ISSN 1870-4069

Simulation of Newtonian Flows on Sudden Contraction Geometries: GPU Implementation

Rigo Alvarado[1], Juan J. Tapia[1], Hector D. Ceniceros[2]

[1] Instituto Politécnico Nacional-CITEDI, Tijuana, BC, Mexico
ralvarado@citedi.mx, jtapiaa@ipn.mx
[2] University of California Santa Barbara, UCSB, Santa Barbara, California, USA
hdc@math.ucsb.edu

Abstract. In this paper, a GPU-based simulation of a Newtonian fluid flow on sudden contraction geometries is presented. The fluid is modeled with the Navier-Stokes equations and solved by the projection method with first order in time and second order in space discretizations. A semi-implicit scheme of finite differences is used in the dicretization process. The solution of the resulting system of linear equations is considered as an optimization problem and is solved by the preconditioned biconjugate gradient stabilized method (BiCGSTAB) implemented on a graphic processor using the CUDA libraries cuSPARSE and cuBLAS.

Keywords: sudden contraction geometry, Navier-Stokes equations, GPU, CUDA.

1 Introduction

Sudden contraction geometries for different fluids are used in many areas of engineering and industrial processes such as heating pipes, polymer processing, tube capillary viscometers, biomedical instruments, thermoforming, injection molding, etc. Therefore, understanding and predicting the behavior of fluids in these geometries is of particular importance within fluid mechanics and is a problem that is continuously studied from different perspectives.

For example, in [6], a hybrid model reduction scheme to approximate the Navier-Stokes equations (NSE) with a low-dimensional model on a contraction geometry is presented. The use of a novel projection method based on midpoint rule for the solution of NSE is discussed in [13]. In [8], Guermond and Minev develop a high-order time approximation for the NSE as an alternative for the classical projection method.

However, the high computational cost involved with this type of studies and simulations is a problem that any investigation related to Computational Fluid Dynamics (CFD) and, therefore, researches related to flow on contraction geometries must overcome. This intensive computation is the product of many factors such as the size of the meshes necessary for the correct discretization, the physical domain that needs to be simulated and the number of variables that must be solved. Therefore, the parallelization of CFD simulations is a topic that constantly receives attention and new proposals.

For example, in [17], Willis presents an MPI implmentation for fluid flow in pipelines. GPU/multicore-based solution to CFD simulations using the NSE are described in [4,10,12] and [16]. An OpenMP-based solution for the NSE on a cavity lid driven is developed in [1].

Therefore, we can state that the motivation for this work and its objective it is very clear: the GPU-based parallelization of the simulation of flow on sudden contraction geometries in order to decrease the computing time and improve the efficiency of the utilization of hardware.

2 Mathematical Model

The Navier-Stokes equation models the fluids movement. The non-dimensional vector form of this equation for incompressible fluids and its incompressibility constraint are defined [11] as:

$$\begin{aligned}\frac{\partial \mathbf{v}}{\partial t} + \nabla \cdot (\mathbf{v} \otimes \mathbf{v}) = -\nabla p + \frac{1}{Re}\nabla^2 \mathbf{v},\\ \nabla \cdot \mathbf{v} = 0,\end{aligned} \tag{1}$$

where $\mathbf{v}$ is the adimensional vectorial field of velocity and p is the adimensional scalar field of pressure. In 2D, NSE involves three equations and three unknowns, u, v and p.

Reynolds number is a dimensionless quantity defined as:

$$Re = \frac{\rho V_o L_o}{\mu}, \tag{2}$$

where V_0 is the reference velocity, L_0 is the characteristic length and ρ is the fluid density. It represents the ratio between inertial and viscous forces. If it is a small value, the flow occurs in parallel lines and is called laminar flow; if the Reynolds number increases, the ordered structure loses its order, giving rise to a flow characterized by eddies and vortexes, called turbulent flow.

3 Numerical Solution

3.1 Projection Method

The main idea of the projection method is to temporarily solve the NSE (Eq. (1)) by omitting the pressure p and then project the result into a vector field of solenoid velocity [2,3,14]. The process comprises three main steps:

– First, a temporary velocity field $\mathbf{v}^* = [u^*, v^*]$ is defined as:

$$\mathbf{v}^* = \mathbf{v}_{sol}^{t+1} + \Delta t \nabla p^{t+1}. \tag{3}$$

Algorithm 1 Projection method for NSE for incompressible fluids

```
Initialization of variables, arrays and differentiation matrices
Initial condition for velocity field
for $iter = 1 : MAX$
 Update of boundary conditions
 Compute $u^*$ and $v^*$
 Solve Poisson equation for pressure $p^{t+1}$
 Calculate new velocities $u^*$ and $v^*$ with $p^{t+1}$
end
```

As pressure is neglected, the rectangular components of (1) are:

$$\frac{u^* - u^t}{\Delta t} + u^t \cdot \nabla u^t = \frac{1}{Re} \nabla u^*, \qquad \frac{v^* - v^t}{\Delta t} + v^t \cdot \nabla v^t = \frac{1}{Re} \nabla v^*, \tag{4}$$

for the horizontal and vertical components, respectively.

– Next, the divergence of (3) is calculated to obtain the Poisson equation:

$$\Delta t \nabla^2 p^{t+1} = \nabla \cdot \mathbf{v}^*. \tag{5}$$

The resulting pressure p^{t+1} calculated with (5) is a scalar field that ensures that the final velocity $\mathbf{v}_{sol}^{t+1}$ will meet the incompressibility constraint.

– At last, the final velocity $\mathbf{v}_{sol}^{t+1}$ is computed as:

$$\mathbf{v}_{sol}^{t+1} = \mathbf{v}^* - \Delta t \nabla p^{t+1}. \tag{6}$$

Algorithm 1 shows the pseudo-code for the described projection method.

3.2 Problem Specification

We choose the 2:1 and 4:1 ratio contraction geometries (Fig. 1) because they are among the most used in investigations and practical applications [6,17]. Furthermore, the dimensions utilized were chosen to improve numerical accuracy at the contraction region. Also, it is important to mention that this 2D representation corresponds to a longitudinal cross-section of a pipe.

Boundary conditions for the numerical solution are: for the velocity field $\mathbf{v}$, no-slip conditions at the wall ($u = v = 0$), $u = u_j$ (parabolic profile) and $v = 0$ at the inlet (inflow) and $\frac{\partial \mathbf{v}}{\partial \mathbf{n}} = 0$ at the outlet (outflow). The parabolic profile of the horizontal component u of velocity is defined as:

$$u_j = u_{max} \left[1 - \left(\frac{r^2}{R^2} \right) \right], \tag{7}$$

where u_{max} is the maximum value of u, right in the middle of the inflow, R is the inlet radius, r is the radius of the j-th element of u at the inlet and j is the vertical index for each one of these elements. For the scalar field of pressure p, homogeneous Neumann boundary conditions are used ($\frac{\partial p}{\partial \mathbf{n}} = 0$) at all boundaries except at the outlet, where $p = 0$. The described boundary conditions are used for both contractions, i.e. geometries 4:1 and 2:1.

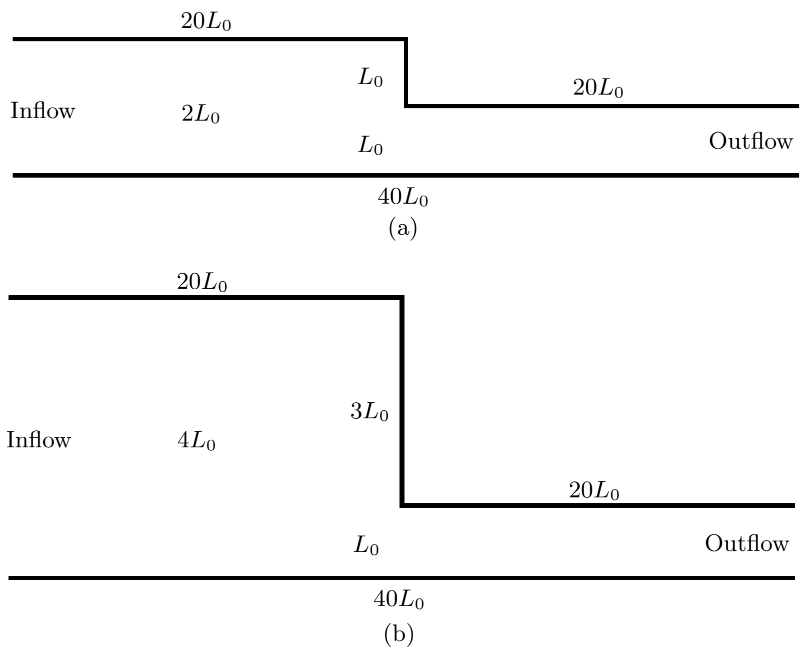


Fig. 1. Sudden contraction geometry (a) 2:1 and (b) 4:1

3.3 Spatial Discretization

Domain is discretized with a regular mesh and centered finite differences of order $O(h^2)$. Spatial step size for both components depends upon the geometry. For contraction 2:1 (Fig. 1 (a)), $\Delta x = \frac{20L_0}{\mathrm{COLS}}$ and $\Delta y = \frac{2L_0}{\mathrm{ROWS}}$ where $COLS = 20N$ represents the maximum number of cells in the horizontal direction (x axis), $ROWS = 2N$ represents the maximum number of cells in the vertical direction (y axis) and $L_0 = 1$ is the width of the outlet. For contraction 4:1 (Fig. 1 (b)), $\Delta x = \frac{20L_0}{\mathrm{COLS}}$ and $\Delta y = \frac{4L_0}{\mathrm{ROWS}}$ where $ROWS = 4N$ represents the maximum number of cells in the vertical direction. Constant N is an integer number, multiple of two, that is utilized to keep the right proportion between the width and length of the domain. In all simulations presented in this work, $\Delta x = \Delta y$. Also, in order to avoid a checkerboard solution, a full-staggered mesh (Fig. 2) is used in all cases [9].

Discretization for $[u^*, v^*]$ (Eq. (4)) and pressure p (Eq. 5) produces the linear systems of equations:

$$Au^* = RHS_{u^*}, \qquad Bv^* = RHS_{v^*}, \qquad Cp^{t+1} = RHS_p, \tag{8}$$

respectively, where A and B are sparse positive definite matrices, at least for Δt reasonably small, and C is a sparse semi-positive definite matrix. Also, A, B and C are non-symmetrical. For these reasons, the solution of these systems is con-

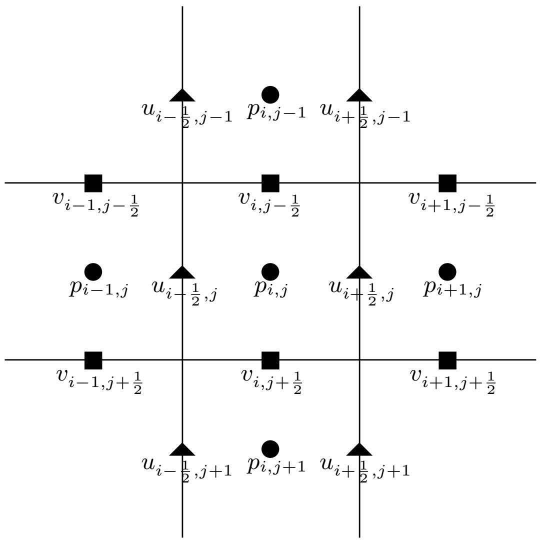


Fig. 2. Full-staggered mesh, $2D$

sidered as an optimization problem and hence, the preconditioned BiCGSTAB method is utilized to solve them.

To compute the final velocity, Eq. (6) is discretized as:

$$\begin{aligned} u^{t+1}_{i+1/2,j} &= u^{*}_{i+1/2,j} - \Delta t \frac{p^{t+1}_{i+1,j} - p^{t+1}_{i,j}}{\Delta x}, \\ v^{t+1}_{i,j+1/2} &= v^{*}_{i,j+1/2} - \Delta t \frac{p^{t+1}_{i,j+1} - p^{t+1}_{i,j}}{\Delta y}, \end{aligned} \tag{9}$$

for its horizontal and vertical components respectively.

4 General Purpose Computing on GPU: GPGPU

The use of GPUs to solve compute-intensive scientific and engineering applications is known as General-Purpose computing on Graphics Processing Units (GPGPU).

CUDA is a combination of a GPU hardware and a parallel programming model that allows the utilization of NVIDIA GPUs in GPGPU applications.

4.1 CUDA Programming Model

The heterogeneous CUDA programming model enables the use of a GPU as a co-processor of the CPU. In this context, GPU is called device and CPU is called host. A CUDA program is composed of serial code sections for the host (on some applications, as in this work, sections of the serial code can be parallelized with

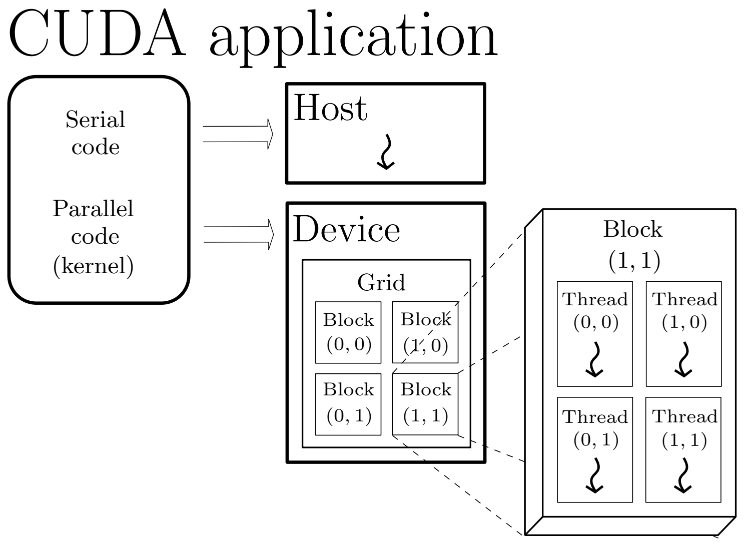


Fig. 3. CUDA programming model

OpenMP, MPI, pthreads, etc.) and parallel code sections for the device, called kernels. The serial code is executed by the main thread on the host; the kernels are executed in parallel within the device by a massive number of CUDA threads. This general structure is shown in Fig. 3.

4.2 cuSPARSE and cuBLAS Libraries

NVIDIA cuSPARSE library contains a set of basic linear algebra subroutines designed for sparse matrix operations that takes advantage of the CUDA parallel programming model as well as of the computational resources of the NVIDIA GPUs in order to perform efficiently its functions.

NVIDIA cuBLAS library is an implementation of BLAS (Basic Linear Algebra Subprograms) for dense matrices on top of the CUDA runtime. The library

Algorithm 2 CUDA-based implementation overview

```
CUSPARSE-CUBLAS initialization
Compute $ilu(0)$ preconditioners for matrices $A$, $B$ and $C$ (cuSPARSE)
Initial condition for $\mathbf{v}$
for $iter = 1 : MAX$
  Update BC
  Compute $RHS_{u^*}$ and $RHS_{v^*}$ (kernels)
  Solve $Au^* = RHS_{u^*}$ and $Bv^* = RHS_{v^*}$ (cuSPARSE-cuBLAS,BICGSTAB)
  Compute $RHS_p$ (kernel)
  Solve $Cp^{t+1} = RHS_p$ (cuSPARSE-cuBLAS,BICGSTAB)
  Update of $u^*$ and $v^*$ with $p^{t+1}$
end
```

includes matrix-vector and matrix-matrix products. The cuBLAS library also provides functions for writing and retrieving data from the GPU.

CUSPARSE and CUBLAS can be used with C and C++ programming languages. To use the functions of both libraries, they should be initialized, the data must be transferred from the host to the GPU memory and then it must be converted to the corresponding format.

5 Parallel Implementation

The systems of linear equations (8) are not suitable to be solved with direct methods due data dependency. Therefore, we choose to solved them as an optimization problem with the BICGSTAB method. Every step of the general BICGSTAB is parallelized with combinations of functions of cuSPARSE and cuBLAS libraries; the $ilu(0)$ preconditioner is also implemented with functions of the cuSPARSE library. In order to use the functions of these libraries, the arrays are copied to the GPU memory and then they are converted to the required formats of sparse matrices. Also, we utilized kernels to compute the RHS vectors for each of the aforementioned systems.

To check the GPU implementation and utilization we used the nvprof and nvvp profilers. The general structure of our implementation is shown in Algorithm 2.

6 Results

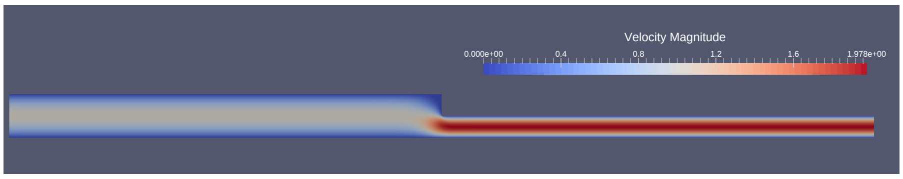
The simulations have been conducted under Ubuntu 14.04 *LTS* using C language on a computer with an *intel i7-4770* processor at 3.4 GHz and 16 GBytes of RAM memory. The CUDA implementation is made with a Maxwell *GeForce GTX* 970 GPU with compute capability 5.2 that has 4 GBytes of global memory. All the results presented are double precision floating point.

The results presented in Section 6.1 and 6.2 are computed with $Re = 1$, which means that there is a balance between inertial and viscous forces. This balance translates into laminar flows, such as the ones shown in those sections. On the other hand, some simulations of flows in which $Re >> 1$ are presented in the Section 6.3.

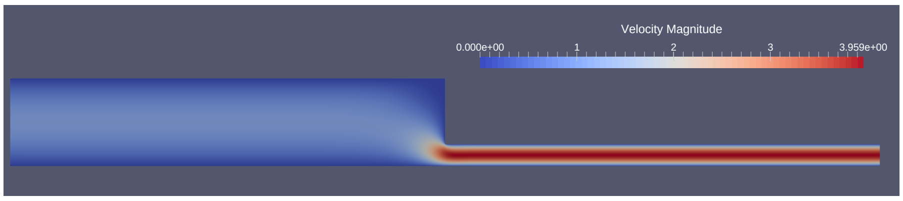
6.1 Contraction 2:1

Magnitude of velocity for a contraction 2:1 after 500 iterations is shown in Fig. 4 (a). Domain is discretized with $N = 128$. Furthermore, $u_{max} = 1$, $\Delta t = 0.003$ and $Re = 1$.

Simulation shows that the maximum velocity at the outflow is twice its value at the inflow, i.e. twice the value of u_{max}. This agrees with the Bernoulli equation, since the pressure in the outlet is smaller than the pressure in the inlet (Fig. 5 (c)) and, therefore, the velocity at the outlet is greater than the velocity at the

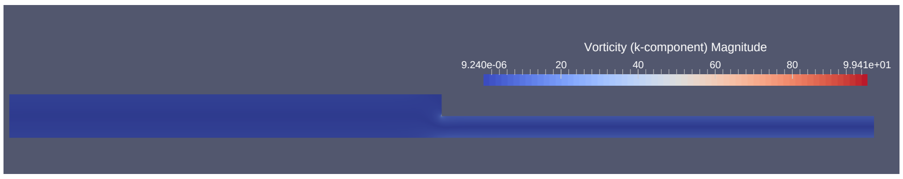


(a)

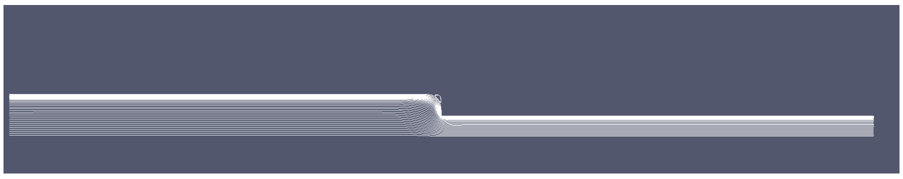


(b)

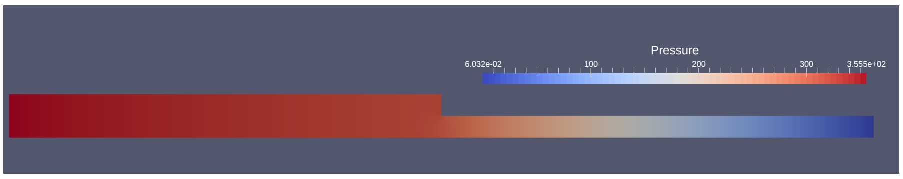
Fig. 4. Magnitude of velocity: (a) Contraction 2:1 and (b) Contraction 4:1.



(a)



(b)



(c)

Fig. 5. Contraction 2:1. (a) Vorticity magnitude. (b) Streamlines. (c) Pressure

inlet to preserve the total energy of the flow. This behavior is known as the Bernoulli effect.

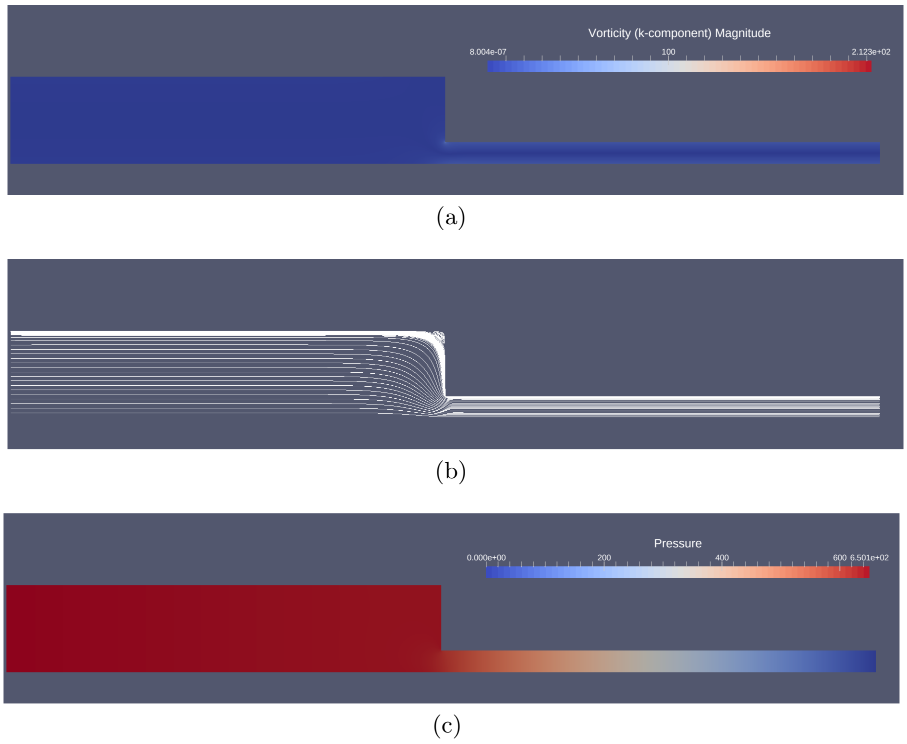


Fig. 6. Contraction 4:1. (a) Vorticity magnitude. (b) Streamlines. (c) Pressure

Magnitude of vorticity, defined as:

$$\omega = \nabla \times \mathbf{v}, \tag{10}$$

is shown in Fig. 5 (a).

It can be seen that the corner at the entrance to the narrow part of the geometry is the point where the vorticity has the largest value. Also, it can be noted that ω has a relatively high value close to the walls of the narrow section of the contraction. This is due to the fact that the fluid velocity is larger in this section and thus, the difference with the null velocity at the walls is greater here than everywhere else.

Streamlines, i.e. the paths or curves in which the function *stream* ψ, computed as:

$$-\,||\omega|| = \nabla^2 \psi, \tag{11}$$

has a constant value, are shown in Fig. 5 (b). These curves represent the path that would follow a massless particle in the flow and they are instantaneously tangent to the velocity vector.

6.2 Contration 4:1

The magnitude of velocity for a contraction 4:1 after 500 iterations is shown in Fig. 4 (b). Domain is discretized with $N = 128$. Furthermore, $u_{max} = 1$, $\Delta t = 0.003$ and $Re = 1$.

Similarly as in the simulation for contraction 2:1, in this case, the velocity and pressure values comply with the Bernoulli effect.

Vorticity magnitude, streamlines and pressure are shown in Fig. 6 (a), (b) and (c) respectively. It can be seen that the point for maximum vorticity, streamlines pattern and pressure behavior are similar to those obtained for contraction 2:1.

6.3 Simulations for High-Re Flows

As the value of Re increases, inertial forces overpower viscosity forces and the flow loses its laminar structure and turns into a turbulent flow, which is characterized by eddies, vortexes and chaotic changes in the velocity and pressure.

Fig. 7 (a) and (b), show a close-up of the eddies that arise at the wall of the contraction 2:1 with $Re = 100$ and $Re = 1000$, respectively. Likewise, turbulences generated in the contraction 4:1 for fluids with $Re = 100$ and $Re = 1000$, are shown in Fig. 7 (c) and (d), respectively. For both geometries, $\Delta x = \Delta y = \frac{1}{128}$ and 500 iterations were computed. For the simulations with $Re = 100$, $\Delta t = 0.003$ and for the simulations with $Re = 1000$, $\Delta t = 0.0003$. The flow patterns obtained coincide with results presented in [7] and [15].

6.4 Convergence Analysis

Order m of the implementation can be calculated with [5]:

$$m = \frac{\log\left(\frac{f_{\frac{h}{2}}(x,y)-f_h(x,y)}{f_{\frac{h}{4}}(x,y)-f_{\frac{h}{2}}(x,y)}\right)}{\log 2}, \tag{12}$$

if h is sufficiently small and succesive meshes are related by a 2:1 proportion. Discretization error e over the finest mesh is defined as [5]:

$$e_{\frac{h}{4}} \approx \frac{f_{\frac{h}{4}}(x,y) - f_{\frac{h}{2}}(x,y)}{2^m - 1}. \tag{13}$$

In this work, for both geometries, m is computed with $N = 32$, $N = 64$ and $N = 128$. Time step Δt is fixed as $(\frac{1}{128})^2$ because the implemented projection method is first order in time and second order in space. As a result, Δt should be equal or smaller than the step size of the finest mesh, i.e. $\Delta t \leq \Delta x$. This is to observe second order in Δx convergence.

Table 1 shows the results of the analysis for both velocity components of both geometries. The results denote a proper implementation.

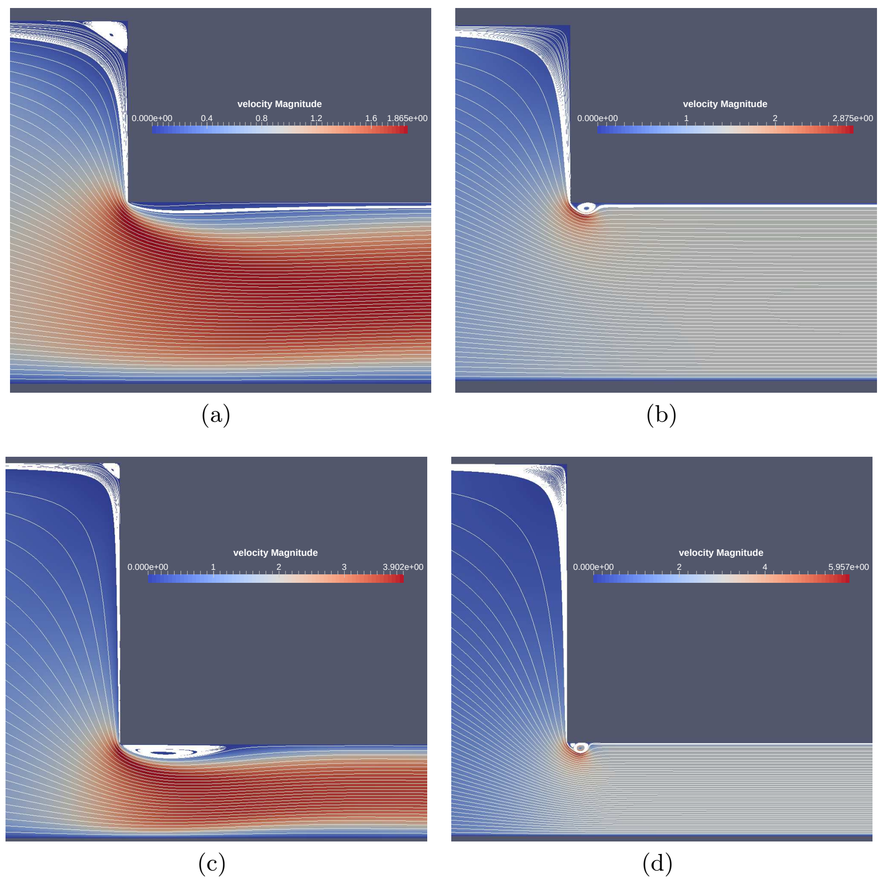


Fig. 7. Contraction 2:1. (a) and (b), zoom in of vortexes with $Re = 100$ and $Re = 1000$ respectively. Contraction 4:1. (c) and (d), zoom in of vortexes with $Re = 100$ and $Re = 1000$ respectively

Table 1. Order m and error e estimation after 500 iterations for $N = 128$

Component	**Contraction 2:1**		**Contraction 4:1**	
	m	e	m	e
u	2.016	1.969×10^{-5}	2.006	2.230×10^{-5}
v	1.897	6.900×10^{-7}	1.959	3.8708×10^{-12}

Execution time for all the meshes utilized in the convergence analysis are shown in Table 2 and 3, for contraction 2:1 and 4:1, respectively. It is important to state that the pressure matrix C has a high value condition number and, because of this characteristic, it converges slower than the other matrices. This issue contributes adversely on the computation time. However, we are currently working on the solution of this problem with the ADI method.

Table 2. Execution time for 500 iterations for different meshes, contraction 2:1

N	Δx, Δy	**Points**	**Cells**	**Execution time** (min)
32	0.03125	62785	61440	247.2
64	0.015625	248449	245760	1117.2
128	0.0078125	988417	983040	5991.7

Table 3. Execution time for 500 iterations for different meshes, contraction 4:1

N	Δx, Δy	**Points**	**Cells**	**Execution time** (min)
32	0.03125	103809	102400	306.2
64	0.015625	412417	409600	1347.5
128	0.0078125	1644033	1638400	8982.5

7 Conclusions and Future Work

The flow simulations on sudden contraction geometries described in this paper, use the functions of the CUSPARSE and CUBLAS libraries to implement on a GPU the preconditioned BiCGSTAB method. This method is used to solve the systems that generates the discretization of the Navier-Stokes equations and, therefore, its appropriate parallelization has a significant impact in the reduction of computation time, which is the main objective of this work.

The described method that numerically estimates the order and error of the discretization, can not only be applied for simulations of contraction geometries; is a general procedure that can be used for other CFD applications as well as for discretizations that arise from the numerical solution of differential equations related to other areas of science.

As future work, we will be working with simulations of non-Newtonian fluxes, e.g. Viscoelastic fluids in the sudden contraction geometries described herein; this is the final stage of our investigation. Meanwhile, we are going to tackle the problem of pressure matrix C with the ADI method and Thomas algorithm, a combination that appropriately fits a GPU application. Also, GPU clusters will be used to simulate larger physical domains.

Aknowledgements. This material is based upon work supported by a grant from the University of California Institute for Mexico and the United States (UC MEXUS) and the Consejo Nacional de Ciencia y Tecnología de México (CONACYT).

References

1. Alamsyah, M.N.A., Simanjuntak, C.A., Bagustara, B.A.R.H., Pradana, W.A., Gunawan, P.H.: Openmp analysis for lid driven cavity simulation using lattice boltzmann method. In: ICoICT '17. pp. 1–6 (May 2017)
2. Chorin, A.J.: The numerical solution of the Navier-Stokes Equations for an incompressible fluid. Bull. Amer. Math. Soc 73(6), 928–931 (1967)
3. Chorin, A.J.: Numerical solution of the Navier-Stokes Equations. Math. Comp. 22(104), 745–762 (1968)
4. Deng, L., Fang, J., Wang, F., Bai, H.: Evaluating multi-core and many-core architectures through accelerating an alternating direction implicit cfd solver. In: ISPDC '16. pp. 1–10 (July 2016)
5. Ferziger, J.H., Peric, M.: Computational methods for fluid dynamics. Springer, Berlin, third edn. (2002)
6. Ge, X., Wen, J.T.: Hybrid model reduction for compressible flow controller design. In: IEEE CDC '11. pp. 6540–6545 (Dec 2011)
7. Griebel, M., Rüttgers, A.: Multiscale simulations of three-dimensional viscoelastic flows in a square-square contraction. J. Nonnewton Fluid Mech. 205, 41–63 (2014)
8. Guermond, J.L., Minev, P.: High-order time stepping for the incompressible Navier-Stokes equations. SISC 37(6), A2656–A2681 (2015)
9. Harlow, F.H., Welch, J.E.: Numerical calculation of time-dependent viscous incompressible flow of fluid with free surface. Phys. Fluids 8(2), 2182–2189 (1965)
10. Hashimoto, T., Yasuda, T., Tanno, I., Tanaka, Y., Morinishi, K., Satofuka, N.: Multi-gpu parallel computation of unsteady incompressible flows using kinetically reduced local Navier-Stokes equations. In: Computers and Fluids. vol. 167, pp. 215–220 (2018)
11. Hoffmann, K.A., Chiang, S.T.: Computational Fluid Dynamics, vol. First. Engineering Education System, Kansas, fourth edn. (2000)
12. Huang, J., Lin, Z., Ma, C., Yuan, X.: Gpu speed-up for the implicit Navier-Stokes solver. In: Proceedings of the ASME Turbo Expo. vol. 6 (Jun 2013)
13. Lovrić, A., Dettmer, W.G., Kadapa, C., Perić, D.: A new family of projection schemes for the incompressible Navier-Stokes equations with control of high-frequency damping. Comput Methods Appl Mech Eng 339, 160–183 (2018)
14. Temam, R.M.: Sur l'approximation de la solution des équations de navier-stokes par la méthode des pas fractionnaires (ii). Arch. Rational Mech. Anal. 33(5), 377–385 (1969)
15. Trebotich, D.P., Colella, P., Miller, G.H.: A stable and convergent scheme for viscoelastic flow in contraction channels. J. Comput. Phys. 205(1), 315–342 (2005)
16. Wang, Y., Baboulin, M., Rupp, K., Le Maître, O., Fraigneau, Y.: Solving 3d incompressible Navier-Stokes equations on hybrid cpu/gpu systems. In: HPC '14. pp. 12:1–12:8. San Diego, CA, USA (2014)
17. Willis, A.P.: The openpipeflow Navier-Stokes solver. SoftwareX 6, 124–127 (2017)