TDT4200 Parallel programming

PS1

Maren Wessel-Berg & Claudi Lleyda Moltó

September 2023

Practical information

Published: 05/09/23

Deadline: 12/09/23 at 22:00

Evaluation: pass/fail

- Completing the problem set is **mandatory**.
- The work must be done **individually** and without help from anyone but the TDT4200 staff.
- **Reference** all sources found on the internet or elsewhere.
- The **requirements**, and **how and what to deliver** is explained in the problem set description found on BlackBoard.
- Start the exercises early!

Where can you get help with the assignment?

- **Recitation lecture**: introduction to the problem set

  (Today)
  Slides will be made available online.
- **TA hours**: ask questions in person

  Friday, September 8, 10:00–12:00 in Cybele
  Monday, September 11, 13:00–15:00 in Cybele
- **Piazza**: question forum

  Ask questions any time (but give us time to answer).
  Select the ps1 folder for questions related to this problem set.
  Do not post full or partial solutions!

Partial differential equations

- Equations where the unknowns are functions.
- We have some information on the derivatives of the unknowns.

Example

Let's solve the following PDE

$$\frac{\partial u}{\partial x} = \cos(x), \qquad \text{such that} \qquad u(0) = 0.$$

We can integrate with respect to x and get that, for some constant C,

$$u(x) = \sin(x) + C.$$

We use the starting condition to deduce $C = 0$, and we find

$$u(x) = \sin(x).$$

Temperature simulation

The heat equation

A harder PDE to solve is the heat equation, given by

$$\frac{\partial u}{\partial t} = K \frac{\partial^2 u}{\partial x^2}.$$

where $K > 0$ is the thermal diffusivity of the medium.

- Intuitively, the term $\frac{\partial^2 u}{\partial x^2}$ corresponds to the average of the temperatures around a point x, minus the temperature at x.
- The derivative $\frac{\partial u}{\partial t}$ at a point x will be larger when the difference is larger, and smaller when the difference is smaller.
- This has a tendency to uniformize the temperature at every point with its neighbours.

Temperature simulation

- Finding a solution to the heat equation can be difficult.

Discretization

- We discretize the spatial domain into the samples $x_0, \ldots, x_N$, with a constant step size of Δx, and we approximate the derivative of u with respect space with the central difference approximation.
- We discretize the time domain into the samples $t_0, \ldots, t_K$, with a constant step size of Δt, and we approximate the derivative of u with respect to time with the one-sided backward difference approximation.

With these we obtain the following approximation

$$\frac{u_i^k - u_i^{k-1}}{\Delta t} = \frac{u_{i-1}^k - 2u_i^k + u_{i+1}^k}{\Delta x^2}.$$

Temperature simulation

From the obtained approximation

$$\frac{u_i^k - u_i^{k-1}}{\Delta t} = K \frac{u_{i-1}^k - 2u_i^k + u_{i+1}^k}{\Delta x^2}$$

we deduce

$$u_i^{k-1} = \left(1 + 2K\frac{\Delta t}{\Delta x^2}\right) u_i^k - K\frac{\Delta t}{\Delta x^2}\left(u_{i-1}^k + u_{i+1}^k\right).$$

We are able to express the temperature at an instant $k-1$ in terms of the temperatures at the instant k.

Watch out for edge cases

We must remember to account for the boundary conditions!

Temperature simulation

Try it yourself!

If instead of using the one-sided **backward** difference approximation we had used the one-sided **forward** difference approximation for the time derivative of u, then we could have solved for u_i^{k+1}, and we would have deduced the following expression

$$u_i^{k+1} = \left(1 - 2K\frac{\Delta t}{\Delta x^2}\right)u_i^k + K\frac{\Delta t}{\Delta x^2}\left(u_{i-1}^k + u_{i+1}^k\right),$$

The very similar expression we had just found, namely,

$$u_i^{k-1} = \left(1 + 2K\frac{\Delta t}{\Delta x^2}\right)u_i^k - K\frac{\Delta t}{\Delta x^2}\left(u_{i-1}^k + u_{i+1}^k\right),$$

leads to very different approaches as to solving the PDE.

Explicit temperature simulation

Finite difference method

Let us start with the equation we obtained from the one-sided forward difference approximation.

$$u_i^{k+1} = \left(1 - 2K\frac{\Delta t}{\Delta x^2}\right)u_i^k + K\frac{\Delta t}{\Delta x^2}\left(u_{i-1}^k + u_{i+1}^k\right)$$

We can use this recurrent expression to directly calculate approximations for each subsequent time-step, based on the values at the previous one.

Pros and cons

- While this method is easy to implement and runs efficiently, it can be numerically unstable.
- It is advisable to use smaller time-steps to reduce the instability, meaning that it will usually require more iterations in comparison to other approaches.

Implicit temperature simulation

Following now from the equation

$$u_i^{k-1} = \left(1 + 2K\frac{\Delta t}{\Delta x^2}\right)u_i^k - K\frac{\Delta t}{\Delta x^2}\left(u_{i-1}^k + u_{i+1}^k\right).$$

If we write

$$D = 1 + 2K\frac{\Delta t}{\Delta x^2} \qquad \text{and} \qquad U = L = -K\frac{\Delta t}{\Delta x^2}$$

we get

$$u_i^{k-1} = Lu_{i-1}^k + Du_i^k + Uu_{i+1}^k.$$

This suggests a linear system $A\vec{u}^k = \vec{u}^{k-1}$ where A is a tridiagonal matrix.

Boundary conditions

If we agree to use the Neumann boundary condition we get

$$u_0^{k-1} = Du_0^k + 2Uu_1^k \qquad \text{and} \qquad u_N^{k-1} = 2Lu_{N-1}^k + Du_N^k.$$

Implicit temperature simulation

Tridiagonal matrix

We can picture the linear system we have obtained, namely,

$$u_i^{k-1} = Lu_{i-1}^k + Du_i^k + Uu_{i+1}^k,$$

$$u_0^{k-1} = Du_0^k + 2Uu_1^k \qquad \text{and} \qquad u_N^{k-1} = 2Lu_{N-1}^k + Du_N^k,$$

in matrix form.
If we set $N = K = 4$ we get the example

$$\begin{pmatrix} D & 2U & 0 & 0 & 0 \\ L & D & U & 0 & 0 \\ 0 & L & D & U & 0 \\ 0 & 0 & L & D & U \\ 0 & 0 & 0 & 2L & D \end{pmatrix} \begin{pmatrix} u_0^k \\ u_1^k \\ u_2^k \\ u_3^k \\ u_4^k \end{pmatrix} = \begin{pmatrix} u_0^{k-1} \\ u_1^{k-1} \\ u_2^{k-1} \\ u_3^{k-1} \\ u_4^{k-1} \end{pmatrix}$$

Implicit temperature simulation

Linear is good :D

Reinterpreting the problem in terms of solving a linear system of equations opens the doors to many methods. In this exercise we will look at

- the Jacobi method,
- the Gauss–Seidel method,
- the Red/Black Gauss–Seidel method.

We can abstract away from most of our specific setting, the heat equation, and reason solely about linear systems of equations.

Pros and cons

Although these methods will generally be slower per iteration, as every iteration will now involve solving a system of equations; they will generally be more stable, and allow us to use larger time-steps, requiring less iterations overall.

The Jacobi method

Informal explanation

Let $A\vec{x} = \vec{b}$ be a linear system where A is strictly diagonally dominant.

- We can use a random vector $\vec{x}'$ as a guess for a solution to the system. Unsurprisingly, this solution will generally not be a very good one.
- If we use the guess to solve the system, that is, taking

$$x_i'' = \frac{b_i - \sum_{j \neq i} A_{i,j} x_j'}{A_{i,i}},$$

  then the new vector $\vec{x}''$ will be a better approximation of the solution than $\vec{x}'$.
- We can perform this operation iteratively, and we will eventually approach the correct answer.

The Jacobi method

Implementation details

- Remember

$$D = 1 + 2K\frac{\Delta t}{\Delta x^2} \qquad \text{and} \qquad U = L = -K\frac{\Delta t}{\Delta x^2},$$

  where K, Δx and Δt are all positive values. Therefore $D > 1$ and $U = L < 0$, and our system matrix is strictly diagonally dominant, and we can use the Jacobi method.
- Notice how every iteration of this method is highly parallelizable. Every value can be computed simultaneously.

The Gauss–Seidel method

Informal explanation

Let $A\vec{x} = \vec{b}$ be a linear system where A is strictly diagonally dominant again.

- This method is similar to the Jacobi method.
- When calculating the values, that is in,

$$x_i'' = \frac{b_i - \sum_{j \neq i} A_{i,j} x_j'}{A_{i,i}},$$

  we use the previously obtained values of x_i'' when calculating x_j'' for $j > i$.
- This introduces a data dependency in every iteration, breaking the high parallelizability of the Jacobi method. In turn, the method converges faster, as our improved guesses are propagated throughout the iteration.

The Gauss–Seidel method

Implementation details

- The Gauss–Seidel method has the advantage that we do not necessarily need to store an additional copy of the matrix, reducing the memory requirements.
- Although the high parallelizability of the Jacobi method is lost, there are some parallelization options available, but not as trivial.

The Red/Black Gauss–Seidel method

Informal explanation

Let $A\vec{x} = \vec{b}$ be a linear system where A is strictly diagonally dominant yet again.

- This method follows the same idea as the Gauss–Seidel method. Notice how in our scenario, when updating the value at a point, we only need information on it's neighbours.
- Therefore, if we split the domain in a checkerboard pattern, colored black and red, we can update the values in each color in any order, provided we do so independently of updating the values in the other color.
- Therefore, we can update the temperatures in the "red" squares in parallel, and then update the temperatures in the "black" squares in parallel.

The Red/Black Gauss–Seidel method

Implementation details

- This method keeps the pros of the Gauss–Seidel method, without greatly reducing the parallelization potential.
- One must remember the values in different partitions of the domain must be updated in a sequential manner. That is, update all the “red” cells before updating all the “black” cells.
- This is only possible because of the data dependencies in out formulas. This must be addressed on a problem-per-problem basis, and the partitions might be different or impossible.

Summary

General considerations for explicit and implicit methods

- Explicit methods are often simpler to implement, but require more iterations with a smaller time-step, and may be more unstable.
- Implicit methods rely on heavier theory, and are often harder to implement. Conversely, they will often come with greater guarantees such as numerical stability, earlier convergence or high parallelizability.

For this problem set we will be working with the three aforementioned implicit methods, but the rest of the exercises will revolve around the explicit method.

- Math is fun, but today you do not need to understand it to do these exercises!

Timing your code

- To time your code it is recommended that you use the `gettimeofday` function from `sys/time.h`.
- Use the manual pages to read the documentation about it. That is, running `man 3 getttimeofday`.
- If you are programming in languages such as Fortran, C, or C++, the manual pages are your friends, and you need to learn how to navigate them.
- They are also extremely useful in Unix-like operating systems.