Implementing a numerical 2D wave equation solver

IN3200/IN4200 obligatory assignment 1, Spring 2023

Note: This is one of the two obligatory assignments that both need to be approved before a student is allowed to take the final exam. (The grade of the final exam is otherwise *not* related to these two assignments.)

Note: Discussions between the students are allowed, but each student should write her/his own implementations. The details about how to submit the assignment can be found at the end of this document.

1 Objectives

This obligatory assignment has the following objectives:

- The student is given a hands-on exercise of implementing a prescribed numerical recipe. (The mathematical details behind the numerical algorithm are not important for this course.)
- The student is asked to implement the needed C functions by strictly following a pre-defined syntax of each function. (It is an important step in writing modular scientific software.)
- The student is asked to learn about dividing an entire code into multiple program files and about compiling a multi-file code.
- The student is given a hands-on exercise of eliminating if-tests inside loops.
- The student is given a hands-on exercise of simple OpenMP parallelization of loops.
- The student is given a hands-on exercise of measuring the time usage of some parts of a code.

2 The numerical recipe

The following numerical recipe can be used to compute the numerical solution of a simple 2D wave equation. (The mathematical and numerical derivations are not important for this mandatory assignment, thus not included.) It suffices to say that the wave equation is valid over a time interval $0 \leq t \leq T$ and inside a 2D spatial domain in the form of a unit square $(x, y) \in [0, 1] \times [0, 1]$.

2.1 Discrete time and spatial points

In contrast to a true solution $u_{\text{true}}(x, y, t)$ that is valid within the entire time-space domain, a numerical solution can typically only be sought at discrete time and spatial points. Specifically, the numerical recipe divides the time interval $0 \leq t \leq T$ into equal-distanced time levels $t_0, t_1, t_2, \ldots, t_{n_t-1}$. That is, $t_0 = 0$, $t_{n_t-1} = T$ and in general $t_\ell = \ell \Delta t$ with $\Delta t = T/(n_t - 1)$. Moreover, the 2D spatial domain is discretized by a uniform 2D grid consisting of $n_y \times n_x$ points, where there are n_y rows and n_x columns. Each grid point with index (i, j) has spatial coordinates $y_i = i\Delta y$ and $x_j = j\Delta x$, with $0 \leq i \leq n_y - 1$, $\Delta y = 1/(n_y - 1)$ and $0 \leq j \leq n_x - 1$, $\Delta x = 1/(n_x - 1)$.

2.2 Notation

As mentioned above, the numerical solution of the 2D wave equation will be sought at the discrete time and space points: t_ℓ, x_j, y_i, where $\ell = 0, 1, 2, \ldots, n_t - 1$, $j = 0, 1, 2, \ldots, n_x - 1$ and $i = 0, 1, 2, \ldots, n_y - 1$. A convention is to denote the numerical solution at these discrete points by $u^\ell_{i,j}$, that is,

$$u^\ell_{i,j} \approx u_{\text{true}}(x_j, y_i, t_\ell).$$

2.3 Numerical recipe

At time level $\ell = 0$. It is conventional that the true solution of the 2D wave equation is known at $t = 0$, so that the discrete values of $u^\ell_{i,j}$ at time level $\ell = 0$ are prescribed. In this mandatory assignment, we will use the following formula:

$$u^0_{i,j} := \cos(2\pi y_i)\cos(2\pi x_j), \quad \text{for } i = 0, 1, 2, \ldots, n_y - 1,\ j = 0, 1, 2, \ldots, n_x - 1. \tag{1}$$

At time level $\ell = 1$. To compute the numerical solution at time level $\ell = 1$, the following formula will be used:

$$u^1_{i,j} := u^0_{i,j} + \frac{\Delta t^2}{16\Delta x^2}\left(u^{0,\text{left}}_{i,j} - 2u^0_{i,j} + u^{0,\text{right}}_{i,j}\right) + \frac{\Delta t^2}{16\Delta y^2}\left(u^{0,\text{below}}_{i,j} - 2u^0_{i,j} + u^{0,\text{above}}_{i,j}\right), \tag{2}$$
$$\text{for } i = 0, 1, 2, \ldots, n_y - 1,\ j = 0, 1, 2, \ldots, n_x - 1,$$

where the definition of $u^{0,\text{left}}_{i,j}$, $u^{0,\text{right}}_{i,j}$, $u^{0,\text{below}}_{i,j}$ and $u^{0,\text{above}}_{i,j}$ is as follows:

$$u^{0,\text{left}}_{i,j} = \begin{cases} u^0_{i,1} & \text{if } j = 0 \\ u^0_{i,j-1} & \text{if } j > 0 \end{cases} \qquad u^{0,\text{right}}_{i,j} = \begin{cases} u^0_{i,n_x-2} & \text{if } j = n_x - 1 \\ u^0_{i,j+1} & \text{if } j < n_x - 1 \end{cases} \tag{3}$$

$$u^{0,\text{below}}_{i,j} = \begin{cases} u^0_{1,j} & \text{if } i = 0 \\ u^0_{i-1,j} & \text{if } i > 0 \end{cases} \qquad u^{0,\text{above}}_{i,j} = \begin{cases} u^0_{n_y-2,j} & \text{if } i = n_y - 1 \\ u^0_{i+1,j} & \text{if } i < n_y - 1 \end{cases} \tag{4}$$

At subsequent time levels $\ell = 2, 3, \ldots, n_t - 1$. To compute the numerical solution on the subsequent time levels, the following formula will be used:

$$u^\ell_{i,j} := 2u^{\ell-1}_{i,j} + \frac{\Delta t^2}{8\Delta x^2}\left(u^{\ell-1,\text{left}}_{i,j} - 2u^{\ell-1}_{i,j} + u^{\ell-1,\text{right}}_{i,j}\right) + \frac{\Delta t^2}{8\Delta y^2}\left(u^{\ell-1,\text{below}}_{i,j} - 2u^{\ell-1}_{i,j} + u^{\ell-1,\text{above}}_{i,j}\right) - u^{\ell-2}_{i,j}, \tag{5}$$
$$\text{for } i = 0, 1, 2, \ldots, n_y - 1,\ j = 0, 1, 2, \ldots, n_x - 1,$$

where the definition of $u^{\ell-1,\text{left}}_{i,j}$, $u^{\ell-1,\text{right}}_{i,j}$, $u^{\ell-1,\text{below}}_{i,j}$ and $u^{\ell-1,\text{above}}_{i,j}$ is the same as formulas (3)-(4) except that all the superscripts 0 are replaced with $\ell - 1$.

Note that to compute on time level ℓ (when $\ell \geq 2$) we make use of the already computed results on the two preceding time levels $\ell - 1$ and $\ell - 2$. This repeats recursively until time level $n_t - 1$ (remember $t_{n_t-1} = T$).

Computation of numercial error. The 2D wave equation chosen for this mandatory assignment actually has an true solution as

$$u_{\text{true}}(x, y, t) = \cos(2\pi x)\cos(2\pi y)\cos(\pi t). \tag{6}$$

Thus, we can quantify the error of the numerical solution, at a certain time point $t = \ell\Delta t$, by the following fomula:

$$\sqrt{\Delta x \Delta y \sum_{i=0}^{n_y-1}\sum_{j=0}^{n_x-1}\left(u_{\text{true}}(x_j, y_i, t) - u^\ell_{i,j}\right)^2} \tag{7}$$

3 Some comments about the implementation

Although all the discrete values of the numerical solution lie on a logical 3D grid (the time direction plus the two space dimensions), it will be a waste of resources to actually allocate a 3D array to store all the numerical solution values. Instead, three 2D arrays (each of dimension $n_y \times n_x$) are sufficient.

As indicated by formula (5), to compute for time level ℓ (when $\ell \geq 2$) we only need two preceding time levels $\ell - 1$ and $\ell - 2$. Thus, it is sufficient in a computer program to only allocate three 2D arrays, for instance named as `u_new`, `u` and `u_prev`. The overall computation, covering time levels $\ell = 2, 3 \ldots, n_t - 1$, can be implemented as a for-loop or while-loop where each iteration computes for one time level. Here, it is important to remember to swap the three array pointers before the next iteration. (An implementation sketch will be provided to the student.)

For convenience, the number of spatial points in the x-direction can be chosen to be the same as that in the y-direction. (That is, $n_x = n_y$ and $\Delta x = \Delta y$.) The size of the time step, Δt, cannot be chosen too large (otherwise the numerical solution may "explode"). For the particular 2D wave equation chosen for this mandatory assignment, the best choice of Δt should equal $2\Delta x$.

4 The mandatory assignment

4.1 Serial programming

The student is required to implement six C functions. These functions should work with a pre-programmed main program, named `serial_main.c`, which is provided inside a downloadeble zip file (see later for details). Each of the six functions should be placed inside a separate file, using the same file name as the function name. The syntax of these functions should be as follows:

```
void first_time_step (int nx, int ny, double dx, double dy, double dt,
                      double **u, double **u_prev);

void one_regular_time_step (int nx, int ny, double dx, double dy, double dt,
                            double **u_new, double **u, double **u_prev);

void one_fast_time_step (int nx, int ny, double dx, double dy, double dt,
                         double **u_new, double **u, double **u_prev);

double compute_numerical_error (int nx, int ny, double dx, double dy,
                                double t_value, double **u);

void allocate_2D_array (double ***array_ptr, int nx, int ny);

void deallocate_2D_array (double **array);
```

Function `first_time_step`**.** It should implement formula (2) to compute for time level $\ell = 1$. Before invoking the function, the 2D array `u_prv` is supposed to already contain the initial values for time level $\ell = 0$. After the function returns, the 2D array `u` will contain the computed discrete numerical values for time level $\ell = 1$. If-tests can be used to faithfully translate formula (2).

Function `one_regular_time_step`**.** It should implement formula (5) to compute <u>one</u> time level with $\ell \geq 2$. Before invoking the function, the 2D arrays `u` and `u_prev` are supposed to already contain the discrete numerical solution values for the two preceding time levels. After the function returns, the 2D array `u_new` will contain the computed discrete numerical values. If-tests should be used to faithfully implement formula (5) .

Function `one_fast_time_step`**.** It should compute the same numerical values as `one_fast_time_step`, but all the if-tests should be avoided.

Function `compute_numerical_error`**.** It should compute the numerical error using formula (7) while assuming the true solution is described by (6). The computed error value is returned by the function.

Function `allocate_2D_array`**.** It should allocate a 2D array of dimension $n_y \times n_x$. All the entries of the allocated 2D array should lie contiguously in memory with row-major storage. This function is supposed to be called as follows:

```
// nx and ny are already assigned with some positive values
double **array;
allocate_2D_array (&array, nx, ny);
// now, the 2D array is allocated with ny rows each with nx entries
```

Hint: A corresponding function that allocates a 1D array can be programmed as follows:

```
void allocate_1D_array (double **array_ptr, int nx)
{
  double *tmp_array = (double*)malloc(nx*sizeof(double));
  *array_ptr = tmp_array;
}
```

Function `deallocated_2D_array`**.** It should deallocate a 2D array which was allocated by function `allocate_2D_array`.

4.2 OpenMP parallelization

The student is required to implement four OpenMP parallelized functions, whose syntax is specified as follows:

```
void omp_first_time_step (int nx, int ny, double dx, double dy, double dt,
      double **u, double **u_prev);

void omp_one_regular_time_step (int nx, int ny, double dx, double dy, double dt,
    double **u_new, double **u, double **u_prev);

void omp_one_fast_time_step (int nx, int ny, double dx, double dy, double dt,
         double **u_new, double **u, double **u_prev);

double omp_compute_numerical_error (int nx, int ny, double dx, double dy,
        double t_value, double **u);
```

As the function names suggest, these should be the OpenMP parallelized counterparts of the corresponding serial functons. Each OpenMP parallelized function should also be placed in a separate file, with the same file name as the function name.

Moreover, the student is required to program an OpenMP parallelized counterpart of `serial_main.c`, named `para_main.c`. The purpose is to call the above four parallelized functions to carry out the same numerical computation, but hopefully at a faster speed due to using multiple threads. Appropriate time measurements should also be implemented inside `para_main.c`.

4.3 Default file folder

There is a pre-prepared zip file that each student can download from

https://www.uio.no/studier/emner/matnat/ifi/IN3200/v23/teaching-material/oblig1_default_folder.zip

Upon unpacking this zip file, you will see a default file folder with the following content:

```
oblig1_src/serial_part/:
allocate_2D_array.c functions.h
compute_numerical_error.c one_fast_time_step.c
deallocate_2D_array.c one_regular_time_step.c
first_time_step.c serial_main.c

oblig1_src/parallel_part/:
omp_compute_numerical_error.c omp_one_regular_time_step.c
omp_first_time_step.c para_functions.h
omp_one_fast_time_step.c para_main.c
```

Three of the files (`serial_main.c`, `functions.h` and `para_functions.h`) are already filled out beforehand. They can be used as they are. The content of the other files needs to be programmed by the student.

5 Submission

Each student is asked to first download the default file folder (see above), and program the serial and OpenMP-parallelized functions as required in separate files in the sub-folders `serial_part` and `parallel_part`. The pre-programmed `serial_main.c` should be used as is. Please insert in each sub-folder a `README.txt` or `README.md` file that describes how the serial or parallel compilation is done.

Before submission, please rename the main `oblig1_src` folder as `oblig1_src_xxx` where `xxx` is your UiO username. (**Note: Don't reveal your candidate number.**) Create a zip file or a tarball of the `oblig1_src_xxx` folder and submit it via Devilry.

The submission deadline is Monday April 3rd, 23:59.

Hint: In case you don't have a proper computer to develop and test your implementations, please use the the "Fox" system that is provided on EduCloud at UiO. (Please read the "Advice" webpage on the IN3200 semester website.)