Final Project of MAE 5032 for 2025 Spring Semester

12431140

June 17, 2025

In this project, we combine some of the theory and practical skills learned in this class to address a problem. In the following, we provide two background problems, from which you may choose one as your project topic.

Problem

We consider the transient heat equation in a two-dimensional unit square $\Omega := (0, 1)$. 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 $\Gamma_g$and heat flux h on Γ_h. The boundary Γadmits a non-overlapping decomposition:$\Gamma = \overline{\Gamma_g \bigcup \Gamma_h}$and $\emptyset = \Gamma_g \bigcap \Gamma_h$. The transient heat equation may be stated as follows.

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

Code Development (60 points)

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, or Runge-Kutta time marching schemes and use other linear algebra libraries. More detailed requirements are listed in below.

1. All your work will be provided in a project report with all technical details provided. Five bonus points will be given to report written in `LaTex`.
2. 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`. Make sure for providing a `README` file on how to build and run your code. Provide a job script to run the project on the cluster.
3. You have to perform version control of your project with `GitHub` as the remote repository. Add the instructor and TA to your repository before the final week so we can see your work history. We want to see a substantial development history. A project with only a few commits will be regarded as cheating.
4. You need to enable a restart facility using `HDF5`. Write the values of the current solutions every a few (say 10) iterations. You need to record necessary data, such as $\Delta x, \Delta t$, etc., as well. 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. You need to test your restart functionality.
5. Your code needs to be coded in a defensive manner. We expect to see detailed comments, error checks, assertions, etc. in your code. There should be absolutely no warning message when compiling your code.
6. In your report, you shall use gnuplot or `VTK` for visualization. Provide your visualization routine with your technical report. MATLAB is not allowed.

Test

The following tests shall be performed and reported in your technical report.

1. Numerical stability (10 points)

Let Δx andΔt be the spatial and temporal mesh size, which are assumed to be uniform in this project. Take a fixed Δx and try different time step sizes 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?

2. Code verification (10 points)

With your code developed, you are responsible for verifying your code using the manufactured solution method or the closed-form analytic solution. 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 a forcing term and the boundary data. With the obtained analytic expressions, you may run your code and obtain numerical solutions with different spatial and temporal resolutions. Let u_{exact} be the vector of the exact solution values at the grid points, and let u_{num} be the vector of the numerical solution values at the grid points. The error is defined as $e := \max_{i \geq i \geq n} |u_{exact,i} - u_{num,i}|$, where $u_{exact,i}$ and $u_{num,i}$ 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 . Report the detailed experimental setting, your obtained error, and the convergence rates.

3. Parallelism (15 points)

Here, we focus on the code implemented with the implicit scheme. There are two major parts of your code that affects its efficiency, that is, the assembly of the matrix and vectors and the solution process for the linear system. Your matrix shall be assembled in a parallel manner using the domain decomposition concept. Each processor will account for only a portion of the domain, and make sure that you understand the matrix and vector layout pattern. For example, in `PETSc`, the ownership of the parallel vector can be setup by the function `VecCreate`. The matrix representation within `PETSc` is the compressed sparse row format. You may set it up by the function `MatCreateAIJ`. Make sure you understand their usage. The `PETSc` library will handle the parallel linear algebra operations.

You need to examine the parallel performance of the code by a fixed-size scalability test. Fix the grid size and the time step size. Progressively increase the number of processors and run your code. Record the time of matrix assembly, solver time, and total time for each run. Repeat each test for at least 3 times to average out noise. Report the speedup and efficiency of the code.

You also need to examine the parallel performance of the code by a isogranular scalability test. Fix the number of unknowns per processor. Increase the number of processors, and record the matrix assembly, solver time, and total time. Repeat each run for 3 times to obtain the averaged computation cost. Report the collected performance results.

4. Postprocessing (5 points)

You need to run your code with a physically/economically reasonable setting. Explain the physical/economical meaning of all data involved in the calculation. Your visualization need to illustrate how the temperature/option price evolves over time. You also need to plot the temperature/option value at selected times ($t = 0, 0.25T, 0.5T, 0.75T, T$). You also need to plot the derivative $\frac{\partial u}{\partial x}$ as a surface of (x, t). Include all visualization plots in the report.