Fully-Explicit 2-Dimension Damped Wave Equation Simulator – Parallel Computing by OpenMP

Graduate Student Project

Yuxuan Jing

1 Introduction

This report is for graduate student final project of the course High Performance Computing. In this program, I implement a fully-explicit 2D damped wave equation simulator, and use OpenMP to achieve parallel computing. Two different OpenMP strategies are used. Two different cases are ran:

- test and compare the different OpenMP strategies and their performance.
- test the scalability of this program.

Finally, some conclusions are drawn.

2 Background

2.1 Wave Equation

The wave equation is an second-order linear partial differential equation for the description of waves. It is a significant equation in fields like acoustics, electromagnetics, and fluid dynamics.

$$\frac{\partial^2 u}{\partial^2 t} = c^2 \frac{\partial^2 u}{\partial^2 x} \tag{1}$$

2.2 Damped Wave Equation

A damped wave is a wave whose amplitude of oscillation decreases with time, eventually going to zero, an exponentially decaying sinusoidal wave. Damped waves are commonly seen in science and engineering, wherever a harmonic oscillator is losing energy faster than it is being supplied.

$$\frac{\partial^2 u}{\partial^2 t} - k\frac{\partial u}{\partial t} = c^2 \frac{\partial^2 u}{\partial^2 x} \tag{2}$$

3 Methods

3.1 Numerical Methods

Finite-difference methods (FDM) are discretization method used for solving differential equations. FDM approximate them with difference equations that finite differences approximate the derivatives. FDMs convert a linear ordinary differential equations or non-linear partial differential equations into a system of equations that can be solved by numerical mathematical technologies.

3.1.1 Temporal Discretization

The FDM discretization for the first order temporal differentiation is:

$$\frac{\partial u}{\partial t} = \frac{1}{\Delta t}(u^{t+1} - u^t) \tag{3}$$

The FDM discretization for the temporal Laplace operator is:

$$\frac{\partial^2 u}{\partial^2 t} = \frac{1}{\Delta t^2}(u^{t+1} - 2u^t + u^{t-1}) \tag{4}$$

3.1.2 Spatial Discretization

The FDM discretization for the spatial Laplace operator is:

$$\frac{\partial^2 u}{\partial^2 x} = \frac{1}{\Delta x^2}(u_{i+1,j} - 2u_{i,j} + u_{i-1,j}) + \frac{1}{\Delta y^2}(u^{i,j+1} - 2u^{i,j} + u^{i,j-1}) \tag{5}$$

3.1.3 Fully Explicit Method

Explicit and implicit methods are numerical approaching methods in simulation system, for the solution of time-depending partial differential equations. Explicit method calculate the state of the next time step form the state of the system at the current time step. Implicit method calculate the state of next time step from both the next time step and the current time step.

Assume $Y(t)$ is the current system state and $Y(t + \Delta t)$ is the state at the later time, where Δt is a small time step. Then, for an explicit method:

$$Y(t + \Delta t) = F(Y(t)), \tag{6}$$

while for an implicit method:

$$G\Big(Y(t), Y(t + \Delta t)\Big) = 0, \tag{7}$$

to find $Y(t+\Delta t)$. Apply fully-explicit Finite Different Method on damped wave

equation the equation become:

$$\frac{1}{\Delta t^2}(u_{i,j}^{t+1} - 2u_{i,j}^{t} + u_{i,j}^{t-1}) - \frac{k}{\Delta t}(u_{i,j}^{t+1} - u_{i,j}^{t}) = \frac{c^2}{\Delta x^2}(u_{i+1,j}^{t} - 2u_{i,j}^{t} + u_{i-1,j}^{t}) + \frac{c^2}{\Delta y^2}(u_{i,j+1}^{t} - 2u_{i,j}^{t} + u_{i,j-1}^{t}). \tag{8}$$

Apply fully-implicit Finite Different Method on damped wave equation the equation become:

$$\frac{1}{\Delta t^2}(u_{i,j}^{t+1} - 2u_{i,j}^{t} + u_{i,j}^{t-1}) - \frac{k}{\Delta t}(u_{i,j}^{t+1} - u_{i,j}^{t}) = \frac{c^2}{\Delta x^2}(u_{i+1,j}^{t+1} - 2u_{i,j}^{t+1} + u_{i-1,j}^{t+1}) + \frac{c^2}{\Delta y^2}(u_{i,j+1}^{t+1} - 2u_{i,j}^{t+1} + u_{i,j-1}^{t+1}). \tag{9}$$

Compared with implicit method, explicit is much more simple to achieve parallel computing. However for damped wave equation, there are two criteria that we need to obey, they are:

- Courant–Friedrichs–Lewy condition

$$k\frac{\Delta u}{\Delta x} < C_{max}. \tag{10}$$

- Von Neumann stability criterion

$$C^2\frac{\Delta u^2}{\Delta x^2} < C_{max}. \tag{11}$$

Typically $C_{max} = 1$, and in the following program, I set C_{max} as $\frac{1}{4}$.

3.2 Parallel computing methods

I used OpenMP to implement the parallel computing of this program. Two directives are using for the main for loop

- # pragma omp parallel for num_threads(thread_count) shared(x_prev, x, t, x_next)
- # pragma omp parallel num_threads(thread_count) default(none) private(my_rank, local_start, local_end) shared(x_prev, x, t, x_next, thread_count)

The first one using OpenMP's default for loop distribution and the second one we manually distribute the for loop elements.

4 Result

Two different cases are ran:

- test and compare the different OpenMP strategies and their performance.
- test the scalability of this program.

4.1 Case 1

This case test and compare the different OpenMP strategies and their performance. In this case we use an 4 node dual-core system. The problem size is 100×100 mesh with 20000 time steps. From figure 1 we can see that, the

N_processor	1	2	3	4	5	6	7	8
Default distribution	2.78	1.60	1.26	1.12	1.38	1.17	1.15	1.04
Manual distribution	2.8	1.52	1.22	1.09	1.27	1.08	1.01	1.04

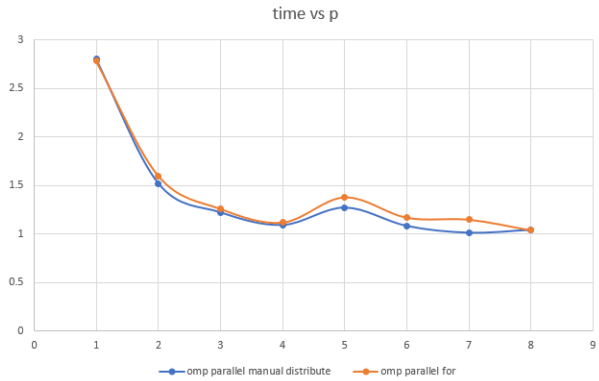

Figure 1: Default loop distribution vs Manual loop distribution.

run-time tendency of both methods is almost the same. The run-time of default method is slightly larger than the run-time of manual method. The reason should be that the manual method can minimize the cache miss rate.

4.2 Case 2

This case test the scalability of the program. I ran the code on Hammer platform and the results are as follows. In this case I use an 12 node quad-core system. The problem size is 200×200 mesh with 40000 time steps. From figure 3 we can see that the total computing time ($p \times$ time) and number of thread have linear relation. $p \times t = A \times p + B$. This means that we can keep the efficiency fixed by increasing the problem size at the same rate as we increase the number of processes/threads. This program is weakly scalable.

N_processor	1	2	3	4	8
time[s]	7.53	4.44	3.33	2.82	2.15
p * time[s]	7.53	8.88	9.99	11.28	17.20
N_processor	12	16	24	32	64
time[s]	1.96	2.22	3.23	6.61	11.59
p * time[s]	23.52	35.52	77.72	211.52	741.76

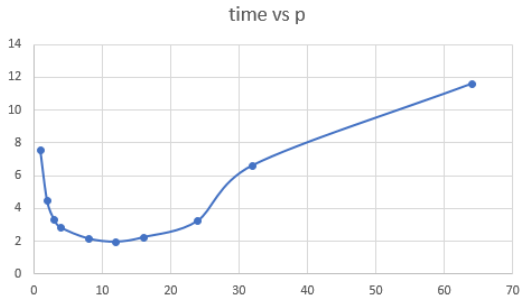

Figure 2: Run-time vs number of threads.

5 Conclusion

- Using fully explicit numerical scheme on 2D Cartesian mesh for damped wave equation is feasible.
- Courant–Friedrichs–Lewy condition and Von Neumann stability condition are both necessary conditions for the stability of this numerical scheme.
- For parallel OpenMP for loop, manually distributing the computation is more effective then the default distribution.
- This program is weakly scalable.

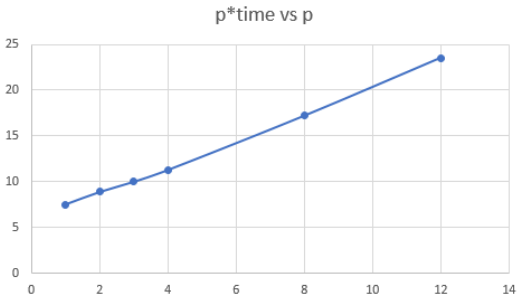

Figure 3: Total computing run-time vs number of threads.