Advanced Scientific Computing: Assignment 5

This assignment must be submitted by Friday May 2, 11 at 5pm.

Question 1: Modern Fortran solver for a time-dependent system of PDEs

Question 1 elaborates on some of the functionalities of modern Fortran for solving time-dependent PDEs.

The shallow water equations (SWE) are a foundational set of equations in fluid dynamics which were initially formulated by Saint-Venant in 1871. The SWEs usually describe the behavior of fluid flow in shallow water bodies such as rivers, lakes, and coastal regions. These equations are pivotal in fields like oceanography, meteorology, and hydraulic engineering, where they are applied to model natural phenomena including wave propagation, tides, and river flows. Due to their simplified nature, the SWEs provide a robust mathematical framework for capturing essential fluid behaviors over large spatial scales without the complexity of full three-dimensional models.
Here, we consider a solver for the linearized SWE

$$u_t + Uu_x + gh_x + Vu_y = 0, \quad (x, y) \in \Omega, \quad t \geq 0, \tag{1a}$$

$$v_t + Uv_x + Vv_y + gh_y = 0, \quad (x, y) \in \Omega, \quad t \geq 0, \tag{1b}$$

$$h_t + Uh_x + Hu_x + Vh_y + Hv_y = 0, \quad (x, y) \in \Omega, \quad t \geq 0, \tag{1c}$$

where U, V and $H > 0$ are constant background flow fields and $g = 1$ dimensionless gravitational constant. Here h is denotes the water height perturbation and (u, v) are the flow velocity perturbations. We consider the zero mean flow velocities $U = V = 0$, mean water height $H = 1$ and enforce the boundary conditions $h = 0$ at the domain boundaries $x \in \partial\Omega$. The final time is $t = 1$ and the initial conditions are

$$u(x, y, 0) = 0, \quad v(x, y, 0) = 0, \quad h(x, y, 0) = 2 \times e^{-d^2(x,y)/0.01}, \quad d^2(x, y) = (x - x_0)^2 + (y - y_0)^2$$

with $(x_0, y_0) = (1/4, 1/2)$.

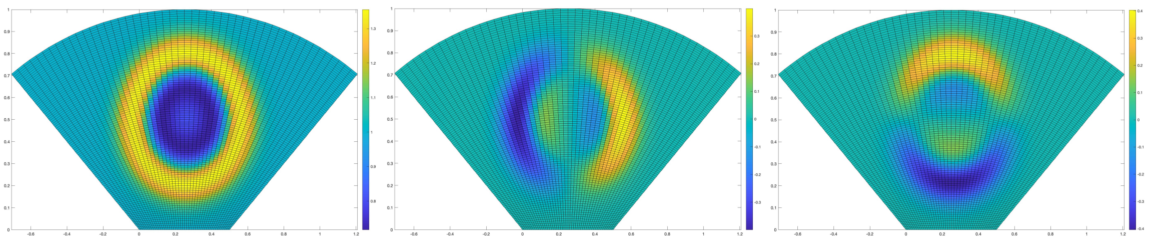

Figure 1: Snapshots of the solutions, $H + h$, u, v, at $t = 0.26$

The spatial domain is a seashell defined by the boundary curves: $(x, y) \in \Omega \subset \mathbb{R}^2$.

$$\begin{bmatrix} x(q,0) \\ y(q,0) \end{bmatrix} = q \begin{bmatrix} 1/2 \\ 0 \end{bmatrix}$$
$$\begin{bmatrix} x(0,r) \\ y(0,r) \end{bmatrix} = r \begin{bmatrix} \cos(3\pi/4) \\ \sin(3\pi/4) \end{bmatrix}$$
$$\begin{bmatrix} x(1,r) \\ y(1,r) \end{bmatrix} = r \begin{bmatrix} \cos(\pi/4) \\ \sin(\pi/4) \end{bmatrix} + \begin{bmatrix} 1/2 \\ 0 \end{bmatrix}$$
$$\begin{bmatrix} x(q,1) \\ y(q,1) \end{bmatrix} = \begin{bmatrix} \cos(q(\pi/4 - 3\pi/4) + 3\pi/4) \\ \sin(q(\pi/4 - 3\pi/4) + 3\pi/4) \end{bmatrix} + q \begin{bmatrix} 1/2 \\ 0 \end{bmatrix}$$

We can use the 2D transfinite interpolation to generate a curvilinear grid on a seashell.

Task 1: Create a directory for the assignment named Assignment5/Q1. Move into the directory and create source and build subdirectories. Download the 14 source files and the CmakefileLists from BlackBoard
CMakeLists.txt
time_step.f90
rhs.f90
mms.f90
main.f90
gradient.f90
domain.f90
block.f90
sbp.f90
plotter.f90
metricderivatives.f90
grid.f90
fields.f90
datatypes.f90
SWE.f90
and store them in the source directory. Use cmake to build the code and generate a Makefile.
Compile and execute the code.

The code uses the 4th order accurate SBP-SAT method with the 4th low-storage (5-stage) Runge-Kutta time-stepping method to solve the IBVP. Please go through the codes and make sure you understand the logical flow and the data structure.

The numerical mesh is store in a file mesh_xy.dat, and the solution is written to a file solution_xy.dat at each time-step.

Prepare a python Jupyter notebook to read the mesh and plot the stored numerical solution data on the mesh. Plot snapshots of the numerical of the total water height $H + h$ at $t = 0.1, 0.25, 0.5, 0.75, 1.0, 1.5, 2$ s, on the mesh.

Provably accuracy is critical for numerical PDE solvers. Now you will verify the accuracy of the solver using the method of manufactured solution. We force the IBVP to have the following exact manufactured solution:

$$u_e = \cos(\pi t)\sin(5\pi x)\sin(5\pi y), \tag{2}$$
$$v_e = \sin(\pi t)\cos(5\pi x)\cos(5\pi y), \tag{3}$$
$$h_e = 1 + 0.2\cos(\pi t)\cos(5\pi x)\cos(5\pi y), \tag{4}$$

and the final time $t = 1.0$. The module mms.f90 contains incomplete implementations of the manufactured solution and the source term, the numerical l_2-error.

Task 2: One of your main task is to complete the implementation of the subroutine update_mms(B, t) to compute the MMS solution and the source terms.

Compute the numerical l_2-error by integrating the point-wise error on the grid using the SBP quadrature rule, that is

$$error(t) = \sqrt{\sum\sum\left((\mathbf{h}_{ij}(t) - \mathbf{h}_{eij}(t))^2 + (\mathbf{u}_{ij}(t) - \mathbf{u}_{eij}(t))^2 + (\mathbf{v}_{ij}(t) - \mathbf{v}_{eij}(t))^2\right) h_i^{(q)} h_j^{(r)} J_{ij}}.$$

The implementation can be realized in the subroutine compute_error$(B, error)$. Note that the data object B%F%F(:,:,:) carries the numerical solution and B%MMS%F%F(:,:,:) carries the analytically manufactured solution.

In the main program, set the final time to $t\_final = 1d0$, the MMS flag $mms\_flag = .true.$ and run the on a sequence of grid points $(n_q, n_r) = [(11, 21), (21, 41), (41, 81), (81, 161), (161, 321)]$. Do the errors converge to zero? What is the order of accuracy?

Question 2: Compile, run and time OpenMP Fortran programs on Jakar

Task 1: Login with your credentials on UTEP's Jakar. Create a directory for the assignment named Assignment5/Q2. Move into the directory and create source and build subdirectories. Download the 6 source files, job submission script, and the CmakefileLists
CMakeLists.txt
job.sh
bvp.f90
grid.f90
main.f90 conjugategradient.f90
loadvector.f90
plotter.f90
from the OpenMP module on BlackBoard and store them in the source directory. Load cmake and the following compilers gnu12 openmpi4 in your work environment. Use cmake to build the code and generate a Makefile. In the job submission script, please replace youremail@utep.edu with your email so that get a message when the job runs.
Compile and execute the code with the job submission script (submit the job with the command sbatch job.sh).

Prepare a python Jupyter notebook to read and plot stored exact and numerical solution data.

The code uses the 4th order SBP-SAT method to solve the BVP

$$-\frac{d^2u}{dx^2} + 0.1u = f, \quad x \in [0, 1],$$

$$\frac{du}{dx} - 0.5u = g_0, \quad x = 0, \quad \frac{du}{dx} + 0.5u = g_1, \quad x = 1,$$

where f is a sufficiently smooth function and $g_0, g_1 \in \mathbb{R}$ are constants. Here the exact solution is $U(x) = e^{(x-0.5)^2/0.25}\cos(10\pi x)$ with $f = -\frac{d^2U}{dx^2} + 0.1U$ and $g_0 = \frac{dU}{dx} - 0.5U, \quad x = 0, \quad g_1 = \frac{dU}{dx} + 0.5U, \quad x = 1,$

The numerical grid, the exact and numerical solution are written to files exa_sol.dat and num_sol.dat

Task 2: We have inserted wall-clock timers into the code. The elapsed time is printed to out.txt
In job submission script, vary the number of threads, for ntasks=1, 2, 4, 6, 8, 10. For each case record the elapsed-time In your Jupyter notebook plot the elapsed-time against the number of threads, ntasks. Does

the elapsed time decrease with increasing number of threads, ntasks. For some constants $C > 0$ and $p > 0$, fit a power law elapsed_time($ntasks$) $= Cntasks^p$, to the elapsed_time. What power p fits the data best?