High Performance Computing Workshop 4

February 23, 2021

The problem

In this exercise we use halo-swapping to solve the 2D wave equation in parallel. All code was written using c++. We make extensive use of point-to-point communication functions of the MPI library.

We aim to discretise and numerically solve the heat equation

$$u_{tt} = u_{xx} + u_{yy}$$

for $x, y \in (-1, 1)$, $t \in (0, T)$, subject to initial condition

$$u(x, , y, t = 0) = e^{-40((x-0.4)^2+y^2)}, u_t(x, y, t = 0) = 0,$$

and homogeneous Dirichlet boundary conditions

$$u(x = \pm 1, y, t) = u(x, y = \pm, t) = 0.$$

As per the worksheet questions we will use the central difference method and leap-frog discretising over the t-dimension. This leads to

$$\frac{U_{i,j}^{n+1} - 2U_{i,j}^{n} + U_{i,j}^{n-1}}{\Delta t} = \frac{U_{i+1,j}^{n} + U_{i-1,j}^{n} + U_{i,j+1}^{n} + U_{i,j-1}^{n} - 4U_{i,j}^{n}}{(\Delta x)^2}$$

for $i, j \in \{1, ..., M-1\}$, $n \in \{0, ..., N-1\}$ with the initial conditions given on the worksheet.

Instead of solving the code in serial we will parallelise the computation using MPI, splitting the load between P different processes.

The halo swapping works in a similar way to the 1D heat equation from the previous workshop. The only difference is now we move a swap a whole boundary of $M + 1$ points between one strip and another one.

The code

We begin by initialising an array U_MPI which will hold the numerical solutions to the PDE computed in parallel.

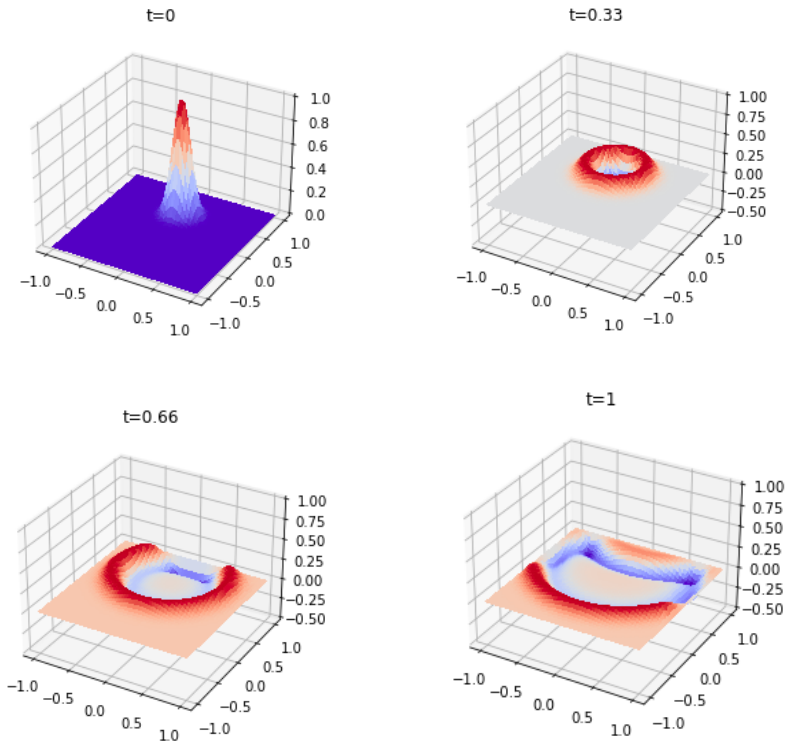

Figure 1: Solution of the second-order wave equation on a square

Each process is allocated $J(M+1)$ points such that

$$J = 2 + \frac{M-1}{\text{size}},$$

where size is the number of processes.

Halo-swapping then ensures that the correct values are swapped between processes so each can compute the next iteration of the wave equation.

Numerical results

We can see that all the solutions of the serial code match those given in the worksheet. The results can be seen in Figure 1. This is grand news as it means our code is working.

For $M = 2305$ when the domain is split into horizontal strips we can see that as we increase the number of processes then the time taken to solve the wave equation decreases. This is as we would expect and can be seen in Figure 2.

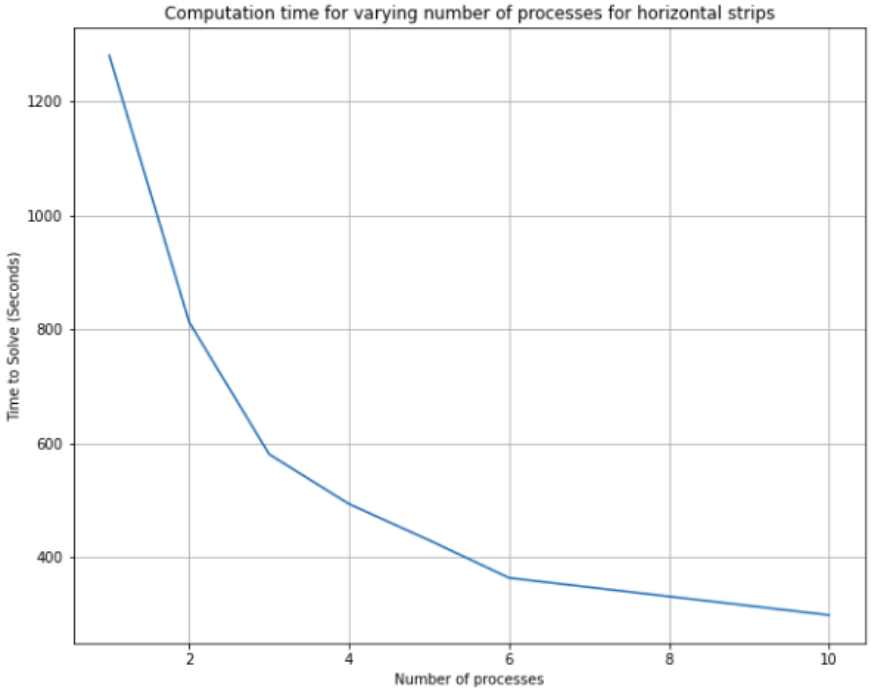

Figure 2: Computation time against number of processors

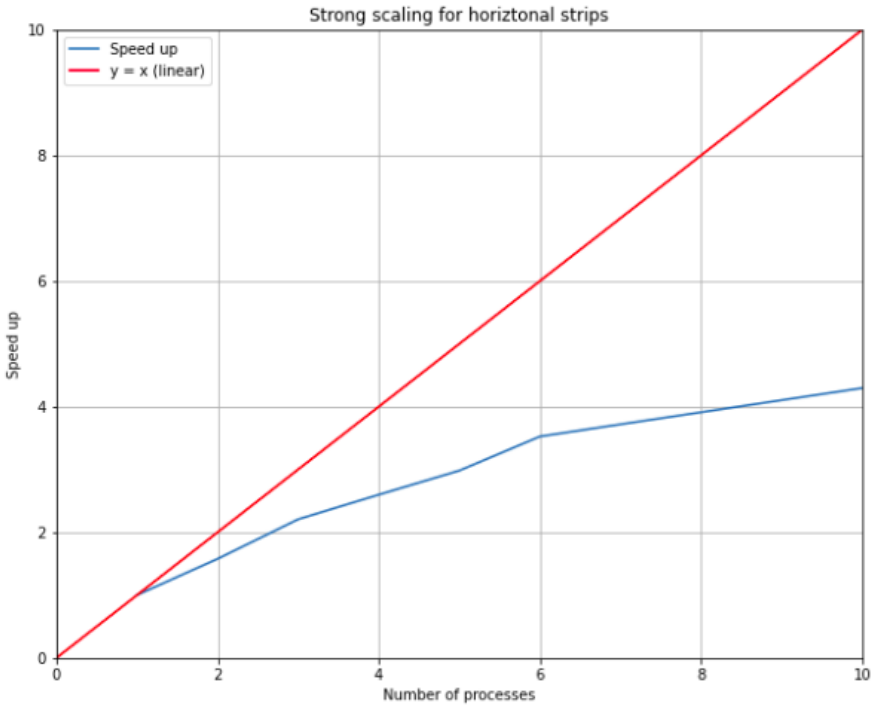

Figure 3: Strong scaling - Speed up against number of processors

To asses the performance of the parallel application as the number of processors is increased we use scaling. Strong scaling is when the problem of the size stays the same as the number of processors increases.

To observe this we first calculate the speed up

$$S(N,P) = \frac{T(N,1)}{T(N,P)},$$

where N is the size of the problem, P is the number of processors, $T(N,1)$ is the time taken on one processors and $T(N,P)$ is the time taken on P processors. We then plot speed up against the number of processors. The results for horizontal strips can be seen in Figure 3.

We can also measure performance by calculating both parallel efficiency

$$E(N,P) := \frac{S(N,P)}{P} = \frac{T(N,1)}{PT(N,P)}$$

and serial efficiency

$$E(N) := \frac{T_{\text{best}(N)}}{T(N,1)}.$$

Notice that $E(N,P) < 1$ and $E(N) \leq 1$. We can see in Figure 4 how the parallel efficiency decreases as the number of processors increases.

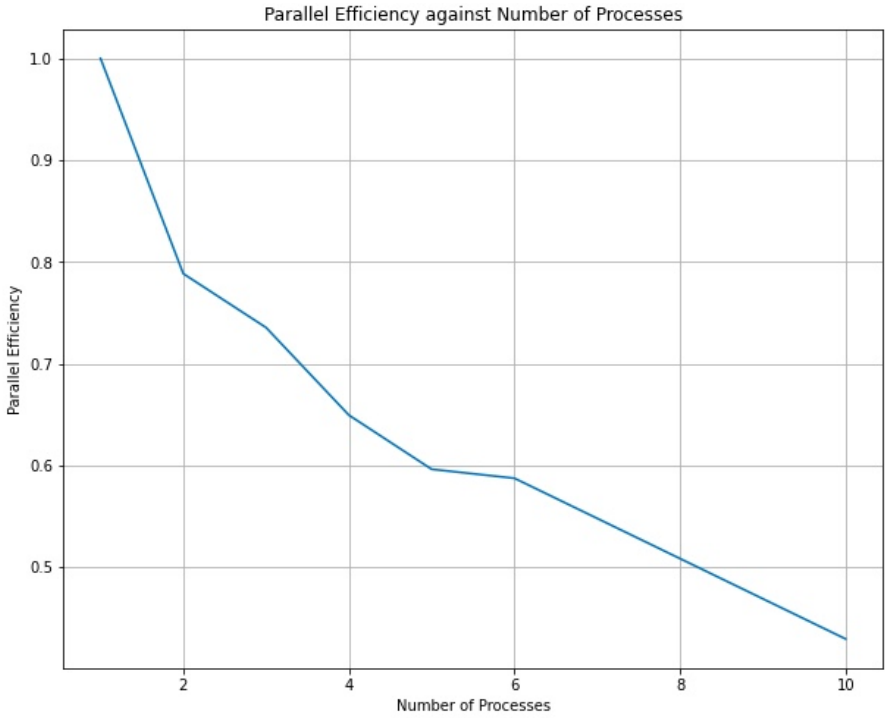

Figure 4: Parallel Efficiency

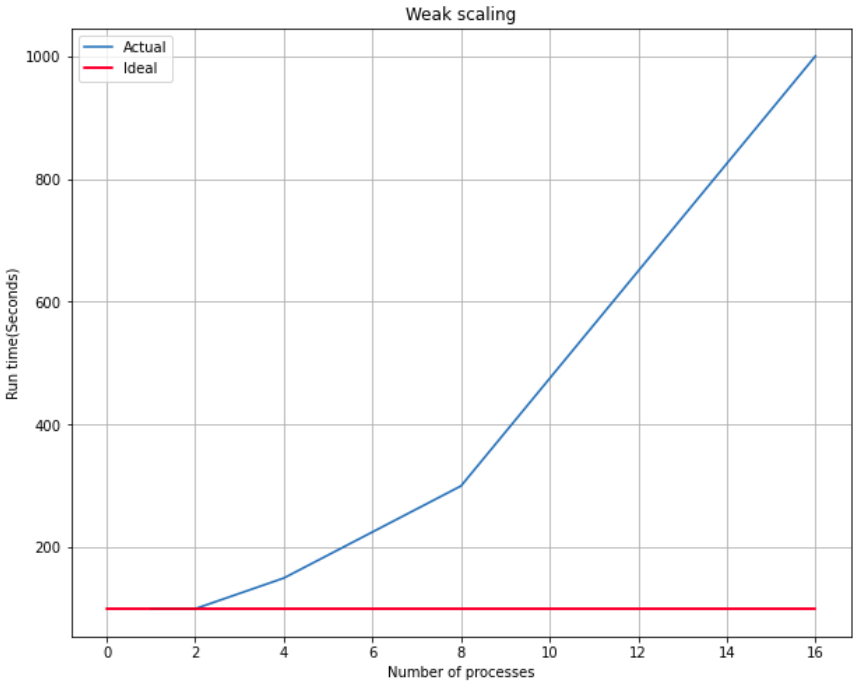

Figure 5: Strong scaling - Speed up against number of processors

For all the processors we tested the code on $P = 10$ gave the fastest run-time $T_{\text{best}}(N) = 298.107$ seconds. The time taken on one processor was $T(N, 1) = 1280.53$ seconds. This gives a serial efficiency for $N = 2306$ of

$$E(N) = \frac{T_{\text{best}}(N)}{T(N, 1)} = \frac{298.107}{1280.53} = 0.23.$$

This seems pretty good but could be made better by rerunning the code on more processors as the run-time appears to have not plateaued yet at $P = 10$.

We can also assess to performance of our code using weak scaling where the problem size increases at the same rate as the number of processors. This will ensure that the amount of work given to each processor is the same each time. If we had ideal parallel code this would mean that the run time would not change. However, we can see from Figure 5 that this is not the case. This is because the serial part of our code prevents the speed up.

Vertical strips

The code was also completed so it solves the wave equation in vertical strips rather than horizontal. The code can be found in this directory.

Initially the code was completed using dynamic memory allocation for each $2D$ array U in each process and U_{sol} in the root process. This proved to be a bit of a nightmare, with segmentation errors all over the place.

I then changed to code to define the arrays statically. This meant to code ran straight away for both the horizontal and vertical strips. I've included code for the horizontal strips with dynamic and static arrays and for vertical just with static arrays. Unfortunately this meant I did not have time to complete the section of work which involved splitting the domain up into small squares. I have a pretty good idea of how to do this though and think with a bit more time I could have completed this task.