Technische Universität München

Parallel Programming (TUM) SS2021 Final Project

Examiner: Prof. Martin Schulz
Project Authors: Shubham Khatri, Maximilian Stadler
Teaching Assistants: Bengisu Elis, Vincent Bode

Project 02 – GE

General Guidelines and Rules

- Your work needs to be submitted through the LRZ GitLab. You will be granted access to your repositories at the kickoff.
- You will submit your solutions in three steps with three different due dates. The individual tasks and their due dates may be found below.
- There are separate repositories under your GitLab group for each task in this document, namely OMP, MPI, Hybrid, Bonus and Documentation. Please make sure to submit your solutions to the corresponding repositories.
- Keep in mind that only the master branches of each repository will be considered as a valid submission and graded. You are expected to submit a single solution file in the repositories that contains your solution code with the best speed-up you have achieved until the deadline. Make sure this single file on the master branch contains the solution that you want to have graded.
- In case you use or adapt code from a work other than yours (internet tutorials, public repositories etc.) cite the source in a code comment. Doing otherwise will be considered as plagiarism and will be taken into account when grading your code.
- For your project presentation, further instructions are provided in the presentation template.
- **Note:** You must not not change the algorithm itself but perform the optimization and parallelization on the algorithm described above.
- To be considered a valid submission, your code should compile on the submission server. Therefore, be careful with the libraries used in your code. In case of unexpected compilation errors, contact your responsible tutor.
- You will be assigned a responsible tutor. Your tutor is available for answering questions and to help resolve any issues you might encounter.
- Write a brief README for each repository which explains the changes or additions in your code other than performance improvements. If you implemented new functions and data structures in your solution, explain these in your README.

Algorithm : Gaussian Elimination

In linear algebra one fundamental task is to solve systems of linear equations. Any system of linear equations with multiple variables can be represented in matrix vector form as follows:

$$7a + 12b + 6c = 2 \tag{1}$$
$$4a + 21b + c = 7 \tag{2}$$
$$3a + 5b + 8c = 9 \tag{3}$$
$$\Rightarrow \begin{bmatrix} 7 & 12 & 6 \\ 4 & 21 & 1 \\ 3 & 5 & 8 \end{bmatrix} \begin{bmatrix} a \\ b \\ c \end{bmatrix} = \begin{bmatrix} 2 \\ 7 \\ 9 \end{bmatrix} \tag{4}$$

Any such equation can be solved using Gaussian Elimination (GE) method. The GE method can be broken down into two steps, the first being the Forward Elimination (FE) and the latter being Backward Substitution (BS). For the sake of ease we are going to refer to the diagonal elements in the matrix as pivots[1].

1 Forward Elimination

The Forward Elimination step involves performing elementary row operations on the matrix such that it can be reduced into *row echelon form*[2] (described below). The row echelon form help us understand if there is a unique solution, an infinite number of solutions or no solution to the system of equation.

Conversion of a system to row echelon form simply means converting the system matrix in an upper triangular form[3]. In order to better describe the conversion of a matrix to row echelon form we take Equation 4 and convert it into upper triangular matrix.

Step 1:

$$\begin{bmatrix} 7 & 12 & 6 \\ 4 & 21 & 1 \\ 3 & 5 & 8 \end{bmatrix} \xrightarrow[R2 \to R2 - \frac{4}{7}R1]{R3 \to R3 - \frac{3}{7}R1} \begin{bmatrix} 7 & 12 & 6 \\ 0 & \frac{99}{7} & -\frac{17}{7} \\ 0 & -\frac{1}{7} & \frac{38}{7} \end{bmatrix} = \begin{bmatrix} 7 & 12 & 6 \\ 0 & 14.142 & -2.428 \\ 0 & -0.143 & 5.428 \end{bmatrix}$$

Step 2:

$$\begin{bmatrix} 7 & 12 & 6 \\ 0 & \frac{99}{7} & -\frac{17}{7} \\ 0 & -\frac{1}{7} & \frac{38}{7} \end{bmatrix} \xrightarrow{R3 \to R3 + \frac{1}{99}R2} \begin{bmatrix} 7 & 12 & 6 \\ 0 & \frac{99}{7} & -\frac{17}{7} \\ 0 & 0 & \frac{535}{99} \end{bmatrix} = \begin{bmatrix} 7 & 12 & 6 \\ 0 & 14.142 & -2.428 \\ 0 & 0 & 5.404 \end{bmatrix}$$

It should be noted that these row operations must also be performed simultaneously on the right hand side of the equation to preserve equality.

Now we can formally define the forward elimination algorithm as follows:

Algorithm 1: Forward Elimination

Data: System Matrix A and Right hand side vector b
Result: Row Echelon Form Matrix
for $k = 1$ *to* $n - 1$ **do**
  for $i = k + 1$ *to* n **do**
    $a_{ik} = \frac{a_{ik}}{a_{kk}}$;
    for $j = k + 1$ *to* n **do**
      $a_{ij} = a_{ij} - a_{ik} * a_{kj}$;
    end
    $b_i = b_i - a_{ik} * b_k$;
  end
end

[1] https://en.wikipedia.org/wiki/Pivot_element
[2] https://en.wikipedia.org/wiki/Row_echelon_form
[3] https://en.wikipedia.org/wiki/Triangular_matrix

2 Backward Substitution

After FE we obtain a upper triangular form for the matrix and to solve the system of equation we can simply start plugging the value from the last rows. This result in the final values of c, b, a, in that order. For better understanding we illustrate this step for Equation 4:

Step 1:

$$\begin{bmatrix} 7 & 12 & 6 \\ 0 & \frac{99}{7} & -\frac{17}{7} \\ 0 & 0 & \frac{535}{99} \end{bmatrix} \begin{bmatrix} a \\ b \\ c \end{bmatrix} = \begin{bmatrix} 2 \\ \frac{41}{7} \\ \frac{812}{99} \end{bmatrix} \Rightarrow c = \frac{812}{535}$$

Step 2:

$$\frac{99}{7}b - \frac{17}{7}\frac{812}{535} = \frac{41}{7} \Rightarrow b = \frac{35739}{3745}$$

Step 3:

$$7a + 12\frac{35739}{3745} + 6\frac{812}{535} = 2 \Rightarrow a = -\frac{455482}{26215}$$

Now we can define the backward substitution algorithm as follows:

Algorithm 2: Backward Substitution

Data: Row Echelon form of A and reduced right hand side vector b
Result: solution vector x
for $i = n$ *to* 1 **do**
  for $j = i + 1$ *to* n **do**
    $x_i = x_i - a_{ij} * x_j$;
  end
  $x_i = \frac{x_i}{a_{ii}}$
end

Note: The algorithm described above is Gaussian Elimination without pivoting and therefore it is necessary that the diagonal elements are largest in the column.

For more details on algorithm itself and other variants of it you can read up on Wikipedia[4].

[4] https://en.wikipedia.org/wiki/Gaussian_elimination

Implementation : Short Documentation

For this exercise we use an interface pattern, where the main is oblivious of the type of parallelization being used. Following is the structure in which the serial code is organized.

- **main.cpp**:
  - The main calls a Compute kernel and outputs the performance metric (speedup).
  - You don't need to do anything here
- **Interface.cpp**
  - This method adds a level of abstraction between the compute kernel and main.
  - Interface performs a conditional compilation and chooses between different kind of parallelization method that are implemented. For example when OMP compute kernel is implemented we can compile the code with *BUILDOMP* flag and the interface would choose the OMP kernel for parallel implementation.
  - You don't need to do anything here
- ***_interface.cpp**:
  - We have 3 interfaces corresponding to the 3 cases: omp_interface.cpp, mpi_interface.cpp and hybrid_interface.cpp
  - This interface performs data initialization for parallel case as well as set up the data structure necessary for each kind of parallelization strategy.
  - *This interface calls the actual kernel that has to be implemented by you.*
  - You don't need to do anything here in case of OMP and MPI implementation, but you should implement the interface for Hybrid case
- ***ge.cpp**:
  - We have 3 kernels corresponding to the 3 cases: ompge.cpp, mpige.cpp and hybridge.cpp
  - These must be implemented by you following the algorithm and data structure pattern provided in the serial part. You are allowed to perform necessary and justifiable changes in data structure (This should be well reasoned and justified in presentation and code documentation).
  - You should implement this.
- **serial.cpp**:
  - This provides a serial implementation of the Gaussian Elimination Algorithm.
  - The *ForwardElimination* method implements the algorithm 1.
  - The *BackwardSubstitution* method implements the algorithm 2.
  - You can perform optimization in this.
- **utility.cpp**:
  - This file contains the methods that are used for setting up the problem such as initialization and command line parsing.
  - *ParseFilesNames* implements command line parsing and finds the file location for the given arguments.
  - *InitializeArray* initializes the memory buffer with the components of the given file.
  - You don't need to do anything here. Any optimization here will not be taken into account on the submission server but you are welcome to be creative and report your findings in the final presentation

- **Makefile**:
  - Implements a recipe (compilation commands) for different kind of parallel implementation.
  - You can use commands such as *make ompge*, *make mpige* and *make hybridge* to compile the source code and build the executable for the three case respectively.
- **Execution instructions**: The code reads the data from the provided matrix, vector and solution files and therefore the path to these files must be provided. For an executable named *ge* and data located in directory *data* we can execute the program using *./ge <path to data directory>/size8x8*. Here the program reads the *size8x8.mat*, *size8x8.vec* and *size8x8.sol* to solve a 8×8 matrix vector equation.

Tasks

1. **General:**

   This task includes preliminary optimisations on the sequential code without parallelising. You are not expected to submit a separate file for this task but by optimising the sequential version provided, this task aims to help you achieve better speed up for following parallelisation tasks .

   Go through the above algorithm and analyze the given sequential code. Where do you see possible improvements through parallelization? Attempt to optimize the sequential part of the code before jumping to parallelization part. Explain the logic for each optimization step.

   Hint: A proper optimization of sequential code should render parallelization useless for smaller matrix size. Show why this happens in your presentation

   Note: It is mandatory that you do not change the algorithm itself but perform the optimization and parallelization on the algorithm described above. Changing the algorithm would result in invalid submission.

2. **OMP:** *Due 15.06.2021 23:59, Submission file "ompge.cpp"*

   Parallelize the computations using OMP. Parallelize as much as possible as long as it does not harm performance. You do not have to parallelize the IO. Explain the parallelization strategies used and the logic behind them. It is recommended that you use a diagram to show parallelization strategy and the data required at each step of parallelized algorithm.

   Remark: With an improved sequential code and 4 OMP threads, you should expect a speedup of at least 1.2 for larger matrices.

3. **MPI:** *Due ~~22.06.2021 23:59~~ 24.06.2021 23:59, Submission file "mpige.cpp*

   Assume (hypothetically) that you have to run your code on a cluster without shared memory access and only one core per compute node. Parallelize the computations using MPI. Parallelize both the Forward Elimination (FE) and Backward Substitution (BS) algorithms. Again there is no need to parallelize the IO. Explain the parallelization strategy, type of communication and the logic behind it. It is recommended to use a diagram to show parallelization strategy and the data required in each step of parallelized algorithm.

   Remark: Assume that the buffer read from the file system is only available on RANK 0. Remark: Your MPI solution should be agnostic to the number of MPI processes used when executing the code. In other words the solution should be executable by any number of MPI processes.

4. **Hybrid (MPI + OMP):** *Due 29.06.2021 23:59, Submission file "hybridge.cpp*

   Assume (hypothetically) that you have to run your code on a cluster with several distributed nodes each having multiple cores with shared memory. How can you leverage this architecture in a hybrid approach using MPI and OMP? Parallelize both the Forward Elimination (FE) and Backward Substitution (BS) algorithms. Again there is no need to parallelize the IO. Explain the parallelization strategy, type of communication and the logic behind it. It is recommended to use a diagram to show parallelization

strategy and the data required in each step of parallelized algorithm. Also justify when and if it makes sense to use hybrid parallelism for this algorithm.

For this task, you are also required to implement the hybrid interface that will connect your implementation of the solver to the main of package. You should use the files *mpi_interface.cpp* and *mpi_interface.h* as reference and perform a similar implementation in *hybrid_interface.cpp* and *hybrid_interface.h*. You are not required to submit these files but they will be necessary for you to test your implementation.

Note: Please stick to the IO pattern described in the *mpi_interface.cpp* file, using a different pattern could cause your submission to fail or could increase the runtime.

5. **Bonus (Optional):** *Due 29.06.2021 23:59, Submission file "bonusge.cpp*

   Given your hybrid implementation (MPI + OMP), can you observe any speedup using one of the following of your choosing: SIMD intrinsics, OMP GPU offloading , CUDA or HIP ? Explain how and provide necessary details.

   Note: Implement this using intrinsic operations and not OMP SIMD.

6. **Presentation Slides:** *Due 06.07.2021 23:59, Submission file "PPSS21_final_project.pdf or .ptx*

   Use the template provided in the Documentation repository for your presentation slides. Submit your presentation into the Documentation repository.