Parallelization of the Multigrid method of solving linear equations

Raul Om Deepak
IIT Madras

Rachit Kumar
IIT Madras

Rudra Panch
IIT Madras

May 12, 2024

1 Abstract

The multi-grid method is a robust iterative approach utilized for efficiently solving large sparse systems of linear equations, particularly in the context of solving partial differential equations. By employing a hierarchy of grids, it accelerates convergence by integrating coarse and fine grids and exploiting the varying length scales inherent in the problem. The method's fundamental strategy involves approximating the solution on a coarse grid, then refining it on finer grids, and further interpolating back to coarser grids to enhance the solution. This iterative process comprises several key steps, starting with an initial guess and proceeding through pre-smoothing to reduce high-frequency error components, restricting the residual to a coarser grid, solving the coarse grid problem, interpolating corrections to finer grids, updating the solution with interpolated corrections, and post-smoothing to further refine error components.

This paper explores parallelisation of the Multi-grid Method using OpenMP , OpenMPI and Open ACC methods to meet the demand for faster computations in scientific and engineering domains. Leveraging shared-memory parallelism with OpenMP ,distributed-memory parallelism with OpenMP and GPU-CPU based distribution with OpenACC, we achieve notable speedup and scalability improvements. We evaluate the efficiency of our implementations, demonstrating their effectiveness in addressing large-scale problems. Our findings provide valuable insights for optimizing complex numerical algorithms on modern parallel architectures.

2 Introduction

The multi-grid method is a powerful numerical technique used to solve large-scale systems of linear equations. It is particularly effective for problems with fine-scale features or high-frequency components. Instead of directly solving the linear system on a single grid, the multi-grid method operates on a hierarchy of grids with varying levels of resolution. By exploiting information at multiple scales, it efficiently corrects errors, leading to rapid convergence.

This paper works on two systems of linear equations, Pade's scheme of formulating the differential of a curve , and a general system of linear equations. We would be dealing with system of linear equations of the form Ax=b.

3 The V scheme : Multi-grid

As the name suggests , the V scheme is a multi-grid algorithm which involves iterating on the present matrix in a V pattern in a up down format. It operates on a hierarchy of grids, moving between coarser and finer levels to efficiently correct errors.

The entire notion of the Multi-grid notion is to achieve a faster convergence.Methods like Jacobi iterative method, or Gauss-Seidel method tend to slow down as the frequency of error decreases. Through interpolating finer matrices into coarser ones , we increase the frequency of errors and hence speedup the convergence

The V-scheme would be involving the following steps:

- Firstly a pre-smoothing would be performed using gauss-seidel method.The main aim of the pre-smoothing operation in the V-cycle of the multi-grid method is to reduce high-frequency errors in the solution on the finest grid level before proceeding to coarser levels for further correction. Pre-smoothing prepares the solution for the multi-grid correction process by damping out oscillations and rapidly reducing local errors.
- Then the residual matrix and the correction matrix would be calculated. The residual matrix (r) is nothing but the difference between the actual 'b' matrix and the 'b' matrix we get by using the current 'x'. It is basically the error in 'b' matrix.
- On the other hand the correction matrix (e) is the error in the 'x' matrix. This correction matrix is what we subject to the restriction process.
- Further the matrices A,e and r would be restricted into coarser matrices using the Restriction matrix (R).
- iteration of gauss-seidel would be performed on these coarser matrices.
- further the matrices would be interpolated back into finer matrices using the Prolongation matrix (I).
- This loop continues until the errors fall under a certain level of tolerance.

4 Formulation of the V scheme: Nomenclature

Let:

- A be the coefficient matrix of the linear system,
- u be the solution vector,
- f be the right-hand side vector,
- r be the residual vector,
- ν be the number of Pre- and post-smoothing iterations.

Residual Calculation:

$$r = f - Au$$

Restriction Operator (R):

Restricts the residual to the coarser grid:

$$r^h = R(r)$$

Prolongation Operator (I):

Interpolates the correction from the coarser grid to the finer grid:

$$e^H = I(e)$$

Smoothing:

Apply relaxation methods (e.g., Jacobi, Gauss-Seidel) to approximate the solution:

$$u^{(i+1)} = S(u^{(i)}, f)$$

V-cycle:

$$u = V(u, f)$$

Where:

- u is the current approximation,
- f is the right-hand side vector,
- V is the V-cycle function,
- The V-cycle function is defined as follows:
  1. Pre-smoothing: Apply ν pre-smoothing iterations.
  2. Restriction: Restrict the residual to a coarser grid: $r^h = R(r)$.
  3. Coarse Grid Solve: Solve the system directly or recursively on the coarser grid: $A^H u^H = r^H$.
  4. Interpolation: Interpolate the correction to the finer grid: $e^h = I(e^H)$.
  5. Correction: Update the solution: $u = u + e^h$.
  6. Post-smoothing: Apply ν post-smoothing iterations.

Initial Guess:

The initial guess $u^{(0)}$ can be set to zero or any other reasonable initial guess.

Stopping Criterion:

The V-cycle iteration can be stopped when a predefined convergence criterion is met, such as reaching a certain residual tolerance or a maximum number of iterations.

Coarse Matrix (A^H) Formulation:

The coarse matrix A^H can be formulated using prolongation and restriction operators:

$$A^H = RAI$$

4.1 Multi-grid method on the Pade's scheme

$$f_0^{'} + 2f_1^{'} = \frac{1}{h}\left(-\frac{5}{2}f_0 + 2f_1 + \frac{1}{2}f_2\right) \qquad \text{Boundary Condition x=0}$$

$$f_n^{'} + 2f_{n-1}^{'} = \frac{1}{h}\left(\frac{5}{2}f_n - 2f_{n-1} - \frac{1}{2}f_{n-2}\right) \qquad \text{Boundary Condition x=n}$$

$$f_{j+1}^{'} + 4f_j^{'} + f_{j-1}^{'} = \frac{3}{h}\left(f_{j+1} - f_{j-1}\right) \qquad \text{Interior points}$$

Treating the above relations as a set of linear equations we can formulate the solution for the pade's scheme using the multi-grid method.The corresponding matrices would look as follows:

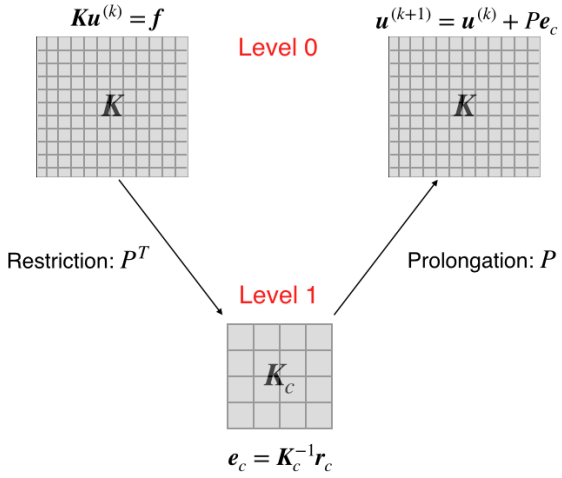

Figure 1: Example Image

$$\begin{pmatrix} 1 & 2 & 0 & \cdots & 0 \\ 1 & 4 & 1 & \cdots & 0 \\ 0 & 1 & 4 & \cdots & 0 \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & \cdots & 2 & 1 \end{pmatrix} \begin{pmatrix} f_0 \\ f_1 \\ \vdots \\ f_{n-1} \\ f_n \end{pmatrix} = \begin{pmatrix} b_1 \\ b_2 \\ \vdots \\ b_{n-1} \\ b_n \end{pmatrix}$$

where the b matrix represents the riht hand side of the above relations for every j from 0 to n.

In order to derive a solution for the Pade's scheme using the multi-grid method, we need to define two matrices :

- Prolongation matrix [N][N/2]
- Restriction matrix [N/2][N]

For our solution we have considered the prolongation matrix and the restriction matrix to be made of zeros and ones. As explained earlier the goal of these matrices would be to prolong and restrict our matrices in order to perform the V-scheme iterations over it. The size of the restriction matrix would be [N/2][N] while that of the prolongation matrix would be [N][N/2] where N denotes the dimension of 'A' matrix.

$$\mathrm{I} = \begin{bmatrix} 1 & 0 & 0 \\ 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \\ 0 & 0 & 1 \end{bmatrix} \qquad \mathrm{R} = \begin{bmatrix} 1 & 1 & 0 & 0 & 0 & 0 \\ 0 & 1 & 1 & 1 & 0 & 0 \\ 0 & 0 & 1 & 1 & 0 & 1 \end{bmatrix}$$

These matrices are the prolongation (I) and restriction (R) matrices with dimension 6 by 3 and 3 by 6 respectively.We would be using the scaled version of these matrices in our code. The restriction matrix would convert the 'correction' matrix into 'correction coarse' and the 'residual' matrix into 'residual coarse' . On the other hand the prolongation matrix does vice versa. It is noticeable that the matrices are transposes of each other , hence storing one of these matrices and then using its transpose to evaluate the other one whenever needed could be an optimal way to reduce memory storage.

4.2 Multi-grid method on a general set of linear equations

We would also be analysing the solution of a general system of linear equations which does not give a derivative function like the Pade's scheme does. This is to compare the performance of the Multi-grid method on different types of matrices.

we would be considering the following matrix :

$$\begin{pmatrix} 2 & -1 & 0 & \cdots & 0 \\ -1 & 2 & -1 & \cdots & 0 \\ 0 & -1 & 2 & \cdots & 0 \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & \cdots & 2 & 1 \end{pmatrix} \begin{pmatrix} x_0 \\ x_1 \\ \vdots \\ x_{n-1} \\ x_n \end{pmatrix} = \begin{pmatrix} 1 \\ 1 \\ \vdots \\ 1 \\ 1 \end{pmatrix}$$

For the 'A' matrix the main diagonal entries are twos, and the sub-diagonal entries are ones. The 'b' matrix is initialised to a column matrix of ones. We would be using the same analogy that we used for the Pade's scheme and would be using the same prolongation and restriction matrices.

5 Plots, Results and Analysis of the serial code

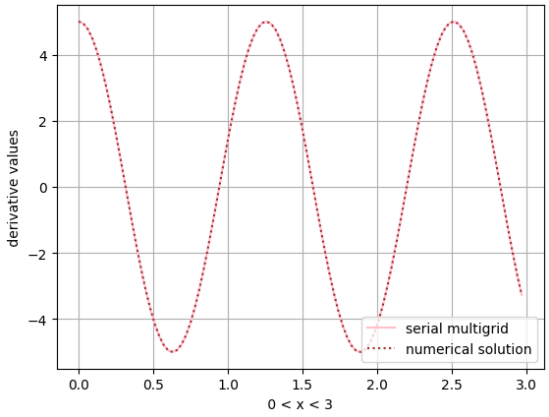

Figure 2: Pade's scheme solution

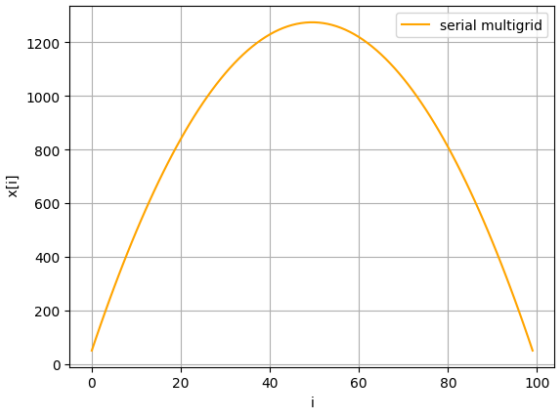

Figure 3: General system of linear equations

We get the run time for the serial codes as 7.1 sec for pade's scheme and 2.2 sec for general system. It is evident from the plots that the Pade's scheme and General system follow an exact plot and have no noise or disturbances. This tells us that the serial code is working perfectly. Moreover the numerical solution of the Pade's scheme perfectly aligns with the solution we get from our working code.

6 Parallelism Strategy

Now we would be trying to parallelize the codes using OpenMP, MPI and OpenACC.We would be discussing in detail about the code snippets that we aim to parallelize and also would be trying to get a time efficient solution for our parallel code.

6.1 OpenMP Implementation

OpenMP provides a shared-memory parallelization technique. We need to analyze the code for loop carried dependencies before paralleling it.

- Gauss-Seidel Iteration: Gauss-Seidel iterations can be parallelized because each iteration updates the solution vector x based on the current residual and system matrix A. The updates to elements of x can be computed independently, which makes it a good candidate for parallelization.OpenMP parallelization is applied to the outer loop of Gauss-Seidel iterations using pragma omp parallel

for. Each thread works on a separate subset of the solution vector x, updating its elements concurrently.

- Coarse Grid Correction: Within each multi-grid level, there is a coarse grid correction step where corrections from the coarser grid are interpolated and added to the current solution vector x. This step involves independent calculations for different elements of x, making it suitable for parallelization. OpenMP parallelization is applied to the coarse grid correction loop using pragma omp parallel for. Each thread computes corrections for a subset of elements in the solution vector x, allowing concurrent updates.
- Matrix-Vector Multiplication for Coarse Grid Residual:This parallelization is applied to calculate the coarse grid residual by performing matrix-vector multiplication using the restriction matrix and the fine grid residual (residual). Since each element of the coarse grid residual can be computed independently, it is parallelizable. OpenMP parallelization is applied using pragma omp parallel for. Each thread computes a subset of elements in the coarse grid residual vector 'residual coarse', enabling parallel computation across different elements.
- Matrix Vector Multiplication for Coarse Grid Correction: This parallelization is used to perform matrix-vector multiplication to compute the coarse grid correction (correction coarse). Similar to the coarse grid residual calculation, each element of the correction vector can be computed independently, making it suitable for parallelization. OpenMP parallelization is applied using pragma omp parallel for. Each thread computes a subset of elements in the correction vector 'correction coarse', enabling parallel computation across different elements.
- By parallelizing these key sections of the code, multiple threads can simultaneously work on different parts of the computation, effectively utilizing the available CPU cores and potentially reducing the overall runtime of the program.

6.2 Open MPI Implementation

OpenMPI provides a distributed memory parallelisation technique. Let us analyze the code and see how could we implement the parallelization of the code via OpenMPI.

- Parallelization of Gauss-Seidel Iteration: The Gauss-Seidel method updates the solution vector x based on the current residual and system matrix A. Since each iteration updates the elements of x using data from the same matrix A and vector b, it can be parallelized across multiple processes. The domain decomposition approach is used here. Each process handles a subset of the solution vector x, and after computation, the updates are exchanged among processes using MPI communication calls (MPI Send and MPI Recv). This allows concurrent updates to the solution vector across processes, reducing the overall runtime.
- Parallelization of Coarse Grid Correction: Similar to Gauss-Seidel, the coarse grid correction step involves updates to the correction vector 'correction coarse' based on the current residual and coarse grid matrix 'A coarse'. Since each iteration updates elements of 'correction coarse' independently, it is parallelizable across multiple processes. Similar to Gauss-Seidel, the domain decomposition approach is used here. Each process handles a subset of the correction vector 'correction coarse', and after computation, the updates are exchanged among processes using MPI communication calls (MPI Send and MPI Recv). This allows concurrent updates to the correction vector across processes, reducing the overall runtime.
- Parallelization of Multigrid Solver: The entire multigrid solver process involves iterations and computations that can be parallelized across multiple processes to distribute the workload and reduce the overall runtime. The parallelization in this function is achieved by distributing the workload of the Gauss-Seidel iterations and coarse grid corrections among different MPI processes. Each process handles a portion of the solution vector x and other associated matrices and vectors. MPI communication calls are used to exchange data between processes when necessary, enabling collaborative computation across the processes.
- By parallelizing these sections of the code using MPI, the workload is effectively distributed across multiple processes, enabling concurrent computation and communication, which reduces the overall runtime of the program, especially for large problem sizes.

6.3 Open ACC implementation

- Parallelization of System Initialization: The initialization of the system matrices (A, b, prolongation, restriction, A coarse) involves setting up the matrices with predefined values. Since these initializations can be done independently for each element of the matrices, parallelization can speed up the initialization process. OpenACC directives are used to parallelize the initialization loops (pragma acc parallel). The present clause ensures that the data referenced within the parallel region is present on the accelerator. The loops are parallelized using the 'num gangs' clause to distribute iterations among gangs of threads.

- Parallelization of Gauss-Seidel Iteration: The Gauss-Seidel iteration updates the solution vector x based on the current residual and system matrix A. Since each iteration updates the elements of x using data from the same matrix A and vector b, it can be parallelized across multiple threads. OpenACC directives are used to parallelize the Gauss-Seidel iteration loop (pragma acc parallel). The present clause ensures that the data referenced within the parallel region is present on the accelerator. The loops are parallelized using the 'num gangs' clause to distribute iterations among gangs of threads.

- Parallelization of Coarse Grid Correction: Similar to the Gauss-Seidel iteration, the coarse grid correction step involves updates to the correction vector 'correction coarse' based on the current residual and coarse grid matrix 'A coarse'. Since each iteration updates elements of 'correction coarse' independently, it is parallelizable across multiple threads. OpenACC directives are used to parallelize the coarse grid correction loop (pragma acc parallel). The present clause ensures that the data referenced within the parallel region is present on the accelerator. The loops are parallelized using the 'num gangs' clause to distribute iterations among gangs of threads.

- Parallelization of Multigrid Solver: The entire multigrid solver process involves iterations and computations that can be parallelized across multiple threads to utilize the computational resources efficiently. OpenACC directives are used to parallelize the multigrid solver loops (pragma acc parallel). The present clause ensures that the data referenced within the parallel region is present on the accelerator. The loops are parallelized using the 'num gangs' clause to distribute iterations among gangs of threads.

- Parallelizing these sections of the code using OpenACC directives enables the execution of computations on accelerators like GPUs, effectively utilizing their parallel processing capabilities.

7 Results and Analysis

These are the resulting plots obtained upon implementing parallelisation of the Multi-grid method.

7.1 OpenMP Solution

The plots are plotted for p=2,4,8 and the serial multigrid code for N=100 (dimension of 'A' matrix). It can be seen that the plots are perfectly similiar providing us the insight that the parallel code converges accurately without any disturbance or noise in the curve for both the Pade's scheme and the General system of linear equations.

7.2 Open MPI Solution

Even in the case of MPI parallelization we get minimal noise and almost perfect convergence indicating that the code is running perfectly and producing efficient results.

7.3 Open ACC solution

The same result can be observed in OpenACC parallelisation indicating efficiency of the code.

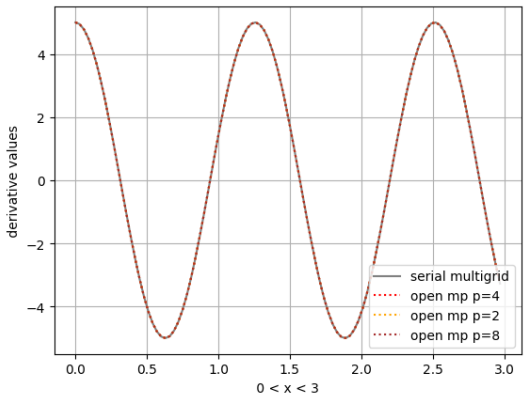

Figure 4: OpenMP Pade's scheme comparision

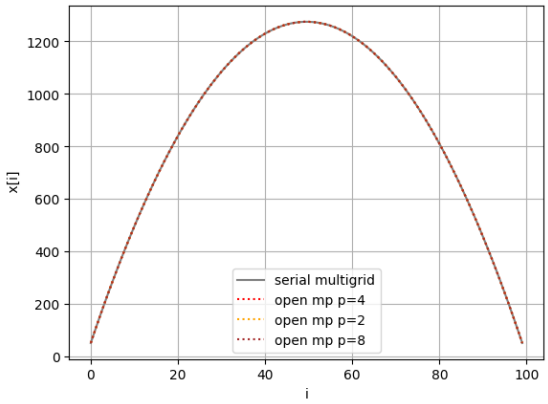

Figure 5: OpenMP comparision for general system

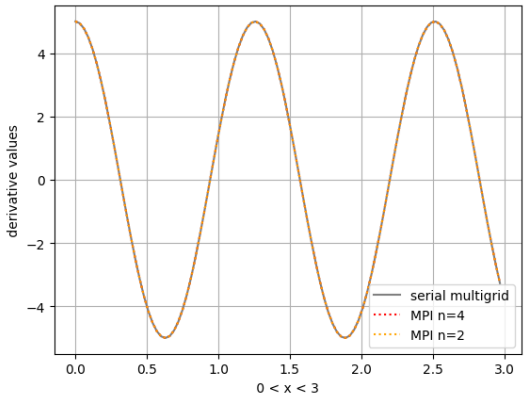

Figure 6: MPI Pade's scheme comparision

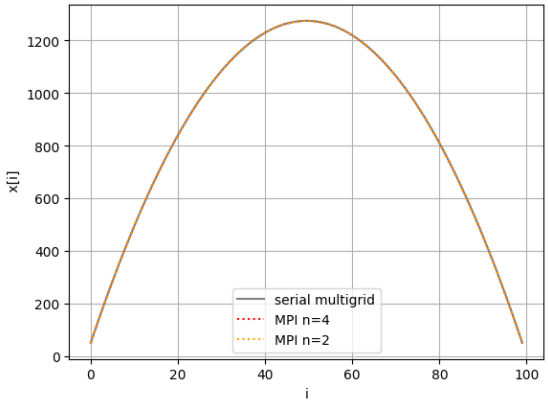

Figure 7: MPI comparision for general system

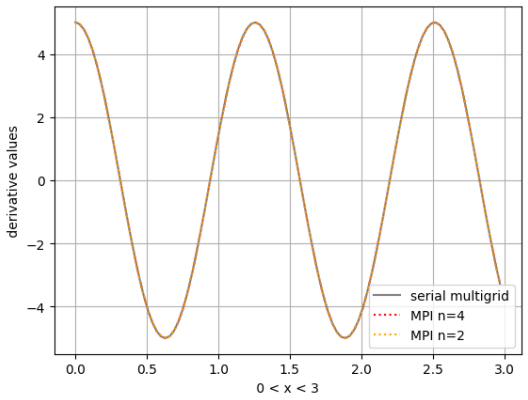

Figure 8: OpenACC Pade's scheme comparision

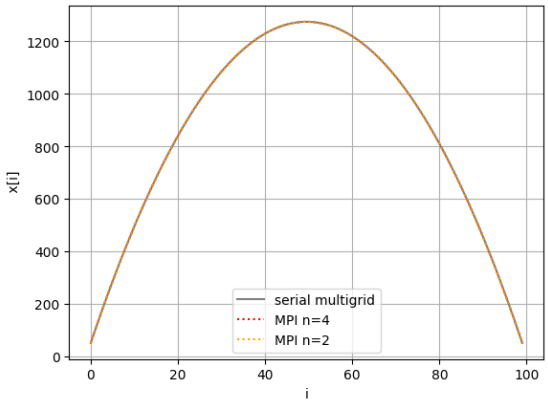

Figure 9: OpenACC comparision for general system

7.4 Speed Up vs cores

The convergence of our codes is perfect but in order to compare the efficiency of the parallelisation we must now compare the Speed up vs number of cores and time of convergence of the codes.

- The Speedup plot obtained for the OpenACC version first dips and then rises. This could be due to the overhead time in the parallel code but as the number of gangs increase, the task becomes more evenly distributed and the time load per gang decreases. Even for the MPI version, we get

increase in Speed up with increase in number of processes.This concludes that the overhead time is relatively less in the MPI parallelization. For the OpenMP version, the Speedup increases for the pade's scheme , but for the general solution , it first increases and then decreases. This tells that a lot of time is being consumed in the forking of threads as number of threads increase. Hence it can be said that the OpenMPI parallelization method has efficiently parallelized our code as it appears to follow the Amdahl's law perfectly for both the plots.

Figure 10: OpenMP Pade

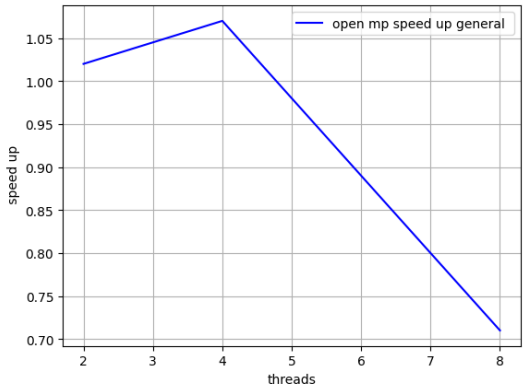

Figure 11: OpenMP general

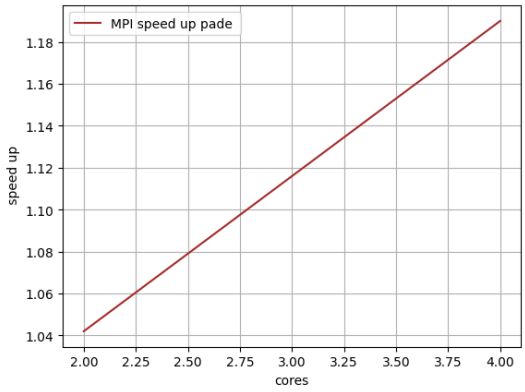

Figure 12: MPI Pade

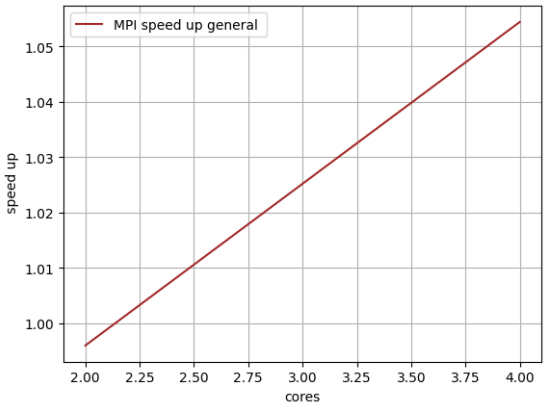

Figure 13: MPI general

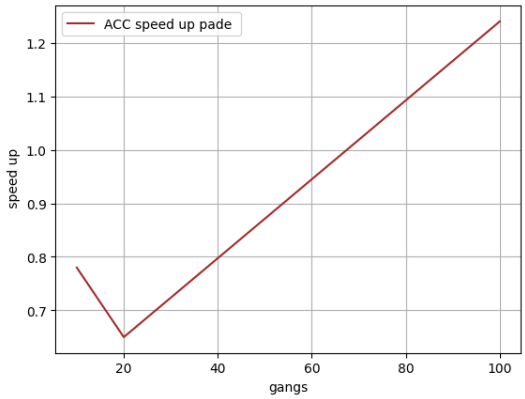

Figure 14: OpenACC pade

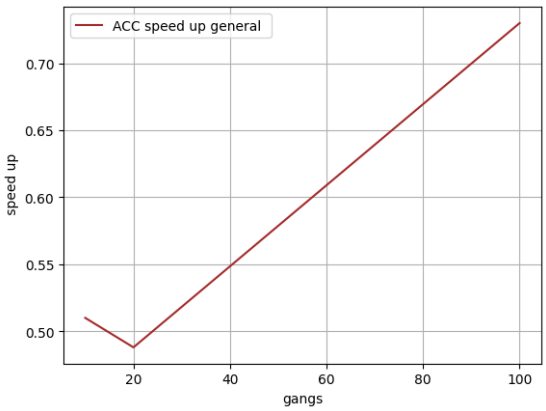

Figure 15: OpenACC general

7.5 Time of execution

The runtimes for all three parallelization schemes have been tabulated below.

7.5.1 OpenMP execution time

System of linear equations	time for p=2	time for p=4	time for p=8
Pade's scheme	3.32 sec	2.59 sec	2.03 sec
General System of linear equations	1.96 sec	1.86 sec	2.81 sec

7.5.2 MPI execution time

System of linear equations	time for cores=2	time for cores=4
Pade's scheme	7.18 sec	6.27 sec
General System of linear equations	2.02 sec	1.91 sec

7.5.3 OpenACC execution time

System of linear equations	time for gangs=10	time for gangs=20	time for gangs=100
Pade's scheme	9.1 sec	11.32 sec	6.87 sec
General System of linear equations	3.87 sec	4.09 sec	2.71 sec

8 Conclusion

This study assessed the efficiency of the Multigrid method by comparing its accuracy on two different systems of linear equations and its parallelization using OpenMP, MPI, and OpenACC. The Multigrid method significantly outperformed traditional iterative methods like Gauss-Seidel and Jacobi, due to its ability to reduce error components across multiple scales, leading to faster convergence and enhanced computational efficiency.

In parallelization, MPI showed the best speedup with increasing cores, making it effective for large-scale problems on distributed systems by minimizing communication overhead and optimizing data distribution. OpenMP achieved faster convergence rates, leveraging efficient shared memory handling and fine-grained parallelism to reduce synchronization overhead. While OpenACC provided performance gains, it did not match the efficiency of MPI and OpenMP, likely due to GPU memory management and kernel execution complexities.

In summary, the Multigrid method is highly efficient for solving large systems of linear equations, outperforming Gauss-Seidel and Jacobi methods. Among parallelization techniques, MPI excels in scalability and speedup for distributed computing, while OpenMP is optimal for rapid convergence in shared memory systems.

9 References

1. Briggs, W. L., Henson, V. E., McCormick, S. F. (2000). A Multigrid Tutorial (2nd ed.).
2. Gabriel, E., Fagg, G., Angskun, T., Dongarra, J. J., Squyres, J. M., Sahay, V., Kambadur, P., Barrett, B., Lumsdaine, A., Castain, R. H., Daniel, D. J., Graham, R. L., Woodall, T. S. (2004). Open MPI: Goals, Concept, and Design of a Next Generation MPI Implementation.
3. Multi-Grid for CFD (Part 2): Restriction and Prolongation Fluid Mechanics 101 [youtube channel]