High Performance Computing for Mathematical Models

2D Fluid Solver

Jan Kirchner, Petr Valenta

February 19, 2016

This document is a part of the assessment work for the subject WMMS5311 High Performance Computing for Mathematical Models lectured at ENSIMAG, Grenoble INP.

Abstract

This document describes a 2D fluid flow solver. The code is written in `C++` programming language and parallelized using `MPI` library. The mathematical description of the flows as well as the implementation and parallelization of the code is briefly discussed. In the last part, the results of several performance tests are presented. All source files can be found at `https://github.com/valenpe7/2d_fluid_solver`.

Contents

1 Introduction

For the final project of the course "High performance computing for mathematical models" we had to apply the methods we learned during the course to parallelize a numerical application that we developed in a previous course using `MPI`[1] library. In this report we are going to explain how to use the final program, how we parallelized it and how the parallelization influenced the performance of the program. We chose to work with fluid solver that computes the solution to the 2D incompressible Navier-Stokes equations for a specified fluid in a specified environment to acquire values for the velocity and the pressure of the fluid at the respective positions in the environment.

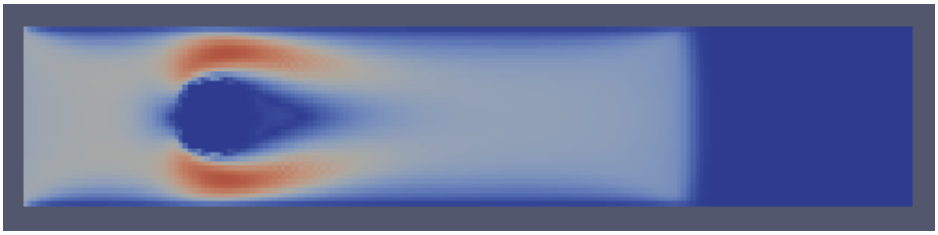

Figure 1: One sample evaluation of the fluid solver at one time step.

The implementation of the solver as well as the mathematical exposition of the problem in this report follows Griebel [2]. At the core of the program in the method `solve_Poisson_Jacobi()` the relaxed Jacobi method is used to approximate a solution for the Poisson equation. The Jacobi method is a classic example for a method that can be efficiently parallelized and that is the reason why we chose this program. We decided furthermore to rewrite the complete program to include `MPI` to improve performance further.

2 Implementation

The behavior of incompressible flows is described by the well known Navier-Stokes equations:

$$\frac{\partial}{\partial t}\mathbf{u} + (\mathbf{u}\cdot \mathrm{grad})\mathbf{u} + \mathrm{grad}p = \frac{1}{Re}\Delta\mathbf{u} + \mathbf{g} \qquad \text{(momentum equation)}, \tag{1}$$

$$\mathrm{div}\mathbf{u} = 0 \qquad \text{(continuity equation)}. \tag{2}$$

where $t > 0$ denotes time, $\mathbf{u}$ is the velocity field, p is the pressure, Re is the Reynolds number and $\mathbf{g}$ denotes body forces such as gravity acting throughout the bulk of the fluid. When solving the 2D case of the system (1), (2) discretized in time with time step Δt, the following *Poisson equation for the pressure* $p^{(n+1)}$ arises:

$$\frac{\partial^2 p^{n+1}}{\partial x^2} + \frac{\partial^2 p^{n+1}}{\partial y^2} = \frac{1}{\Delta t}\Big(\frac{\partial F^n}{\partial x} + \frac{\partial G^n}{\partial y}\Big), \tag{3}$$

where

$$F^n = u_x^n + \Delta t\left[\frac{1}{Re}\left(\partial_x^2 u_x^n + \partial_y^2 u_y^n\right) - \partial_x (u_x^2)^n - \partial_y (u_x u_y)^n + g_x\right],$$

$$G^n = u_y^n + \Delta t\left[\frac{1}{Re}\left(\partial_x^2 u_y^n + \partial_y^2 u_x^n\right) - \partial_y (u_y^2)^n - \partial_x (u_x u_y)^n + g_y\right].$$

The equation (3) can reliably be solved with the *Jacobi relaxation method*. To achieve the standard form of the equation, first the right hand side of the equation is computed at each grid point $(i\Delta x, j\Delta y)$ and time step Δt. The update of the pressure then occurs in the method `solve_Poisson_Jacobi()` according to the updating formula:

$$p_{i,j}^{\mathrm{it}+1} := (1-\omega)p_{i,j}^{\mathrm{it}} + \frac{\omega}{\frac{\varepsilon_i^E + \varepsilon_i^W}{\Delta x^2} + \frac{\varepsilon_j^N + \varepsilon_j^S}{\Delta y^2}}\left(\frac{\varepsilon_i^E p_{i+1,j}^{\mathrm{it}} + \varepsilon_i^W p_{i-1,j}^{\mathrm{it}}}{\Delta x^2} + \frac{\varepsilon_j^N p_{i,j+1}^{\mathrm{it}} + \varepsilon_j^S p_{i,j-1}^{\mathrm{it}}}{\Delta y^2} - \mathrm{RHS}_{i,j}\right)$$

for $i = 1, \ldots, i_{\max}$ and $j = 1, \ldots, j_{\max}$, where $\omega \in (0,1)$ and the parameters $\varepsilon \in \{0,1\}$ indicate whether cell (i,j) lies adjacent to the domain boundary. The superscripts W, E, N, and S indicate in which direction (west, east, north, or south) the boundary lies. These updates are repeated until convergence is achieved. These computations are the most expensive part of the solver and parallelizing

them should significantly increase the code performance.

We chose to consider the problem column-wise, thus the domain is divided into columns and workload is distributed between individual processes accordingly. In our version of code, the number of processes has to be divisor of the number of cells in the horizontal direction. Wherever during computation quantity relies on values from the neighboring columns, information is exchanged between the processes using `MPI` library routines.

3 Compilation and executing

For the compilation, one can use the attached bash script "compile" which executes the prepared makefile. Due to the usage of features introduced by the latest revision of C++ standards, the minimum required version of GCC is 4.8.

To execute the fluid solver, a `.png` file is required to specify the environment [1]. The compiled application takes the filename of the input image and output path for the exported `.vtk` files as an argument. Other parameters like fluid properties, the geometry of the simulation domain, boundary conditions, etc. can be specified in the code before the compilation. After simulation, the `.vtk` files can be displayed with a software tools such as `ParaView` or `VisIt`.

4 Results

As a part of the project, several performance test were done on a clusters operated and managed by the Czech National Grid Organization MetaCentrum. We chose the cluster minos.zcu.cz, which contains 50 nodes, where each of the nodes has the following hardware specification:

CPU	2 × 6-core Intel® Xeon® E5645 2.40 GHz
RAM	24 GB
Disk	2 × 600 GB
Net	1 Gbit/s Ethernet, Infiniband. IB switches of clusters are connected by 6 × 40Gb/s
Owner	CESNET

Table 1: Specification of used computational resources

We made a model scene of size 1600×400 cells with obstacles created using an image, which is shown in Figure (2). The flow is coming from the north to the south boundary, in order to distribute workload between processes equally, since we have divided the domain in columns. The temporary solution of exporting data from all processes into one `.vtk` file for visualization purposes is turned off during the performance tests, because it significantly influences the computational time. The results of performed simulations can be seen in the Table (2).

From the measured times we calculated a speedup. The result is shown in Figure (3). It can be clearly seen, that up to 4 processes we achieved almost linear speedup. With the increase of the number of computational units, the value of speedup grows more slowly. Using more than 64 processors is

[1] Obstacles in the environment are specified via the alpha channel of the prepared `.png` file. The value 0 indicates an empty field, while any other value indicates an obstacle.

inefficient in this case. The computational domain is too small, and too much time is spent for the communication between processes.

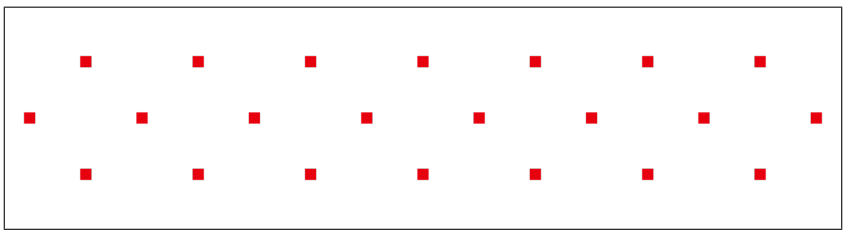

Figure 2: `.png` image of size 1600×400 used for creating model scene for the performance tests of the code. Red squares are obstacles.

P	T_{total} (s)	T_{calcul} (s)
1	415	404
2	208	202
4	123	112
8	87	74
16	55	42
32	31	23
64	23	13
80	32	12

Table 2: Total duration (T_{total}) and duration of calculation without setup and completion (T_{calcul}) of simulations according to the number of processors P on which had been executed.

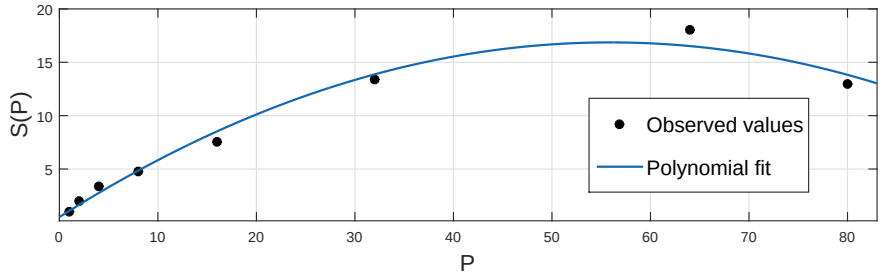

Figure 3: Speedup S(P) of the code depending on the number of processes P.

5 Conclusion

Even though the results of parallelization indicates significant improvement of the code, there is definitely still a lot of work to do. First, the domain should be divided into sub-domains not only in the x-direction, but also in the y-direction, to be sure, that the workload is distributed correctly independently of the fluid flow direction. Afterwards, we should handle the situation, when the number

of cells is not a divisor of the chosen number of processors. The next step should be the rewriting of the routine for exporting the data. The current approach is inefficient and for the parallel computing completely useless. It has been used only to check if the solver works correctly in the parallel regime. The option could be exporting into parallel `.vtk` files.

Acknowledgements

Authors would like to thank Mr. Christophe Picard, for the guidance and valuable advice he has provided throughout the course.

Access to computing and storage facilities owned by parties and projects contributing to the National Grid Infrastructure MetaCentrum, provided under the programme "Projects of Large Infrastructure for Research, Development, and Innovations" (LM2010005), is greatly appreciated.

Access to the CERIT-SC computing and storage facilities provided under the programme Center CERIT Scientific Cloud, part of the Operational Program Research and Development for Innovations, reg. no. CZ. 1.05/3.2.00/08.0144, is greatly appreciated.

Jan Kirchner, Petr Valenta

References

[1] Edgar Gabriel, Graham E. Fagg, George Bosilca, Thara Angskun, Jack J. Dongarra, Jeffrey M. Squyres, Vishal Sahay, Prabhanjan Kambadur, Brian Barrett, Andrew Lumsdaine, Ralph H. Castain, David J. Daniel, Richard L. Graham, and Timothy S. Woodall. Open MPI: Goals, concept, and design of a next generation MPI implementation. In *Proceedings, 11th European PVM/MPI Users' Group Meeting*, pages 97–104, Budapest, Hungary, September 2004.

[2] Michael Griebel, Thomas Dornseifer, and Tilman Neunhoeffer. *Numerical Simulation in Fluid Dynamics: A Practical Introduction.* Society for Industrial and Applied Mathematics, Philadelphia, PA, USA, 1998.