OpenMP Problem Set

Anastasios-Faidon Retselis (AEM: 4394)

15/2/2021

1 Exercise 1

In exercise 1, the wave equation $u_{tt} - \alpha^2 u_{xx} = 0$ is to be solved using the Lax-Wendroff method.

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

where $c = \alpha\frac{\Delta t}{\Delta x}$ and the initial conditions and time step are defined in the problem set statement. First, the code is implemented as a serial program and can be found under **Ex1_retselis_single.c**. Running the code for $N = 200$ grid points, the program returns a .csv file containing the required information to plot the solution $u(x,t)$ as well as the corresponding surface.

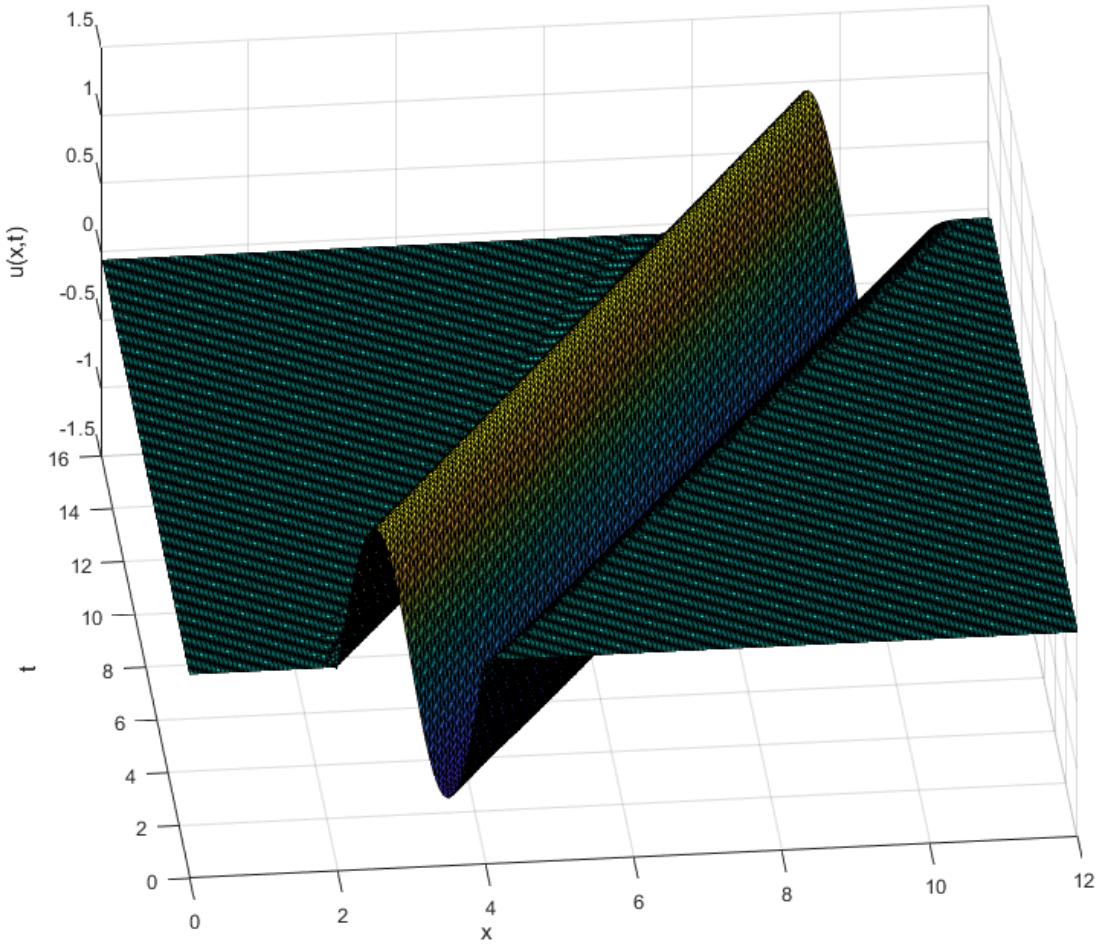

Figure 1: Surface plot of $u(x,t)$

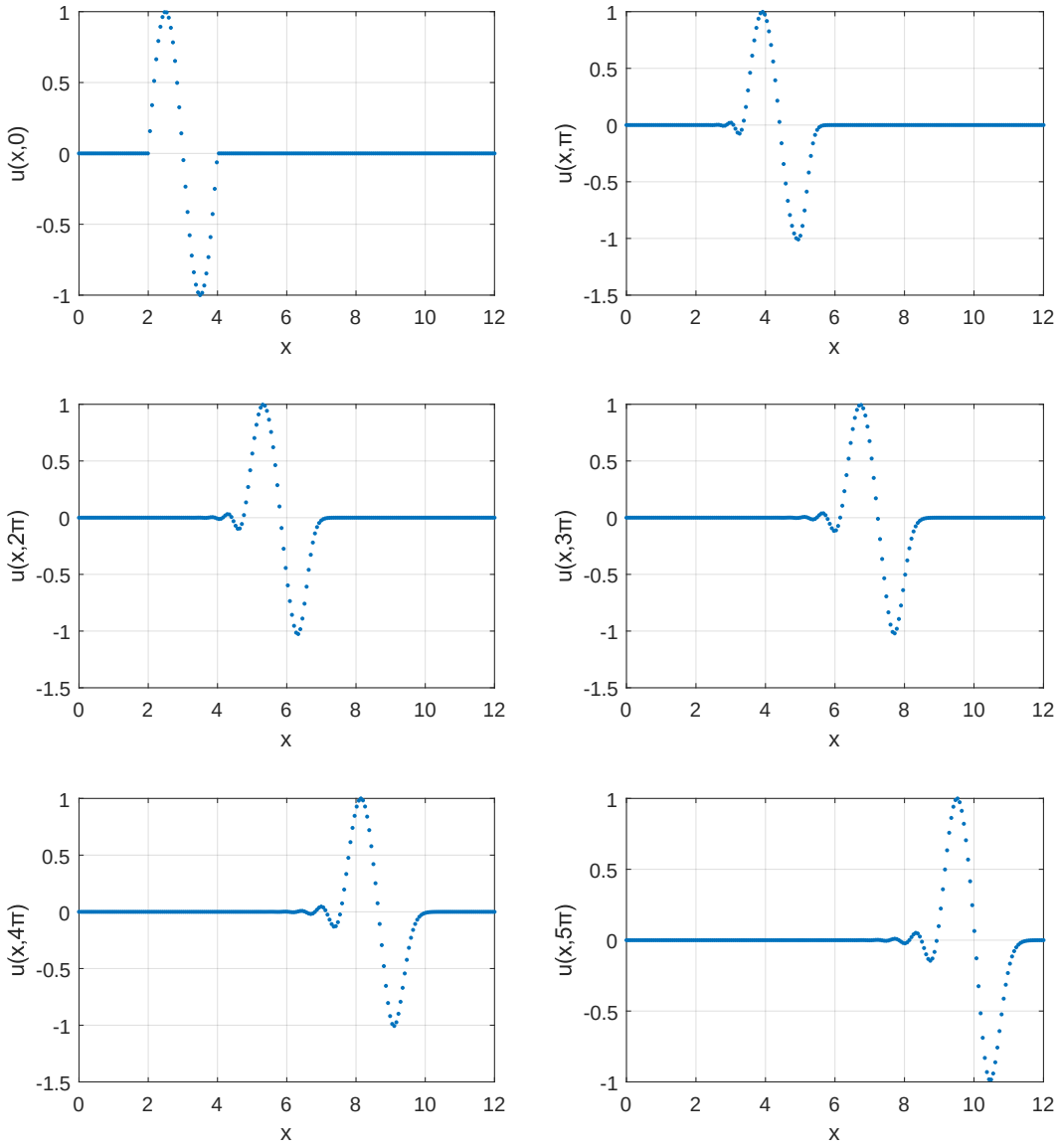

Figure 2: Solution $u(x,t)$ for $t = 0, \pi, 2\pi, ..., 5\pi$

Next, the program is converted in order to utilize OpenMP. The solution can be found in the file **Ex1_retselis_parallel.c**. After validating that the program runs correctly for $N = 2000$ points, we can increase the number of points to evaluate our code. Note that for the requested $N = 2000$ points no noticeable change occurs by utilizing several threads and therefore the number of grid points was increased to $N = 20000$. The results can be found in Table 1, while the speedup is visualized in Figure 3

Table 1: Results for N=20000 grid points

Threads	Time (sec)	Speedup
1	42.239	1
2	29.231	1.445
4	18.851	2.241
8	17.753	2.379

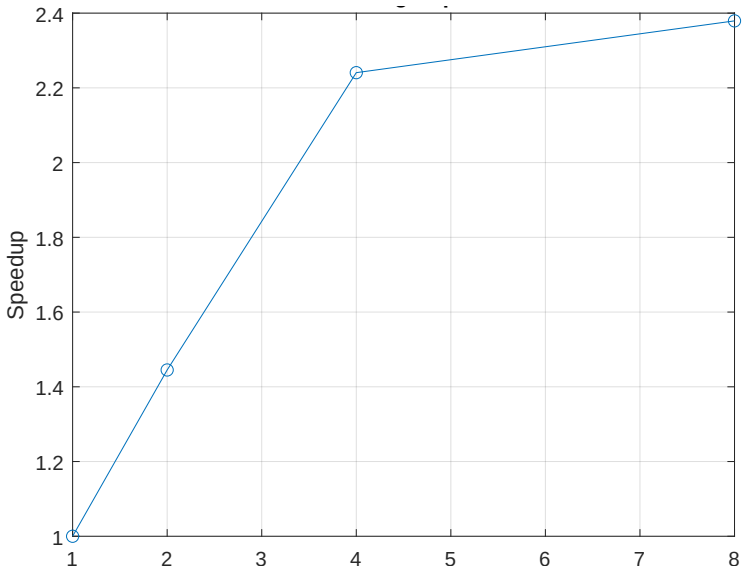

Figure 3: Speedup vs Threads for $N = 20000$

2 Exercise 2

In Exercise 2, the file *matmul.c* is to be converted to a parallel program. The solution can be found in **Ex2_retselis_parallelmatmul.c**. Let us consider the multiplication of the 3x3 matrices:

$$A = \begin{pmatrix} 3 & 3 & 3 \\ 3 & 3 & 3 \\ 3 & 3 & 3 \end{pmatrix} \tag{2}$$

$$B = \begin{pmatrix} 2 & 2 & 2 \\ 2 & 2 & 2 \\ 2 & 2 & 2 \end{pmatrix} \tag{3}$$

$$A \cdot B = \begin{pmatrix} 18 & 18 & 18 \\ 18 & 18 & 18 \\ 18 & 18 & 18 \end{pmatrix} \tag{4}$$

The above example is used as a reference case to ensure that the parallel program developed runs without producing an error. Having confirmed the good functionality, we can now run for several rows and columns to determine the speed required for the program to run by using different numbers of threads. We will use 1000 rows and 1000 columns for each matrix.

Table 2: Results for 1000×1000 square matrices

Threads	Execution Time (sec)	Speedup
1	8.145	1
2	4.384	1.445
4	3.012	2.241
8	2.635	2.379

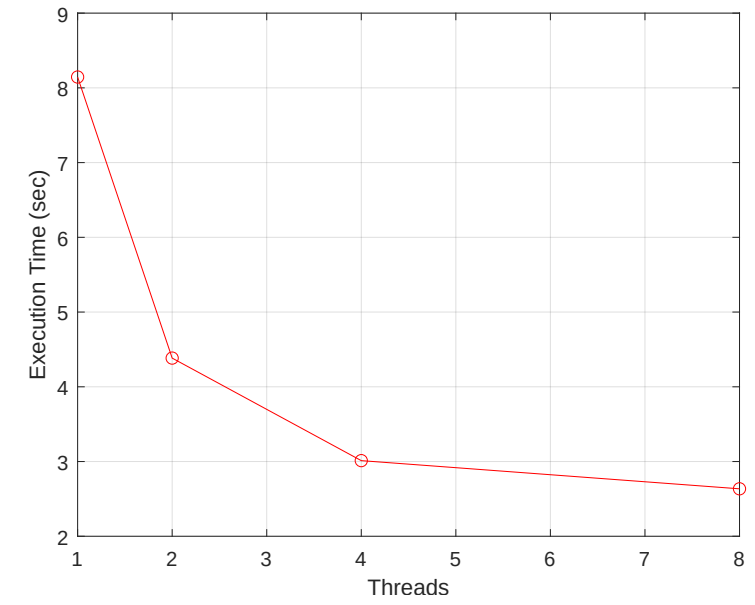

Figure 4: Execution time vs Threads for 1000×1000 square matrices

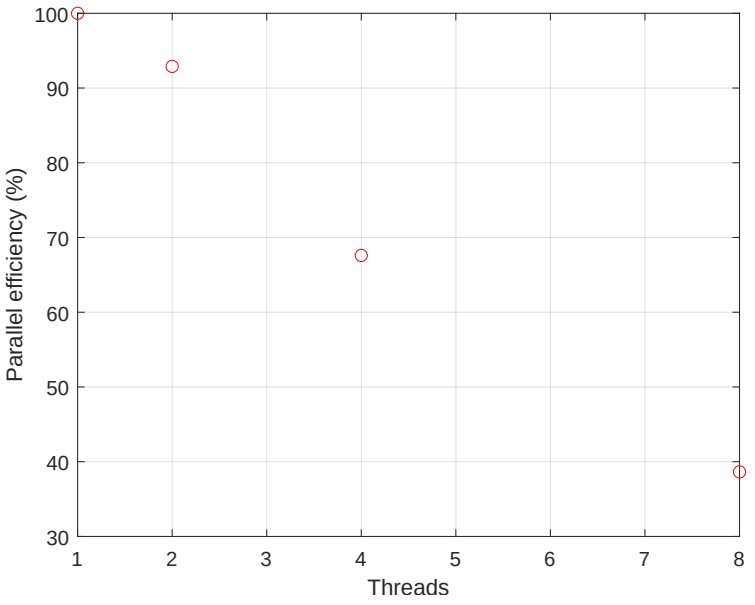

Figure 5: Parallel efficiency vs Threads for 1000×1000 square matrices

Although the computation time is decreased based on Figure 4, from Figure 5 we can deduce that there might be a possibility to further optimize the code to increase parallel efficiency.

3 Exercise 3

A typical problem solved using numerical methods is the problem of integration. There are different ways to implement numerical integration and one of the most known ones is the **Simpson's rule**. More specifically, we will investigate the multiple application of Simpson's 1/3 Rule, which can be used to compute the interval of a given function $f(x)$ at a specific interval a, b. If we have n data points of x and $f(x)$ inside $[a, b]$, there are $n-1$ segments in the given interval. The integral can be approximated using the formula:

$$I \cong (b-a)\frac{f(x_0) + 4\sum_{i=1,3,5,\ldots}^{n-1} f(x_i) + 2\sum_{j=2,4,6,\ldots}^{n-2} f(x_j) + f(x_n)}{3n} \tag{5}$$

Let us use this formula to determine the value of $ln(2)$. We can use the function:

$$f(x) = \frac{1}{1+x} \tag{6}$$

and we know that integrating this function from 0 to 1 returns $ln(2)$.

$$\int_0^1 \frac{1}{1+x} dx = ln(2) \tag{7}$$

We will split the interval $[0, 1]$ in $n = 100000001$ data points, resulting in 100000000 segments to be computed. The code implementing the solution can be found in file **Ex3_retselis_simpsonrule.c**. The results of running the code in multiple threads can be found in Table 3 and Figure 6.

Table 3: Results for $N = 100000000$ intervals

Threads	Time (sec)	Speedup
1	0.956	1
2	0.556	1.719
4	0.434	2.203
8	0.415	2.304

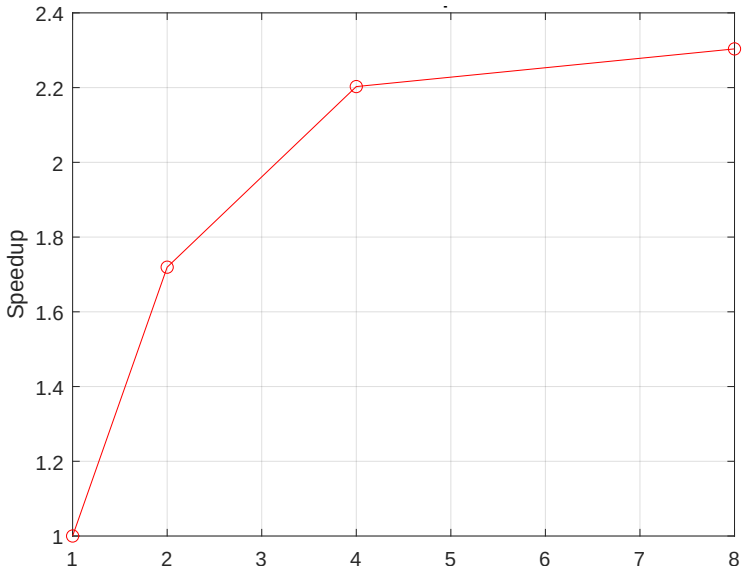

Figure 6: Speedup vs Threads for $N = 100000000$ intervals

This is an example of a use case. Note that this number is probably an overkill for this integral but it serves only as an example to demonstrate the speedup from having parallel code. For different integrals, with a larger interval this number might be adequate.