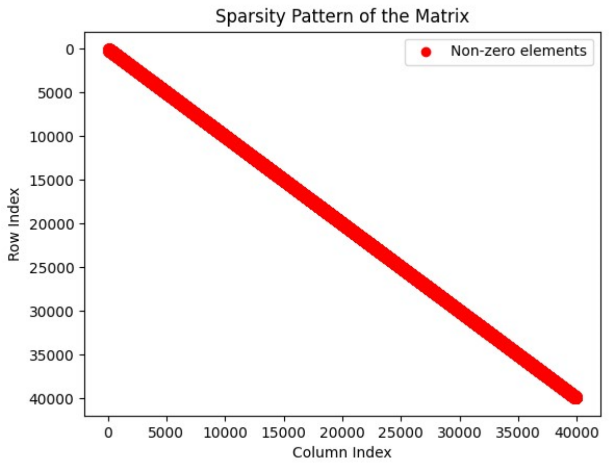


Figure 5: Visualize sparsity pattern of matrix jnlbrng1.mtx

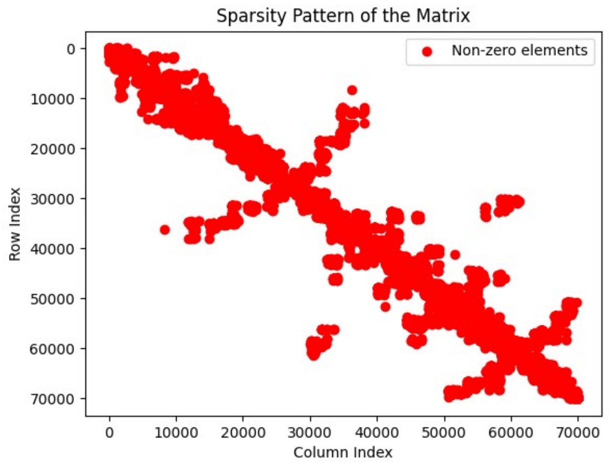


Figure 6: Visualize sparsity pattern of matrix ACTIVSg70k.mtx

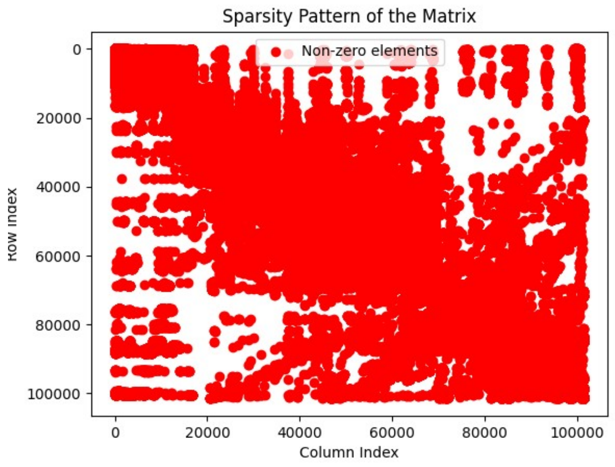


Figure 7: Visualize sparsity pattern of matrix 2cubes sphere.mtx

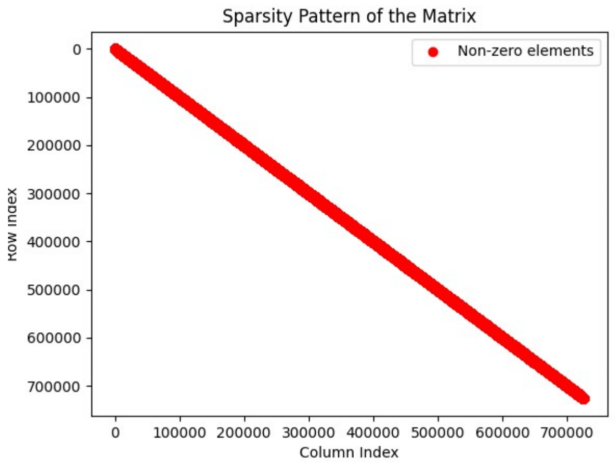


Figure 8: Visualize sparsity pattern of matrixtmt sym.mtx

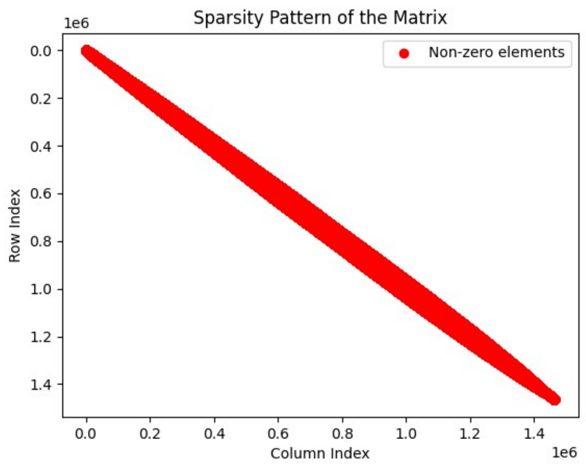


Figure 9: Visualize sparsity pattern of StocF-1465.mtx

the plots, just the name of the file to be read was changed.

Listing 2: Python Plotting Code

```
from scipy.io import mmread
import matplotlib.pyplot as plt
import numpy as np

#ChatGPT was referenced to make to make the following:

#Reading the MM file
matrix = mmread('tmt_sym.mtx')

#Shape of the matrix
num_rows, num_cols = matrix.shape

#Creating scatter plot
nonzero_r, nonzero_c = matrix.nonzero()
plt.scatter(nonzero_c, nonzero_r, marker = 'o', color='red', label=
    'Non-zero elements')

plt.title('Sparsity Pattern of the Matrix')
plt.xlabel('Column Index')
plt.ylabel('Row Index')
plt.gca().invert_yaxis()
plt.legend()
plt.grid(False)
plt.show()
```

5 GDB

GDB can be used to debug the program. It can be used to set breakpoints, and to print values for variables at different points throughout the program. To run this debugger, first the flag -g has to be included when compiling the program with gcc. This flag allows the compiler to include debugging information in the executable. Therefore the line to compile would be the following: `gcc -g main.c functions.c -o main -lm`. The code can then be run to debug using `gdb main`. Then any file can be run by typing `run (file name)`. A break point can be set with the command `break (file):(line number` and the value of a variable can be printed with the command `p (varible)`. An example of these can be seen below

Listing 3: GDB Example

```
alexis@AlexisXPS:~/MECHTRON2MP3/Assignments/Assignment 3$ gcc -g
   main.c functions.c -o main -lm
alexis@AlexisXPS:~/MECHTRON2MP3/Assignments/Assignment 3$ gdb main
GNU gdb (Ubuntu 12.1-0ubuntu1~22.04) 12.1
Copyright (C) 2022 Free Software Foundation, Inc.
License GPLv3+: GNU GPL version 3 or later <http://gnu.org/licenses
   /gpl.html>
This is free software: you are free to change and redistribute it.
There is NO WARRANTY, to the extent permitted by law.
Type "show copying" and "show warranty" for details.
This GDB was configured as "x86_64-linux-gnu".
Type "show configuration" for configuration details.
For bug reporting instructions, please see:
<https://www.gnu.org/software/gdb/bugs/>.
Find the GDB manual and other documentation resources online at:
    <http://www.gnu.org/software/gdb/documentation/>.

For help, type "help".
Type "apropos word" to search for commands related to "word"...
Reading symbols from main...
(gdb) run LFAT5.mtx
Starting program: /home/alexis/MECHTRON2MP3/Assignments/Assignment
   3/main LFAT5.mtx
[Thread debugging using libthread_db enabled]
```

```
Using host libthread_db library "/lib/x86_64-linux-gnu/libthread_db
   .so.1".
Number of non-zeros: 46
Row pointer: 0 3 5 7 11 15 18 21 26 31 33 35 39 43 46
Column Index: 0 3 4 1 5 2 6 0 3 7 8 0 4 7 8 1 5 9 2 6 10 3 4 7 11
   12 3 4 8 11 12 5 9 6 10 7 8 11 13 7 8 12 13 11 12 13
Values: 1.570880 -94.252800 0.785440 12566400.000000
   -6283200.000000 0.608806 -0.304403 -94.252800 15080.448000
   -7540.224000 94.252800 0.785440 3.141760 -94.252800 0.785440
   -6283200.000000 12566400.000000 -6283200.000000 -0.304403
   0.608806 -0.304403 -7540.224000 -94.252800 15080.448000
   -7540.224000 94.252800 94.252800 0.785440 3.141760 -94.252800
   0.785440 -6283200.000000 12566400.000000 -0.304403 0.608806
   -7540.224000 -94.252800 15080.448000 94.252800 94.252800
   0.785440 3.141760 0.785440 94.252800 0.785440 1.570880
16

The matrix name: LFAT5.mtx
The dimension of the matrix: 14 by 14
Number of non-zeros (read from file): 30
Number of non-zeros (from symmetry): 46
Converged after 2412 iterations
CPU time taken to solve Ax=b: 0.000521 seconds
Residual Norm: 3.5910866945954524e-14
[Inferior 1 (process 51131) exited normally]
(gdb) break functions.c:97
Breakpoint 1 at 0x555555555a73: file functions.c, line 98.
(gdb) run
Starting program: /home/alexis/MECHTRON2MP3/Assignments/Assignment
   3/main LFAT5.mtx
[Thread debugging using libthread_db enabled]
Using host libthread_db library "/lib/x86_64-linux-gnu/libthread_db
   .so.1".

Breakpoint 1, ReadMMtoCSR (filename=0x7fffffffdf4b "LFAT5.mtx",
   matrix=0x7fffffffdb10, not_on_diagonal=0x7fffffffdab4) at
   functions.c:98
98          *not_on_diagonal = not_on_diagonal_count;
(gdb) p not_on_diagonal
```

```
$1 = (int *) 0x7fffffffdab4
(gdb) p not_on_diagonal_count
$2 = 16
(gdb) quit
```

6 GCOV

`gcov` is used for code coverage analysis. This tool provides insight into how much of the code is executed during testing and gathers data about which lines of code were executed along with how many times they were executed. This can be useful to identify untested sections of code and to improve the quality of the program.

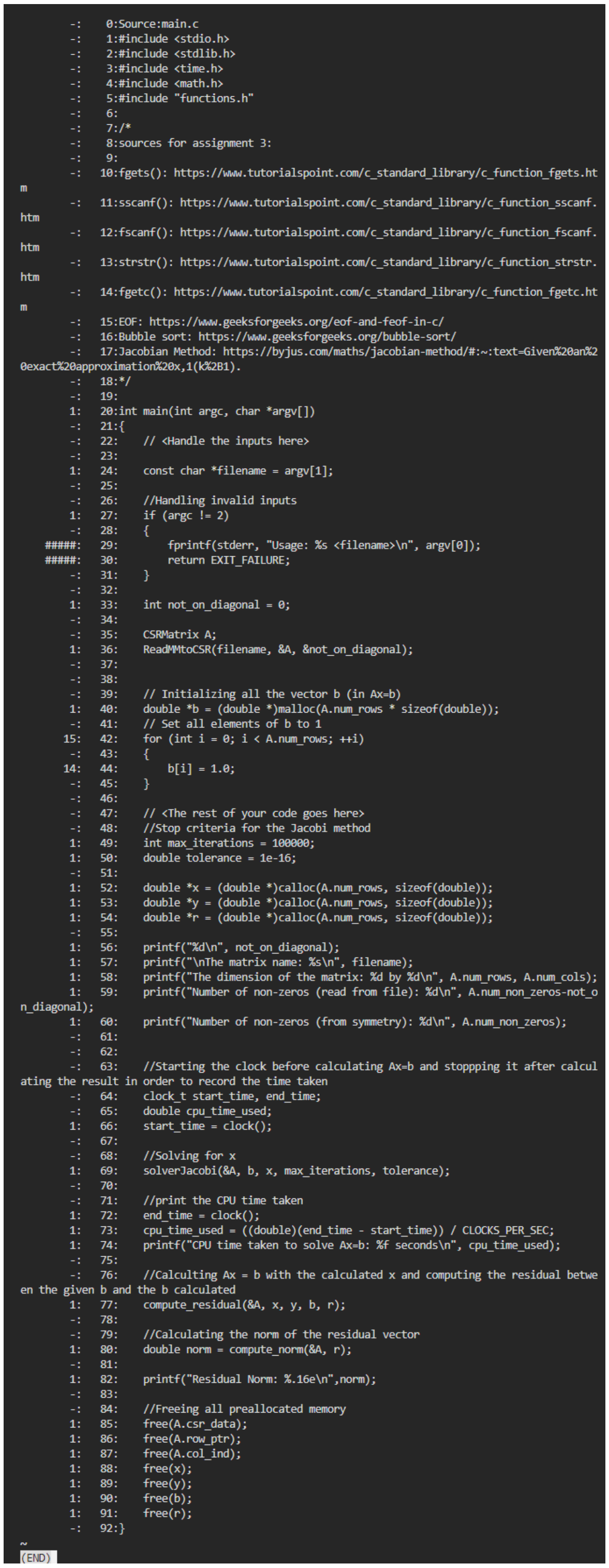
This tool generates two different files to output coverage data, a `.gcda` (coverage data) and a `.gcno` (coverage notes) file.

To use this tool the flags `-fprofile-arcs` and `-ftest-coverage` have to be included. For my program I used `gcc-9 -fprofile-arcs -ftest-coverage main.c functions.c -o main -lm`. The -9 is the version of gcc that I used. The program can the be executed normally with `./main (filename)`. To generate the coverage data files the I used the commands `gcov main.c functions.` To then view these coverage files I used the command `less main.c.gcov` and `less functions.c.gcov`. These coverage reports can be seen below:

7 Vtune

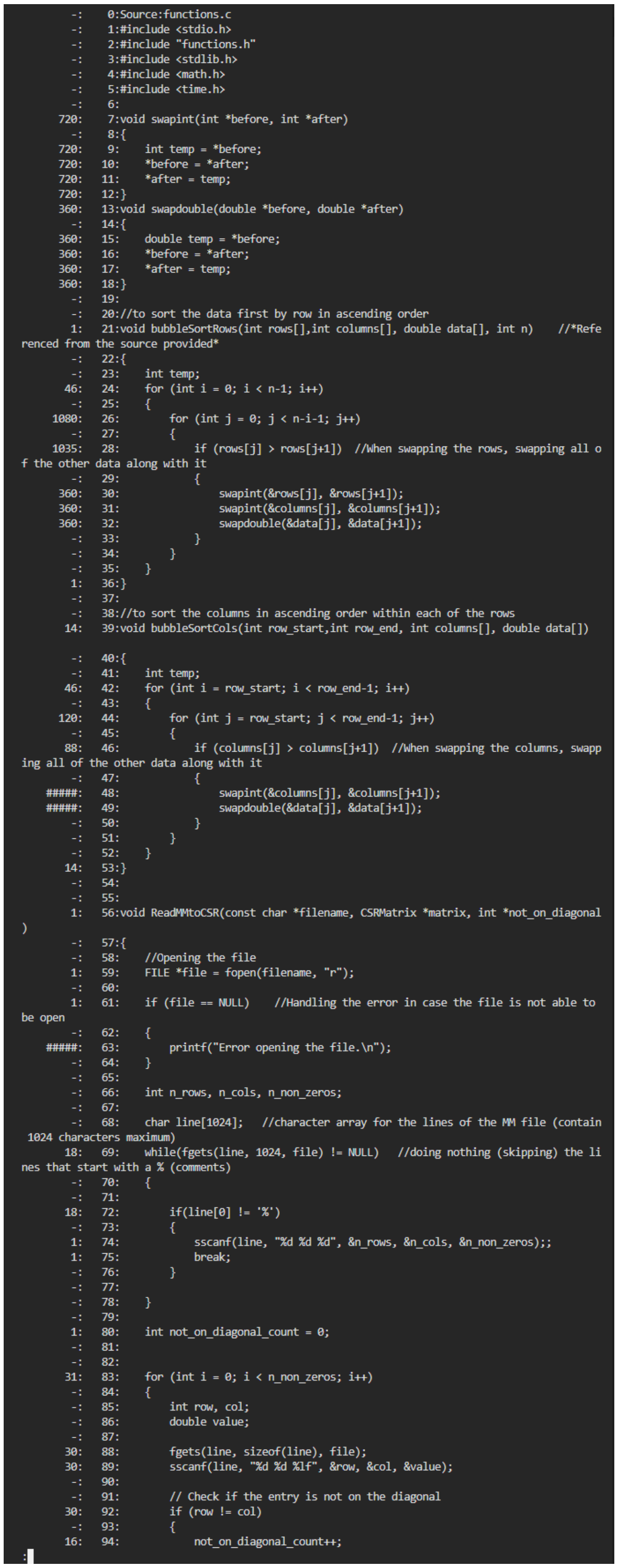
The analysis with Vtune for the MM files that are not too long can be seen below.

Because LFAT5.mtx and LF10.mtx are relatively smaller matrices, the time for an analysis to be generated is too short. However for the two longer examples below, an analysis was able to be created.



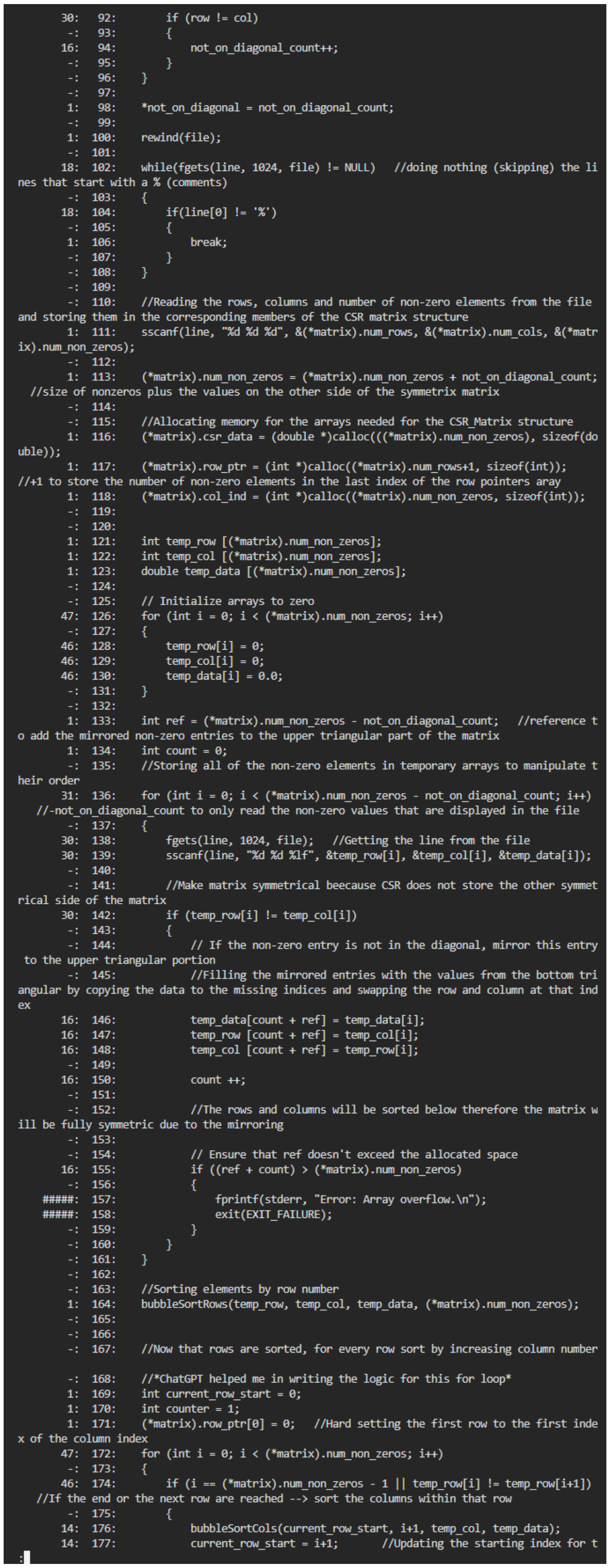
```
        -:    0:Source:main.c
        -:    1:#include <stdio.h>
        -:    2:#include <stdlib.h>
        -:    3:#include <time.h>
        -:    4:#include <math.h>
        -:    5:#include "functions.h"
        -:    6:
        -:    7:/*
        -:    8:sources for assignment 3:
        -:    9:
        -:   10:fgets(): https://www.tutorialspoint.com/c_standard_library/c_function_fgets.ht
m
        -:   11:sscanf(): https://www.tutorialspoint.com/c_standard_library/c_function_sscanf.
htm
        -:   12:fscanf(): https://www.tutorialspoint.com/c_standard_library/c_function_fscanf.
htm
        -:   13:strstr(): https://www.tutorialspoint.com/c_standard_library/c_function_strstr.
htm
        -:   14:fgetc(): https://www.tutorialspoint.com/c_standard_library/c_function_fgetc.ht
m
        -:   15:EOF: https://www.geeksforgeeks.org/eof-and-feof-in-c/
        -:   16:Bubble sort: https://www.geeksforgeeks.org/bubble-sort/
        -:   17:Jacobian Method: https://byjus.com/maths/jacobian-method/#:~:text=Given%20an%2
0exact%20approximation%20x,1(k%2B1).
        -:   18:*/
        -:   19:
        1:   20:int main(int argc, char *argv[])
        -:   21:{
        -:   22:    // <Handle the inputs here>
        -:   23:
        1:   24:    const char *filename = argv[1];
        -:   25:
        -:   26:    //Handling invalid inputs
        1:   27:    if (argc != 2)
        -:   28:    {
    #####:   29:        fprintf(stderr, "Usage: %s <filename>\n", argv[0]);
    #####:   30:        return EXIT_FAILURE;
        -:   31:    }
        -:   32:
        1:   33:    int not_on_diagonal = 0;
        -:   34:
        -:   35:    CSRMatrix A;
        1:   36:    ReadMMtoCSR(filename, &A, &not_on_diagonal);
        -:   37:
        -:   38:
        -:   39:    // Initializing all the vector b (in Ax=b)
        1:   40:    double *b = (double *)malloc(A.num_rows * sizeof(double));
        -:   41:    // Set all elements of b to 1
       15:   42:    for (int i = 0; i < A.num_rows; ++i)
        -:   43:    {
       14:   44:        b[i] = 1.0;
        -:   45:    }
        -:   46:
        -:   47:    // <The rest of your code goes here>
        -:   48:    //Stop criteria for the Jacobi method
        1:   49:    int max_iterations = 100000;
        1:   50:    double tolerance = 1e-16;
        -:   51:
        1:   52:    double *x = (double *)calloc(A.num_rows, sizeof(double));
        1:   53:    double *y = (double *)calloc(A.num_rows, sizeof(double));
        1:   54:    double *r = (double *)calloc(A.num_rows, sizeof(double));
        -:   55:
        1:   56:    printf("%d\n", not_on_diagonal);
        1:   57:    printf("\nThe matrix name: %s\n", filename);
        1:   58:    printf("The dimension of the matrix: %d by %d\n", A.num_rows, A.num_cols);
        1:   59:    printf("Number of non-zeros (read from file): %d\n", A.num_non_zeros-not_o
n_diagonal);
        1:   60:    printf("Number of non-zeros (from symmetry): %d\n", A.num_non_zeros);
        -:   61:
        -:   62:
        -:   63:    //Starting the clock before calculating Ax=b and stoppping it after calcul
ating the result in order to record the time taken
        -:   64:    clock_t start_time, end_time;
        -:   65:    double cpu_time_used;
        1:   66:    start_time = clock();
        -:   67:
        -:   68:    //Solving for x
        1:   69:    solverJacobi(&A, b, x, max_iterations, tolerance);
        -:   70:
        -:   71:    //print the CPU time taken
        1:   72:    end_time = clock();
        1:   73:    cpu_time_used = ((double)(end_time - start_time)) / CLOCKS_PER_SEC;
        1:   74:    printf("CPU time taken to solve Ax=b: %f seconds\n", cpu_time_used);
        -:   75:
        -:   76:    //Calculting Ax = b with the calculated x and computing the residual betwe
en the given b and the b calculated
        1:   77:    compute_residual(&A, x, y, b, r);
        -:   78:
        -:   79:    //Calculating the norm of the residual vector
        1:   80:    double norm = compute_norm(&A, r);
        -:   81:
        1:   82:    printf("Residual Norm: %.16e\n",norm);
        -:   83:
        -:   84:    //Freeing all preallocated memory
        1:   85:    free(A.csr_data);
        1:   86:    free(A.row_ptr);
        1:   87:    free(A.col_ind);
        1:   88:    free(x);
        1:   89:    free(y);
        1:   90:    free(b);
        1:   91:    free(r);
        -:   92:}
~
(END)
```

Figure 10: Coverage Report for main.c



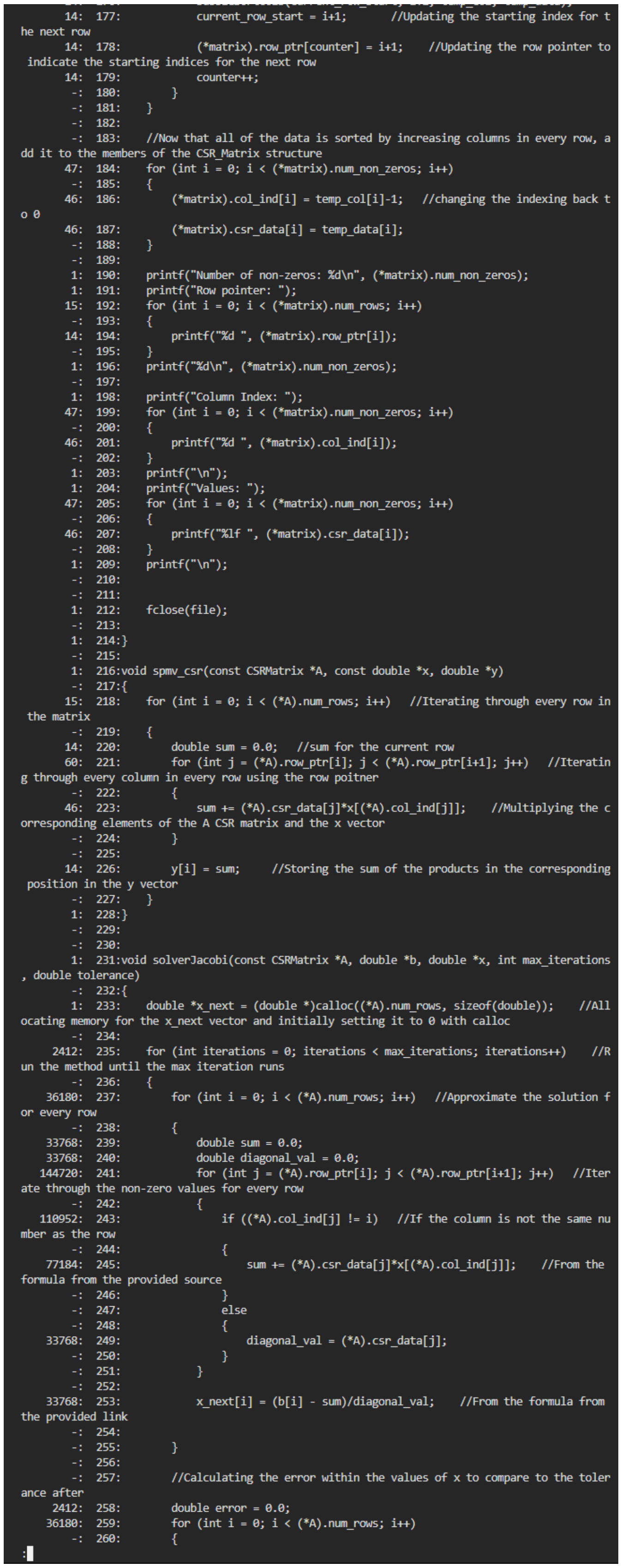
```
        -:    0:Source:functions.c
        -:    1:#include <stdio.h>
        -:    2:#include "functions.h"
        -:    3:#include <stdlib.h>
        -:    4:#include <math.h>
        -:    5:#include <time.h>
        -:    6:
      720:    7:void swapint(int *before, int *after)
        -:    8:{
      720:    9:    int temp = *before;
      720:   10:    *before = *after;
      720:   11:    *after = temp;
      720:   12:}
      360:   13:void swapdouble(double *before, double *after)
        -:   14:{
      360:   15:    double temp = *before;
      360:   16:    *before = *after;
      360:   17:    *after = temp;
      360:   18:}
        -:   19:
        -:   20://to sort the data first by row in ascending order
        1:   21:void bubbleSortRows(int rows[],int columns[], double data[], int n)    //*Refe
renced from the source provided*
        -:   22:{
        -:   23:    int temp;
       46:   24:    for (int i = 0; i < n-1; i++)
        -:   25:    {
     1080:   26:        for (int j = 0; j < n-i-1; j++)
        -:   27:        {
     1035:   28:            if (rows[j] > rows[j+1])  //When swapping the rows, swapping all o
f the other data along with it
        -:   29:            {
      360:   30:                swapint(&rows[j], &rows[j+1]);
      360:   31:                swapint(&columns[j], &columns[j+1]);
      360:   32:                swapdouble(&data[j], &data[j+1]);
        -:   33:            }
        -:   34:        }
        -:   35:    }
        1:   36:}
        -:   37:
        -:   38://to sort the columns in ascending order within each of the rows
       14:   39:void bubbleSortCols(int row_start,int row_end, int columns[], double data[])

        -:   40:{
        -:   41:    int temp;
       46:   42:    for (int i = row_start; i < row_end-1; i++)
        -:   43:    {
      120:   44:        for (int j = row_start; j < row_end-1; j++)
        -:   45:        {
       88:   46:            if (columns[j] > columns[j+1])  //When swapping the columns, swapp
ing all of the other data along with it
        -:   47:            {
    #####:   48:                swapint(&columns[j], &columns[j+1]);
    #####:   49:                swapdouble(&data[j], &data[j+1]);
        -:   50:            }
        -:   51:        }
        -:   52:    }
       14:   53:}
        -:   54:
        -:   55:
        1:   56:void ReadMMtoCSR(const char *filename, CSRMatrix *matrix, int *not_on_diagonal
)
        -:   57:{
        -:   58:    //Opening the file
        1:   59:    FILE *file = fopen(filename, "r");
        -:   60:
        1:   61:    if (file == NULL)    //Handling the error in case the file is not able to
be open
        -:   62:    {
    #####:   63:        printf("Error opening the file.\n");
        -:   64:    }
        -:   65:
        -:   66:    int n_rows, n_cols, n_non_zeros;
        -:   67:
        -:   68:    char line[1024];   //character array for the lines of the MM file (contain
 1024 characters maximum)
       18:   69:    while(fgets(line, 1024, file) != NULL)   //doing nothing (skipping) the li
nes that start with a % (comments)
        -:   70:    {
        -:   71:
       18:   72:        if(line[0] != '%')
        -:   73:        {
        1:   74:            sscanf(line, "%d %d %d", &n_rows, &n_cols, &n_non_zeros);;
        1:   75:            break;
        -:   76:        }
        -:   77:
        -:   78:    }
        -:   79:
        1:   80:    int not_on_diagonal_count = 0;
        -:   81:
        -:   82:
       31:   83:    for (int i = 0; i < n_non_zeros; i++)
        -:   84:    {
        -:   85:        int row, col;
        -:   86:        double value;
        -:   87:
       30:   88:        fgets(line, sizeof(line), file);
       30:   89:        sscanf(line, "%d %d %lf", &row, &col, &value);
        -:   90:
        -:   91:        // Check if the entry is not on the diagonal
       30:   92:        if (row != col)
        -:   93:        {
       16:   94:            not_on_diagonal_count++;
:
```

Figure 11: Coverage Report (Part 1) for functions.c



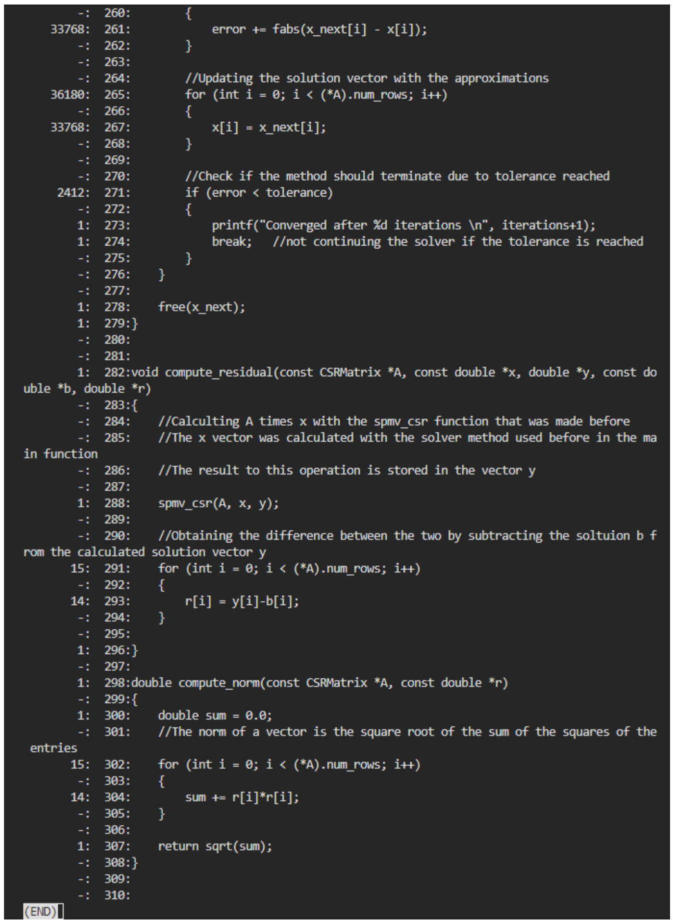
```
       30:   92:        if (row != col)
        -:   93:        {
       16:   94:            not_on_diagonal_count++;
        -:   95:        }
        -:   96:    }
        -:   97:
        1:   98:    *not_on_diagonal = not_on_diagonal_count;
        -:   99:
        1:  100:    rewind(file);
        -:  101:
       18:  102:    while(fgets(line, 1024, file) != NULL)   //doing nothing (skipping) the li
nes that start with a % (comments)
        -:  103:    {
       18:  104:        if(line[0] != '%')
        -:  105:        {
        1:  106:            break;
        -:  107:        }
        -:  108:    }
        -:  109:
        -:  110:    //Reading the rows, columns and number of non-zero elements from the file
and storing them in the corresponding members of the CSR matrix structure
        1:  111:    sscanf(line, "%d %d %d", &(*matrix).num_rows, &(*matrix).num_cols, &(*matr
ix).num_non_zeros);
        -:  112:
        1:  113:    (*matrix).num_non_zeros = (*matrix).num_non_zeros + not_on_diagonal_count;
  //size of nonzeros plus the values on the other side of the symmetrix matrix
        -:  114:
        -:  115:    //Allocating memory for the arrays needed for the CSR_Matrix structure
        1:  116:    (*matrix).csr_data = (double *)calloc(((*matrix).num_non_zeros), sizeof(do
uble));
        1:  117:    (*matrix).row_ptr = (int *)calloc((*matrix).num_rows+1, sizeof(int));
//+1 to store the number of non-zero elements in the last index of the row pointers aray
        1:  118:    (*matrix).col_ind = (int *)calloc((*matrix).num_non_zeros, sizeof(int));
        -:  119:
        -:  120:
        1:  121:    int temp_row [(*matrix).num_non_zeros];
        1:  122:    int temp_col [(*matrix).num_non_zeros];
        1:  123:    double temp_data [(*matrix).num_non_zeros];
        -:  124:
        -:  125:    // Initialize arrays to zero
       47:  126:    for (int i = 0; i < (*matrix).num_non_zeros; i++)
        -:  127:    {
       46:  128:        temp_row[i] = 0;
       46:  129:        temp_col[i] = 0;
       46:  130:        temp_data[i] = 0.0;
        -:  131:    }
        -:  132:
        1:  133:    int ref = (*matrix).num_non_zeros - not_on_diagonal_count;   //reference t
o add the mirrored non-zero entries to the upper triangular part of the matrix
        1:  134:    int count = 0;
        -:  135:    //Storing all of the non-zero elements in temporary arrays to manipulate t
heir order
       31:  136:    for (int i = 0; i < (*matrix).num_non_zeros - not_on_diagonal_count; i++)
   //-not_on_diagonal_count to only read the non-zero values that are displayed in the file
        -:  137:    {
       30:  138:        fgets(line, 1024, file);   //Getting the line from the file
       30:  139:        sscanf(line, "%d %d %lf", &temp_row[i], &temp_col[i], &temp_data[i]);
        -:  140:
        -:  141:        //Make matrix symmetrical beecause CSR does not store the other symmet
rical side of the matrix
       30:  142:        if (temp_row[i] != temp_col[i])
        -:  143:        {
        -:  144:            // If the non-zero entry is not in the diagonal, mirror this entry
 to the upper triangular portion
        -:  145:            //Filling the mirrored entries with the values from the bottom tri
angular by copying the data to the missing indices and swapping the row and column at that ind
ex
       16:  146:            temp_data[count + ref] = temp_data[i];
       16:  147:            temp_row [count + ref] = temp_col[i];
       16:  148:            temp_col [count + ref] = temp_row[i];
        -:  149:
       16:  150:            count ++;
        -:  151:
        -:  152:            //The rows and columns will be sorted below therefore the matrix w
ill be fully symmetric due to the mirroring
        -:  153:
        -:  154:            // Ensure that ref doesn't exceed the allocated space
       16:  155:            if ((ref + count) > (*matrix).num_non_zeros)
        -:  156:            {
    #####:  157:                fprintf(stderr, "Error: Array overflow.\n");
    #####:  158:                exit(EXIT_FAILURE);
        -:  159:            }
        -:  160:        }
        -:  161:    }
        -:  162:
        -:  163:    //Sorting elements by row number
        1:  164:    bubbleSortRows(temp_row, temp_col, temp_data, (*matrix).num_non_zeros);
        -:  165:
        -:  166:
        -:  167:    //Now that rows are sorted, for every row sort by increasing column number

        -:  168:    //*ChatGPT helped me in writing the logic for this for loop*
        1:  169:    int current_row_start = 0;
        1:  170:    int counter = 1;
        1:  171:    (*matrix).row_ptr[0] = 0;   //Hard setting the first row to the first inde
x of the column index
       47:  172:    for (int i = 0; i < (*matrix).num_non_zeros; i++)
        -:  173:    {
       46:  174:        if (i == (*matrix).num_non_zeros - 1 || temp_row[i] != temp_row[i+1])
   //If the end or the next row are reached --> sort the columns within that row
        -:  175:        {
       14:  176:            bubbleSortCols(current_row_start, i+1, temp_col, temp_data);
       14:  177:            current_row_start = i+1;       //Updating the starting index for t
:
```

Figure 12: Coverage Report (Part 2) for functions.c



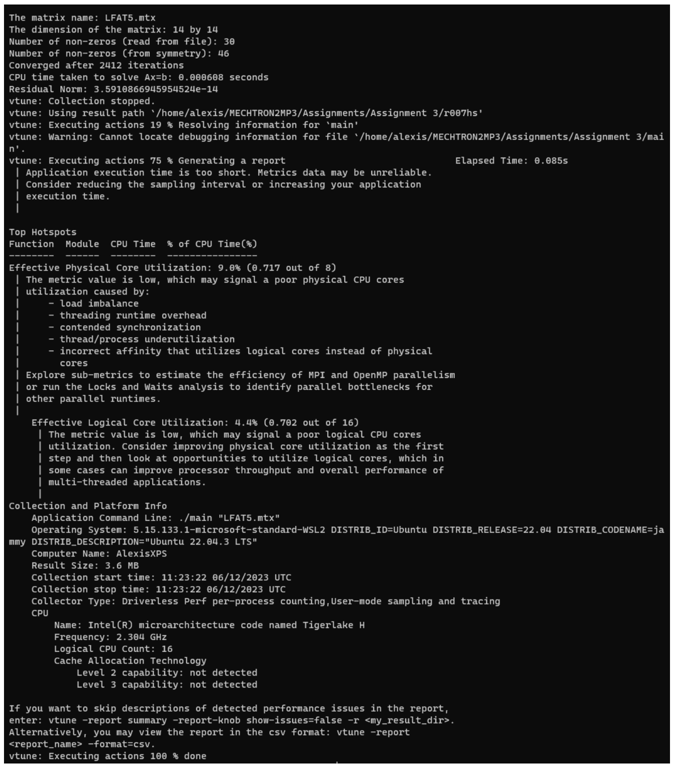
```
       14:  177:            current_row_start = i+1;       //Updating the starting index for t
he next row
       14:  178:            (*matrix).row_ptr[counter] = i+1;    //Updating the row pointer to
 indicate the starting indices for the next row
       14:  179:            counter++;
        -:  180:        }
        -:  181:    }
        -:  182:
        -:  183:    //Now that all of the data is sorted by increasing columns in every row, a
dd it to the members of the CSR_Matrix structure
       47:  184:    for (int i = 0; i < (*matrix).num_non_zeros; i++)
        -:  185:    {
       46:  186:        (*matrix).col_ind[i] = temp_col[i]-1;   //changing the indexing back t
o 0
       46:  187:        (*matrix).csr_data[i] = temp_data[i];
        -:  188:    }
        -:  189:
        1:  190:    printf("Number of non-zeros: %d\n", (*matrix).num_non_zeros);
        1:  191:    printf("Row pointer: ");
       15:  192:    for (int i = 0; i < (*matrix).num_rows; i++)
        -:  193:    {
       14:  194:        printf("%d ", (*matrix).row_ptr[i]);
        -:  195:    }
        1:  196:    printf("%d\n", (*matrix).num_non_zeros);
        -:  197:
        1:  198:    printf("Column Index: ");
       47:  199:    for (int i = 0; i < (*matrix).num_non_zeros; i++)
        -:  200:    {
       46:  201:        printf("%d ", (*matrix).col_ind[i]);
        -:  202:    }
        1:  203:    printf("\n");
        1:  204:    printf("Values: ");
       47:  205:    for (int i = 0; i < (*matrix).num_non_zeros; i++)
        -:  206:    {
       46:  207:        printf("%lf ", (*matrix).csr_data[i]);
        -:  208:    }
        1:  209:    printf("\n");
        -:  210:
        -:  211:
        1:  212:    fclose(file);
        -:  213:
        1:  214:}
        -:  215:
        1:  216:void spmv_csr(const CSRMatrix *A, const double *x, double *y)
        -:  217:{
       15:  218:    for (int i = 0; i < (*A).num_rows; i++)   //Iterating through every row in
 the matrix
        -:  219:    {
       14:  220:        double sum = 0.0;   //sum for the current row
       60:  221:        for (int j = (*A).row_ptr[i]; j < (*A).row_ptr[i+1]; j++)   //Iteratin
g through every column in every row using the row poitner
        -:  222:        {
       46:  223:            sum += (*A).csr_data[j]*x[(*A).col_ind[j]];    //Multiplying the c
orresponding elements of the A CSR matrix and the x vector
        -:  224:        }
        -:  225:
       14:  226:        y[i] = sum;     //Storing the sum of the products in the corresponding
 position in the y vector
        -:  227:    }
        1:  228:}
        -:  229:
        -:  230:
        1:  231:void solverJacobi(const CSRMatrix *A, double *b, double *x, int max_iterations
, double tolerance)
        -:  232:{
        1:  233:    double *x_next = (double *)calloc((*A).num_rows, sizeof(double));    //All
ocating memory for the x_next vector and initially setting it to 0 with calloc
        -:  234:
     2412:  235:    for (int iterations = 0; iterations < max_iterations; iterations++)    //R
un the method until the max iteration runs
        -:  236:    {
    36180:  237:        for (int i = 0; i < (*A).num_rows; i++)   //Approximate the solution f
or every row
        -:  238:        {
    33768:  239:            double sum = 0.0;
    33768:  240:            double diagonal_val = 0.0;
   144720:  241:            for (int j = (*A).row_ptr[i]; j < (*A).row_ptr[i+1]; j++)   //Iter
ate through the non-zero values for every row
        -:  242:            {
   110952:  243:                if ((*A).col_ind[j] != i)   //If the column is not the same nu
mber as the row
        -:  244:                {
    77184:  245:                    sum += (*A).csr_data[j]*x[(*A).col_ind[j]];    //From the
formula from the provided source
        -:  246:                }
        -:  247:                else
        -:  248:                {
    33768:  249:                    diagonal_val = (*A).csr_data[j];
        -:  250:                }
        -:  251:            }
        -:  252:
    33768:  253:            x_next[i] = (b[i] - sum)/diagonal_val;    //From the formula from
the provided link
        -:  254:
        -:  255:        }
        -:  256:
        -:  257:        //Calculating the error within the values of x to compare to the toler
ance after
     2412:  258:        double error = 0.0;
    36180:  259:        for (int i = 0; i < (*A).num_rows; i++)
        -:  260:        {
:
```

Figure 13: Coverage Report (Part 3) for functions.c



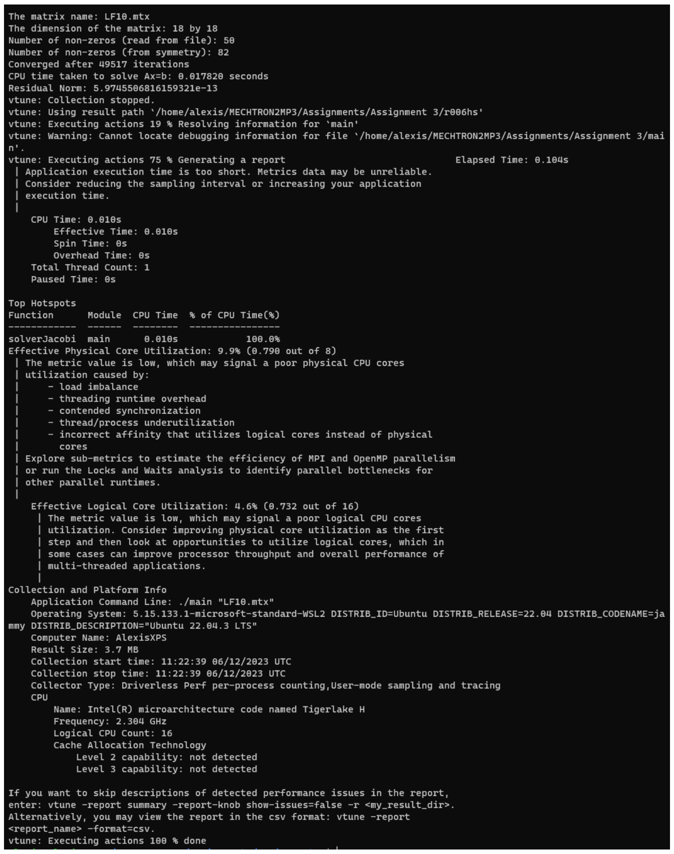
```
        -:  260:        {
    33768:  261:            error += fabs(x_next[i] - x[i]);
        -:  262:        }
        -:  263:
        -:  264:        //Updating the solution vector with the approximations
    36180:  265:        for (int i = 0; i < (*A).num_rows; i++)
        -:  266:        {
    33768:  267:            x[i] = x_next[i];
        -:  268:        }
        -:  269:
        -:  270:        //Check if the method should terminate due to tolerance reached
     2412:  271:        if (error < tolerance)
        -:  272:        {
        1:  273:            printf("Converged after %d iterations \n", iterations+1);
        1:  274:            break;    //not continuing the solver if the tolerance is reached
        -:  275:        }
        -:  276:    }
        -:  277:
        1:  278:    free(x_next);
        1:  279:}
        -:  280:
        -:  281:
        1:  282:void compute_residual(const CSRMatrix *A, const double *x, double *y, const do
uble *b, double *r)
        -:  283:{
        -:  284:    //Calculting A times x with the spmv_csr function that was made before
        -:  285:    //The x vector was calculated with the solver method used before in the ma
in function
        -:  286:    //The result to this operation is stored in the vector y
        -:  287:
        1:  288:    spmv_csr(A, x, y);
        -:  289:
        -:  290:    //Obtaining the difference between the two by subtracting the soltuion b f
rom the calculated solution vector y
       15:  291:    for (int i = 0; i < (*A).num_rows; i++)
        -:  292:    {
       14:  293:        r[i] = y[i]-b[i];
        -:  294:    }
        -:  295:
        1:  296:}
        -:  297:
        1:  298:double compute_norm(const CSRMatrix *A, const double *r)
        -:  299:{
        1:  300:    double sum = 0.0;
        -:  301:    //The norm of a vector is the square root of the sum of the squares of the
 entries
       15:  302:    for (int i = 0; i < (*A).num_rows; i++)
        -:  303:    {
       14:  304:        sum += r[i]*r[i];
        -:  305:    }
        -:  306:
        1:  307:    return sqrt(sum);
        -:  308:}
        -:  309:
        -:  310:
(END)
```

Figure 14: Coverage Report (Part 4) for functions.c



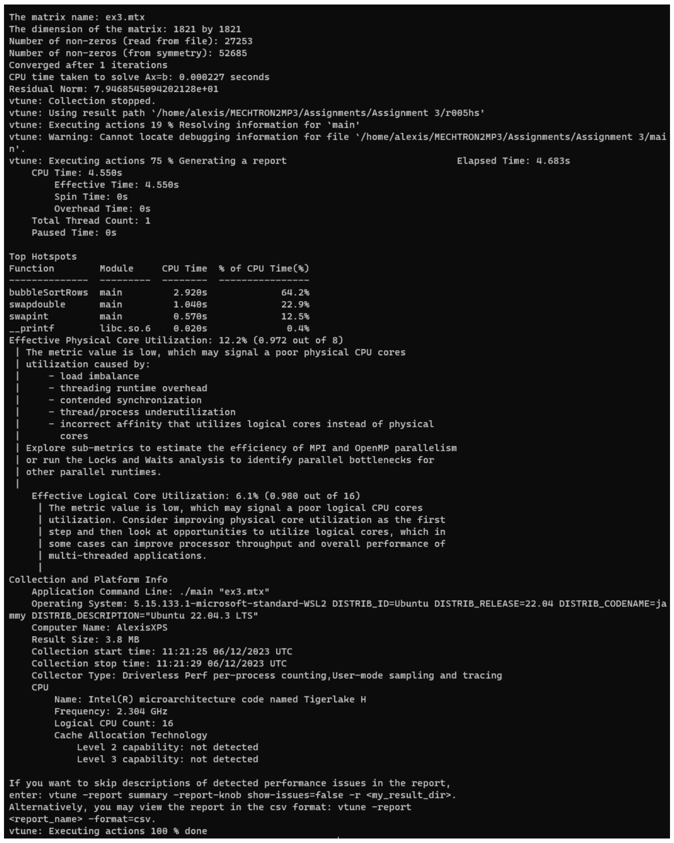
```
The matrix name: LFAT5.mtx
The dimension of the matrix: 14 by 14
Number of non-zeros (read from file): 30
Number of non-zeros (from symmetry): 46
Converged after 2412 iterations
CPU time taken to solve Ax=b: 0.000608 seconds
Residual Norm: 3.5910866945954524e-14
vtune: Collection stopped.
vtune: Using result path `/home/alexis/MECHTRON2MP3/Assignments/Assignment 3/r007hs'
vtune: Executing actions 19 % Resolving information for `main'
vtune: Warning: Cannot locate debugging information for file `/home/alexis/MECHTRON2MP3/Assignments/Assignment 3/mai
n'.
vtune: Executing actions 75 % Generating a report                              Elapsed Time: 0.085s
 | Application execution time is too short. Metrics data may be unreliable.
 | Consider reducing the sampling interval or increasing your application
 | execution time.
 |

Top Hotspots
Function  Module  CPU Time  % of CPU Time(%)
--------  ------  --------  ----------------
Effective Physical Core Utilization: 9.0% (0.717 out of 8)
 | The metric value is low, which may signal a poor physical CPU cores
 | utilization caused by:
 |     - load imbalance
 |     - threading runtime overhead
 |     - contended synchronization
 |     - thread/process underutilization
 |     - incorrect affinity that utilizes logical cores instead of physical
 |       cores
 | Explore sub-metrics to estimate the efficiency of MPI and OpenMP parallelism
 | or run the Locks and Waits analysis to identify parallel bottlenecks for
 | other parallel runtimes.
 |
    Effective Logical Core Utilization: 4.4% (0.702 out of 16)
     | The metric value is low, which may signal a poor logical CPU cores
     | utilization. Consider improving physical core utilization as the first
     | step and then look at opportunities to utilize logical cores, which in
     | some cases can improve processor throughput and overall performance of
     | multi-threaded applications.
     |
Collection and Platform Info
    Application Command Line: ./main "LFAT5.mtx"
    Operating System: 5.15.133.1-microsoft-standard-WSL2 DISTRIB_ID=Ubuntu DISTRIB_RELEASE=22.04 DISTRIB_CODENAME=ja
mmy DISTRIB_DESCRIPTION="Ubuntu 22.04.3 LTS"
    Computer Name: AlexisXPS
    Result Size: 3.6 MB
    Collection start time: 11:23:22 06/12/2023 UTC
    Collection stop time: 11:23:22 06/12/2023 UTC
    Collector Type: Driverless Perf per-process counting,User-mode sampling and tracing
    CPU
        Name: Intel(R) microarchitecture code named Tigerlake H
        Frequency: 2.304 GHz
        Logical CPU Count: 16
        Cache Allocation Technology
            Level 2 capability: not detected
            Level 3 capability: not detected

If you want to skip descriptions of detected performance issues in the report,
enter: vtune -report summary -report-knob show-issues=false -r <my_result_dir>.
Alternatively, you may view the report in the csv format: vtune -report
<report_name> -format=csv.
vtune: Executing actions 100 % done
```

Figure 15: Vtune analysis for LFAT5.mtx



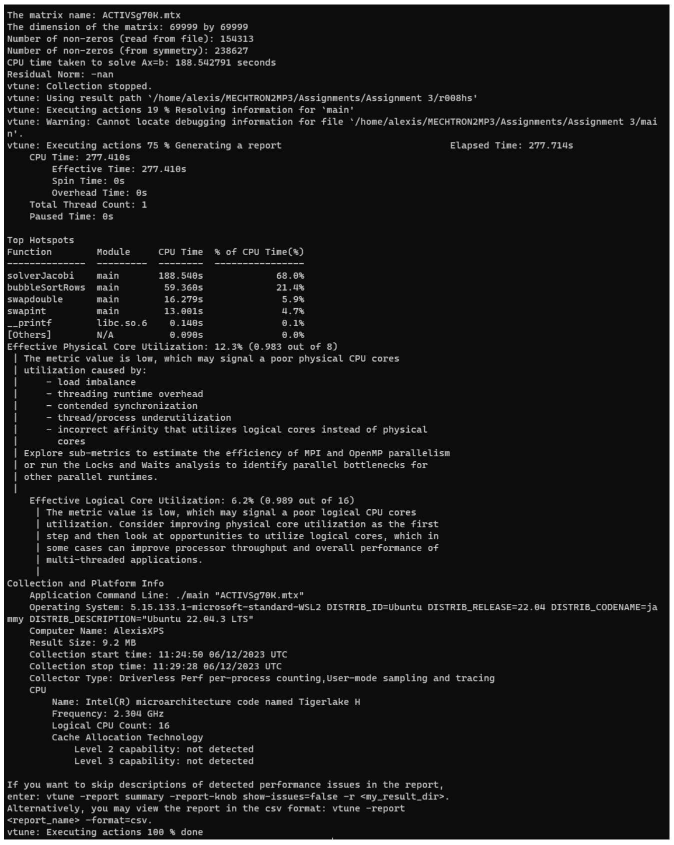
```
The matrix name: LF10.mtx
The dimension of the matrix: 18 by 18
Number of non-zeros (read from file): 50
Number of non-zeros (from symmetry): 82
Converged after 49517 iterations
CPU time taken to solve Ax=b: 0.017820 seconds
Residual Norm: 5.9745506816159321e-13
vtune: Collection stopped.
vtune: Using result path `/home/alexis/MECHTRON2MP3/Assignments/Assignment 3/r006hs'
vtune: Executing actions 19 % Resolving information for `main'
vtune: Warning: Cannot locate debugging information for file `/home/alexis/MECHTRON2MP3/Assignments/Assignment 3/mai
n'.
vtune: Executing actions 75 % Generating a report                              Elapsed Time: 0.104s
 | Application execution time is too short. Metrics data may be unreliable.
 | Consider reducing the sampling interval or increasing your application
 | execution time.
 |
    CPU Time: 0.010s
        Effective Time: 0.010s
        Spin Time: 0s
        Overhead Time: 0s
    Total Thread Count: 1
    Paused Time: 0s

Top Hotspots
Function      Module  CPU Time  % of CPU Time(%)
------------  ------  --------  ----------------
solverJacobi  main      0.010s            100.0%
Effective Physical Core Utilization: 9.9% (0.790 out of 8)
 | The metric value is low, which may signal a poor physical CPU cores
 | utilization caused by:
 |     - load imbalance
 |     - threading runtime overhead
 |     - contended synchronization
 |     - thread/process underutilization
 |     - incorrect affinity that utilizes logical cores instead of physical
 |       cores
 | Explore sub-metrics to estimate the efficiency of MPI and OpenMP parallelism
 | or run the Locks and Waits analysis to identify parallel bottlenecks for
 | other parallel runtimes.
 |
    Effective Logical Core Utilization: 4.6% (0.732 out of 16)
     | The metric value is low, which may signal a poor logical CPU cores
     | utilization. Consider improving physical core utilization as the first
     | step and then look at opportunities to utilize logical cores, which in
     | some cases can improve processor throughput and overall performance of
     | multi-threaded applications.
     |
Collection and Platform Info
    Application Command Line: ./main "LF10.mtx"
    Operating System: 5.15.133.1-microsoft-standard-WSL2 DISTRIB_ID=Ubuntu DISTRIB_RELEASE=22.04 DISTRIB_CODENAME=ja
mmy DISTRIB_DESCRIPTION="Ubuntu 22.04.3 LTS"
    Computer Name: AlexisXPS
    Result Size: 3.7 MB
    Collection start time: 11:22:39 06/12/2023 UTC
    Collection stop time: 11:22:39 06/12/2023 UTC
    Collector Type: Driverless Perf per-process counting,User-mode sampling and tracing
    CPU
        Name: Intel(R) microarchitecture code named Tigerlake H
        Frequency: 2.304 GHz
        Logical CPU Count: 16
        Cache Allocation Technology
            Level 2 capability: not detected
            Level 3 capability: not detected

If you want to skip descriptions of detected performance issues in the report,
enter: vtune -report summary -report-knob show-issues=false -r <my_result_dir>.
Alternatively, you may view the report in the csv format: vtune -report
<report_name> -format=csv.
vtune: Executing actions 100 % done
```

Figure 16: Vtune analysis for LF10.mtx



```
The matrix name: ex3.mtx
The dimension of the matrix: 1821 by 1821
Number of non-zeros (read from file): 27253
Number of non-zeros (from symmetry): 52685
Converged after 1 iterations
CPU time taken to solve Ax=b: 0.000227 seconds
Residual Norm: 7.9468545094202128e+01
vtune: Collection stopped.
vtune: Using result path `/home/alexis/MECHTRON2MP3/Assignments/Assignment 3/r005hs'
vtune: Executing actions 19 % Resolving information for `main'
vtune: Warning: Cannot locate debugging information for file `/home/alexis/MECHTRON2MP3/Assignments/Assignment 3/mai
n'.
vtune: Executing actions 75 % Generating a report                              Elapsed Time: 4.683s
    CPU Time: 4.550s
        Effective Time: 4.550s
        Spin Time: 0s
        Overhead Time: 0s
    Total Thread Count: 1
    Paused Time: 0s

Top Hotspots
Function        Module     CPU Time  % of CPU Time(%)
--------------  ---------  --------  ----------------
bubbleSortRows  main         2.920s             64.2%
swapdouble      main         1.040s             22.9%
swapint         main         0.570s             12.5%
__printf        libc.so.6    0.020s              0.4%
Effective Physical Core Utilization: 12.2% (0.972 out of 8)
 | The metric value is low, which may signal a poor physical CPU cores
 | utilization caused by:
 |     - load imbalance
 |     - threading runtime overhead
 |     - contended synchronization
 |     - thread/process underutilization
 |     - incorrect affinity that utilizes logical cores instead of physical
 |       cores
 | Explore sub-metrics to estimate the efficiency of MPI and OpenMP parallelism
 | or run the Locks and Waits analysis to identify parallel bottlenecks for
 | other parallel runtimes.
 |
    Effective Logical Core Utilization: 6.1% (0.980 out of 16)
     | The metric value is low, which may signal a poor logical CPU cores
     | utilization. Consider improving physical core utilization as the first
     | step and then look at opportunities to utilize logical cores, which in
     | some cases can improve processor throughput and overall performance of
     | multi-threaded applications.
     |
Collection and Platform Info
    Application Command Line: ./main "ex3.mtx"
    Operating System: 5.15.133.1-microsoft-standard-WSL2 DISTRIB_ID=Ubuntu DISTRIB_RELEASE=22.04 DISTRIB_CODENAME=ja
mmy DISTRIB_DESCRIPTION="Ubuntu 22.04.3 LTS"
    Computer Name: AlexisXPS
    Result Size: 3.8 MB
    Collection start time: 11:21:25 06/12/2023 UTC
    Collection stop time: 11:21:29 06/12/2023 UTC
    Collector Type: Driverless Perf per-process counting,User-mode sampling and tracing
    CPU
        Name: Intel(R) microarchitecture code named Tigerlake H
        Frequency: 2.304 GHz
        Logical CPU Count: 16
        Cache Allocation Technology
            Level 2 capability: not detected
            Level 3 capability: not detected

If you want to skip descriptions of detected performance issues in the report,
enter: vtune -report summary -report-knob show-issues=false -r <my_result_dir>.
Alternatively, you may view the report in the csv format: vtune -report
<report_name> -format=csv.
vtune: Executing actions 100 % done
```

Figure 17: Vtune analysis for ex3.mtx



```
The matrix name: ACTIVSg70K.mtx
The dimension of the matrix: 69999 by 69999
Number of non-zeros (read from file): 154313
Number of non-zeros (from symmetry): 238627
CPU time taken to solve Ax=b: 188.542791 seconds
Residual Norm: -nan
vtune: Collection stopped.
vtune: Using result path `/home/alexis/MECHTRON2MP3/Assignments/Assignment 3/r008hs'
vtune: Executing actions 19 % Resolving information for `main'
vtune: Warning: Cannot locate debugging information for file `/home/alexis/MECHTRON2MP3/Assignments/Assignment 3/mai
n'.
vtune: Executing actions 75 % Generating a report                              Elapsed Time: 277.714s
    CPU Time: 277.410s
        Effective Time: 277.410s
        Spin Time: 0s
        Overhead Time: 0s
    Total Thread Count: 1
    Paused Time: 0s

Top Hotspots
Function        Module     CPU Time  % of CPU Time(%)
--------------  ---------  --------  ----------------
solverJacobi    main       188.540s             68.0%
bubbleSortRows  main        59.360s             21.4%
swapdouble      main        16.279s              5.9%
swapint         main        13.001s              4.7%
__printf        libc.so.6    0.140s              0.1%
[Others]        N/A          0.090s              0.0%
Effective Physical Core Utilization: 12.3% (0.983 out of 8)
 | The metric value is low, which may signal a poor physical CPU cores
 | utilization caused by:
 |     - load imbalance
 |     - threading runtime overhead
 |     - contended synchronization
 |     - thread/process underutilization
 |     - incorrect affinity that utilizes logical cores instead of physical
 |       cores
 | Explore sub-metrics to estimate the efficiency of MPI and OpenMP parallelism
 | or run the Locks and Waits analysis to identify parallel bottlenecks for
 | other parallel runtimes.
 |
    Effective Logical Core Utilization: 6.2% (0.989 out of 16)
     | The metric value is low, which may signal a poor logical CPU cores
     | utilization. Consider improving physical core utilization as the first
     | step and then look at opportunities to utilize logical cores, which in
     | some cases can improve processor throughput and overall performance of
     | multi-threaded applications.
     |
Collection and Platform Info
    Application Command Line: ./main "ACTIVSg70K.mtx"
    Operating System: 5.15.133.1-microsoft-standard-WSL2 DISTRIB_ID=Ubuntu DISTRIB_RELEASE=22.04 DISTRIB_CODENAME=ja
mmy DISTRIB_DESCRIPTION="Ubuntu 22.04.3 LTS"
    Computer Name: AlexisXPS
    Result Size: 9.2 MB
    Collection start time: 11:24:50 06/12/2023 UTC
    Collection stop time: 11:29:28 06/12/2023 UTC
    Collector Type: Driverless Perf per-process counting,User-mode sampling and tracing
    CPU
        Name: Intel(R) microarchitecture code named Tigerlake H
        Frequency: 2.304 GHz
        Logical CPU Count: 16
        Cache Allocation Technology
            Level 2 capability: not detected
            Level 3 capability: not detected

If you want to skip descriptions of detected performance issues in the report,
enter: vtune -report summary -report-knob show-issues=false -r <my_result_dir>.
Alternatively, you may view the report in the csv format: vtune -report
<report_name> -format=csv.
vtune: Executing actions 100 % done
```

Figure 18: Vtune analysis for ACTIVSg70K.mtx