Concurrent Scientific Computing Using Python

Project 2: Irrotational Fluid Flow

Randriamahefasoa Tsinampoizina Marie-Sophie

April 5, 2011

Executive Summary

The purpose of this work is to emphasis the importance of high performance computing in numerical methods. The problem that we are dealing with is to solve the stream function of an irrotational fluid flow through a contraction duct.

The stream function is given by a Poisson equation which is be solved numerically by finite element method and Jacobi iteration.

Statement of the Problem

As stated in [1], a fluid flows into a contraction duct (in the Figure 1) with a uniform velocity of 3m/s end exits at a velocity of 12m/s. Solve the Stream Function in the case of irrotational fluid.

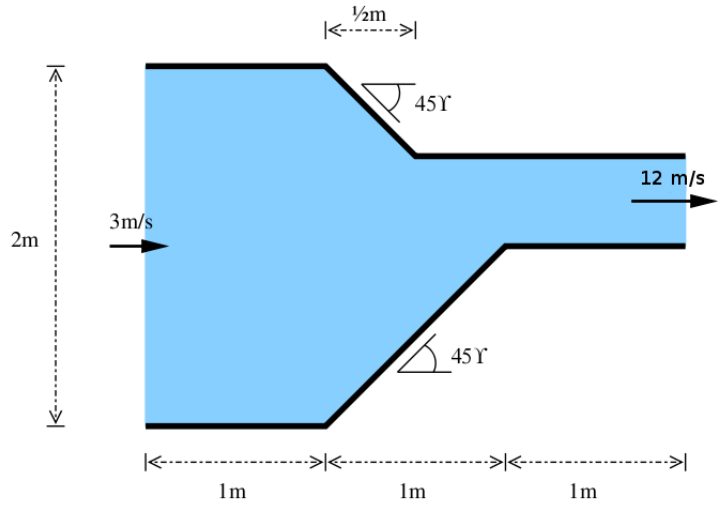

Figure 1: Incompressible fluid flows through a contraction duct

Description of the Mathematics

Following the analysis described in [1], the stream function $u(x, y)$ for irrotational fluid flow is governed by the Poisson equation:

$$\frac{\partial^2 u}{\partial x^2} + \frac{\partial^2 u}{\partial y^2} = 0$$

with the boundary conditions

- $u(0, y) = 3y$ in the left
- $u(3, y) = 12(y - 1)$ in the right

- $u(x, y) = 0$ along the lower wall

- $u(x, y) = 6$ along the upper wall

Description of the Algorithm

A mesh grid representing the duct stores the value of the stream function in its nodes

Consider the spacing of the mesh grid is $h = \frac{1}{2N}$ where N is an integer. With this condition, the grid always follows the lower and the upper wall.

The standard Poisson finite difference element method is implemented as follows

$$u_{i,j} = 0.25 \left(u_{i-1,j} + u_{i+1,j} + u_{i,j-1} + u_{i,j+1} \right)$$

The Jacobi method: we apply the following iteration to the matrix $u^{(n)}$

$$u_{i,j}^{(n+1)} = 0.25 \left(u_{i-1,j}^{(n)} + u_{i+1,j}^{(n)} + u_{i,j-1}^{(n)} + u_{i,j+1}^{(n)} \right)$$

where $u^{(n+1)}$ is the new matrix and $u^{(n)}$ is the old matrix.

The matrix is divided in vertical strips. The division into those strips is done as follows:

- we count all the internal nodes `all_int_nodes` and initialise `remaining_int_nodes = all_int_nodes` and the processes `remaining_proc`
- for each process: we vary the width of the strips until it has a number of internal nodes more than the approximated ratio `remaining_int_nodes/remaining_proc`, then we subtract the nodes in that strip from the `remaining_int_nodes`

Results

This is the plotting obtained

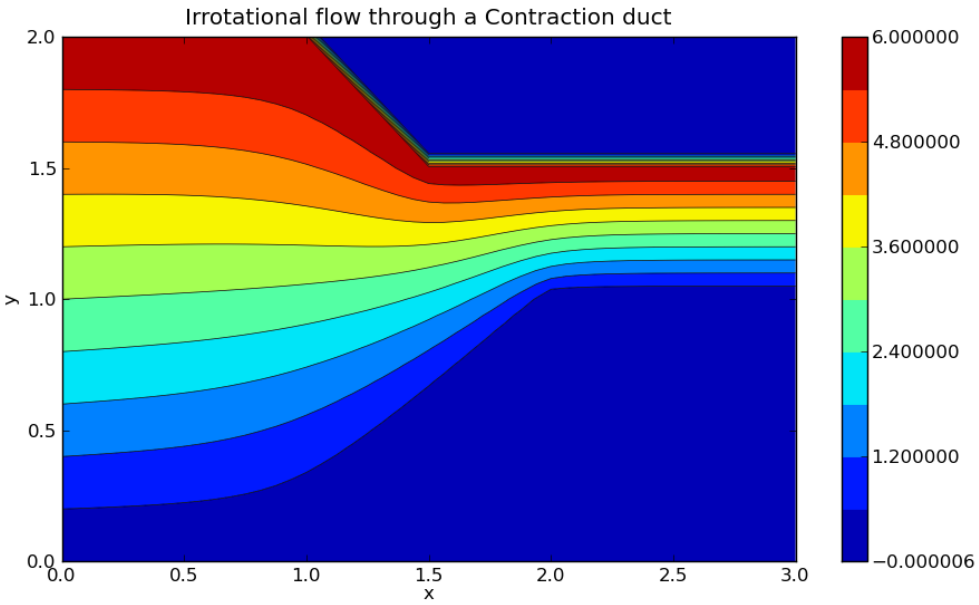

Figure 2: Incompressible fluid flows through a contraction duct: graph of the streamlines

Using the values

- `N = 20`
- `max_error` $= 10^{-3}$

we obtained

P	T_P[ms]	S_P	S_P^C
S	20690.0	.	.
1	24487.0	1.0	0.845
2	12960.0	1.889	1.596
4	18431.0	1.329	1.122
8	18106.0	1.352	1.143

Table 1: Table of times

Conclusion

The benefit from the use of multi-processes becomes obvious when we are dealing with large subdivision of the grid (large N). In our case the optimal number of processes to use is 2, but with more powerful computer that would increase.

Bibliography

[1] Brian Brodie. *A Friendly Introduction to Numerical Analysis*, chapter 9,"Eliptic Partial Differential Equations". Prentice Hall., 2006.