MECHTRON 2MP3 Assignment 3

Developing a Biconjugate Gradient Stabilized Algorithm in C

Andrew De Rango, 400455362

1 Introduction

1.1 Assignment Overview

The goal of this assignment was to develop a program in C that is capable of solving first-order linear systems of equations. These systems are of the form Ax=b, where A is a known matrix, b is a known vector, and x is a vector whose elements must be solved for. The algorithm should be capable to solving very large and sparse matrices, up to millions of rows. In developing a solution, two parameters must be optimized/minimized: The runtime and the norm of the residual vector. The residual norm, or rather the magnitude of the residual vector, was the measure used to illustrate the accuracy of the solution given and can be defined by $||Ax - b||$.

It should be noted that very large matrices can impose significant memory issues when running the program. As the matrices being dealt with are sparse, it is better to store the matrices in CSR format, rather than storing every element of the 2D array in memory. This not only minimizes the memory allocated and reduces the incidence of segmentation fault, but it also improves runtime.

Furthermore, it would take a very long time to read the given matrices if each of their elements were given in a file. This would entail reading trillions of values, which is simply infeasible. To overcome this issue, files were provided and read from MTX format. For further information regarding MTX formatting, visit the Given Files folder of this repo.

1.2 Approach

The general process that the program developed for this assignment undergoes is described in this section.

First, it reads information from the MTX file and converts it directly to the CSR format. Then, it checks if the matrix is triangular. If the matrix is triangular, then it asks the user if they intended to solve the linear system where A is a simply the triangular matrix as provided, or if A was intended to be the corresponding symmetrical matrix. This occurs because a triangular matrix and its symmetrical counterpart, obtained by reflecting one triangular obliquely onto the other triangle, would have the same MTX file. In MTX format, there is no way to discern the

triangular matrix from the symmetrical. If the user decides to symmetrise the matrix, or reflect the triangle obliquely to the other triangle, then the program completes the rest of the operations using the symmetrical matrix.

Next, the program uses the `png.h` library to generate a PNG file that shows the sparsity of the matrix. This is an image that shows the sparsity pattern of the matrix, represented by black and white pixels showing where non-zero values exist in the matrix. White represents non-zero elements, while black pixels represent zeros. It creates a new directory if one has not already been created to add the PNG file to. The PNG file is always added to a folder in the directory that the user is currently in called *Sparsity Pattern Images*. The program notifies the user of this and specifies the file name and directory.

Following this, the program undergoes the iterative Biconjugate Gradient Stabilized (BiCGSTAB) algorithm and returns a solution vector x. It then computes and stores the residual and its norm. The program then executes the Conjugate Gradient algorithm and stores its residual norm. The program then compares the residual norms and takes the better (lower) one. The program calculates the program runtime, which accounts for both algorithms combined.

1.3 Biconjugate Gradient Stabilized Algorithm Overview

BiCGSTAB is a robust algorithm used to solve first-order linear systems of equations. It is an extension of the standard conjugate graduate method, in that it supports solutions for matrices that are both nonsymmetrical and indefinite. That is, the eigenvalues of the A matrix are not required to be positive in order to derive an accurate solution to the system. As an iterative method that uses the residual to define two new search directions (biconjugate gradient vectors), BiCGSTAB also has functionality that enforces stabilization which renders faster convergence.

2 Implementation in C

2.1 Structure Overview

The code can be broken down into four files:

- `Makefile`: Defines rules to be employed upon compilation.
- `functions.h`: Contains function prototypes that are formally defined in `functions.c`
- `main.c`: Contains the main function that calls functions from `functions.c`

- `functions.c`: Defines all functions other than `int main` such as `spvm_csr`, `bicgstab`, `conjugate_gradient`, and more.

2.2 Parameter Adjustment

There exists four arbitrary parameters within the program that may significantly change the resultant solution for x, and thus the residual vector and its norm. They are defined in the following lines:

```
bicgstab(csrMatrix, b, x, 1e-7, 10000);
conjugate_gradient(csrMatrix, b, x, 1e-7, 10000);
```

Both the BiCGSTAB and Conjugate Gradient algorithms take in the parameters `tolerance` and `max_iterations`. Above, they are defined as 0.0000001 and 10,000 respectively.

- `tolerance`: The iterations stop once the residual norm converges to below this specified tolerance. This should depend on the accuracy that the user is looking for in their specific circumstance.
- `max_iterations`: If the algorithms can't converge, then the iterations will stop after `max_iterations` iterations. This helps deal with cases in which BiCGSTAB will take a long time to converge to the true solution.

2.3 Makefile Summary

The Makefile supplied with the BiCGSTAB repository can be seen below:

```
CC = gcc
CFLAGS = -Wall -Wextra -g -lm -O2 $(shell pkg-config --cflags libpng)
LIBS = $(shell pkg-config --libs libpng)

all: bicgstab

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

clean:
```

```
rm -f bicgstab
rm -rf bicgstab.dSYM
```

This Makefile is a set of instructions instigated from the command line that aids in the compilation and linking processes of potentially multiple source code files. An example is shown in the code block above. Here are the roles of the individual components within the Makefile:

`CC = gcc`: `CC` sets the compiler that will be used to compile the program. In this case, we are using gcc.

`CFLAGS = -Wall -Wextra -g -lm -O2 $(shell pkg-config --cflags libpng)`: `CFLAGS` lists the flags that will be used by the compiler defined above. Each dash represents a precursor for another flag. `-Wall` enables the compiler to display warning messages upon compilation, such as declared but unused variables within the program. `-Wextra` provides more potential warnings. `-g` enables the compiler to provide debugging information, and `-lm` helps connect the `math.h` header used for other files. `-O2` enables possible optimizations within the program to further reduce the computation time. The elements within the brackets configure the `libpng` package, which is needed for the sparse patterns.

`LIBS=$(shell pkg-config --libs libpng)`: Necessary for ligpng, helps configure libpng via the linker flag.

`all: bicgstab`: Defines bicgstab as the default target when running `make` without any following arguments.

`bicgstab`: `main.c`, `functions.c`, `functions.h`: This is the rule for building the target `bicgstab`. The target is built depending on `main.c`, `functions.c`, and `functions.h`. This is the part of the Makefile responsible for creating just one object file despite multiple C files. The line below, `$(CC) $(CFLAGS) -o bicgstab main.c functions.c $(LIBS)`, is responsible for building the target if it needs to be rebuilt.

`clean`: This is an independent rule. It does not depend on any of the parameters defined above. If the user runs the command `make clean` in the command line, then the listed files will be removed from the system. In the Makefile shown above, the `bicgstab` and `bicgstab.dSYM`, which are created upon compilation, will be deleted if the user executes the command `make clean`.

2.4 VTune Analysis

The VTune and gcov portions of this assignment could not be completed because VTune and gcov are not available on Apple silicon processors.

2.5 CSR Formatting

The CSR format of LFAT5.mtx can be seen below:

```
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
CSR Data: 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
```

2.6 Dependencies

This program requires `png.h` in order to execute.

To install the dependency on Debian-based Linux, run the following:

`sudo apt-get update`

`sudo apt-get install libpng-dev`

On MacOS, use Homebrew:

`brew update`

`brew install libpng`

2.7 Running the Program

To run the program, the following commands must be run in the program's directory:

`make`

`./bicgstab <filename.mtx>`

For example,

`make`

`./bicgstab LFAT5.mtx`

3 Results

3.1 Performance

Table 1: BiCGSTAB Result for Different Matrices

Matrix	Dimensions	Non-Zeros	CPU Time (s)	Residual Norm
b1_ss.mtx	7x7	15	0.000010	0.000000
LFAT5.mtx	14x14	46	0.000038	0.000000
LF10.mtx	18x18	82	0.000141	0.000000
ex3.mtx	1821x1821	52685	1.011970	0.000112
jnlbrng1.mtx	40000x40000	199200	0.041605	0.000000
ACTIVSg70K.mtx	69999x69999	238627	14.121449	7595.457651
2cubes_sphere.mtx	101492x101492	1647264	66.212776	87.843514
tmt_sym.mtx	726713x726713	2903837	91.981226	0.182784
StocF-1465.mtx	1465137x1465137	11235263	84.903552	8179.010486

3.2 Sparse Pattern Visualization

As mentioned in Chapter 1, sparse pattern visualizations were implemented to determine the pattern of non-zero numbers in the input matrix. These can be seen below:

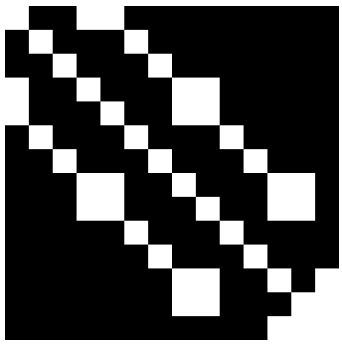

Figure 1: Sparse Pattern Image of `LFAT5.mtx`.

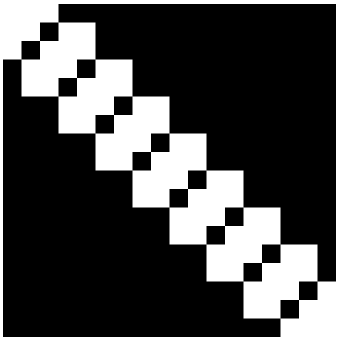

Figure 2: Sparse Pattern Image of `LF10.mtx`.

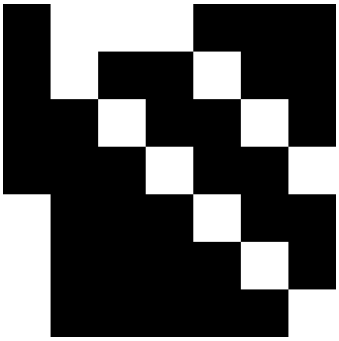

Figure 3: Sparse Pattern Image of `b1_ss.mtx`.

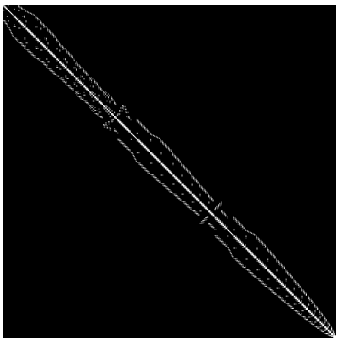

Figure 4: Sparse Pattern Image of `ex3.mtx`.