MAE5032 Final Project

In this project, we combine some of the theory and practical skills learned in this class to address a simple problem. You have two options. You can work as a group, with 2 people in each group, or you can work individually. For people working in pairs, you are required to work on two-dimensional problems (Problem A by default).

Problem A

We consider the transient heat equation in a two-dimensional (2D) unit square $\Omega := (0,1) \times (0,1)$. The boundary of the domain Γ can be decomposed into

the bottom face $\Gamma_b := \{(x,y) : x \in (0,1), y = 0\}$;

the top face $\Gamma_t := \{(x,y) : x \in (0,1), y = 1\}$;

the left face $\Gamma_l := \{(x,y) : x = 0, y \in (0,1)\}$;

the right face $\Gamma_r := \{(x,y) : x = 1, y \in (0,1)\}$.

Let f be the heat supply per unit volume, u be the temperature, ρ be the density, c be the heat capacity, u_0 be the initial temperature, κ be the conductivity, n_x and n_y be the Cartesian components of the unit outward normal vector . The boundary data involves the prescribed temperature g on Γ_g and heat flux h on Γ_h. The boundary Γ admits a non-overlapping decomposition: $\Gamma = \overline{\Gamma_g \cup \Gamma_h}$ and $\emptyset = \Gamma_g \cap \Gamma_h$. The transient heat equation may be stated as follows.

$$
\begin{aligned}
\rho c \frac{\partial u}{\partial t} - \kappa \left(\frac{\partial^2 u}{\partial x^2} + \frac{\partial^2 u}{\partial y^2} \right) &= f \quad \text{on } \Omega \times (0,T) \\
u &= g \quad \text{on } \Gamma_g \times (0,T) \\
\kappa \frac{\partial u}{\partial x} n_x + \kappa \frac{\partial u}{\partial y} n_y &= h \quad \text{on } \Gamma_h \times (0,T) \\
u|_{t=0} &= u_0 \quad \text{in } \Omega.
\end{aligned}
$$

Problem B

We consider the transient heat equation in a one-dimensional (1D) domain $\Omega := (0,1)$. The boundary of the domain is $\Gamma = \{0,1\}$. Let f be the heat supply per unit volume, u be the temperature, ρ be the density, c be the heat capacity, u_0 be the initial temperature, κ be the conductivity, n_x be the Cartesian components of the unit outward normal vector . The boundary data involves the prescribed temperature g on Γ_g and heat flux h on Γ_h. The boundary Γ admits a non-overlapping decomposition: $\Gamma = \overline{\Gamma_g \cup \Gamma_h}$ and $\emptyset = \Gamma_g \cap \Gamma_h$. The transient heat equation may be stated as follows.

$$
\begin{aligned}
\rho c \frac{\partial u}{\partial t} - \kappa \frac{\partial^2 u}{\partial x^2} &= f \quad \text{on } \Omega \times (0, T) \\
u &= g \quad \text{on } \Gamma_g \times (0, T) \\
\kappa \frac{\partial u}{\partial x} n_x &= h \quad \text{on } \Gamma_h \times (0, T) \\
u|_{t=0} &= u_0 \quad \text{in } \Omega.
\end{aligned}
$$

With the instructor's permission, you may choose a different physical problem associated with your own research. You are encouraged to work on problems or projects from your computational solid/fluid mechanics class.

1 Code development

You are expected to develop a numerical software that can address this mathematical model. In specific, you can choose finite difference method for spatial discretization; you are expected to use both explicit Euler and implicit Euler methods to treat the problem in time. The space and temporal resolutions are assumed to be **uniform**. For both time marching methods, you need to perform basic numerical analysis to delineate the stability property of the adopted scheme. In your code, you will have to assemble a sparse matrix representing the discrete heat equation. You are recommended to use PETSc to handle your sparse matrix and the subsequent linear system solution procedure.

The above numerical methods and the PETSc library are **recommended**. With the instructor's permission, you may choose finite element, finite volume, and other explicit/implicit time marching schemes and use other linear algebra library. More detailed requirement as listed in below.

1. You need to enable a restart facility using HDF5. Write the values of the current solutions every 10 iterations. You need to record necessary data, such as Δx, Δt, etc. Add a command line argument to your program that causes it to read the restart file and resume execution by reading all data from the restart file. Test the correctness of this implementation by comparing restarted and non-restarted simulation results.

2. Your code shall be coded in a defensive manner. We expect to see detailed comments, error checks, assertions, etc. in your code. There should be no warning message when compiling your code.

3. You need to provide a Makefile or CMake for building your software on Tai-Yi. You shall be able to test different compiling flags in your Makefile or CMake. Makesure for provding a README file on how to build and run your code. Provide a job script to run the project on the cluster.

4. You shall perform a profiling analysis of your code using either gprof or Valgrind. Make an analysis of your code performance.

5. You shall do version control of your project with GitHub or Gitee. Add the teaching team to your repository before the final week so we can see your work history. **In particular, we expect to see a collaborative work over weeks in your version control system.**

6. In your report, you shall use gnuplot or a related visualization utility for graphs for 1D problems. You shall use VTK and Paraview for visualization for 2D problems. Provide your visualization routine with your technical report. MATLAB is **not** allowed.

7. All your work will be provided in a project report with all technical details provided. Bonus points will be given to report written in LaTex.

2 Code testing

2.1 Method stability

Run your code with time-independent data. In the 1D case, you may consider the following options.

$$f = \sin(l\pi x), \quad u_0 = e^x, \quad u(0,t) = u(1,t) = 0, \quad \kappa = 1.0.$$

In the above, l is a constant. With the boundary and initial data being time-independent, the solution of the partial differential equation will converge to a steady state solution as time $t \to \infty$. In particular, the steady state is characterized by $\partial u/\partial t = 0$. Substitute this into the differential equation, you may actually obtain the analytical form of the steady state solution for the 1D case. You can design similar boundary and initial data for the 2D problem.

Let Δx and Δt be the spatial and temporal mesh size, which are assumed to be uniform. Take $\Delta x = 10^{-2}$ and try various time steps with the explicit and implicit methods. The calculation may diverge with certain time step sizes. What is the maximum time step size that can deliver stable calculations? Does the result match with the theoretical analysis?

Since our goal is to obtain the steady state solution, we wish to use large time step sizes. What is the influence of the time step on the speed with which implicit method converges to the steady state? Explore a low convergence tolerance and larger time steps in particular.

2.2 Manufactured solution method

With your code developed, you are responsible for verifying your code using the manufactured solution method. Assume you have a solution taken an analytic form, you may put the solution into the original differential equation and obtain the analytic forms of f, g, and h. With the obtained analytic forms, you may run your code and obtain numerical solutions with different spatial and temporal resolutions. Let $\boldsymbol{u}_{exact}$ be the vector of the exact solution values at the grid points, and let $\boldsymbol{u}_{num}$ be the vector of the numerical solution values at the grid points. The error is defined as $e := \max_{1\le i\le n}|u_{\text{exact},i} - u_{\text{num},i}|$, where $u_{\text{exact},i}$ and $u_{\text{num},i}$ are the i-th component

of the solution vectors with the vector length being n. The error is related to the mesh resolution as $e \approx C_1 \Delta x^{\alpha} + C_2 \Delta t^{\beta}$. To determine the value of α and β, you need to progressively refine your mesh and document the error value with the corresponding mesh size. Since there are two error sources, you need to first keep a very fine temporal mesh to rule out the temporal error source and run with different Δx. Use a straight line to approximate the values of $\log(e)$ against $\log(\Delta x)$. The slope of the straight line gives α. In a similar fashion, fix the spatial mesh to be reasonably fine, and calculate the error with different Δt. Use the results to determine β.

3 Parallelism

In this section, we will examine and improve the parallel efficiency of the developed code. In your code, there are two parts that need to be parallelized. One is the assembly of the matrix, and one is the iteration over time. Monitor the time spent on those sections. For the explicit method, matrix-vector multiplication is performed over time; for the implicit method, a linear system is solver over time. The PETSc library can help you automatically handle these. You shall pay attention to parallelize the matrix assembly, making sure that part is properly parallelized.

Perform the fixed-size scalabiling and isogranular scaling analysis for both explicit and implicit solvers.

In your implicit solver, you will have to solve a linear system in each time step. Thus, the choice of linear solver (iterative method and preconditioning technique) can impact the parallel performance. Investigate the following PC options from PETSc for your parallel test: (a) Jacobi; (b) Additive Schwarz; (c) LU with MUMPS. You may refer the PETSc manual for references.