Solving 2D wave equation on a parallel computer

This is the first mandatory assignment of INF3380. Each student should work independently and write a very short report, to be submitted together with the source codes and, preferably, some representative plots of the numerical solution.

Note: We've corrected quite a number of typos from the corresponding version of Spring 2010!

1 The 2D wave equation

The following 2D wave equation is to be solved on a parallel computer:

$$\frac{\partial^2 u}{\partial t^2} = \frac{1}{2}\left(\frac{\partial^2 u}{\partial x^2} + \frac{\partial^2 u}{\partial y^2}\right), \tag{1}$$

for which $u(x, y, t)$ is the unknown solution.

2 The numerical method

We adopt the unit square $(x, y) \in (0, 1)\times(0, 1)$ as the spatial solution domain, and introduce $M+2$ equally-spaced mesh points in the x-direction and $N+2$ points in the y-direction. The spatial mesh spacing is therefore $\Delta x = \frac{1}{M+1}$ and $\Delta y = \frac{1}{N+1}$. In the temporal direction, we divide the time domain $0 < t \leq T$ evenly into L time steps, such that $\Delta t = \frac{T}{L}$.

In our numerical method, superscript ℓ will be used to denote a time level and subscripts i, j refer to a spatial mesh point. That is, $u^{\ell}_{i,j}$ denotes the numerical approximation of $u(i\Delta x, j\Delta y, \ell\Delta t)$.

Like for the 1D wave equation, which was discussed in detail in the lecture slides of week 7, we will also use central finite differences. The numerical algorithm for solving Equation (1) thus reads as follows:

$$\begin{aligned} u_{i,j}^{\ell+1} &= 2u_{i,j}^{\ell} - u_{i,j}^{\ell-1} \\ &\quad + \frac{\Delta t^2}{2\Delta x^2}\left(u_{i-1,j}^{\ell} - 2u_{i,j}^{\ell} + u_{i+1,j}^{\ell}\right) \\ &\quad + \frac{\Delta t^2}{2\Delta y^2}\left(u_{i,j-1}^{\ell} - 2u_{i,j}^{\ell} + u_{i,j+1}^{\ell}\right) \quad 1 \le i \le M,\ 1 \le j \le N. \end{aligned} \tag{2}$$

The above formula is the main computational work to compute $u_{i,j}^2, u_{i,j}^3, \ldots, u_{i,j}^L$ on the interior points. On the boundary points, i.e., for $i = 0$, $i = M + 1$, $j = 0$ and $j = N + 1$, the value of u^ℓ is given as

$$\sin(2\pi x + 2\pi y)\cos(2\pi t) \tag{3}$$

on the physical boundary at all time levels.

The initial shape of u is prescribed as

$$u_{i,j}^0 = \sin(2\pi x + 2\pi y) \quad 0 \le i \le M + 1,\ 0 \le j \le N + 1. \tag{4}$$

In addition, we need a formula for computing $u_{i,j}^1$ so that the numerical algorithm (2) can start iterating. The following formula can be used:

$$\begin{aligned} u_{i,j}^1 = u_{i,j}^0 &+ \frac{\Delta t^2}{4\Delta x^2}\left(u_{i-1,j}^0 - 2u_{i,j}^0 + u_{i+1,j}^0\right) \\ &+ \frac{\Delta t^2}{4\Delta y^2}\left(u_{i,j-1}^0 - 2u_{i,j}^0 + u_{i,j+1}^0\right) \quad 1 \le i \le M,\ 1 \le j \le N. \end{aligned} \tag{5}$$

3 Serial implementation

The following code skeleton can be used:

```
/* Choose values of M, N, T, L, and calculate dx,dy,dt */
/* ... */

/* Allocation of three 3D arrays up, u, um*/
/* ... */

/* Enforcing initial condition 1 */
```

```
for (j=0; j<=N+1; j++) {
  y = j*dy;
  for (i=0; i<=M+1; i++) {
    x = i*dx;
    u[j][i] = sin(2*M_PI*(x+y));
  }
}

/* Enforcing initial condition 2 (only needed for the interior points) */
for (j=1; j<=N; j++)
  for (i=1; i<=M; i++)   /* interior points */
    um[j][i] = u[j][i]+((dt*dt)/(4*dx*dx))*(u[j][i-1]-2*u[j][i]+u[j][i+1])
                      +((dt*dt)/(4*dy*dy))*(u[j-1][i]-2*u[j][i]+u[j+1][i]);

/* main time loop */
t = 0;
while (t<T) {
  t += dt;

  for (j=1; j<=N; j++)
    for (i=1; i<=M; i++)   /* interior points */
      up[j][i] = 2*u[j][i]-um[j][i]
                +((dt*dt)/(2*dx*dx))*(u[j][i-1]-2*u[j][i]+u[j][i+1])
                +((dt*dt)/(2*dy*dy))*(u[j-1][i]-2*u[j][i]+u[j+1][i]);

  /* Compute up[0][i], up[N+1][i], up[j][0], up[j][M+1] using
     boundary conditions: sin(2*M_PI*(x+y))*cos(2*M_PI*t) */
  /* ... */

  /* data shuffle */
  tmp = um;
  um = u;
  u = up;
  up = tmp;
}

/* Deallocatiion of data arrays */
/* ... */
```

As *Task 1* of the mandatory assignment, each student should program the serial 2D wave solver by adding parts that are missing in the above code skeleton. Note that the three 2D arrays `up`, `u` and `um` are meant to store values of $u^{\ell+1}$, u^{ℓ} and $u^{\ell-1}$, respectively. Each 2D array has dimension $(M+2) \times (N+2)$.

To test whether the serial implementation works, you can choose $T = 1$ and $M = N = L - 1$, so that $\Delta x = \Delta y = \Delta t$. The numerical solution can be stored into data files at selected time levels, for the purpose of later analysis and visualization. As a check, the numerical solution should closely resemble the true solution $u(x, y, t) = \sin(2\pi x + 2\pi y)\cos(2\pi t)$.

4 Parallel implementation

4.1 Work division

For any given number of MPI processes P, it is assumed that P is the product of two integers Q and R. Accordingly, all the interior mesh points, i.e., $1 \leq i \leq M$ and $1 \leq j \leq N$, are divided into Q parts in the x-direction and R parts in the y-direction.

As *Task 2* of the mandatory assignment, a balanced work division scheme should be devised, such that the division of the $M \times N$ interior points is as even as possible among the $P = Q \times R$ processes. (Note that M may not be divisible by Q, likewise N may not be divisible by R.)

4.2 MPI implementation using blocking sends/receives

As *Task 3* of the mandatory assignment, an MPI implementation of the 2D wave equation solver should be made. We note that each process should add one layer of "ghost points" around its assigned rectangular region of interior points. This is for either enforcing the physical boundary condition or facilitating data exchanges between nearest neighbors.

The needed data exchanges between nearest neighbors can either be implemented using pairs of `MPI_Send` and `MPI_Recv` or be implemented using `MPI_Sendrecv`. (Care should be given to the sequence of the `MPI_Send` and `MPI_Recv` calls for avoiding possible deadlocks.)

Run the parallel code using different values of P, and if possible different combinations of Q and R for each P. Measure the time usage and make comparison with the serial implementation to form a table of speedup values.

Before doing detailed time measurements, it is of course important to check that the parallel implementation produces same numerical results as the serial implementation, independent of the P value. Visualization can be a handy tool. There are two options for data storage. We can either let a master process collect and merge all the small pieces before writing to one single large data file, or let all processes directly write their assigned pieces to a set of small data files. For the second option, a post-processing code is needed to "sew" the small pieces together later.

4.3 MPI implementation using non-blocking sends/receives

As *Task 4* (optional) of the mandatory assignment, the MPI implementation can be modified so that non-blocking `MPI_Isend` and `MPI_Irecv` commands are used instead. The purpose is to allow communication to happen at the same time while computation is carried out. Redo the time measurements.