ID5130: Parallel Scientific Computing

Assignment 2 - MPI

Date: April 12, 2024 Abhijeet Nair (AE20B101)

Question 1 - Wave Equation

Part (a) - Serial Program

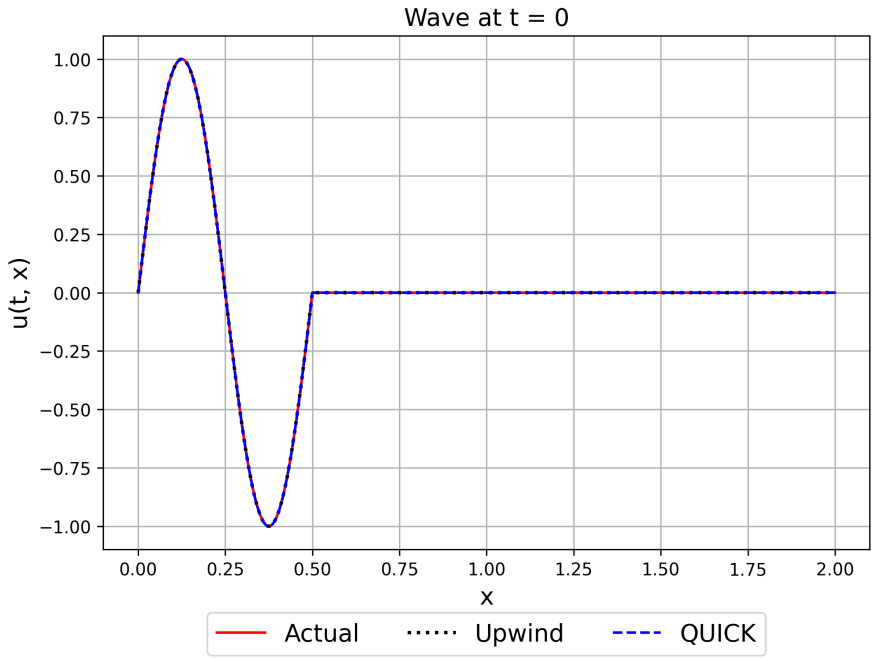

Figure 1: Comparison of analytical and numerical solution at $t = 0$

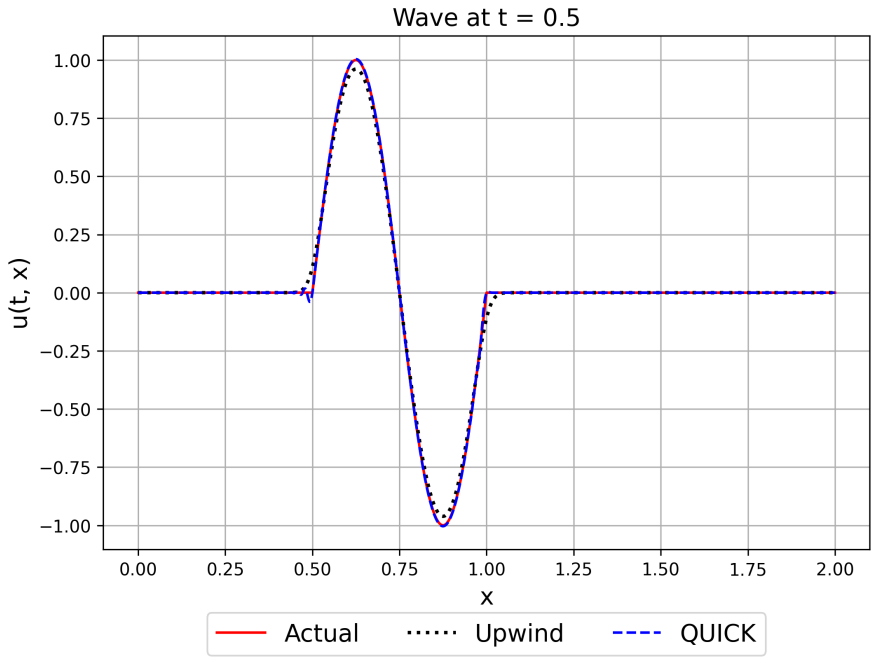

Figure 2: Comparison of analytical and numerical solution at $t = 0.5$

Fig. 1 compares the solutions of the wave equation at $t = 0$. Since this is at the initial time instant, all there plots coincide, as expected. Fig. 2 compares the solutions of the wave equation at $t = 0.5$.

Here, certain differences are noticed. The solution computed using the upwind scheme is observed to be having a smaller amplitude and starts the wave before and ends after the actual solution. The QUICK solution follows the actual solution more accurately, except for a small oscillation at the beginning of the wave.

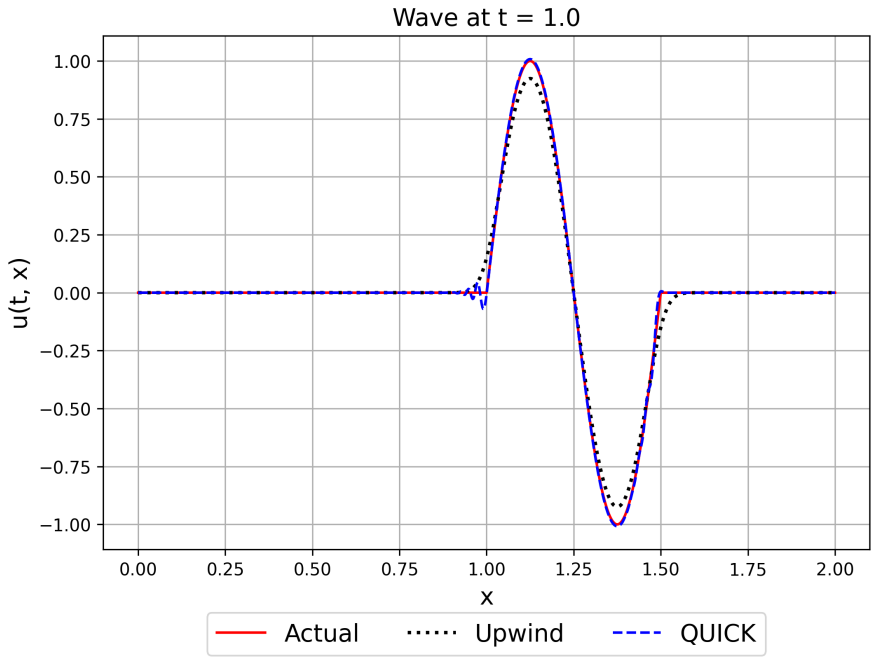

Figure 3: Comparison of analytical and numerical solution at $t = 1$

Fig. 3 compares the solutions of the wave equation at $t = 1$. Here, the differences noticed earlier have become larger.

Part (b) - Parallel Program

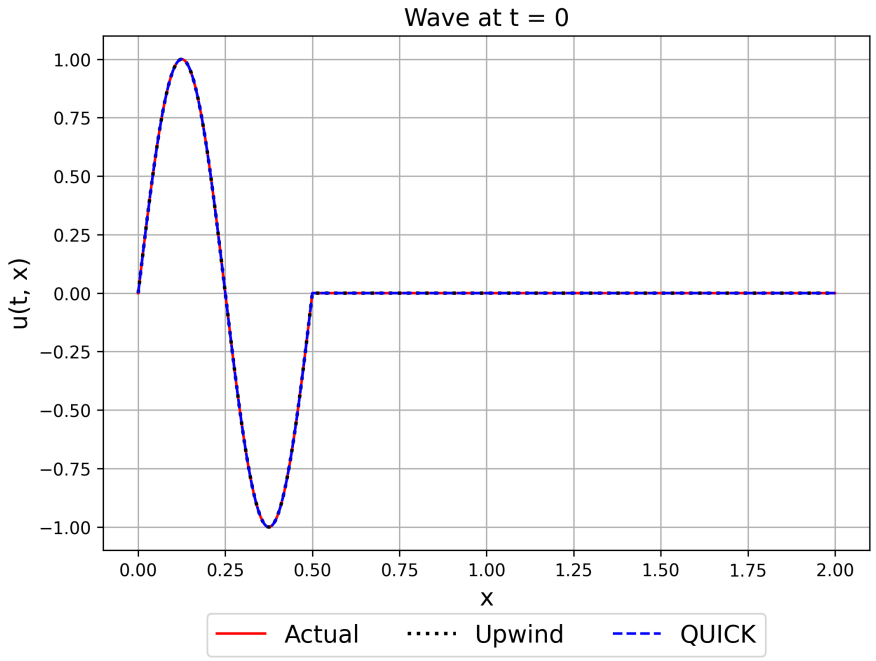

Figure 4: Comparison of analytical and numerical solution at $t = 0$

Figs. 4, 5 and 6 have been generated using parallel code written using MPI. The same results as generated by the serial code have been reproduced using the parallel code as well.

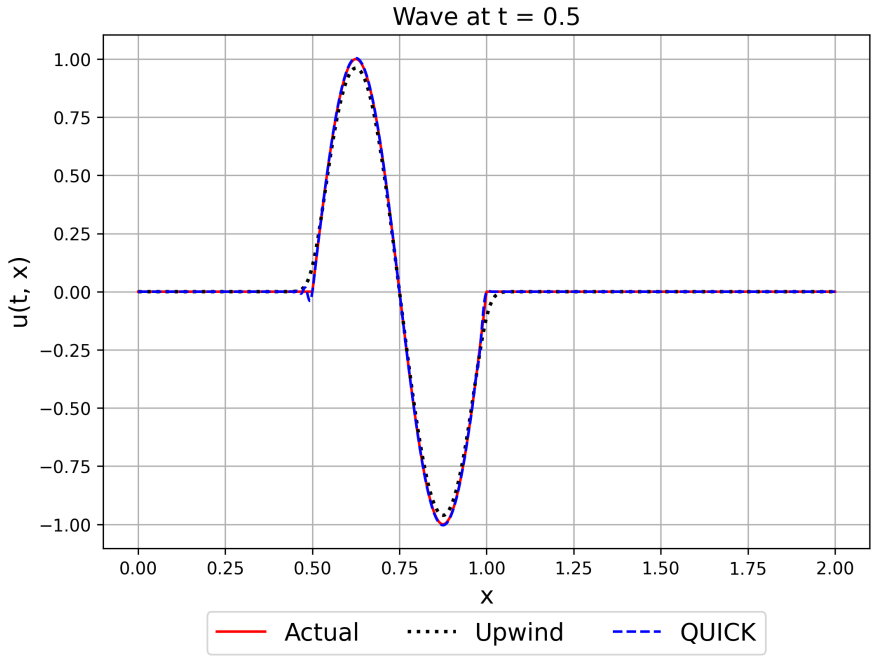

Figure 5: Comparison of analytical and numerical solution at $t = 0.5$

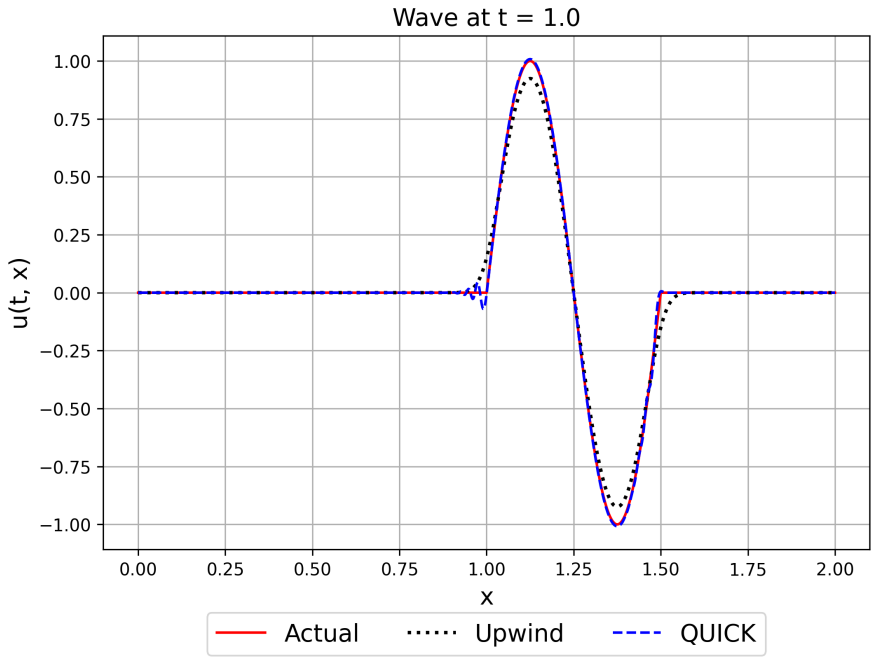

Figure 6: Comparison of analytical and numerical solution at $t = 1$

Part (c) - Comparison of Upwind and QUICK Solutions

As explained above, it is noticed that the both of these schemes lead to a smoother solution at the corner. But, the QUICK scheme's solution is more accurate, closely following the actual solution except at the ends. There is also a small oscillation at the start of the wave. The solution of the Upwind scheme also has a smaller amplitude as compared to the actual solution. In both cases, the deviation from the actual solution seems to be increasing as time progresses.

Question 2 - Poisson Equation

Part (a) - Serial Jacobi Method

Fig. 7 shows the numerical solution of ϕ vs x at $y = 0$ and ϕ vs y at $x = 0$. The number of iterations taken for $\Delta = \Delta_x = \Delta_y = 0.1$ was **688**. The termination condition was the 2-norm of the elements of the difference matrix being less than 10^{-4}.

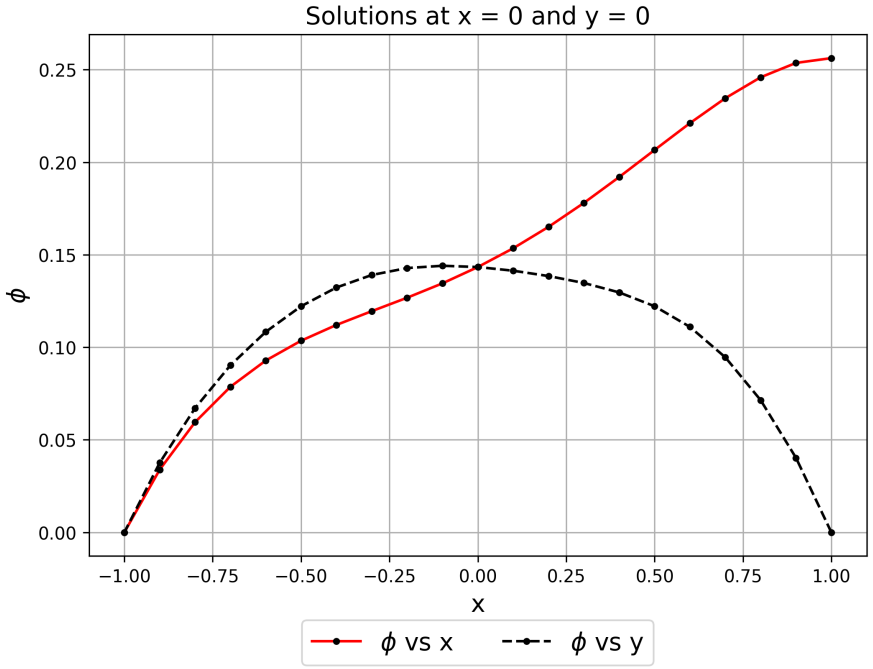

Figure 7: ϕ vs x at $y = 0$ and ϕ vs y at $x = 0$

Part (b) - Parallel Jacobi Method

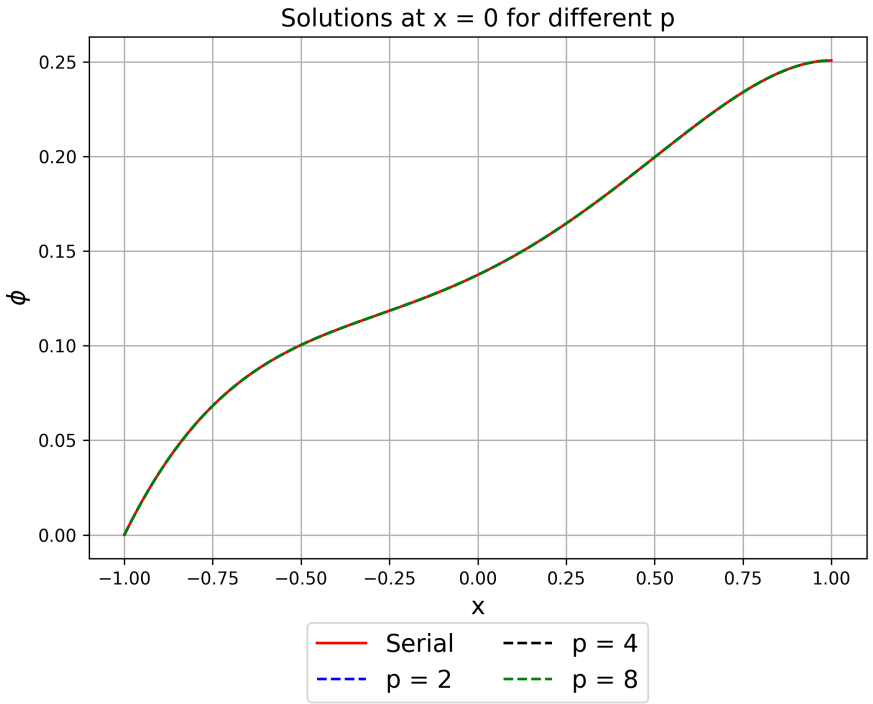

Figure 8: ϕ vs x at $y = 0$ for different number of processors

Fig. 8 compares the solution of ϕ vs x at $y = 0$, computed using different number of processors in parallel and serial. The number of iterations taken for $\Delta = 0.01$ was **38886** for all versions. It can be seen that all the parallel solutions and the serial solution is perfectly matching in this case, thus verifying the parallel code for the Jacobi method. Similarly, Fig. 9 compares the solution of ϕ vs y at $x = 0$. Again, the solutions are perfectly matching.

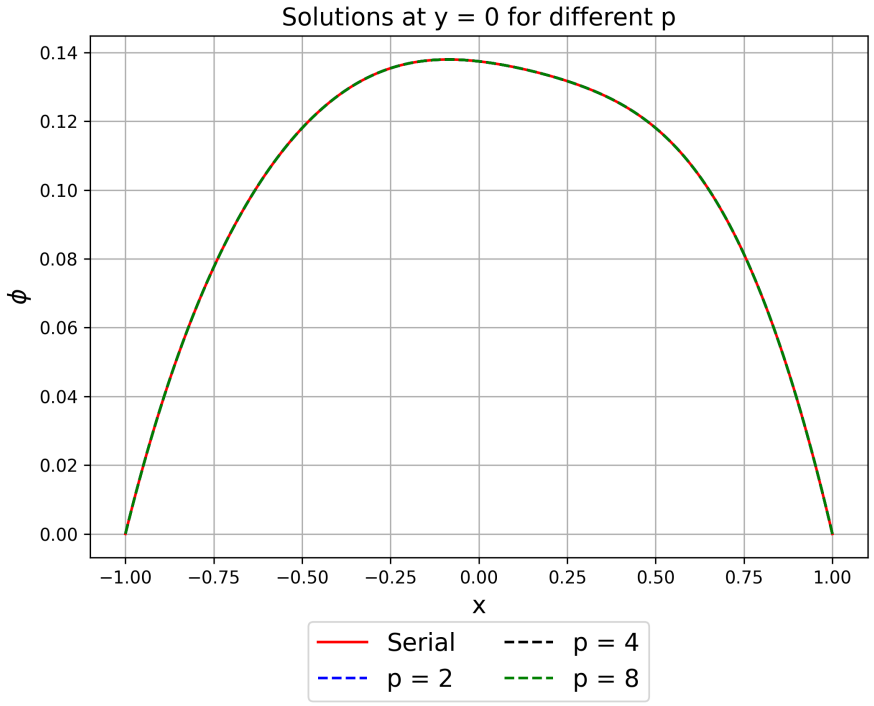

Figure 9: ϕ vs y at $x = 0$ for different number of processors

Part (c) - Parallel Gauss-Seidel Method

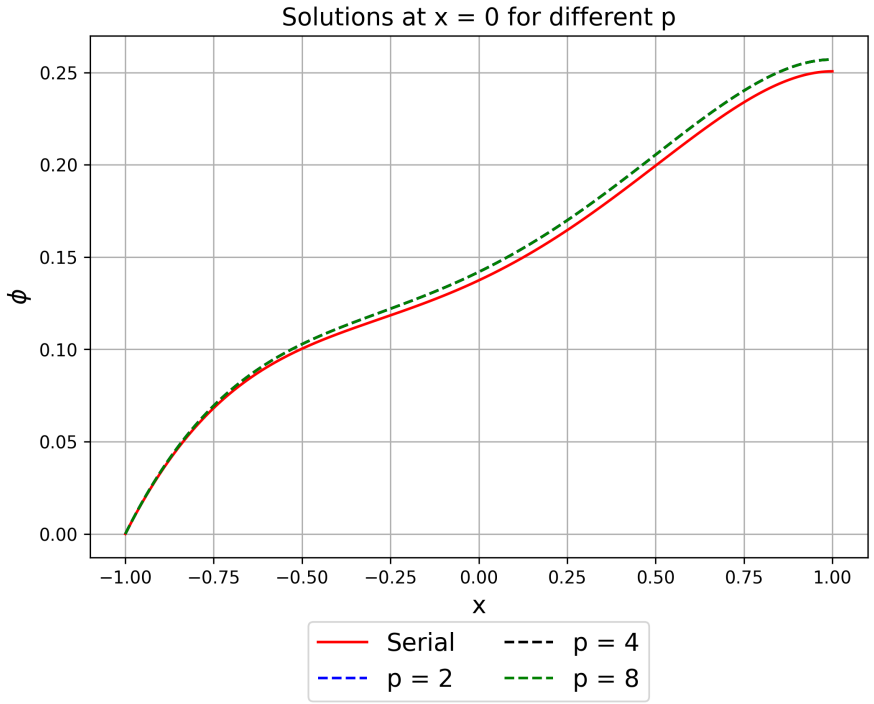

Figure 10: ϕ vs x at $y = 0$ for different number of processors (Jacobi Serial)

Fig. 10 shows the comparison between the solution of ϕ vs x at $y = 0$, computed using different number of processors in parallel using the Red-Black Colouring approach for Gauss-Seidel and

serial using the Jacobi Method. Here, a small difference is there between the plots, suggesting a small difference between the solutions of Jacobi and Gauss-Seidel Methods.

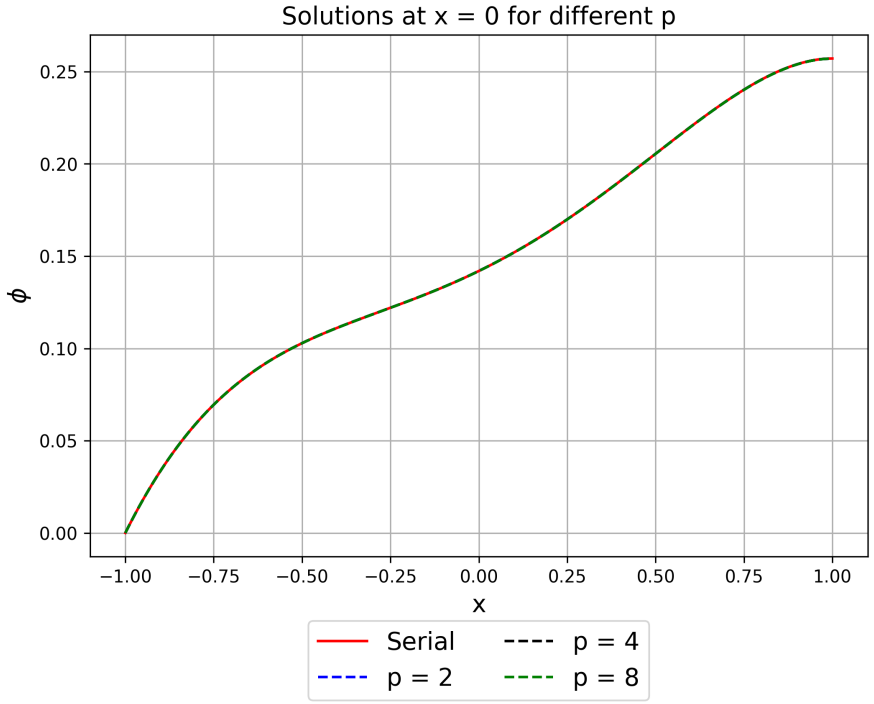

Figure 11: ϕ vs x at $y = 0$ for different number of processors (GS Serial)

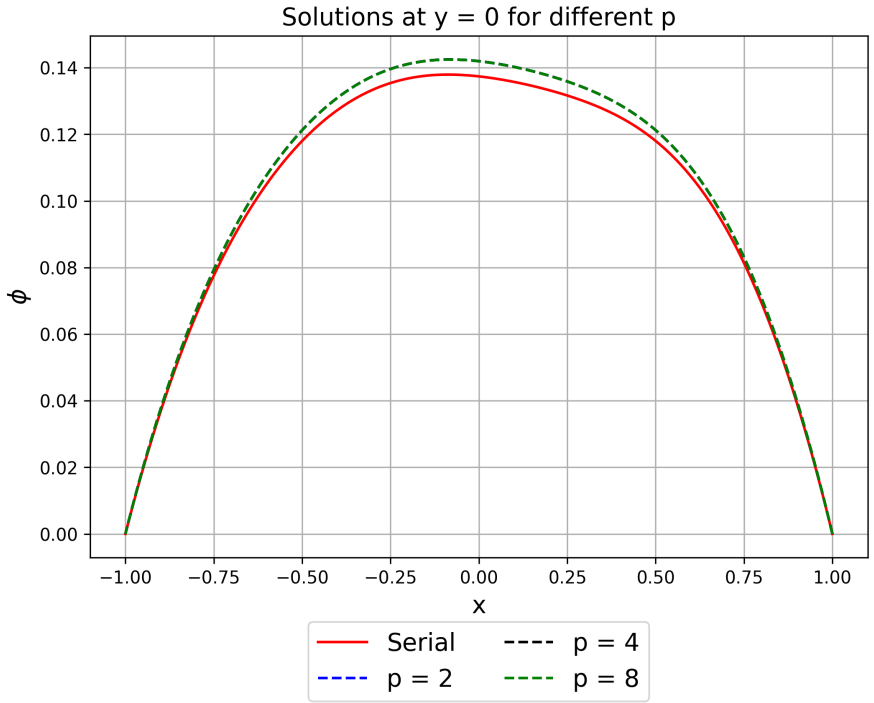

Figure 12: ϕ vs y at $x = 0$ for different number of processors (Jacobi Serial)

Fig. 11 makes the same comparison but with serial computation has been made using Gauss-Seidel. This confirms that the two methods indeed produce slightly different results. The number of iterations taken for $\Delta = 0.01$ was **24008** for $p = 2, 4$ and **24095** for $p = 8$. The serial code for GS method took **23895** iterations for the same Δ. Figs. 12 and 13 show the results for ϕ vs y. The same thing can be observed here as well. In all cases, the Gauss-Seidel method converges in fewer

number of iterations, leading to faster computations. This is because of the additional information being conveyed about the computations which have already taken place in the same iteration.

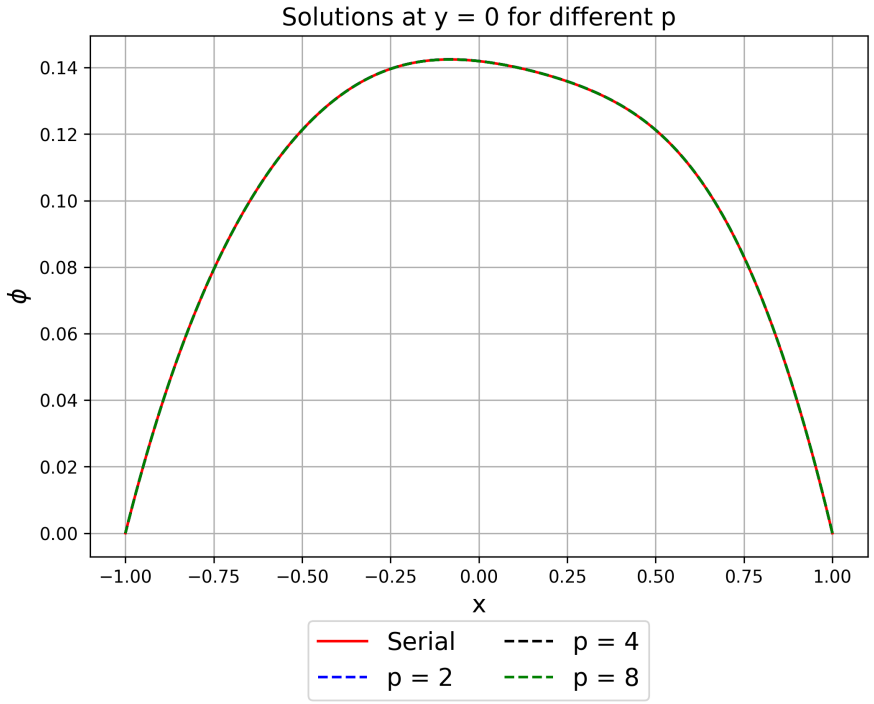

Figure 13: ϕ vs y at $x = 0$ for different number of processors (GS Serial)

Part (d) - Performance Analysis

Fig. 14 shows the speed-up $\psi(n, p)$ of the parallel versions of the Jacobi and Gauss-Seidel codes as a function of the processor count. It can been seen that for both the methods, there is significant speed-up as processor number increases. But we observe that there is more speed-up for the Jacobi method. This can be attributed to the fact that in each iteration, Jacobi method does 2 send/receive (in general) while in Gauss-Seidel (red-black method), there are 4 of them, 2 for sending blue data, and 2 for sending red data. Thus, as the processor count is increased, the overhead associated with sending more number of times causes Gauss-Seidel to have lower speed-up.

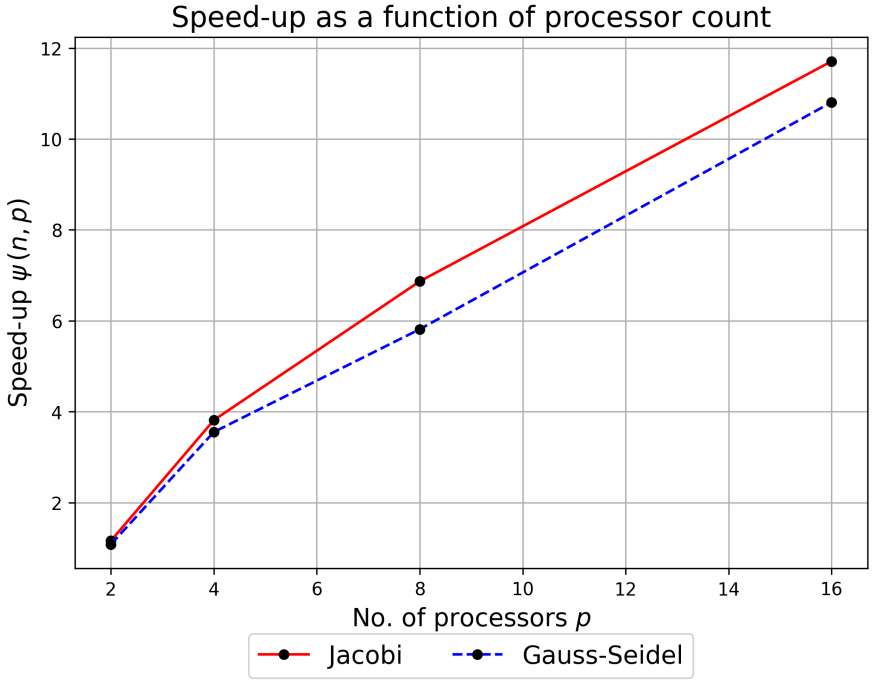

Figure 14: Speed-up $\psi(n, p)$ of the parallel codes as a function of p

Fig. 15 shows the speed-up $\psi(n,p)$ both methods as a function of the problem size, which is represented by the grid size. Here, the parallel version considered was run with $p = 4$, that is, the plotted data is $\psi(n,4)$. The plots show the similar results, with the Jacobi method having better speed-up as compared to Gauss-Seidel. Here, the plot seems to be flattening out as the problem size keeps increasing.

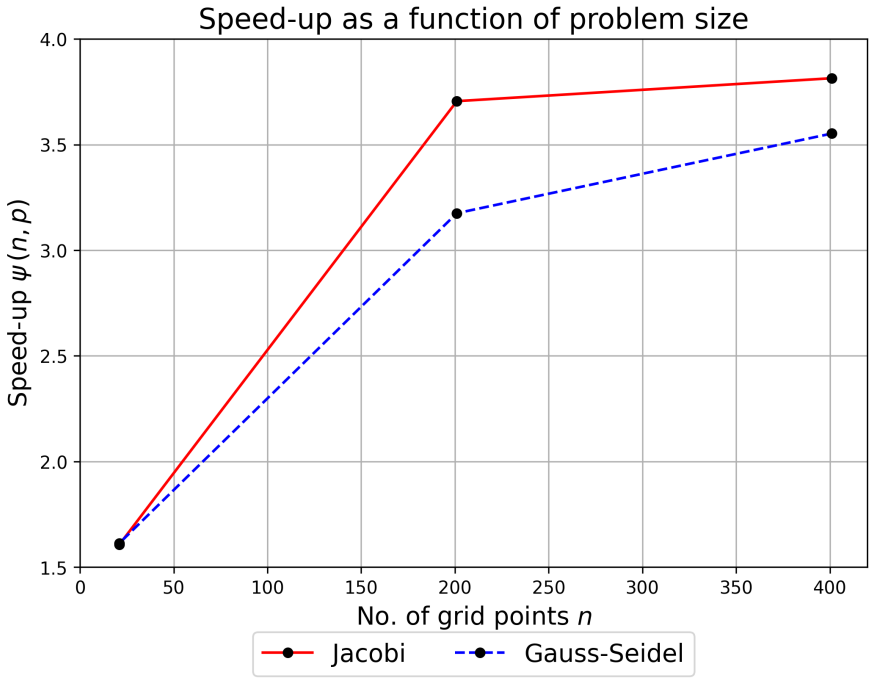

Figure 15: Speed-up $\psi(n,p)$ of the parallel codes as a function of n