2D wave solver in OpenMP

The program solves the following 2D wave equation within a parallel programming enviroment (OpenMP), in the 2+1 $\Omega \times (0,T] \equiv (0,L_x) \times (0,L_y) \times (0,T]$ spacetime grid, using the finite difference method with $N_x = N_y = 40$ mesh points,

$$u_{tt} - (u_{xx} + u_{yy}) = f(x,y,t), \quad (x,y) \in \Omega,\ t \in (0,T] \tag{1}$$

with the following initial conditions,

$$u(x,y,0) = I(x,y) = x(L_x - x)y(L_y - y), \quad (x,y) \in \Omega \tag{2}$$

$$u_t(x,y,0) = V(x,y) = \frac{1}{2}x(L_x - x)y(L_y - y), \quad (x,y) \in \Omega \tag{3}$$

and boundary condition,

$$u(x,y,t) = 0, \quad (x,y) \in \partial\Omega,\ t \in (0,T] \tag{4}$$

where

$$f(x,y,t) = 2\left(1 + \frac{1}{2}t\right)\left[y(L_y - y) + x(L_x - x)\right] \tag{5}$$

$$L_x = L_y = 10, \quad T = 20 \tag{6}$$

The 2D wave equation is a hyperbolic partial differential equation, for which we use the finite difference method as mentioned above. First we choose an appropriate time step Δt such that $\Delta t < h = \Delta x = \Delta y$. In our case we set $N_t = 250$. Rewriting the original equation (1) in a finite difference form we take the following recursive equation,

$$u_{i,j}^{n+1} = 2u_{i,j}^{n}(1 - 2g) - u_{i,j}^{n-1} + g(u_{i+1,j}^{n} + u_{i,j+1}^{n} + u_{i-1,j}^{n} + u_{i,j-1}^{n}) + \Delta t^2 f_{i,j}^{n} \tag{7}$$

where i, j, n refer to x,y and t iterations respectively and $g = \left(\frac{\Delta t}{h}\right)^2$.

The next step is to write the second initial condition in a finite difference form ($n = 0$),

$$u_{i,j}^{-1} = u_{i,j}^{1} - 2\Delta t V_{i,j}^{0} \tag{8}$$

Setting $n = 0$ in the wave equation and applying the boundary condition we take,

$$u^1_{i,j} = 2u^0_{i,j}(1-2g) + \Delta t V^0_{i,j} - u^1_{i,j} + \frac{1}{2}g(u^0_{i+1,j} + u^0_{i,j+1} + u^0_{i-1,j} + u^0_{i,j-1}) + \frac{1}{2}\Delta t^2 f^0_{i,j} \tag{9}$$

Throughout the code, solutions at n+1,n and n-1 time points are stored in three different arrays: u, u_1 and u_2 respectively.

The solution algorithm goes as follows. First we set the initial condition $u^0_{i,j} = I_{i,j}$. Next we compute $u^1_{i,j}$ using the above equation (9), and apply the boundary condition. Now we switch the array variables before proceeding to the next time iteration, as follows

$$u^{n-2}_{i,j} \leftarrow u^{n-1}_{i,j} \tag{10}$$

$$u^{n-1}_{i,j} \leftarrow u^{n}_{i,j} \tag{11}$$

Following we compute $u^{n+1}_{i,j}$ recursively for $n = 1, 2, ..., N_t$. Between each time step, we apply the boundary condition and switch the array variables as shown above.

The numerical solution is compared to the analytical solution,

$$u(x, y, t) = x(L_x - x)y(L_y - y)\left(1 + \frac{1}{2}t\right) \tag{12}$$

Since we are making use of parallel programming, the above procedure is executed repeatedly using a different number of threads (1, 2, 4, 8). The goal is to compute the parallel efficiency and parallel speedup for each execution and compare the results.

$$parallel\ speedup\ (n) = \frac{time\ elapsed\ with\ 1\ thread}{time\ elapsed\ with\ n\ threads}$$

$$parallel\ efficiency\ (n) = \frac{parallel\ speedup}{n}$$

Time elapsed, parallel speedup, parallel efficiency and the last (t = 20) central value for each thread number case, are printed as output.