MPI Wave Equation Report

The aim of this assignment is to implement a parallel solver for the wave equation below:

$$\frac{\partial^2 u}{\partial t^2} = c^2 \nabla^2 u$$

Implementation of this solver within C++ involved the use of several MPI techniques for domain decomposition, peer-to-peer communications with the aim to increase efficiency/scalability with increasing processors. With this, the domain was decomposed following from the lecture example, aiming to ascertain an evenly distributed sub-grids amongst processors.

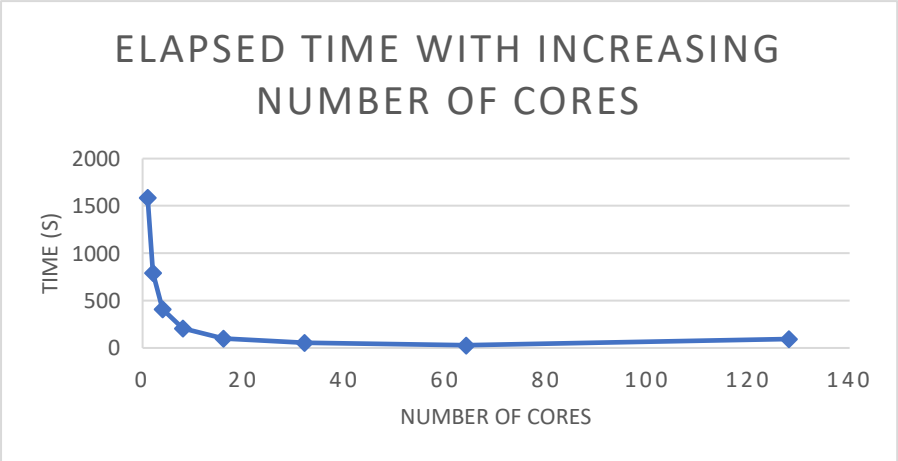
The analysis was performed on the provided DUG High-Performance Clusters, running over several grid sizes (using 500x500 as a standard), for a solver over the pre-defined 30 second period, computing 25 frames per second.
These pre-defined grid sizes were objectively chosen due to the limitation of the available per job allocated run-time coupled with being optimal in showing the expected result. Performance was further measured on slightly small grid (e.g., 100x100) – showing the inefficiencies that can occur due to using a too large number of cores for a particular problem set. This is evaluated further below.

Efficiency/Scalability speed-up of the system solution with increasing processors is carried out through timings with increasing cores using the DUG HPC machine available. Testing was carried out on a series of domain grid sizes of 100x100, 250x250 and 500x500. A single node with 2, 4, 8, 16, 32, 64 cores was used as this remained the limit; with further testing of 128 cores (split between 2 nodes running with 64 cores).

Within this analysis we will be discussing the two main measures of performance:

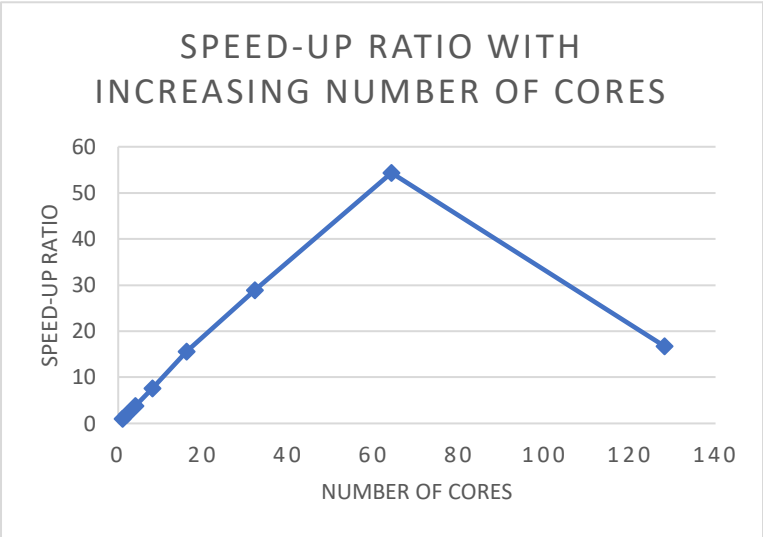
1. Speed-up ratio
2. Parallel Efficiency

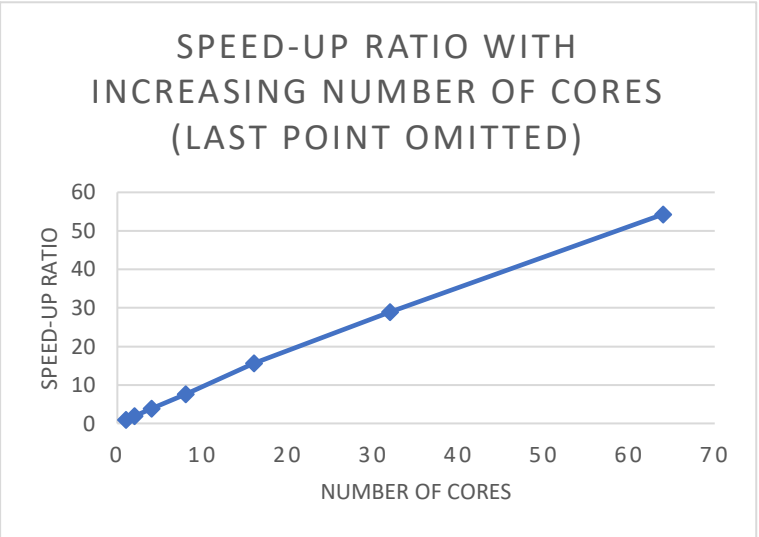
Below shows the elapsed run-time of the solution as number of cores increases for a given 500x500 grid size.



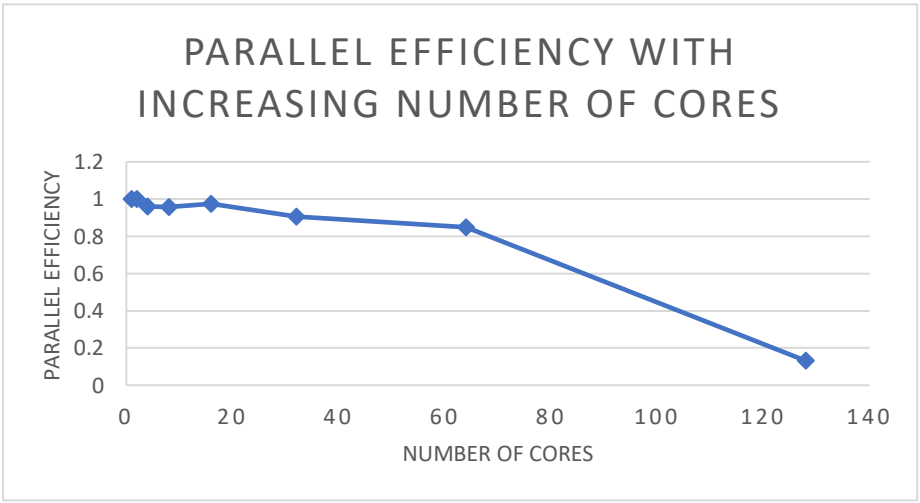
As depicted in the graph above, there is a significant decrease in the elapsed run-time as the number of cores scale. This, however, shows a general plateau region within the 32-64 core range as limitations of the MPI scalability is overturned by serial dependencies such as creating and destroying variables as well as the overheads of increasing cores becoming more apparent.

There is a however, an increase in elapsed run-time with 128 cores to a time of 94.42s. This can be explained due to the setup of 128 cores requiring 2 HPC nodes and thereby needing extra communications to process, incurring a dramatically increased run-time.



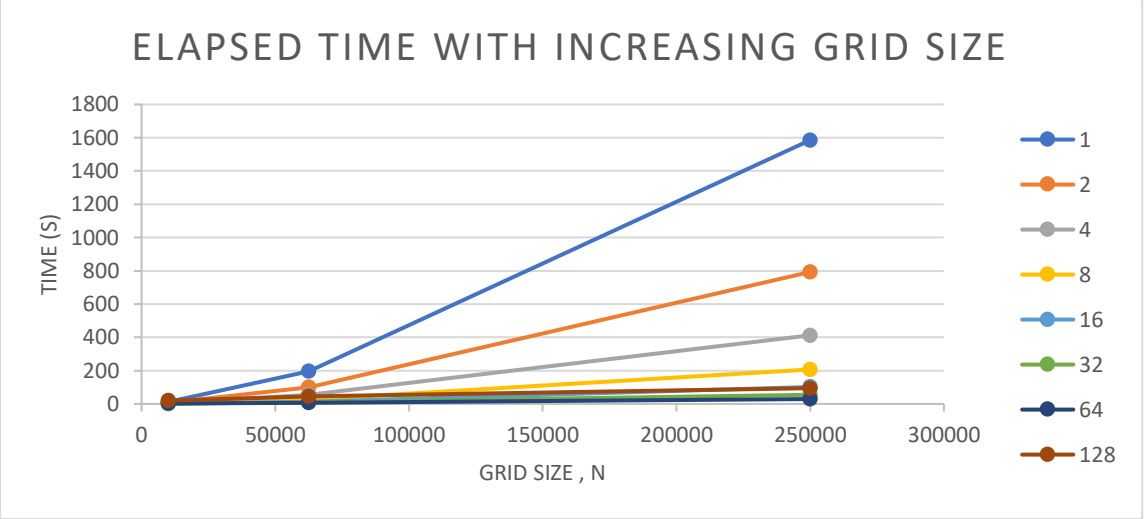


The graphs shown above express the speed-up: how much faster the solver runs in parallel compared to running at serial (1 core). The figure on the right-hand side shows a linear trend when analysing cores up until n=64. This was used to illustrate the linear speed-up of the runs as opposed to the left figure which considers the data point at n=128. As mentioned above, this however is not a true measure at n=128 comparable to the rest, due to the hardware limitations requiring 2 separate machines communicating back and forth adding a layer of overheads.

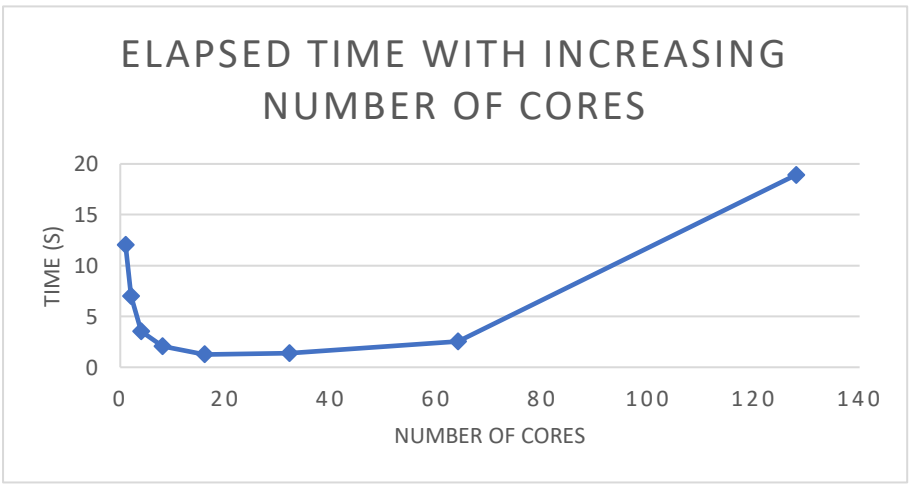


The second important measure of performance describes the parallel efficiency. This is the speed of the system relative to the ideal speed-up for that number of cores. Typically, and as shown in the graph above, with increasing number of cores, the efficiency begins to decline. The graph above shows a near consistent efficiency rating with being over 0.95 from two until 16 cores, dropping slightly at 32 and 64 cores with values of 0.9 and 0.85 respectively.

This expresses a minimal decline in the parallel efficiency with increasing cores subject to not falling below 85% the expected. This again, is with knowledge to omit the last data point of n=128 cores due to the points iterated previously.



The graph above shows the behaviour of the listed cores while the grid size is increased. The trend shows that time increases linearly with grid size, and that this required compute time decreases exponentially with increasing cores. This figure details the difference between serial and 2 cores in run-time shows a dramatic difference as opposed to the comparison of the larger cores tested.



Further, this graph illustrates the elapsed run-time on the HPC cluster with increasing number of cores at a smaller grid domain of 100x100. The figure shows that there is a significant (expected) decrease in run-time as cores increase. However, the timings begin to increase at n = 32, and signifying an increasing run-time trend with increasing cores at 64 and (expected) 128.

This is because of domain decomposition occurring inefficiently due to too many cores called for this grid size. Being unable to distribute the overall domain data segments into significantly appropriate sections renders the overall run-time to increase, further coupled by increased overheads of increased cores.

This thereby demonstrates the issue of requiring knowledge of some specifics of the problem-set/problem domain wishing to solve, to remain increasing parallelism and use of cores to be advantageous.

To conclude, through analysis of the derived solution using DUG's HPC system, the results show it coincides with the expected knowledge of increasing the number of cores, thereby decreases the elapsed run-time. The solution and results discussed magnifies a near to ideal speed-up with a great efficiency rating at around 90% across the cores. However, we have further seen the effects of parallelism across two nodes – whereby communication across the systems produces an obscene overhead resulting in inefficiency and lack of computational gain. Further to this, the choice of cores must represent an appropriate choice for a given grid size, or it will in turn give little to no computational improvement; and seldom unnecessary overhead becoming detrimental to elapsed time. As a result, running on particularly large grids with core increments from 2 until 64 showed parallelism effectively implemented.