MECHTRON 2MP3 - Porgramming for Mechatronics

Creating a Matrix Solver in C

Marco Tan, 400433483

1 Introduction

For this assignment, we created C code that reads a Matrix Market file, saves it as some CSRMatrix structure, and uses it to solve for $A * x = b$.

2 Algorithm

The algorithm employed in my program was the Jacobi method, a method of solving for matrices iteratively. We set the solution matrix x to some initial guess. In my code, we set the values to be zero initially. Then, on each iteration, we calculate the next value of x with the following algorithm:

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

This works for most cases. However, there are cases where the solution never converges. Such an example is when the matrix is singular, or there exists a zero somewhere down the diagonal. In my code, I also added a Jacobi preprocessor, which attempts to swap rows around until there are zeros down the diagonal.

Another way this method could fail is if the matrix is not strictly diagonally dominant. In a strictly diagonally dominant system, the following inequality is satisfied down the diagonal:

$$|a_{ii}| \geq \sum_{j \neq i} |a_{ij}|$$

3 Results

Table 1: Results of Jacobi Method, Max 10,000 iterations, Residual Threshold of 1e-7

Problem	Size		Non-zeros	CPU time (sec)	Norm-Residual
	row	column			
LFAT5.mtx	14	14	30	0.008	9.823480e-08
LF10.mtx	18	18	50	0.203	1.656052e-05
ex3.mtx	1821	1821	27253	57.678	3.636329e+01
jnlbrng1.mtx	40000	40000	119600	19.635	9.906812e-08
ACTIVSg70K	69999	69999	154313	6:22.76	-nan
2cubes_sphere.mtx	101492	101492	874373	10.353	8.122470e-08
tmt_sym.mtx	726713	726713	2903837	N\A	N\A
StocF-1465.mtx	1465137	1465137	11235263	N\A	N\A

Values were not gotten for `tmt_sym.mtx` and `StocF-1465.mtx` because iteration was way too slow for me to get values within a reasonable amount of time.

The residual was poor for `ex3.mtx` potentially because the matrix is not strictly diagonally dominant. However, considering that the norm of the residual was pretty low, maybe I just needed to give it more time to iterate through to converge to a nice solution. In a similar vain, `ACTIVSg70K.mtx` is also not strictly diagonally dominant, and we can see by the norm of the residual that our solution has diverged to negative infinity, and so this isn't solvable via the Jacobi method.

3.1 VTune

Here is the VTune CLI output:

```
vtune -collect hotspots -report summary ./bin/main ./test/LFAT5.mtx
vtune: Warning: Only user space will be profiled due to credentials lack. Consider changing
vtune: Collection started. To stop the collection, either press CTRL-C or enter from anothe
Method: Jacobi
Number of iterations: 10000
Threshold: 1.000000e-07
Preconditioner: Jacobi

Raw print of CSRMatrix:
        - Number of rows: 14
        - Number of columns: 14
```

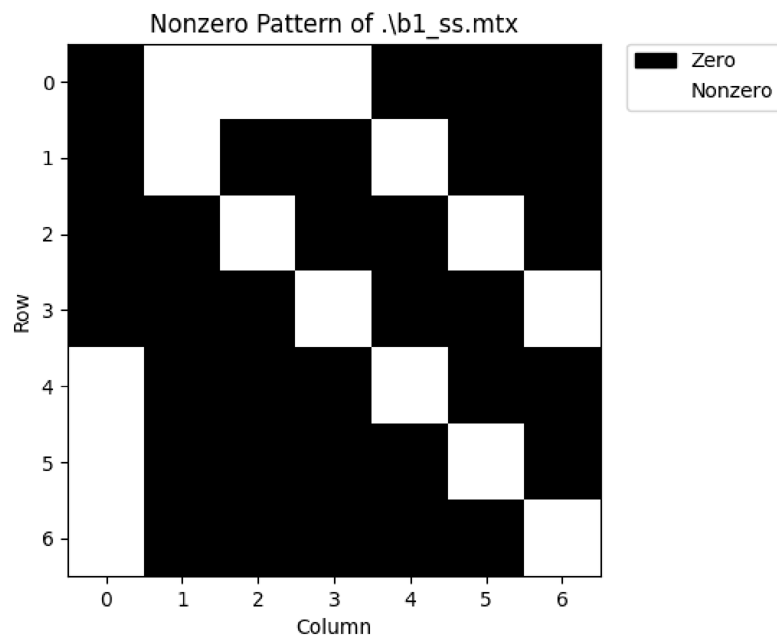
```
        - Number of non-zero elements: 30
The matrix is lower triangular

The matrix is not strictly diagonally dominant
Solving the system...
Warning: preconditioning is enabled. The original matrix will not be preserved.
Preconditioning complete.
Iteration: 518
Residual reached threshold. Stopping iterations.
Solver complete.


Residual: 9.823480e-08
vtune: Collection stopped.
vtune: Using result path `/home/ionicargon/MECHTRON-2MP3/github/MT2MP3-MatrixSolver/r001hs
vtune: Executing actions 75 % Generating a report                              Elapsed Time
 | 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: 7.8% (0.467 out of 6)
 | 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: 4.7% (0.560 out of 12)
     | The metric value is low, which may signal a poor logical CPU cores
```

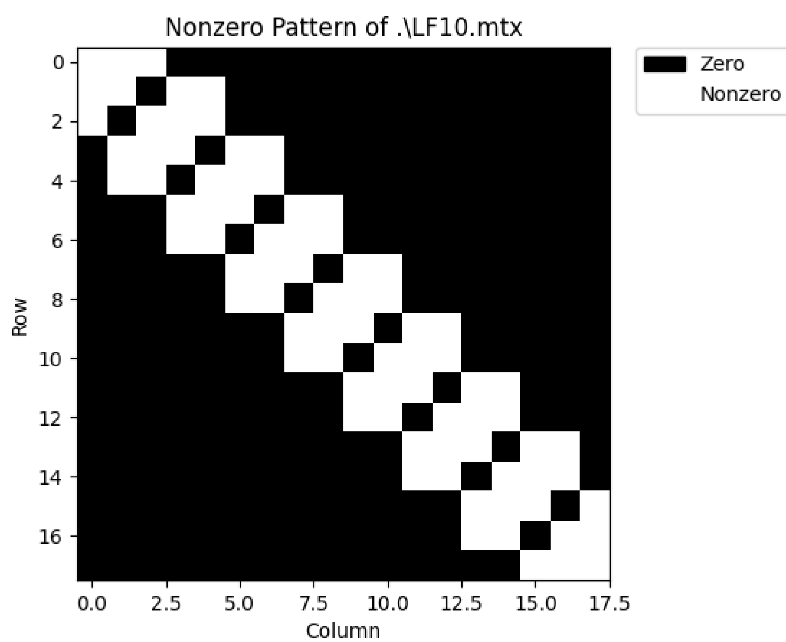
```
     | utilization. Consider improving physical core utilization as the first
     | step and then look at opportunities to utilize logical cores, which in
     | some cases can improve processor throughput and overall performance of
     | multi-threaded applications.
     |
Collection and Platform Info
    Application Command Line: ./bin/main "./test/LFAT5.mtx"
    Operating System: 5.15.133.1-microsoft-standard-WSL2 DISTRIB_ID=Ubuntu DISTRIB_RELEASE=
    Computer Name: DESKTOP-J8KNLS6
    Result Size: 3.6 MB
    Collection start time: 04:36:39 04/12/2023 UTC
    Collection stop time: 04:36:40 04/12/2023 UTC
    Collector Type: Driverless Perf per-process counting,User-mode sampling and tracing
    CPU
        Name: Intel(R) microarchitecture code named Coffeelake
        Frequency: 2.208 GHz
        Logical CPU Count: 12
        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
```

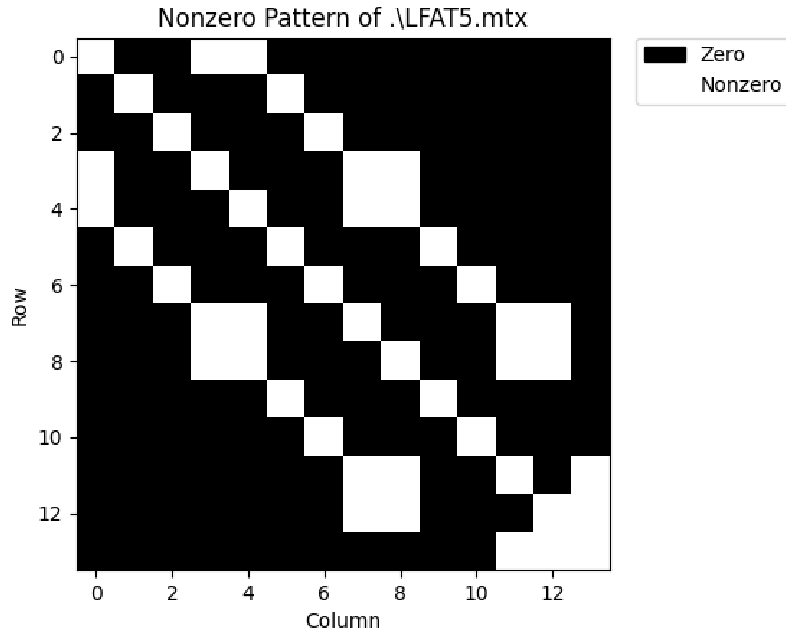
The VTune outputs for all the other files (except for the ones that I couldn't solve) were pretty similar. We have poor core usage, probably because we're not parallelizing the code. It's hard to effectively parallelize this algorithm though because it is iterative and relies on past values to predict future values, which means it's difficult for the threads to reliably communicate between each other.

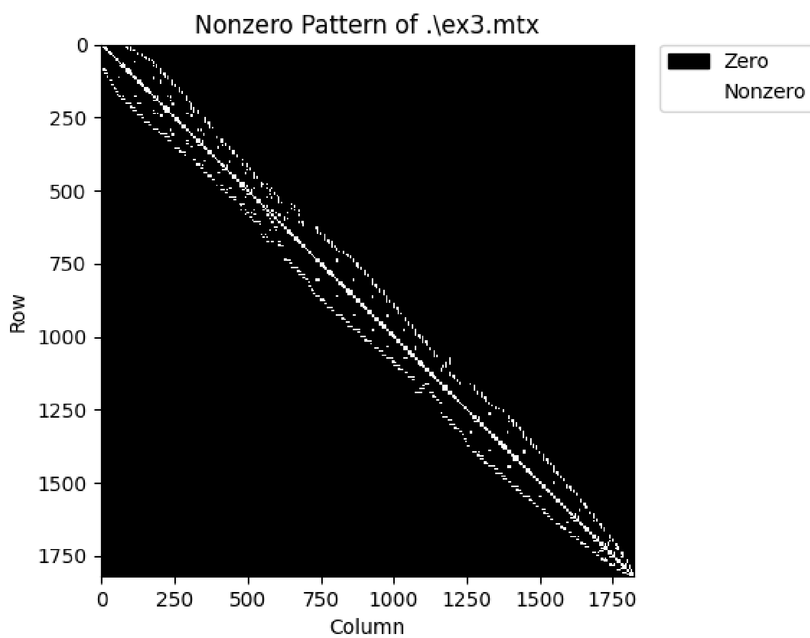
4 Matrix Visualization

These matrices were visualized using `scipy` and `matplotlib` in Python 3.12. I could not link the Python properly to the C program because WSL does not come installed with an X server, so any GUI program does not work properly in WSL. Further, matrices that were larger than `ex3.mtx` required exponentially more memory than I have, so I couldn't plot them.






Nonzero Pattern of .\LFAT5.mtx
Zero
Nonzero
Row
Column
0
2
4
6
8
10
12


Nonzero Pattern of .\ex3.mtx
Zero
Nonzero
Row
Column
0
250
500
750
1000
1250
1500
1750

Here is the code used to visualize these matrices:

```
import numpy as np
import matplotlib.pyplot as plt
import matplotlib.colors as colors
from matplotlib.patches import Patch
import scipy.io as sio
from scipy.sparse import tril, triu

# get file from command line
import sys
file = sys.argv[1]

# custom colormap
cmap = colors.ListedColormap(['black', 'white'])

# load the Matrix Market file
matrix = sio.mmread(file)

# only reflect if the original matrix is lower triangular (csfmatrix stores
                                        lower triangular for symmetric matrices
                                        )
if matrix.shape[0] == matrix.shape[1] and (triu(matrix, 1).nnz == 0):
    # reflect the matrix
    matrix = triu(matrix, 1) + tril(matrix, -1).T

# create a binary mask of the matrix
mask = matrix.astype(bool).toarray()

# display the mask using matplotlib
plt.imshow(mask, cmap=cmap, interpolation='nearest')
plt.title(f'Nonzero Pattern of {file}')
plt.xlabel('Column')
plt.ylabel('Row')
legend_elements = [Patch(facecolor='black', edgecolor='black', label='Zero'),
                     Patch(facecolor='white', edgecolor='white', label='Nonzero
                                                                      ')]
plt.legend(handles=legend_elements, bbox_to_anchor=(1.05, 1), loc='upper left',
                                         borderaxespad=0.)
plt.show()
```

5 Makefile

The following code is my Makefile:

```
CC=gcc
CFLAGS=-lm -Ofast
DFLAGS=-Wall -Wextra -W -g -O0 -lm


#FLAGSGaussSeidel= -DGAUSS_SEIDEL -DMAX_ITER=10000 -DTHRESHOLD=1e- #-DPRECONDITIONING
FLAGSJacobi= -DJACOBI -DMAX_ITER=10000 -DTHRESHOLD=1e-7 -DPRECONDITIONING
FLAGSPrint= -DPRINT=1 -DPYTHON

FLAGS= $(CFLAGS) $(FLAGSJacobi) # $(FLAGSPrint) #-DUSER_INPUT

SDIR=src
IDIR=include
ODIR=obj
BDIR=bin

_DEPS = functions.h
DEPS = $(patsubst %,$(IDIR)/%,$(_DEPS))

_OBJ = main.o functions.o
OBJ = $(patsubst %,$(ODIR)/%,$(_OBJ))

all: check_dir $(BDIR)/main

$(ODIR)/%.o: $(SDIR)/%.c $(DEPS)
$(CC) -c -o $@ $< -I$(IDIR) $(FLAGS)

$(BDIR)/main: $(OBJ)
$(CC) -o $@ $^ -I$(IDIR) $(FLAGS)


.PHONY: clean check_dir


clean:
```

```
rm -f $(ODIR)/*.o $(BDIR)/main
if [ -d "./r000hs/" ]; then rm -rf ./r000hs/; fi
if [ -f solution.txt ]; then rm solution.txt; fi
if [ -f smvp_output.txt ]; then rm smvp_output.txt; fi

check_dir:
if [ ! -d "./$(ODIR)/" ]; then mkdir $(ODIR); fi
if [ ! -d "./$(BDIR)/" ]; then mkdir $(BDIR); fi
```

This Makefile is similar to the one I had for assignment 2, but with a few differences:

- I have a set of flags for choosing between different methods. Originally, my code worked for the two methods, but I ran out of time when I forgot that the implementation of the Matrix Market file requires that lower triangular matrices be treated as symmetric matrices.
- the `check_dir` checks to see if the build and object directories exist before building the program. If they don't exist, they are automatically created.
- I used a bunch of flags to tell the compiler which sections of the code to compile (i.e. printing raw or pretty, printing or outputting to a file, defining values for max iterations, threshold, etc.)