ASSIGNMENT 3

Solving System of Linear Equations

NATHAN SINHA

1 Introduction

Assignment 3 required that I create a function that is able to calculate a vector x in the equation Ax = b where A is a given symmetrical matrix and b is a matrix of only the value 1. To solve for x the Gauss-Seidel method was used. Additionally, before solving for x, the matrix A was converted from matrix market format to csr format. After x was found its accuracy tested via the residual norm.

2 Guass-Seidel Method

The Guass-Seidel Method is a method that improves on the Jacobi method. It operates under the assumption that the diagonal is filled with non-zero numbers. This method can be represented by the following:

$$\begin{aligned} aX_{11} + bX_{21} + cX_{31} &= Y_1 \\ aX_{12} + bX_{22} + cX_{32} &= Y_2 \\ aX_{13} + bX_{23} + cX_{33} &= Y_3 \end{aligned}$$

Where a, b, and c are the values in 'x'. These equations are then rewritten as follows:

$$\begin{aligned} a &= (Y_1 - (bX_{21} + cX_{31}))/X_{11} \\ b &= (Y_2 - (aX_{12} + cX_{32}))/X_{22} \\ c &= (Y_3 - (aX_{13} + bX_{23}))/X_{33} \end{aligned}$$

To begin we make an assumption for the values of 'a', 'b', and 'c'. It is common to initialize them as zero. We then sub the values of 'b' and 'c' to solve for 'a', this will update 'a'. We repeat this step to solve for 'b', this time using the new value of 'a' just calculated. This is repeated for 'a', 'b', and 'c' however many times. After it has finished iterating the most recent results for 'a', 'b', and 'c' is your solution.

Additionally if the difference between the previous result and the new results is below a tolerance the method is complete.
Recall that the diagonals cannot be zero, this is to avoid a 0 division error. Also note that large matrices will not solve because the iterations require for the x vector to converge is to large.

3 Makefile

```
CC = gcc
CFLAGS = -Wall -Wextra -std=c99
SOURCES = functions.c main.c
EXECUTABLE = myprogram
LIBS = -lm

all: $(EXECUTABLE)

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

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

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

This Makefile runs similarly to examples found in lecture. The first 6 lines of the compiler are used to set the inputs when compiling. These lines set the compiler, flags, additional library and the name of the output. It also takes the source files it will be using and creates objects for them.

The following lines organizes the previously gathered information such that the executable will compile with it.

A clean command is used to remove the compiled output should you wish.

To run the Makefile use 'make' in the terminal, to use the clean function use 'make clean'.

4 TABLES

Table 1: Results

Porblem (.mtx)	Size		Non-Zeros	CPU time (Sec)	Norm-Residual
	row	*column*			
LFAT5	14	14	30	0.000162	2.66e-14
LF10	18	18	50	0.005846	6.01e-13
ex3	1821	1821	27253	133.910588	29.93
jnlbrng1	40000	40000	119600	0.714396	1.25e-12
ACTIVSg70K	-	-	-	-	-
2cubes$_{s}phere$	-	-	-	-	-
tmt$_{s}ym$	-	-	-	-	-
StocF-1465	-	-	-	-	-

5 Plots

5.1 LFAT5

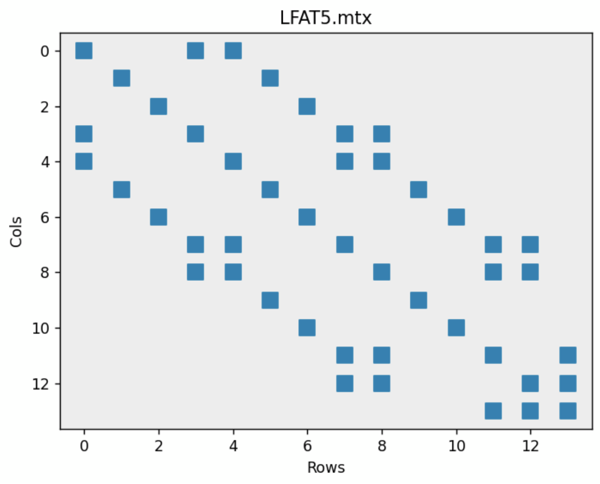


Figure 1: Enter Caption

5.2 LF10

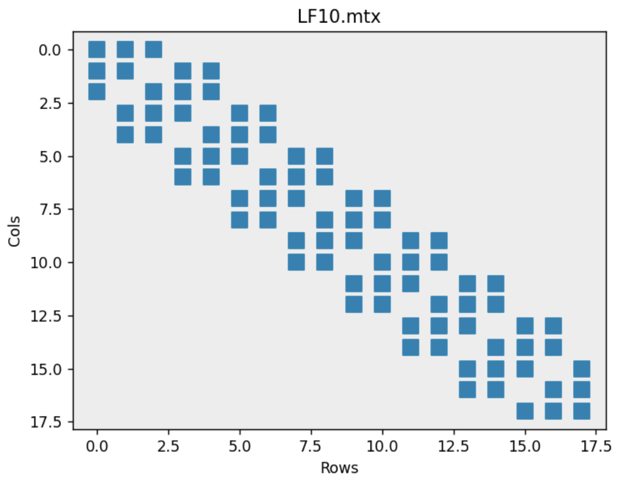


Figure 2: Enter Caption

5.3 ex3

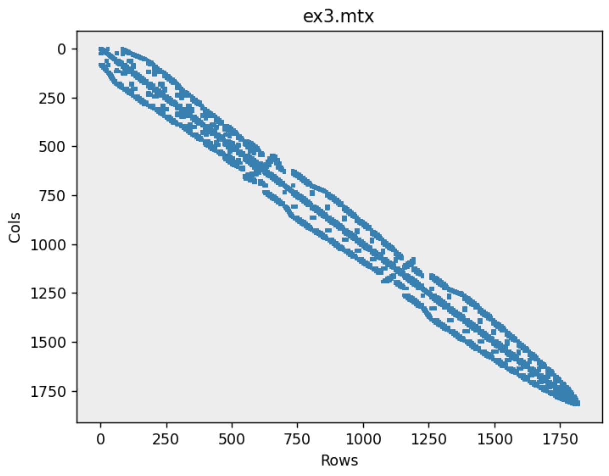


Figure 3: Enter Caption

5.4 jnlbrng1

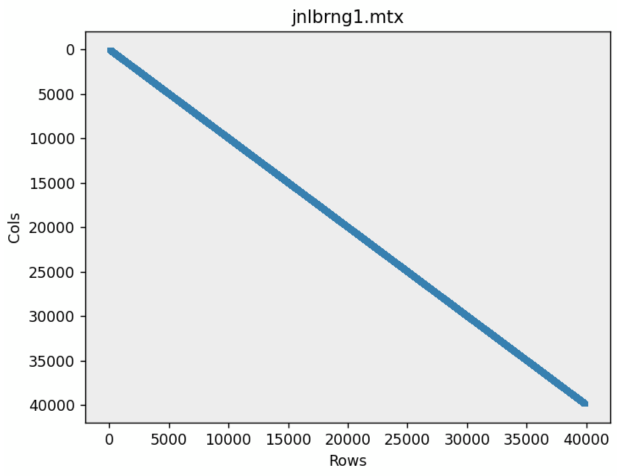


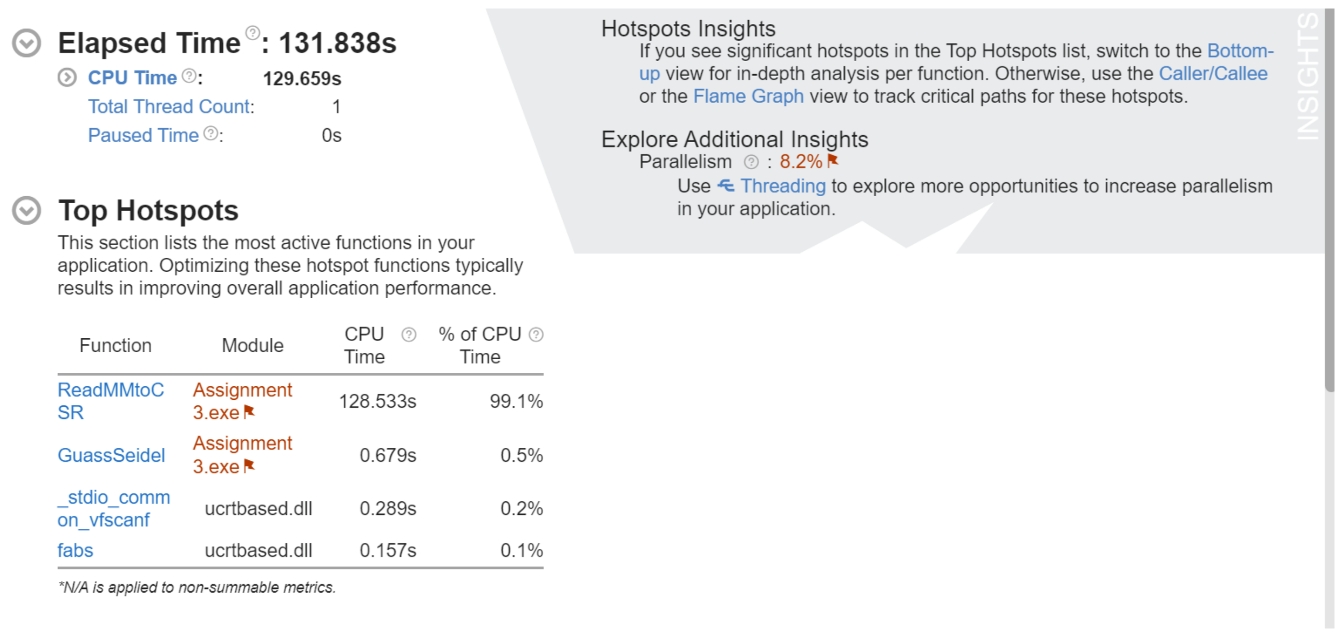
Figure 4: jnlbrng1 Sparce Matrix

5.5 Plotting Code

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

# Gets file to use
matrix = mmread('C:/Users/Windows/Downloads/mtx/...').toarray() #
   change directory as needed

# Finds location of NZE
non_zero_indices = np.nonzero(matrix)

# Plots NZE
plot.scatter(non_zero_indices[1], non_zero_indices[0], marker='s',
   s=x, c='#1e81b0') # adjust size as needed

# Formatting
plot.gca().invert_yaxis() # origin at top left
plot.gca().set_facecolor('#eeeeee')
plot.title('x.mtx') # change title as needed
plot.xlabel('Rows')
plot.ylabel('Cols')

# display plot
plot.show()
```

Note: plot was made using python.

6 VTune



Elapsed Time ⓘ: 131.838s

CPU Time ⓘ:	**129.659s**
Total Thread Count:	1
Paused Time ⓘ:	0s

Top Hotspots

This section lists the most active functions in your application. Optimizing these hotspot functions typically results in improving overall application performance.

Function	Module	CPU Time ⓘ	% of CPU Time ⓘ
ReadMMtoCSR	Assignment3.exe	128.533s	99.1%
GuassSeidel	Assignment3.exe	0.679s	0.5%
_stdio_common_vfscanf	ucrtbased.dll	0.289s	0.2%
fabs	ucrtbased.dll	0.157s	0.1%

**N/A is applied to non-summable metrics.*

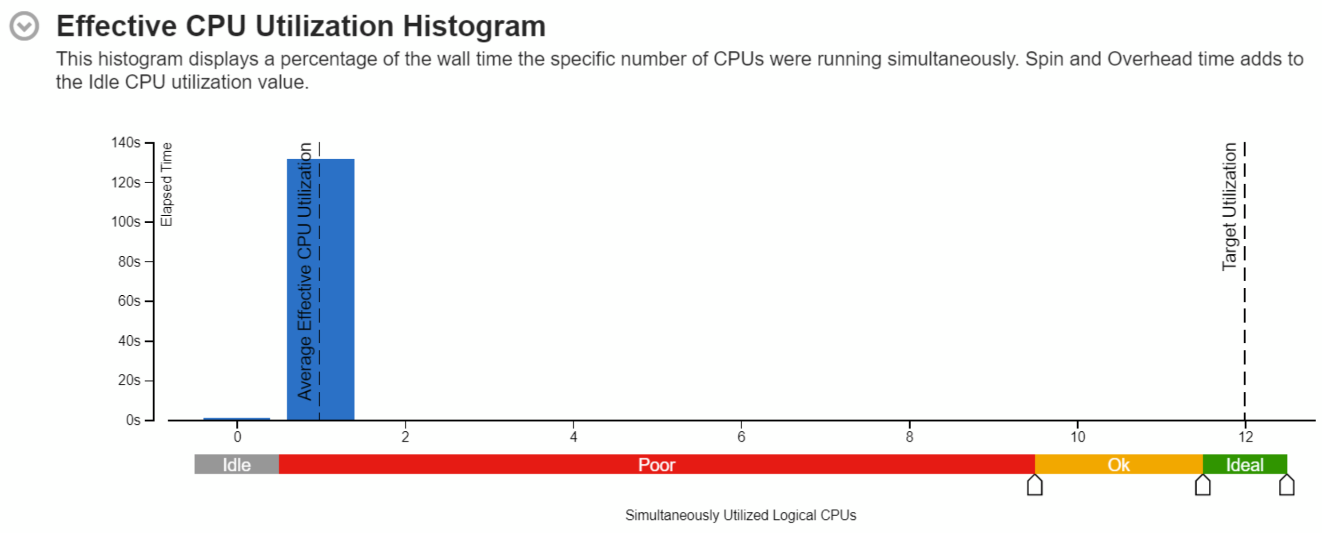
Hotspots Insights

If you see significant hotspots in the Top Hotspots list, switch to the Bottom-up view for in-depth analysis per function. Otherwise, use the Caller/Callee or the Flame Graph view to track critical paths for these hotspots.

Explore Additional Insights

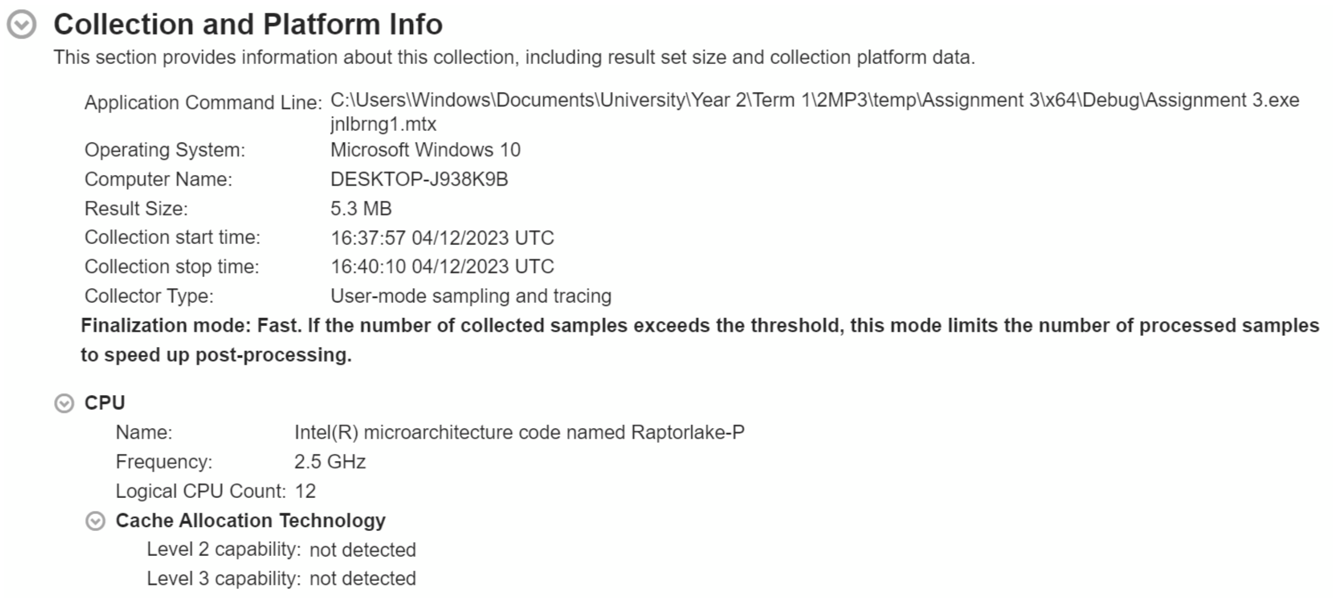
Parallelism ⓘ : 8.2%

Use Threading to explore more opportunities to increase parallelism in your application.



Effective CPU Utilization Histogram

This histogram displays a percentage of the wall time the specific number of CPUs were running simultaneously. Spin and Overhead time adds to the Idle CPU utilization value.



Collection and Platform Info

This section provides information about this collection, including result set size and collection platform data.

Application Command Line:	C:\Users\Windows\Documents\University\Year 2\Term 1\2MP3\temp\Assignment 3\x64\Debug\Assignment 3.exe jnlbrng1.mtx
Operating System:	Microsoft Windows 10
Computer Name:	DESKTOP-J938K9B
Result Size:	5.3 MB
Collection start time:	16:37:57 04/12/2023 UTC
Collection stop time:	16:40:10 04/12/2023 UTC
Collector Type:	User-mode sampling and tracing

Finalization mode: Fast. If the number of collected samples exceeds the threshold, this mode limits the number of processed samples to speed up post-processing.

CPU

Name:	Intel(R) microarchitecture code named Raptorlake-P
Frequency:	2.5 GHz
Logical CPU Count:	12

Cache Allocation Technology

Level 2 capability:	not detected
Level 3 capability:	not detected

By observing the results of vTune we can conclude that the function ReadMMtoCSR is highly inefficient.

7 gcov

Output after using gcov:

```
File 'main.c'
Lines executed:97.06% of 34
Creating 'main.c.gcov'

File 'functions.c'
Lines executed:88.66% of 97
Creating 'functions.c.gcov'

Lines executed:90.84% of 131
```

Here we can see that the code is working as intended. The lines that are not executed are likely due to them being skipped over on the assumption they are if, else if, and/or else loops that's requirements to not get meet