INFO 0939 Project 2022

Last updated: 04/10/2022

Goals of the Project

- Solve partial differential equations using a finite difference scheme coded in C.
- Experiment with the stability of explicit time integration schemes.
- Parallelize the code on shared and distributed memory systems using OpenMP and MPI.
- Run scalability analyses.
- Learn about the physics of acoustic wave propagation through numerical simulations.

Project statement

The propagation of sound waves is governed by the following system of first order partial differential equations:

$$\frac{\partial P}{\partial t} = -\rho c^2 \operatorname{div} \mathbf{v}, \tag{1}$$

$$\frac{\partial \mathbf{v}}{\partial t} = -\frac{1}{\rho} \mathbf{grad}\, P, \tag{2}$$

where, in three dimensions and assuming a Cartesian coordinate system and MKS units, $P = P(x,y,z,t)$ is the scalar pressure field (in Pa) and $\mathbf{v} = \mathbf{v}(x,y,z,t)$ is the vector velocity (in m/s). Both the pressure and the velocity are function of the position (x,y,z) and of the time t. The material parameters are the speed of sound $c = c(x,y,z)$ (in m/s) and the density $\rho = \rho(x,y,z)$ (in kg/m^3), which can be function of the position but are assumed to be independent of time.

Expanded in terms of the three components v_x, v_y, v_z of the velocity $\mathbf{v}$, system (1)–(2) can be rewritten as

$$\frac{\partial P}{\partial t} = -\rho c^2 \left(\frac{\partial v_x}{\partial x} + \frac{\partial v_y}{\partial y} + \frac{\partial v_z}{\partial z} \right), \tag{3}$$

$$\frac{\partial v_x}{\partial t} = -\frac{1}{\rho} \frac{\partial P}{\partial x}, \tag{4}$$

$$\frac{\partial v_y}{\partial t} = -\frac{1}{\rho} \frac{\partial P}{\partial y}, \tag{5}$$

$$\frac{\partial v_z}{\partial t} = -\frac{1}{\rho} \frac{\partial P}{\partial z}. \tag{6}$$

You are asked to solve this system of partial differential equations using the Finite Difference Time Domain (FDTD) method (see e.g. [1]). The FDTD method is based on a discretization of the pressure and the components of the velocity in both space and time, on regularly spaced grid nodes. A pressure node is surrounded by velocity components such that the components are oriented along the line joining the component and the pressure node. The distance (offset) between pressure nodes along the

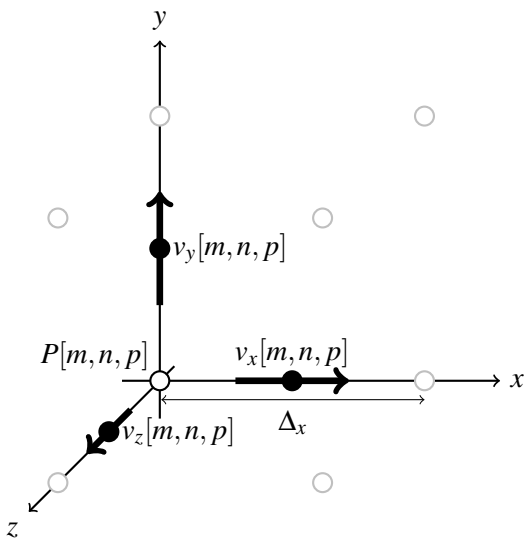

Figure 1: Spatial discretization in three dimensions: arrangement of velocity nodes relative to the pressure node with the same spatial indices $[m,n,p]$ (the time index q is omitted for clarity).

x, y and z directions is supposed to be constant, and equal respectively to Δ_x, Δ_y and Δ_z. In addition to these spatial offsets, the pressure nodes are assumed to be offset by half of a temporal step Δ_t from the velocity nodes. Given four integer indices m, n, p and q, we introduce the following notations for the representation of the discretized pressure and velocity components (see Figure 1):

$$P^q[m,n,p] := P(m\Delta_x, n\Delta_y, p\Delta_z, q\Delta_t), \tag{7}$$

$$v_x^{q+1/2}[m,n,p] := v_x((m+1/2)\Delta_x, n\Delta_y, p\Delta_z, (q+1/2)\Delta_t), \tag{8}$$

$$v_y^{q+1/2}[m,n,p] := v_y(m\Delta_x, (n+1/2)\Delta_y, p\Delta_z, (q+1/2)\Delta_t), \tag{9}$$

$$v_z^{q+1/2}[m,n,p] := v_z(m\Delta_x, n\Delta_y, (p+1/2)\Delta_z, (q+1/2)\Delta_t). \tag{10}$$

For this project it will be assumed that the spatial grid offsets are equal in all directions, i.e. $\Delta_x = \Delta_y = \Delta_z = \delta$. Replacing the derivatives in (3) with finite differences and using the discretization (7)–(10) yields the following update equation for the pressure:

$$\begin{aligned} P^q[m,n,p] = P^{q-1}[m,n,p] - \rho c^2 \frac{\Delta_t}{\delta} \Big(& v_x^{q-1/2}[m,n,p] - v_x^{q-1/2}[m-1,n,p] + \\ & v_y^{q-1/2}[m,n,p] - v_y^{q-1/2}[m,n-1,p] + \\ & v_z^{q-1/2}[m,n,p] - v_z^{q-1/2}[m,n,p-1] \Big). \end{aligned} \tag{11}$$

The update equation for the x component of the velocity is obtained from the discretized version of (4), which yields

$$v_x^{q+1/2}[m,n,p] = v_x^{q-1/2}[m,n,p] - \frac{1}{\rho}\frac{\Delta_t}{\delta}\left(P^q[m+1,n,p] - P^q[m,n,p]\right). \tag{12}$$

The update equations for the other two components are derived in a similar way.

For this project a homogeneous initial condition is always assumed for both the pressure and the velocity, i.e.

$$P^0[m,n,p] = v_x^{1/2}[m,n,p] = v_y^{1/2}[m,n,p] = v_z^{1/2}[m,n,p] = 0, \quad \forall m,n,p, \tag{13}$$

and a Dirichlet condition is specified on selected pressure nodes to impose a sound source:

$$P^q[m,n,p] = P^q_{\text{source}}[m,n,p], \quad m,n,p \in S, \tag{14}$$

where S denotes the subset of nodes where $P^q_{\text{source}}[m,n,p]$ is imposed. For example, if there are N_x, N_y and N_z nodes respectively along x, y and z, a point-wise sinusoidal source at frequency f can be imposed in the middle of the domain by choosing

$$P^q_{\text{source}}[\lfloor N_x/2 \rfloor, \lfloor N_y/2 \rfloor, \lfloor N_z/2 \rfloor] = \sin(2\pi f q \Delta_t). \tag{15}$$

On the boundary of the domain, one assumes homogeneous Neumann boundary conditions for the pressure (where any node "outside" the domain by one offset is given the same value as the one associated to the closest node on the boundary, e.g. $P^q[-1,n,p] := P^q[0,n,p]$ or $P^q[m,N_y,p] := P^q[m,N_y-1,p]$) and zero normal velocity. Finally, note that a two-dimensional problem in the $z=0$ plane can be solved simply by setting $p=0$ and $v_z^{q+1/2}[m,n,p] = 0, \forall m,n,p,q$, and by ignoring the update equation for v_z.

Instructions

By group of **two students** you are asked to:

1. Write the remaining FDTD update equations for v_y and v_z.
2. Write a sequential C program that solves the discretized equations **in two dimensions** over the unit square ($1m \times 1m$) with constant material parameters $c = 340$ m/s and $\rho = 1.225$ kg/m^3, and with the pressure imposed as a sinusoidal source at frequency $3400Hz$ at a single node in the middle of the grid. Validate the solution and study the effect of the spatial grid size δ and the time step Δ_t. In particular, study the stability of the scheme depending on the ratio $c\Delta_t/\delta$.
3. Generalize the previous code so that it
   - reads an input ASCII parameter file (see "ASCII parameter file format" below) that defines the main parameters of the simulation. The name of this parameter file should be the only argument taken by your program when executed from the command line;
   - reads binary input map files (see "Binary data file format" below) describing the spatial profile of the speed of sound and density. The corresponding file names will be specified in the input parameter file;
   - creates binary output files for the pressure and for each component of the velocity for selected time steps, using the same format as for the binary input map files (the base names for these files will again be specified in the input parameter file).

   The resulting program is what is expected for **deadline 1**.

4. Do a rough analysis of the arithmetic intensity of the FDTD and determine which is the main limiting factor for the speed of your algorithm. What do you expect if you run the FDTD on a machine capable of 250 GB/s of memory bandwidth and 1.3 TFLOPS/s of processing power?

5. Parallelize the code for shared memory architectures using OpenMP. This should be done for **deadline 2**.

6. Parallelize the code for distributed memory architectures using MPI. You are completely free to imagine how to split the computation for the parallel implementation. Several approaches are possible: prefer simple solutions and document your choices. The resulting code is expected for **deadline 3**.

7. Perform a scalability analysis of your parallel code, by evaluating both the strong and weak scaling.

8. Imagine a few combinations of input parameters, speed and density maps and sources to study the propagation of sound in interesting configurations.

Here are some optional enhancements that you can work on if you would like to go further:

1. (Difficulty level: easy) Extend your code to the three-dimensional case;

2. (Difficulty level: moderate) Add a new source type where the pressure is computed from an audio file (you can use e.g. `ffmpeg` to extract the raw waveform), extract the computed sound at one or more interesting location(s) and reconstruct the audio file from the simulated data;

3. (Difficulty level: moderate to difficult, depending on the choice of condition) Implement more sophisticated boundary conditions on the boundary of the domain to simulate a transparent boundary, allowing waves to exit the simulation domain without reflection.

ASCII parameter file format

The input parameter file is an ASCII text file that contains the simulation parameters, structured as follows:

```
delta
delta_t
max_t
sampling_rate
source_type
input_speed_filename
input_density_filename
output_pressure_base_filename
output_velocity_x_base_filename
output_velocity_y_base_filename
output_velocuty_z_base_filename
```

with

- `delta` (double): spatial grid offset δ;
- `delta_t` (double): time step Δ_t;
- `max_t` (double): maximum simulation time; the maximum number of time steps is $\lfloor$`max_t`$/\Delta_t \rfloor$;
- `sampling_rate` (int): sampling rate at which output files should be created (0: never save, 1: save all steps, 2: save 1 step out of 2, etc.);
- `source_type` (string): type of source; the value `point_source_middle_3400` should correspond to the 3400 Hz sine wave excitation on a single node in the middle of the domain, as described above. Add additional source types as necessary for your experiments: for example, the value `double_source_ping` could correspond to two sources with a short "ping" sound, and `myfile.wav` could correspond to an audio file.
- `input_speed_filename` (string): name of the input file containing the sound speed profile (the file format is described in "Binary data file format" below);
- `input_density_filename` (string): same as `input_speed_filename` but for the density profile;
- `output_pressure_base_filename` (string): base name of the output file for the pressure; the full file name consists of the base name to which the time step q is appended (the file format is described in "Binary data file format" below);
- `output_velocity_x_base_filename` (string): same as `output_pressure_base_filename`, but for the x component of the velocity;
- `output_velocity_y_base_filename` (string): same as `output_pressure_base_filename`, but for the y component of the velocity;
- `output_velocity_z_base_filename` (string): same as `output_pressure_base_filename`, but for the z component of the velocity.

Sample parameter files will be provided in class. They can be written and read using the `fprintf()` and `fscanf()` functions from the standard C library.

Binary data file format

The input and output binary files are structured as follows:

```
nx ny nz xmin xmax ymin ymax zmin zmax values...
```

with

- `nx` (int): number of nodes along x;
- `ny` (int): number of nodes along y;
- `nz` (int): number of nodes along z;

- `xmin` (double): minimum x coordinate of the domain;
- `xmax` (double): maximum x coordinate of the domain;
- `ymin` (double): minimum y coordinate of the domain;
- `ymax` (double): maximum y coordinate of the domain;
- `zmin` (double): minimum z coordinate of the domain;
- `zmax` (double): maximum z coordinate of the domain;
- `values` (`nx` $\times$ `ny` $\times$ `nz` doubles). The values are assumed to be given "by row", i.e.

```
for(int p = 0; p < nz; p++)
  for(int n = 0; n < ny; n++)
    for(int m = 0; m < nx; m++)
      printf("value[%d][%d][%d] = %g\n", m, n, p,
              value[ny * nx * p + nx * n + m]);
    }
  }
}
```

Sample binary data files will be provided in class. They can be written and read using the `fwrite()` and `fread()` functions from the standard C library. You can use various packages to visualize these files: Matlab, Octave, Python, ...

Evaluation

You will need to submit your code, as well as input parameter and data files, for each of the **three intermediate deadlines**, together with the exact command line to compile and run the code on the NIC5 cluster. Detailed instructions will be given in class. The codes submitted for the intermediate deadlines will not be graded, but failure to submit your code will lead to a penalty on your final grade.

The **final version of your code will need to be submitted no later than one week before the oral exam**, which will take place during the January exam session. You will need to prepare a 10 minute presentation for the oral exam, with a summary of the main results of your project. A hands-on examination will follow, where you will be asked technical questions on your program, and you will be tasked with running it on a new input dataset.

General remarks on C coding style

Your coding style will be evaluated. As such, some quality in the submitted code is expected, and you should strive to have a clean and neat coding style. In general, think about who will read your code and ask yourself if it is clear and understandable. Please beware of some common mistakes that will lead to penalties on your final grade:

- Do all your computations in `double`. There is no reason to use single precision in this project.

- If you `malloc()` something, remember to `free()` it.
- Check return values for failure, especially from `malloc()`, `fopen()` or similar calls. Don't assume that those calls will always succeed.
- Don't use Variable Length Arrays, i.e.

```
int function(int N) {
    int array[N];
}
```

  The value of `N` must be known at compile time. If it is not the case, use `malloc()`.
- Don't abuse comments. If something is obvious, don't comment it. If you feel the need to comment some code, ask yourself if perhaps you can write the code in a clearer way instead of putting that comment.
- Try to use relevant and appropriate variable and function names. Be coherent in naming things, in order to make it easy to track down where data goes.
- Avoid global variables. There are very few valid use cases for global variables and they are one of the factors that make your code not thread safe.
- Finally, a function *should* fit on your screen. If it doesn't, perhaps you should split it in smaller pieces. Also, if you have a function that takes more than 10 parameters, perhaps you need to think if you really need all of them, or it might be time to create a `struct`?

References

[1] John B. Schneider, *Understanding the Finite-Difference Time-Domain Method*, 2010. Available online at: `http://www.eecs.wsu.edu/~schneidj/ufdtd/` .