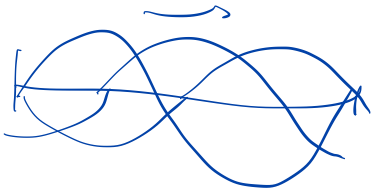
1D Wave Equation Worksheet

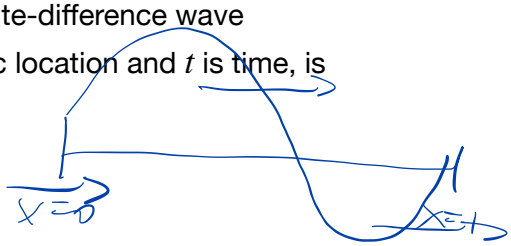
Dr. Michal A. Kopera

ME 471 / 571, Spring 2021



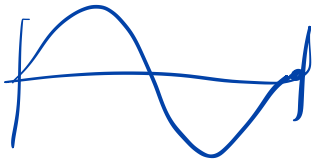
1. One-dimensional wave equation

In this assignment you will parallelize a serial implementation of a finite-difference wave equation solver. The wave described by $u(x,t)$, where x is geometric location and t is time, is governed by partial differential equation given by



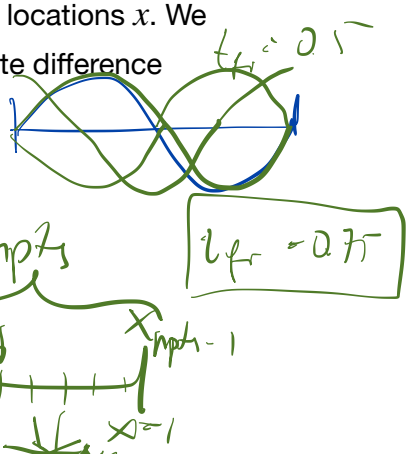
$$\frac{\partial^2 u}{\partial t^2} = c^2 \frac{\partial^2 u}{\partial x^2}, \qquad x \in [0,\ 1], \quad t \in [0,\ t_{final}],$$

where c is the wave speed. We apply a periodic boundary condition, meaning that if a wave travels to the right and crosses the right boundary at $x = 1$, it comes back in at $x = 0$. We will also need the following initial conditions to start the simulation at time $t = 0$:



$$u(x,\ 0) = \sin(2\pi x), \qquad \left.\frac{\partial u}{\partial t}\right|_{t=0} = -2\pi c \cos(2\pi x).$$

The task of the program is to find the solution $u(x,t)$ for any time t_{final} and all locations x. We can approximate both temporal and spatial derivatives using second order finite difference formula:



$$\left.\frac{\partial^2 u}{\partial x^2}\right|_{x=x_i} \approx \frac{u_{i-1} - 2u_i + u_{i+1}}{\Delta x^2}, \quad \text{for } i = 0,\ldots,\text{npts} - 1,$$

$$\left.\frac{\partial^2 u}{\partial t^2}\right|_{t=t_n} \approx \frac{u_i^{n-1} - 2u_i^n + u_i^{n+1}}{\Delta t^2},$$

where $\Delta x = x_{i+1} - x_i$ is the distance between consecutive points, and is assumed constant; npts is the number of spatial points we chose in our approximation; u_i^n is the value of our wave function at spatial location x_i and time t_n; Δt is the size of the time step. Putting both approximations into the equation, yields:

$$\frac{u_i^{n-1} - 2u_i^n + u_i^{n+1}}{\Delta t^2} = c^2 \frac{u_{i-1}^n - 2u_i^n + u_{i+1}^n}{\Delta x^2}.$$

Rearranging such that the terms on the left hand side are the "future" values u^{n+1}, and on the right hand side only the present and past values u^n, u^{n-1} gives:

$$u_i^{n+1} = \left(c\frac{\Delta t}{\Delta x}\right)^2 \left(u_{i-1}^n - 2u_i^n + u_{i+1}^n\right) + 2u_i^n - u_i^{n-1}$$

and we can set $\alpha = c\frac{\Delta t}{\Delta x}$, which is a parameter which relates the speed of the wave, size of the time step and spatial resolution. We want to make sure that out choice of Δt and Δx are such that $\alpha \leq 1$ (this is called Courant-Friedriechs_Levy condition, or CFL). We finally arrive at a formulation:

$$u_i^{n+1} = \alpha^2 \left(u_{i-1}^n - 2u_i^n + u_{i+1}^n\right) + 2u_i^n - u_i^{n-1}$$

In this scheme, to compute a future value of u_i^{n+1} we need the values at the current time-step n, as well as at the previous time-step $n-1$. For any given point i, we need values u_{i-1}^n, u_{i+1}^n from neighboring points at current time-step n. This is not a problem in the interior of the domain, but we have to consider what to do for the periodic boundary conditions. Since the wave that exits one end of the domain, comes back through the other, it is like both ends were connected to each other, forming a loop. We can express the equations for both endpoints as:

$$u_0^{n+1} = \alpha^2 \left(u_{npts-1}^n - 2u_0^n + u_1^n\right) + 2u_0^n - u_0^{n-1}$$

$$u_{npts-1}^{n+1} = \alpha^2 \left(u_{npts-2}^n - 2u_{npts-1}^n + u_0^n\right) + 2u_{npts-1}^n - u_{npts-1}^{n-1}$$

Note how we use the other end-point value for the right or left neighbor.

I have provided for you a serial code wave.c and partially parallelized code wave_mpi.c with a matching Makefile. The serial code works, but is not parallel. The partially parallelized code initializes MPI, creates the data and makes sure the simulation is set-up and ready to go. All it is missing is the communication routines. You job is to fill in missing commands and make sure that the code executes correctly in parallel. To help you judge that, the program prints the value of L2 norm of the difference between your solution and the exact solution, and also writes a file results_wave.dat with values of your solution and exact solution at all the points x at the final time. You can plot those values to see whether you have a match using the plot_wave.mlx Live Script. You need to provide three input parameters to the program: number of points, size of time-step, and final time for the simulation. You can run the program as follows:

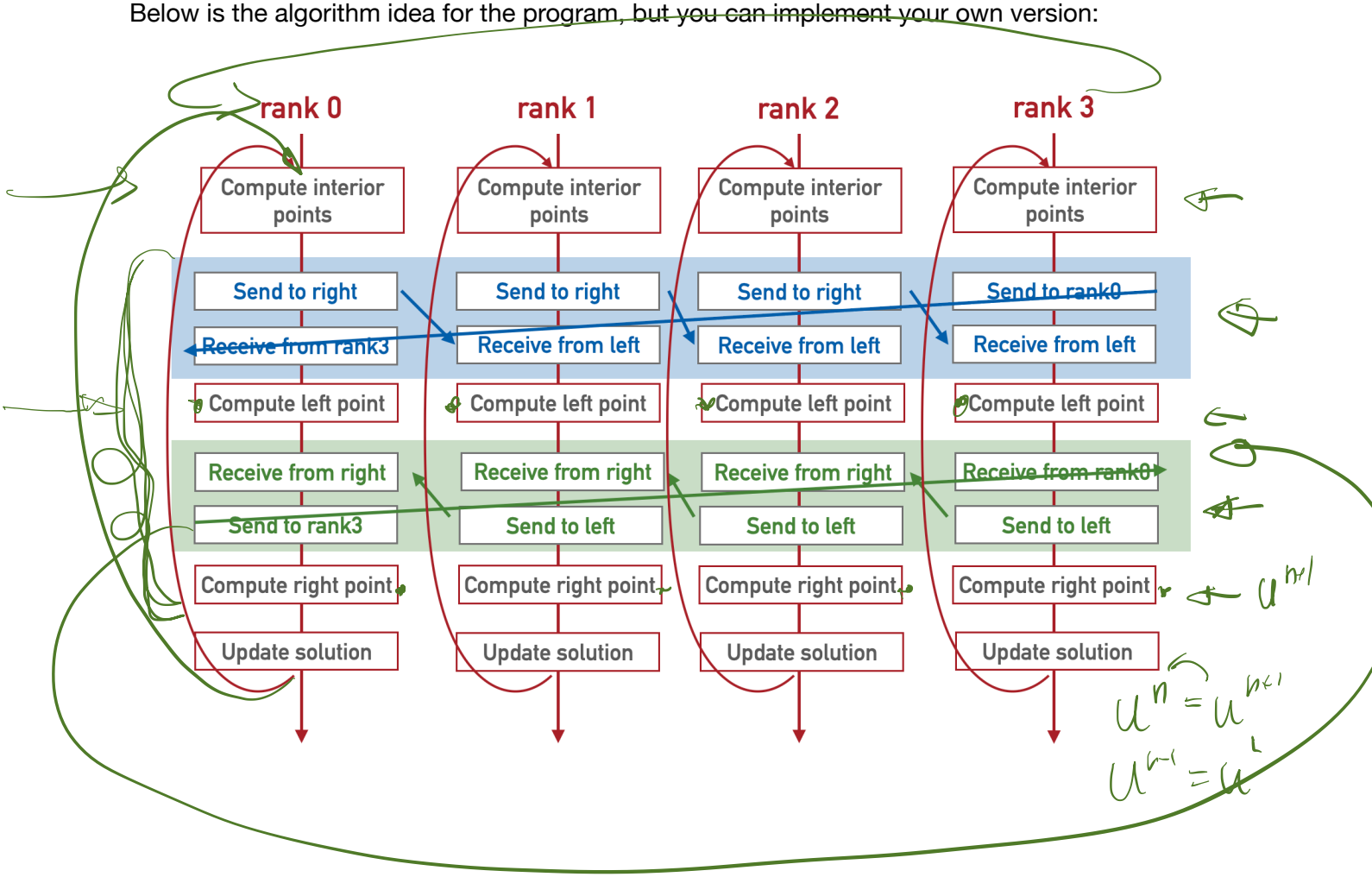
```
mpirun -np p wave npts dt t_final
```

where p is the number of processes you want to use, npts is the number of points, dt is the size of the time-step, and t_final is the final simulation time. A good choice of parameters to try is:

```
npts = 1024
dt = 0.0005
```

which would result in $\alpha \approx 0.5$. Remember, that some choices of dt may be too big for the dx resulting from your choice of npts, as dx = 1/(npts-1). The program will complain if this happens. Note that the npts has to be divisible by p. You can choose any p which satisfies that. The choice of t_{final} is arbitrary, depending on how long you want your simulation to run. $t_{final} = 0.5$ will result with the initial wave moving across half of the domain, while $t_{final} = 1$ will bring the wave back to the starting location (as we have a periodic boundary condition).

Below is the algorithm idea for the program, but you can implement your own version:



Task 1

Fill in the missing communication routines in the wave_mpi.c code. You can choose your parallelization strategy. Are you going to use MPI_Send and MPI_Recv pairs, or

MPI_Sendrecv? How would you handle the communication at ranks 0 and nproc-1? After you are done check whether you get a correct result by using the plot_wave.mlx.

For simplicity, you only need to parallelize the actual computation, which happens in the time loop (clearly marked in the code), and not pre- and post-processing (allocation of arrays, initial condition, computing L2 norm, writing results). You are welcome to take a look at some of those functions and code.

Task 2

After you successfully parallelize the program (remember to check results for correctness), try a few different settings for npts and p. Run a strong and weak scaling experiments and plot the results.

A note for weak scaling: computing speedup for weak scaling makes little sense, as our simulation does not really speed-up. We try to keep the time constant as we increase the size of the problem. For weak scaling, we can express the efficiency as:

$$E = \frac{T_1}{T_p},$$

where T_1 is the time to solve a small problem on 1 processor, and T_p is the time to solve a p -times bigger problem on p processors.

Task 3

Create another version of the wave_mpi.c code, where you choose a different communication strategy. If you went with Sendrecv, maybe you can try regular Send and Recv pairs, or explore what happens when you switch the order of Recv and Send, or locate them in a different location in the code?

Create efficiency plots for weak and strong scaling, and compare them with those in Task 2. Does the choice of t_{final} affect the efficiency of your code?