Grundlagen des wissenschaftlichen Rechnens (GwR)

Assignment 1 – Report

1) Introduction

This study endeavors to implement a parallel Gauss-Seidel (GS) solver for addressing elliptic partial differential equations (PDEs) on a two-dimensional domain. The discretization of the PDE utilizes finite difference techniques, yielding a linear system of equations. The GS solver iteratively approximates solutions to this system, with parallelization facilitated through OpenMP for shared memory systems to enhance computational efficiency.

The executable program, denoted as "gssolve," is designed to receive command-line arguments specifying the number of grid points in each dimension (nx, ny) and the desired number of iterations. Subsequently, it executes the GS solver for the specified iterations while measuring the overall wall clock time for computation. Moreover, the computed solution is stored in a file named "solution.txt," formatted suitably for visualization using gnuplot. Upon completion of the final iteration, the discrete L2-norm of the residual is determined and reported.

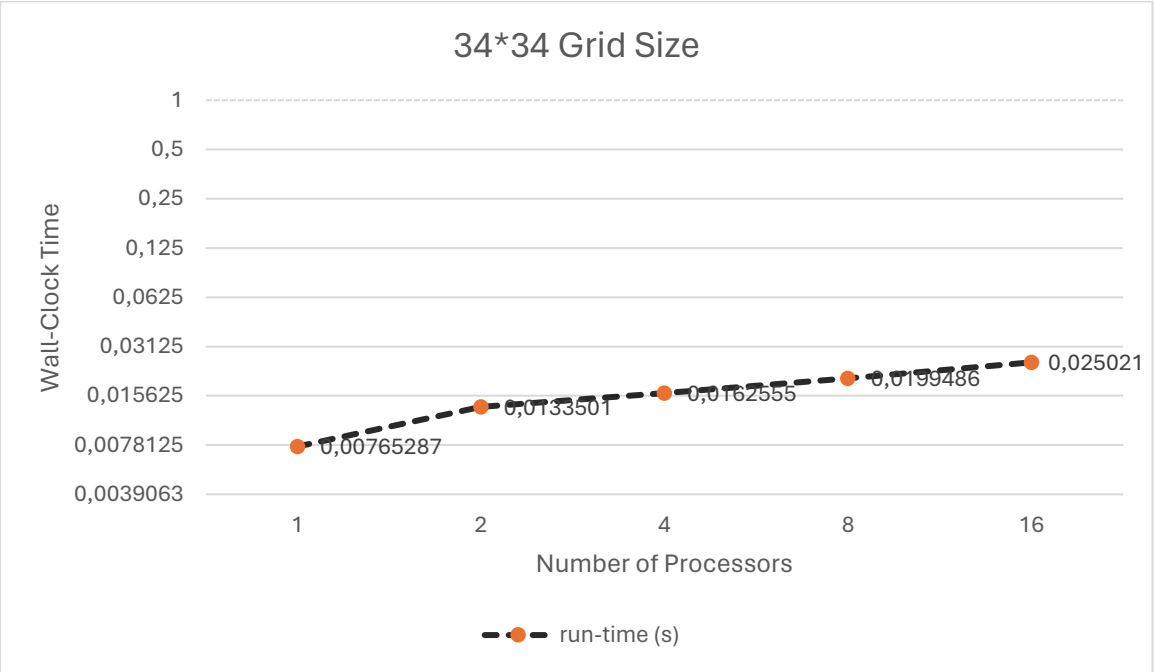
The parallel implementation is subjected to rigorous testing on RRZE's Meggie cluster, encompassing varying thread counts to assess scalability. A comprehensive evaluation entails a strong scaling experiment, wherein the number of processes (np) is systematically varied from 1 to 16, while maintaining fixed grid sizes (nx = ny) at two distinct values, 34 and 1026, and the number of iterations at 1000. Wall-clock time, speed-up, and parallel efficiency metrics are scrutinized across differing process counts, with graphical representations elucidating the findings.

In essence, this research aims to showcase an adept parallelization of the Gauss-Seidel solver leveraging OpenMP and to scrutinize its scalability within a distributed computing environment.

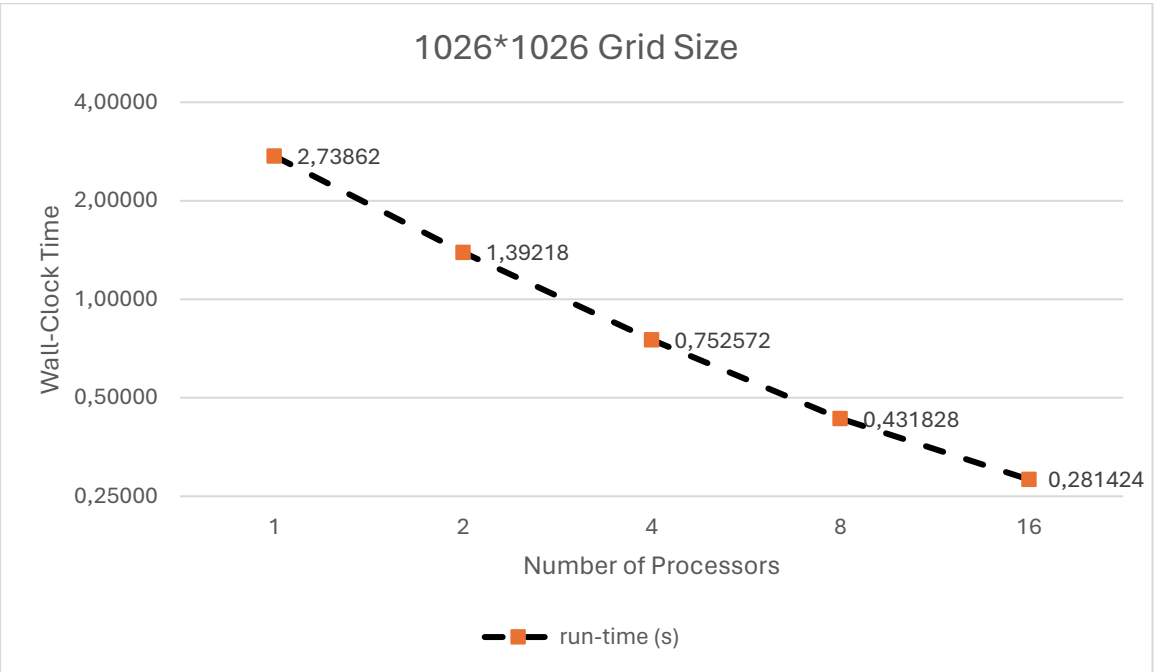
2) Wall-Clock Time

Wall time is the amount of time that a program or process takes to run from start to finish. In our case, wall clock time is the duration of the calculation which is done by Gauss-Seidel and residual norm functions.

Graph 1.0 and Graph 1.1 show the total run-time of the programs for 34*34 and 1026*1026 grid size in seconds.



Graph 2.0
Wall-Clock Time for 34*34 Grid Size



Graph 2.1
Wall-Clock Time for 1026*1026 Grid Size

3) Speed-Up

Speed-Up is a metric which express the performance improvments as parallel calculation increases. This metric is calculated as follows.

$$S(p) = T(1)/T(p) \leq T(1) / (T(1)*(f+(1-f)/p)$$

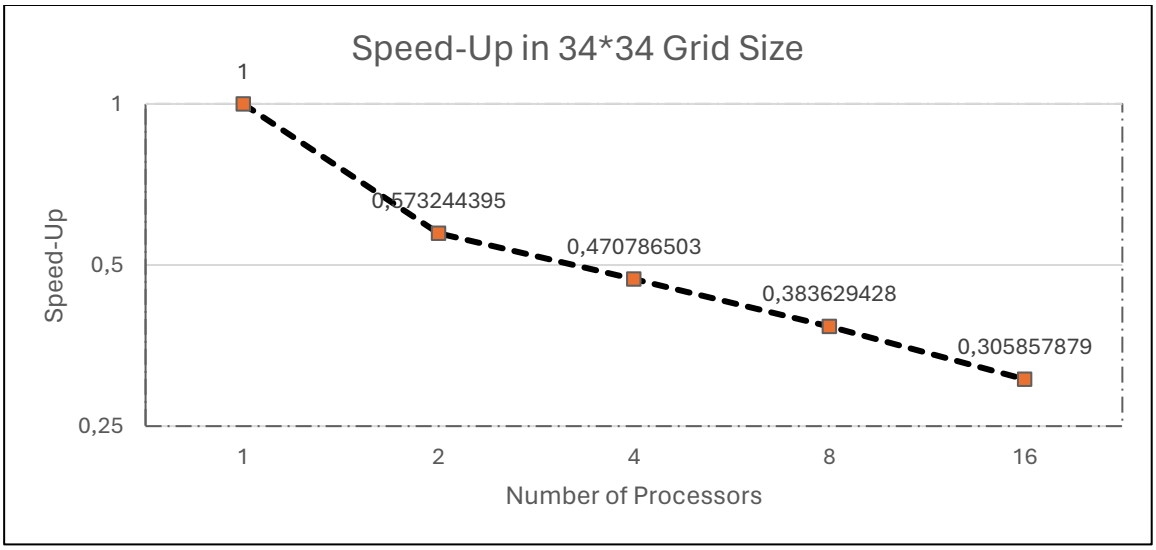
where

S(p) : Speed-Up value

$T(p)$: runtime of the code fragment using p threads

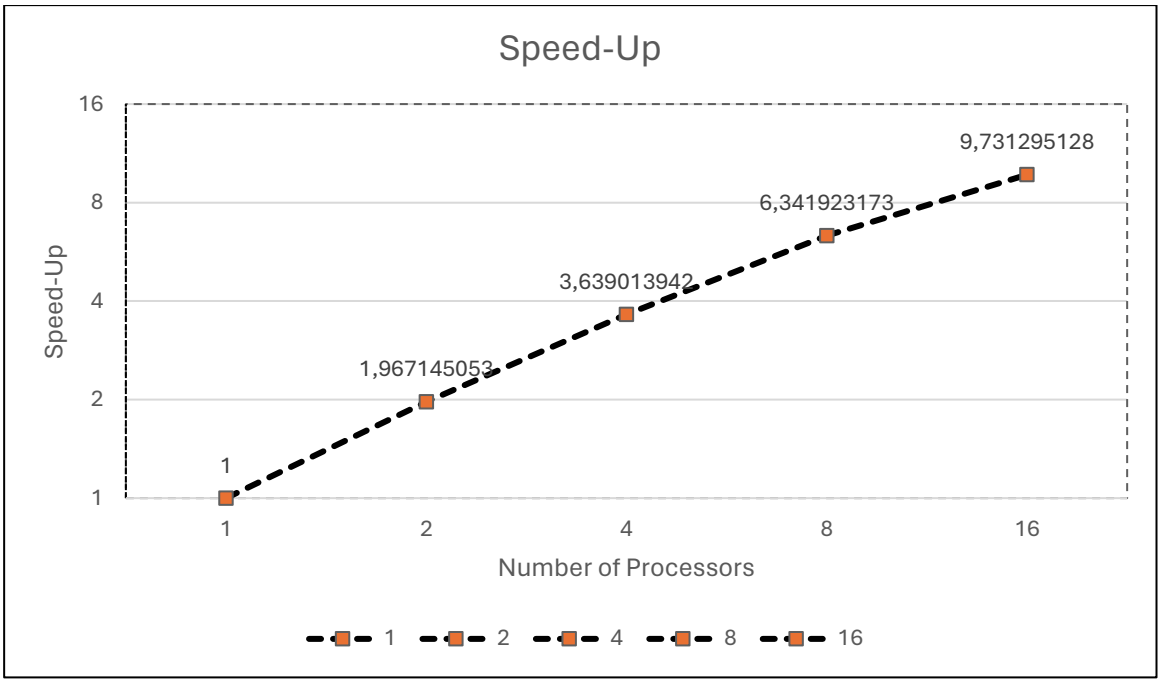
f : the fraction of $T(1)$ that can not be parallelized

In this assignment, Speed-Up value is calculated for both 34*34 grid size and 1026*1026 grid size using 1000 iterations.



Graph 3.0
Speed-Up for 34*34 Grid Size

The Graph 3.0 shows the result of Speed-Up value of 34*34 grid size with 1000 iterations. As indicated in the chart, it is possible to see that total speed-up decreases as the number of processors increase. A compelling reason behind this result is, computer expends more time for communication between processors and as a result, results with values lower than 1 are obtained. This is also known as *elbow effect*.



Graph 3.1
Speed-Up for 1026*1026 Grid Size

The Graph 3.1 shows the result of Speed-Up value of 1026*1026 grid size with 1000 iterations. As indicated in the chart, it is possible to see that total speed-up increases the number of processors increase. This is an expected situation because as the grid size increased, the parallel code worked more effectively and the time spent on communication remained at a negligible level compared to this efficiency.

4) Parallel Efficiency

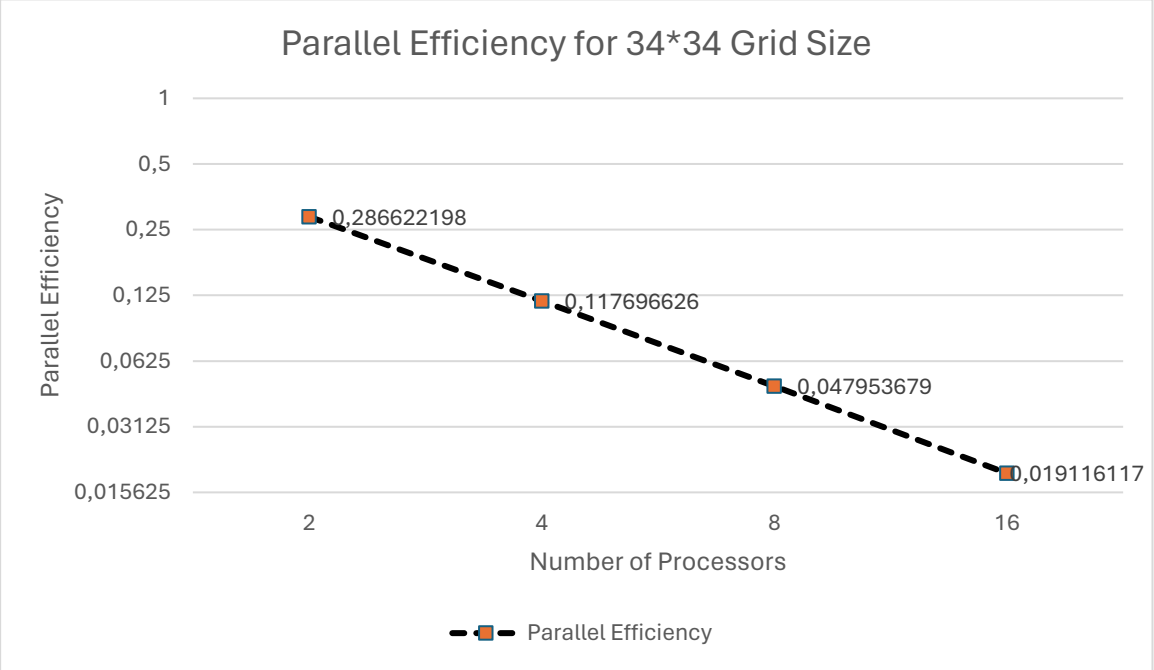
The goal of parallel computing is to reduce the time-to-solution of a problem by running it on multiple cores. In order to understand the effectivness of the solution, it is possible to calculate the parallel efficiency value -$E(p)$- as follows.

$$\boldsymbol{E(p) = S(p) / p}$$

Where $S(p)$ is the speed-up that using p processors.

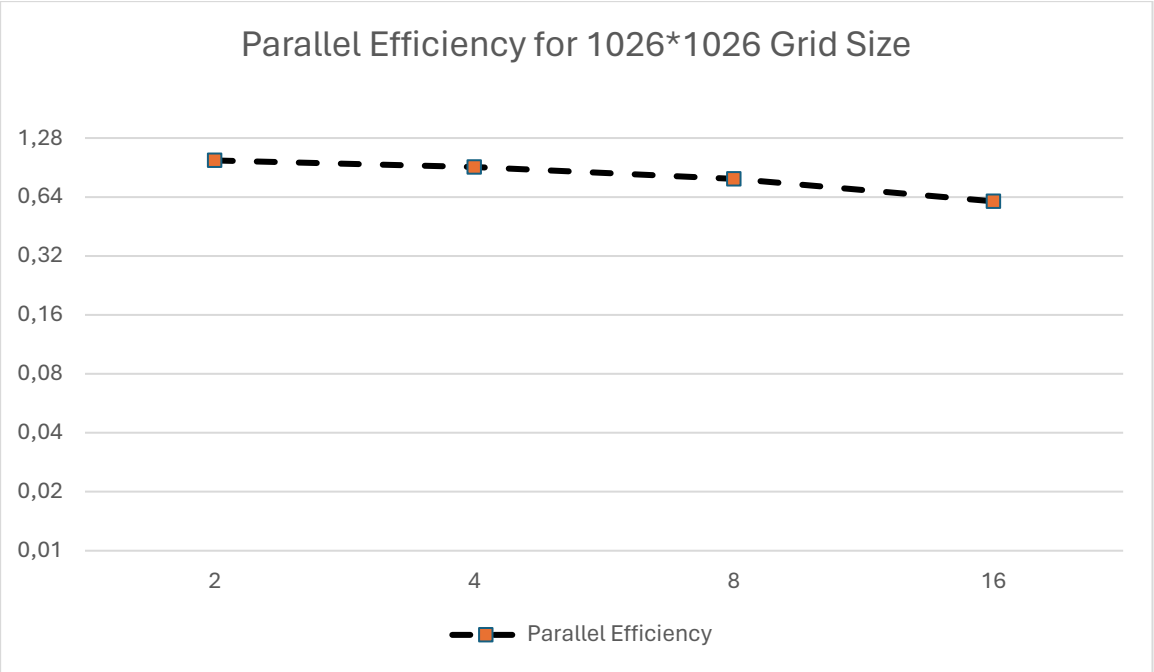
Ideally $\boldsymbol{S(p)}$, in other words, $\boldsymbol{E(p) * p}$, should be equal to 1. This situation required that everything can be parallelized (f=0).

In this assignment, Parallel Efficency value is calculated for both 34*34 grid size and 1026*1026 grid size using 1000 iterations.



Graph 4.0
Parallel Efficiency for 34*34 Grid Size

The Graph 4.0 shows the result of parallel efficiency of 34*34 grid size with 1000 iterations. As is seen clearly, parallelizing the solution to the grid problem, that is, increasing the number of processors, prolongs the solution time of the problem. In other words, parallelizing the problem is not a more effective solution.



Graph 4.1
Parallel Efficiency for 1026*1026 Grid Size

The Graph 4.1 shows the result of parallel efficiency of 1026*1026 grid size with 1000 iterations. As is seen clearly, parallel efficenct is close to 1 compared to the previous small grid size calculations. This means, parallelizing code is more purposeful for bigger problems. Nontheless, it is the most efficent way to use 2 processors to achive the biggest parallel efficency value for this grid size.