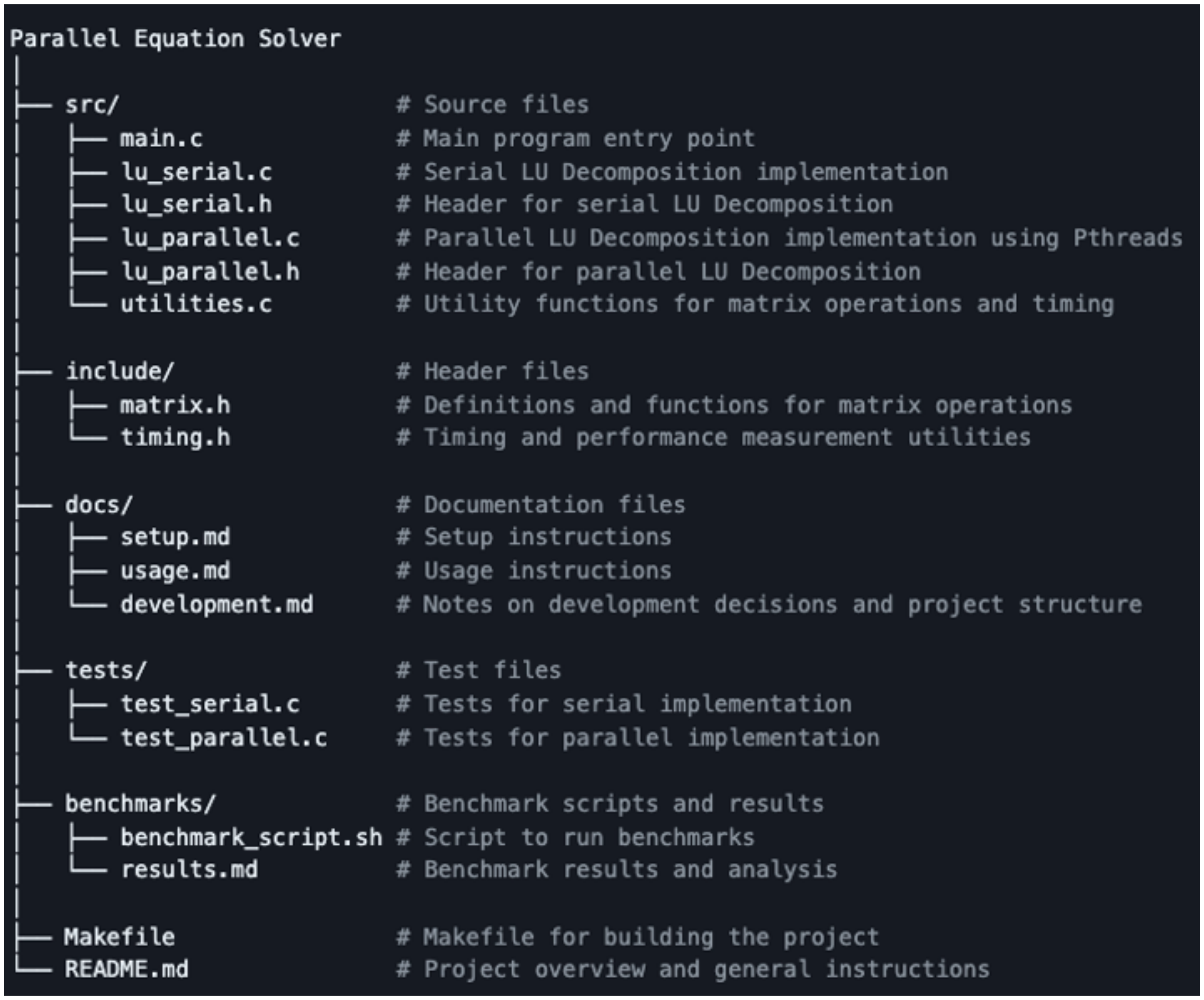
Advanced Parallel Equation [Matrix] Solver for Large-Scale Systems

Tasks completed so far:

1. **Git Repository and File Structure Setup**:

   First, we started a git repo and created directories with file structures that could be useful in our project. We may not end up using all those files, but this structure will be helpful in enforcing modularity into our program.

   Top view structure:



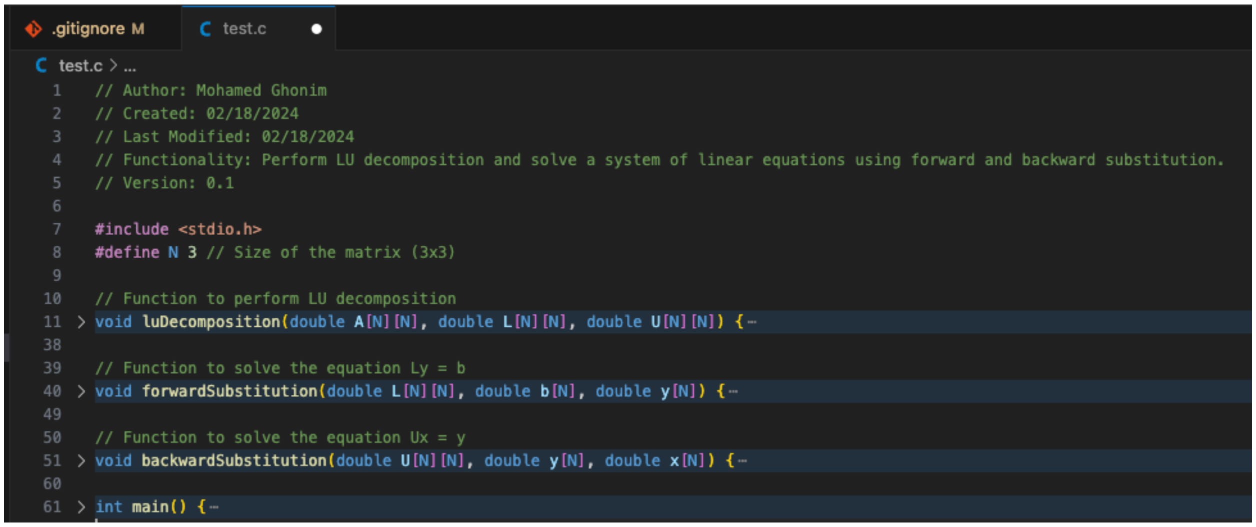
```
Parallel Equation Solver
|
├── src/                    # Source files
│   ├── main.c              # Main program entry point
│   ├── lu_serial.c         # Serial LU Decomposition implementation
│   ├── lu_serial.h         # Header for serial LU Decomposition
│   ├── lu_parallel.c       # Parallel LU Decomposition implementation using Pthreads
│   ├── lu_parallel.h       # Header for parallel LU Decomposition
│   └── utilities.c         # Utility functions for matrix operations and timing
│
├── include/                # Header files
│   ├── matrix.h            # Definitions and functions for matrix operations
│   └── timing.h            # Timing and performance measurement utilities
│
├── docs/                   # Documentation files
│   ├── setup.md            # Setup instructions
│   ├── usage.md            # Usage instructions
│   └── development.md      # Notes on development decisions and project structure
│
├── tests/                  # Test files
│   ├── test_serial.c       # Tests for serial implementation
│   └── test_parallel.c     # Tests for parallel implementation
│
├── benchmarks/             # Benchmark scripts and results
│   ├── benchmark_script.sh # Script to run benchmarks
│   └── results.md          # Benchmark results and analysis
│
├── Makefile                # Makefile for building the project
└── README.md               # Project overview and general instructions
```

2. **Algorithm Exploration and Initial Coding**:

We wanted to make sense of the LU decomposition algorithm ourselves, so we solved a few simple 3x3 and 4x4 matrix examples representing systems of linear equations. This

helped us understand how to implement the program, and how to make it computationally efficient.
The answers obtained were used to validate our initial version of the program.

3. **Testing and Versioning**:
To get started, we will code a simple C code implementing this algorithm, then we'll use those same examples above to verify the correctness of that algorithm. Next, we'll improve the algorithm to work on bigger matrices, and will use pthreads.



```c
// Author: Mohamed Ghonim
// Created: 02/18/2024
// Last Modified: 02/18/2024
// Functionality: Perform LU decomposition and solve a system of linear equations using forward and backward substitution.
// Version: 0.1

#include <stdio.h>
#define N 3 // Size of the matrix (3x3)

// Function to perform LU decomposition
void luDecomposition(double A[N][N], double L[N][N], double U[N][N]) {…

// Function to solve the equation Ly = b
void forwardSubstitution(double L[N][N], double b[N], double y[N]) {…

// Function to solve the equation Ux = y
void backwardSubstitution(double U[N][N], double y[N], double x[N]) {…

int main() {…
```

Version: 0.1: initial version of the program all implemented in main.c and operated using the command line.

Version 0.2:
We then made the program more structured but adding the matrices in the /matrices directory as text files passed to the program using the command-line terminal as the first argument.

Version: 0.3 : read in the matrix from a file passed as an argument to the program
: added a function to free the allocated memory
: the matrix size is parameterized, and passed as the first line in the matrix file

Version: 0.4 : allow writing the solution to a file passed as an argument to the program
: if no file is passed, the solution will be printed in the terminal
: example usage: matrices/py_generated/5000x5000.txt matrices_solution/5000x5000.txt

4. **Progress on Serial Program Development**:
we parameterized the size of the matrix, such that it's passed to the program from the first line in the matrix txt file.
We tested the program on multiple 3x3 and 4x4 matrices which we know the answers to previously as a way to debug the code. The program solved all the matrices correctly.

To get and test much bigger matrices, we wrote a python script <utilities/matrix_generator.py> which takes in an argument of the dimension of nodes, and generates a matrix which gates saved in the <matrices/py_generated> directory with $matrix_dimension.txt tile.
Currently, we have a functional serial program that takes in a matrix of dimensions N and solves it using LU decomposition and saves the solution to a text file or prints it on the screen.

Next steps:

At this point, we need to focus on parallelizing the program, so that we can use pthreads to run it on multiple cores.
We also need to implement a way to measure the time it takes to find a solution, so that we can use that to collect benchmarks and compare the serial performance to the parallel performance.
We will then use perl or python to automate running the program once serially, then in parallel on multiple processors from 1 to 16, and produce an output file in the benchmark directory with the time it takes to find the solutions, as well as the speedup achieved.

References:

[1] K. Hartnett and substantive Quanta Magazine moderates comments to facilitate an informed, “New algorithm breaks speed limit for solving linear equations,” Quanta Magazine, https://www.quantamagazine.org/new-algorithm-breaks-speed-limit-for-solving-linear-equations-20210308/ (accessed Jan. 28, 2024).

[2] T. J. Dekker, W. Hoffmann, and K. Potma, “Parallel algorithms for solving large linear systems,” *Journal of Computational and Applied Mathematics*, vol. 50, no. 1–3, pp. 221–232, 1994. doi:10.1016/0377-0427(94)90302-6

[3] W. by: Q. Chunawala, “Fast algorithms for solving a system of linear equations,” Baeldung on Computer Science, https://www.baeldung.com/cs/solving-system-linear-equations (accessed Jan. 28, 2024).

[4] D. Kaya and K. Wright, “Parallel algorithms for LU decomposition on a shared memory multiprocessor,” *Applied Mathematics and Computation*, vol. 163, no. 1, pp. 179–191, Apr. 2005. doi:10.1016/j.amc.2004.01.027

[5] E. E. Santos and M. Muralcetharan, “Analysis and Implementation of Parallel LU-Decomposition with Different Data layouts,” *University of California, Riverside*, Jun. 2000.