CS 5220 Final Project Proposal

Bob Chen [kc 853]

Last Updated on November 2, 2015

1 Alternating Direction Implicit (ADI)

1.1 Basic Problem

The initial problem is to solve a system given by

$$div(a(\mathbf{x}\nabla\mathbf{u})) = f, \quad u(x) = g(x), \quad \text{for } x \in \partial\Omega$$

We are treating a as a uniform constant throughout the entire space Ω which is defined to be a rectangular region in $\mathbb{R}^2$. We are using Dirichlet boundary conditions for this problem.

Instead of solving this problem directly, we seek a solution as the steady state of

$$\frac{\partial\tilde{u}}{\partial t} = div(a(\mathbf{x})\nabla\tilde{u}) - f$$

The steady state will occur when we have the exact solution and the error

$$\tilde{e}(x,t) = \tilde{u} - u$$

will converge as we move forward in time. Essentially, we will use numerical methods to move the state $\tilde{u}$ in time and stop when its difference from the true solution is sufficiently small.

1.2 Finite Difference Approximation

In the problem, we have a grid on the space Ω with the same stepsize h in both x and y direction (for simplicity). We first look at 1D case for simple discussion of the solver. The problem we try to solve is

$$\alpha\frac{d^2u}{dx^2} = f$$

which can be expressed as

$$\begin{bmatrix} 1 & & & & \\ \frac{\alpha}{h^2} & -2\frac{\alpha}{h^2} & \frac{\alpha}{h^2} & & \\ & \ddots & \ddots & \ddots & \\ & & \frac{\alpha}{h^2} & -2\frac{\alpha}{h^2} & \frac{\alpha}{h^2} \\ & & & & 1 \end{bmatrix} \begin{bmatrix} u_1 \\ \vdots \\ u_{m-1} \end{bmatrix} = \begin{bmatrix} f_1 - g_a\frac{\alpha}{h^2} \\ f_2 \\ \vdots \\ f_{m-1} - g_a\frac{\alpha}{h^2} \end{bmatrix}$$

We simply solve this tridiagonal linear system using Thomas's algorithm.

The discussion of 2D case will be a little more involved but it has the same idea. The problem becomes solving the following system

$$\begin{aligned} (I - \frac{dt}{2}dx\alpha_x\Delta x)u^*_{k+1} &= (I + \frac{dt}{2}dx\Delta x + dt\alpha_y\Delta y)u_k - dt\mathbf{f} \\ (I - \frac{dt}{2}dx\alpha_y\Delta x)u_{k+1} &= u^*_{k+1} - \frac{dt}{2}\alpha_y\Delta y u_k \end{aligned}$$

The method marches back solvers in the x direction and then in the y direction, thus giving rise to the name alternating direction implicit.

1.3 Parallelization

Each time we cross the domain, we have to repeat the same computation for each row/column of grid values. It is possible to parallelize these computations and get a speedboost from multithreads. Unlike shallow water simulation, we are using implicit solver for each row/column, which makes the basic computation expensive. We expect to see better results from domain decomposition methods.

2 Objectives

2.1 Serial Code

It is tricky to set up the data structre for efficient computation because we need to march both in the x and y direction. We certainly do not want to make explicit copies for each timestep. So we will have to change the data structure for better vectorization of the inner loops.

For the serial code, we will use profiling tools such as amplifier to find the bottleneck of our code and vectorize the inner loops for our kernel solvers. Also, we will try different compiler flags for faster serial code.

2.2 Parallel Code

For the parallel code, we will rearrange the serial code if necessary and set up the computation so that it incurs as little overhead cost as possible. Then we will implement a functional version of parallel code using OpenMP. Then we will vary the number of threads used for parallel computing and report our findings.

2.3 What to Do

We have a code for naive serial implementation and we will need to optimize it. We will divide the project into two stages: parallel code implementation and tuning. To reach the first milestone, we need to

- The original code barely works, so we will need to clean it up and organize it in better way for debugging.
- We need to write our own Makefile to get the program working on the Totient cluster, along with pbs files for it to run on the computing node.
- Basic profiling to get the performance analysis and identify bottleneck of the current code.
- Optimization of the serial code. Both vectorize the loops and try different compiler flags
- Use OpenMP to parallelize the code.

Then, we want to study the behavior of our code and tune it for best performance:

- Compare the performance of our new code with original naive code.
- Change the number of threads used for parallel code.
- Run this code on Phi Board to see what difference it makes.
- Tune the code.
- Write a report.