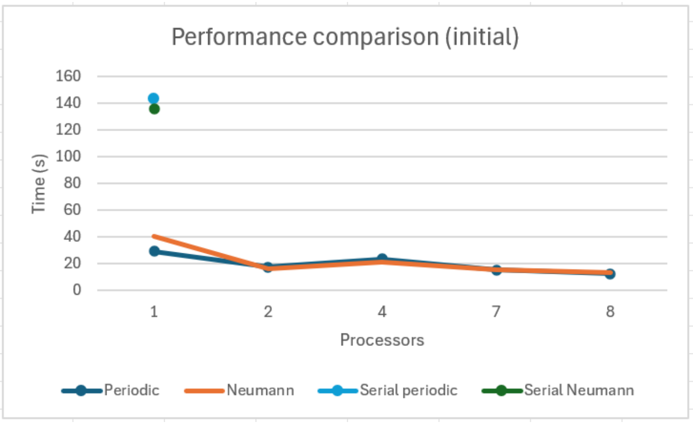
## Improvements and Optimisation

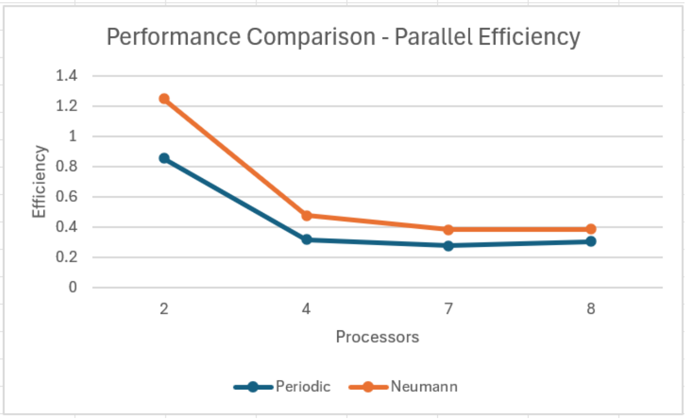
To allow the code to multiple processors following key implementation changes were made:

- The domain is split onto a rectangular grid depending on the number of processors
- Each processor is assigned a subsection of the domain with a padding of 1 on each side to store edge information from neighbouring processors
- C1 and C2 arrays are now stored as C type arrays in the *CMatrix* class. This allows to create custom MPI_Datatypes for easier communication at the edges of the subdomains
- Each processor finds its neighbours up, right, left, and down and stores them for communication
- Periodic – changes to the for loop such that instead of looping over their domain, edge processors have to communicate with the processors on the other end of the domain in order to recreate periodicity
- Neumann – boundary conditions are only implemented for processors on the edge of the domain and only on the respective edge. For the rest, calculations are continuous to the end of their domain. This is implemented by changing the range of the for loops accordingly. Inner processors iterate over their entire domain, except for the padding, which is still used in calculations but is not written to.

**Quantification of Performance:**

After the parallel implementation we can see that with 1 process we are already performing better than the serial code. This is probably due to switching from vector of vectors for the C1 and C2 arrays we switch to C arrays which are contiguous and thus faster to access by the computer.





As we increase the processors the time for the simulation decreases overall. However, we can see it does not do so linearly. For 4 processors the time even increases. This can be due to the increase in number of communication costing more than the speedup from the parallelisation. Additionally, assessing the parallel efficiency, it signifantly drops as the number of processors increases – this can be due to the increased overhead of setting up communications and could potentially be improved by reducing the number of communications where possible.

**Optimisation**

To reduce the overhead from the communications I tried to create a temporary datatype which would send C1 and C2 boundary at the same time. Initially, tried implementing only for sending the upper boundary in the periodic simulation but did not manage to create the communication correctly. See Parallel_Reaction_Equations_Joint_Comm.cpp file and do_iteration_periodic function.