This document was created by **Christian Buth**.

It is made available on my homepage

http://www.Christian.Buth.mysite.de

for *free*.

If you encounter problems in displaying this file or

you find mistakes feel free to contact me

cbuth@ix.urz.uni-heidelberg.de .

Department of Physics and Astronomy

High Performance Computing in Physics Project
Physics 4

Practical: Fluid Dynamics

Christian Buth

May 1999

Abstract

This project investigates the fluid flow in a cavity and is an example of computational fluid dynamics. Two algorithms, the Jacobi and the Gauss-Seidel, are used to calculate the velocity field inside the cavity. To simplify the calculations a two-dimensional model of the system is used. A first approach is to solve the Navier-Stokes equation for inviscid flow. Afterwards the viscous flow is examined. The problem is inherent parallel and the implementation investigates and exploits this.

Declaration

I declare that this project and report is my own work.

Signature: Date:

Supervisor: Dr. David Henty

3 Weeks

Contents

1 Introduction

This project deals with computational fluid dynamics and is a tiny example of what is done by engeneers and scientist on supercomputers to solve "real" problems. The aim of the exercise is to simulate a two-dimensional fluid flow through a cavity which is modelled as a square. The fluid enters the system through an inlet at the right and leaves the cavity through an outlet at the bottom.

The properties of fluid flows are governed by the Navier-Stokes equation. This equation is solved for inviscid and viscous flow. This is done by implementing a variety of partial differential equation (PDE) solver. The PDE solver use two major techniques the Jacobi and the Gauss-Seidel algorithm and many modified versions of both.

The convergence rate of the PDE solvers for the inviscid flow is investigated thoroughly and is compared with results obtained from theory.

The viscous flow is more difficult to solve because the governing equations are no longer linear. The stability of the Gauss-Seidel algorithm turns out to be poor and insufficient for highly turbulent flows. Thus an underrelaxed Gauss-Seidel algorithm is used to obtain results in the regimes.

Afterwards the scaleability of the parallelised algorithms is examined. *Speed-up* and *efficiency* are measured and analysed.

2 Theoretical Background

The motion of incompressible fluids is described by the NAVIER-STOKES and the simplified *continuity* equation[1]

$$\frac{\partial \vec{u}}{\partial t} + (\vec{u} \cdot \vec{\nabla})\vec{u} = \vec{F} - \frac{1}{\varrho}\vec{\nabla}P + \nu\vec{\nabla}^2\vec{u} \qquad \text{and} \qquad \vec{\nabla} \cdot \vec{u} = 0. \tag{1}$$

In this project a steady flow ($\frac{\partial}{\partial t} = 0$), P = constans, $\vec{F} = 0$ and $\varrho = 1$ is assumed.

The *vorticity* of a fluid flow is $\vec{\omega} = \vec{\nabla} \times \vec{u}$. Using $\vec{u} \times \omega = \frac{1}{2}\vec{\nabla}\vec{u}^2 - (\vec{u} \cdot \vec{\nabla})\vec{u}$ equation (1) becomes

$$-\vec{u} \times \vec{\omega} = -\frac{1}{2}\vec{\nabla}\vec{u}^2 - \frac{1}{\varrho}\vec{\nabla}P + \nu\vec{\nabla}^2\vec{u}.$$

$\vec{\nabla}\times$ on both sides yields

$$\nu\vec{\nabla}^2\vec{u} = -\vec{\nabla} \times (\vec{u} \times \vec{\omega}) = (\vec{u} \cdot \vec{\nabla})\vec{\omega} - (\vec{\omega} \cdot \vec{\nabla})\vec{u} - \vec{\omega}(\vec{\nabla} \cdot \vec{\omega}).$$

The z-component of this equation is

$$\nu\vec{\nabla}^2\zeta = u_{\mathrm{x}}\frac{\partial \zeta}{\partial x} + u_{\mathrm{y}}\frac{\partial \zeta}{\partial y}, \tag{2}$$

an equation for the two-dimensional vorticity

$$\zeta = \frac{\partial u_{\mathrm{x}}}{\partial y} - \frac{\partial u_{\mathrm{y}}}{\partial x}. \tag{3}$$

For a two-dimensional flow one can introduce the *stream function* Ψ by setting $u_{\mathrm{x}} = \frac{\partial \Psi}{\partial y}$ and $u_{\mathrm{y}} = -\frac{\partial \Psi}{\partial x}$. Using the *stream function* with equations (3) and (2) gives

$$\zeta = \vec{\nabla}^2\Psi \qquad \nu\vec{\nabla}^2\zeta = \frac{\partial \Psi}{\partial y}\frac{\partial \zeta}{\partial x} - \frac{\partial \Psi}{\partial x}\frac{\partial \zeta}{\partial y}. \tag{4}$$

2.1 Inviscid flow

In the case of the inviscid ($\nu = 0$) flow the equation (2) gets a simpler form

$$\vec{\nabla}^2\Psi = \zeta,$$

where $\zeta = 0$ in this project[2]. Discretising this partial differential equation using a five point stencil yields

$$\Psi(i,j) = \frac{1}{4} \cdot \left(\Psi(i-1,j) + \Psi(i+1,j) + \Psi(i,j-1) + \Psi(i,j+1)\right).$$

[1] These equations are deduced in [3].

[2] ζ describes the initial vorticity which is conserved in inviscid flows. If $\zeta \neq 0$ the boundary condition is not of a DIRICHLET type and one would have to reimpose them for every iteration. At every time ζ must satisfy both equations (2) but the left hand side of the right equation is zero, because $\nu = 0$.

2.2 Viscous flow

The viscous flow is described by equation (2) which is discretised using a five point stencil for the LAPLACE operator and central differences for the derivatives. This gives

$$\Psi(i,j) = \frac{1}{4} \cdot (\Psi(i-1,j) + \Psi(i+1,j) + \Psi(i,j-1) + \Psi(i,j+1) - \zeta(i,j))$$

and

$$\begin{aligned} \zeta(i,j) &= \frac{1}{4} \cdot (\zeta(i-1,j) + \zeta(i+1,j) + \zeta(i,j-1) + \zeta(i,j+1)) \\ &\quad -\frac{R}{16} \cdot ((\Psi(i,j+1) - \Psi(i,j-1))(\zeta(i+1,j) - \zeta(i-1,j)) \\ &\quad -(\Psi(i+1,j) - \Psi(i-1,j))(\zeta(i,j+1) - \zeta(i,j-1))). \end{aligned}$$

3 Program

3.1 Usage

Initially, the program displays a menu from which the user can select a partial differential equation solver. Afterwards the grid size is asked for. The grid size is the number of cells of one edge of the cavity. The cavity is defined internally for a grid size of 30. If a higher size is specified the REYNOLDS number and other parameters are scaled automatically by the program to compute the same system, i.e. the system defined using a size of 30, but with a finer grid. If the PDE solver is overrelaxed then a value for the overrelaxation parameter will be requested. In the case that a PDE solver for viscous flow was chosen the program prompts for a REYNOLDS number. Afterwards the program commences computing the velocity field.

3.2 Structure

The program is written in FORTRAN77 and comprises of four parts. The first part is the main program which interacts with the user and contains the main loop to compute the fluid flow. In the second part you find supporting procedures. These set up the initial configuration, impose the boundary conditions, compute the velocity field and calculate the residue of a given stage in the solution of the equations. The third and fourth part are the routines for solving the partial differential equations for both the inviscid and the viscous flow in the cavity.

Parts of the program are implemented parallelly. The parallelisation is done using *OpenMP*[3]. All of the supporting procedures are implemented serially and parallelly. Some of the partial differential equations solver are, too, parallelised.

[3] OpenMP is described in [2].

4 Results and Discussion

4.1 Inviscid flow

Figure 1 shows the inviscid flow in the cavity. On this graph there are no vortices visible. This is due to the inviscid approximation, because the friction at the walls of the cavity are not taken into consideration. In an inviscid flow vortices are neither created nor destroyed. This is called *permanence of irrotational motion*[4]. As there are no whirlpools in the beginning, the array of the stream function is set to zero initially, vortices cannot be observed. Figure 1 represents a very slow flow, i. e. a flow with small REYNOLDS number.

The residue of the computation of the inviscid flow is shown in figure 2. This result was obtained using single precision floating point arithmetic in an early version of the program. The residue drops rapidly in the beginning. Afterwards it decreases linearly on the half-logarithmic scale used in the plot. At high numbers of iteration rounding errors start to become important and cause the curve to fluctuate irregularly. This is not the case for the version of the program using double precision numbers, confer figure 6.

Figure 3 and 4 show the residue of the GAUSS-SEIDEL algorithm both with lexicographic and red-black checkerboard updating scheme. The small difference between the numbers of iteration of the two updating schemes, 1157 versus 1155 is strongly influenced by effects of rounding errors. Carrying out the same calculations with the double precision version of the program one needs 1172 iterations for lexicographic and 1203 iterations for red-black ordering. In this case which is much more reliable than the former the lexicographic updating was faster. This is fairly obvious because under red-black update every second site is updated, so the effect of the boundary conditions spreads more slowly in the beginning. Figure 5 shows that the difference in the residue of the two algorithms becomes constant after a couple of updates. Then both PDE solver are equally fast.

The convergence rate of JACOBI versus GAUSS-SEIDEL is visualised in figure 6. For the JACOBI algorithm the predicted number of iterations[5] required to reduce the error by a factor 10^{-p} is

$$n_{\mathrm{J}}(p) = \frac{p \ln 10}{\ln \rho_{\mathrm{J}}} \qquad \rho_{\mathrm{J}} = 1 - \frac{\pi^2}{2L^2},$$

where L denotes the grid size and ρ_{J} is the spectral radius of the matrix describing the discretised partial differential equation. Setting $p = 6$ and $L = 30$ the predicted number of iterations is $n_{\mathrm{J}}(6) = 2513$. This is in satisfactory agreement with the actual result 2346 for the double precision version of the program. The GAUSS-SEIDEL algorithm is approximately twice as fast as the JACOBI code. Dividing $n_{\mathrm{J}}(6)$ by two gives $n_{\mathrm{GS}}(6) = 1257$ which compares well to 1172 iterations for lexicographic and 1203 iterations for red-black ordering.

To improve the convergence rate of the GAUSS-SEIDEL algorithm overrelaxation was implemented. Figure 7 shows a graph of the number of iterations required to achieve an accuracy of $\approx 10^{-6}$ with varying ω. The minimum of the graph is $\omega_{opt} = 1.82 \pm 0.01$. An

[4] See page 114 of [3]

[5] See page 35 of [1] for a discussion.

estimate of the expected value of ω_{opt} is[6]

$$\omega_{opt} \approx \frac{2}{1 + \frac{\pi}{L}}.$$

The grid is size $L = 30$ yielding $\omega_{opt} = 1.810$ in good agreement with the formerly found value. In the case that there is no known analytic value for the optimal overrelaxation parameter a good approximation can be found using a coarse grid or an otherwise simplified system and then employing the techniques of this paragraph.

4.2 Viscous flow

The viscous flow is much more fascinating than the inviscid flow due to the formation of vortices. Figures 12, 14, 16, 18 and 20 show the velocity field in the cavity with varying REYNOLDS numbers. The vortices grow with increasing REYNOLDS numbers.

For $Re > 3.4$ the standard GAUSS-SEIDEL algorithm becomes unstable and the program terminates abnormally due to floating point errors. Employing under-relaxation one can compute the velocities for much higher REYNOLDS numbers. See figures 22, 23 and 24. Here the grid size is 100.

The two algorithms are only suitable to solve small problems with stable equations. For bigger tasks one would prefer to chose for example a *conjugate gradient* algorithm which is more sophisticated but yields better results.

4.3 Parallelisation

The JACOBI code is easily parallelised because the algorithm is independent of the order in which the individual cells are updated.

To parallelise the GAUSS-SEIDEL algorithm one has to loop over the cells in red-black ordering[7]. With this scheme one can update the red cells in random order keeping the black cells fixed. Afterwards the black cells are updated and the red cells are kept fixed.

The *speed-up* and the *efficiency*[8] were measured using the real elapsed time for the program execution. The reference time for one processor was measured using the serial versions of the algorithms. The data points for one processor in the graphs 8, 9, 10 and 11 are the *speed-up* and the *efficiency* of the parallel against the serial version of the algorithms. Why both values are better than theoretically possible is not clear. Probably this is due to errors, the operating system or the *OpenMP* implementation.

The fact that the real elapsed time was used for all measurements causes a considerable amount of error. For example the curve for grid size 30 on figure 10 is heavily influenced by errors. Nevertheless some trends can be seen. Figures 8 and 10 show that both JACOBI and GAUSS-SEIDEL algorithm scale well. This is confirmed by figures 9 and 11 which show the efficiency of the two algorithms. The far better *speed-up* and *efficiency* for the grid size 60 compared to the grid size 30 comes from the problem itself. The amount of work that needs to be done to achieve a given accuracy scales like L^2.

[6] This is equation (4.20) on page 36 in [1].

[7] For a discussion see page 33 of [1].

[8] For a definition of these terms see page 27 of [1].

5 Conclusion

Although the project deals with computational fluid dynamics it is a representative of a whole class of similar problems where the underlying physics is described by *partial differential equations*. Thus the exercise shows the possibilities and constraints of the employed algorithms not only in this special case but also in many areas of physics.

The advantages of the JACOBI and the GAUSS-SEIDEL algorithm are their simplicity and paralleliseability but they are slow and instable.

The project shows that even very small physical problems may need powerful computers to solve them numerically.

6 References

[1] Lecture notes to *High Performance Computing in Physics*.
[2] Technology Watch Report of the Edinburgh Parallel Computing Centre (EPCC) on OpenMP:http://www.epcc.ed.ac.uk/epcc-tec/documents/techwatch-openmp/form2.html .
[3] D. J. Tritton, Physical Fluid Dynamics, Second Edition 1988, Oxford University Press