REVISTA
INTERDISCIPLINAR DE
PESQUISA EM
ENGENHARIA

HIGH PERFORMANCE COMPUTING ARCHITECTURE FOR FLUID DYNAMICS AND FLUID-STRUCTURE INTERACTION

Luiz Felipe Marchetti do Couto

Henrique Campelo Gomes

Paulo de Mattos Pimenta

luiz.couto@usp.br

henrique.campelo@usp.br

ppimenta@usp.br

Departament of Structural and Geotechnical Engineering, Polytechnic School at University of São Paulo

Av. Prof. Almeida Prado, travessa 2, 83, Cidade Universitária, São Paulo, SP, 05508-900, Brazil

Abstract. *One of the biggest challenges of engineering is enable computational solutions that reduce processing time and provide more accurate numerical solutions. Proposals with several approaches that explore new ways of solving such problems or improve existing solutions emerge. Some of the areas dedicated to propose such improvements is the parallel and high performance computing. Techniques that improve the processing time, more efficient algorithms and faster computers open up new horizons allowing to perform tasks that were previously unfeasible or would take too long to complete. We can point out, among several areas of interest, Fluid Dynamics and Fluid-Structure Interaction. In this work it was developed a parallel computing architecture in order to solve numerical problems more efficiently, compared to sequential architecture (e.g. Fluid Dynamics and Fluid-Structure Interaction problems) and it is also possible to extend this architecture to solve different problems (e.g. Structural problems). The objective is to develop an efficient computational algorithm in scientific programming language C ++, based on previous work carried out in Computational Mechanics Laboratory (CML) at Polytechnic School at University of São Paulo, and later with the developed architecture, execute and investigate Fluid Dynamics and Fluid-Structure Interaction problems with the aid of CML computers. A sensitivity analysis is executed for different problems in order to assess the*

best combination of elements quantity and speedup, and then a perfomance comparison. Using only one GPU, we could get a 10 times speedup compared to a sequential software, using Finite Element with Immersed Boundary Method and a direct solver (PARDISO).

Keywords: *Fluid-Structure Interaction, Finite Elements, High Performance Computing, GPU, CUDA*

1 INTRODUCTION

Nowadays the biggest challenges of engineering is to enable computational solutions that reduce processing time and provide more accurate numerical solutions. Proposals that explore new ways of solving such problems or improve existing solutions emerge. One of the biggest areas that we can point out is the parallel and high performance computing. Techniques that improve the processing time, more efficient algorithms and faster computers open up new horizons allowing to perform tasks that were previously unfeasible or would take too long to complete.

Fluid-Structure Interaction (FSI) is a problem that involves fluid dynamics and structures, and the solution of one problem depends on the solution of the other. Turning it into a coupled system. If solving a Fluid Dynamics problem with different boundary conditions is a difficult task, a FSI adds the challenge of solving a simultaneous solution of the coupled system where the boundary conditions of the interface between the fluid and the structure are unknown a priori, since they depends on the solution of the problem itself. Besides, some problems of interest in engineering involves big displacements of the structure and fluid convection, so that FSI is a strongly non-linear problem.

Some of the challenges that numerical modeling of FSI offers are, among others, the spatial domain occupied by the fluid changes in time as the interface moves and the mathematical model to handle that. Accurate representation of the flow field near the fluid-structure interface requires that the mesh be updated to track the interface and this requires special attention in three-dimensional problems with complex geometries (Tezduyar *et al.*, 2005).

One difficult that we could mention is simulating problems of this nature due to high speed fluid flow that normally would require complex turbulence models (Gamnitzer *et al.*, 2010). Add to that the fact that simulation time, in general, needs to be wide, so we could visualize the physical phenomena, and short time steps, one demand of any numerical simulation of fluids.

This combination, naturally, demands high performance computing and a model reduction order (Lieu *et al.*, 2006), and parallel computing.

This work is divided into four main sections. The first section will cover the parallel computing. The second section will cover the FSI with Immersed Boundary method theory. The third section will handle the polygon integration using the theory described in (Sudhakar *et al.*, 2014). The last main section will present some numerical simulations (only Fluid Dynamics with Immersed Boundary Methods) in order to test and validate the software implementation and compare the results from a sequential and a parallel execution.

2 PARALLEL COMPUTING

2.1 Sequential computing

Sequential computing has been used for more that 60 years, since John Von Neumann created digital computing in 50s. It is defined as a system that has a central processing unit (CPU)

and a memory unit. The processing speed of any application depends, mainly, on the instruction execution rate, cycles per second (clock) and transfer rate (bandwidth) between memory and CPU.

Gordon E. Moore.

Moore's law was established by Gordon Earl Moore and says that the number of transistors inside a microprocessor will double every two years.

> "The complexity for minimum component costs has increased at a rate of roughly a factor of two per year. Certainly over the short term this rate can be expected to continue."

Microprocessors based on a single CPU (i.e. Intel Pentium family and AMD) obtained an exponential growing in processing speed and cost reduction in the last decades. CPUs got to GFLOPs (Giga Floating-point Operations per Second) in usual desktops to hundreds GFLOPs in clusters.

2.2 Parallel computing

Developers believed that improvements in hardware would increase the processing speed and software execution. However this growing has shrunk since 2003 due to high energy consumption and heat dissipation problems inside processors, limiting the clock frequency and the efficiency on each clock period in a single CPU.

This way, microprocessors suppliers and producers changed CPU internal technology so that they could use multiple processing unit (Multi-core processors). This change in hardware had a major impact in developers software community.

Latency vs. Throughput.

The following concepts are important and contradictories and are usually used to decide which approach could be applied to a specific hardware or software.

- Latency or execution time is the time needed to a task to be concluded. It is measured in time units or clock periods.
- Throughput or bandwidth is the number of tasks that must be executed in a specific time. It is measured in units of something that is been produced (I/O, iterations, transfered memory) per unit of time.

Some examples:

- Clock frequency: $100\,\mathrm{MHz}$
- Throughput of a device: $640\,\mathrm{Mbits/second}$

Speedup.

Speedup is a performance metric to determine how much the performance of a system is inferior or superior to other system.

Example:

- System A executes a specific task in 200 cycles;
- System B executes a specific task in 350 cycles;
- $350/200 = 1.75$, system B é 1.75 times faster than system A.

In Molyneaux *et al.* (2009) an extensive analysis of a series of performance metrics is discussed that could be used whether in hardware or in software, as well as in computer networks among other applications.

Gene M. Amdahl.

In Amdahl *et al.* (1967), Gene M. Amdahl says:

> "... For over a decade, single computer has reached its limits and truly significant advances can be made only by interconnection of a multiplicity of computers in such a manner as to permit cooperative solution. Demonstration of the single processor approach and the weaknesses of the multiple processor approach in terms of application to real problems ..."

This quote is known as "Amdahl's Law", and is frequently used in parallel computing to predict the maximum theoretical speedup obtained when multiple processors are used in a system. This law shows that, unless the software (or part of the software) is 100% efficient when multiple processors are used, the system will benefit less even when adding more processors to this system.

The following equation shows the global speedup of the system when "Amdahl's Law" is applied to a system:

$$S(n) = \frac{1}{(1-P) + \frac{P}{n}} \tag{1}$$

where:

- $S(n)$ is the theoretical speedup;
- P is the fraction of an algorithm that can be parallelized;
- n is the number of processors or threads used in the process.

Fig. 1 shows the theoretical speedup of a parallelized system related to the number of processors or threads used in this system. It can be seen that even with the increment of the number of processors or threads that will be directed, in theory, to the system, the theoretical speedup does not increase, limiting the system.

2.3 GPU

In this section is presented the different type of memories available in GPU. Section 2.3 describes the assembly of global matrices used in the software developed in this work. In Farber (2011) can be found a complete description of all available memories in GPU, the best way to use them, how to get the best performance in GPGPU applications and some examples with CUDA programming language.

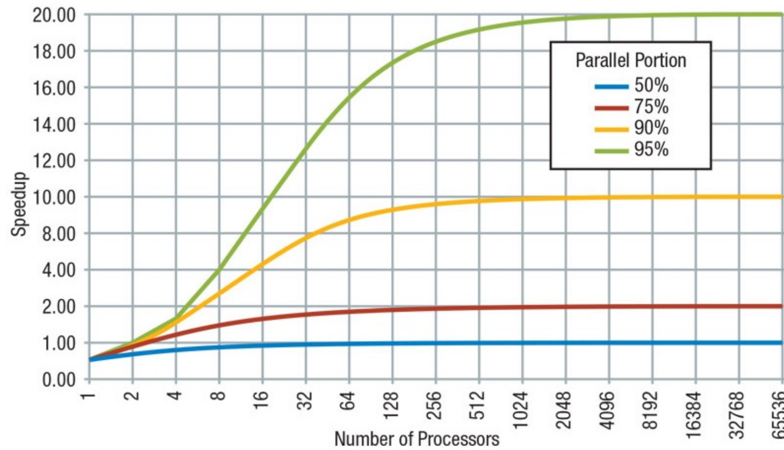


Figure 1: Graphic with theoretical speedup with the number of processors or threads used in a system related to the fraction of a code that could be effectively parallelized (http://www.rtcmagazine.com/articles/view/103209).

GPU Memory.

GPGPU applications has available memories inside graphic card microprocessor and the card itself. The fastest and more scalable ones are shared memories. The only limitation are the available size for storage of information (some KB) and only some threads inside a block can access it.

The global memory is a system of shared memories that can be accessed by all threads of the GPU. The available size for storing information, usually, is measured in GB, turning them into the biggest memory of the GPU, more used, but the slowest one.

Table 1 shows the bandwidth of different memories available in the GPU.

Table 1: Bandwidth of different memories available in GPU (Farber *et al.*, 2011).

Register Memory	$\approx 8000\,\mathrm{GB/s}$
Shared Memory	$\approx 1600\,\mathrm{GB/s}$
Global Memory	$170\,\mathrm{GB/s}$
Mapped Memory	$\approx 8\,\mathrm{GB/s}$ (unidirectional)

It can be seen that the global memory must be well used in order to obtain the maximum performance in GPGPU application. The same way, informations that are stored in shared memory must be well dimensioned so that it could fit in the few KB available.

Memory types.

Table 2 shows the different types of memories available in GPU e their characteristics

Fig. 2 illustrates a schematic diagram of different types of memories and how transfer between them occurs.

Table 2: Memory types available in GPU and their characteristics (Farber *et al.*, 2011).

Type	Location	Access	Scope
Register	On-chip (Inside graphic card microprocessor)	Read/Write	One thread
Local	On-chip	Read/Write	One thread
Shared	On-chip	Read/Write	All threads in a block
Global	Off-chip (Graphic card)	Read/Write	All threads + host
Constant	Off-chip	Read	All threads + host
Texture	Off-chip	Read/Write	All threads + host

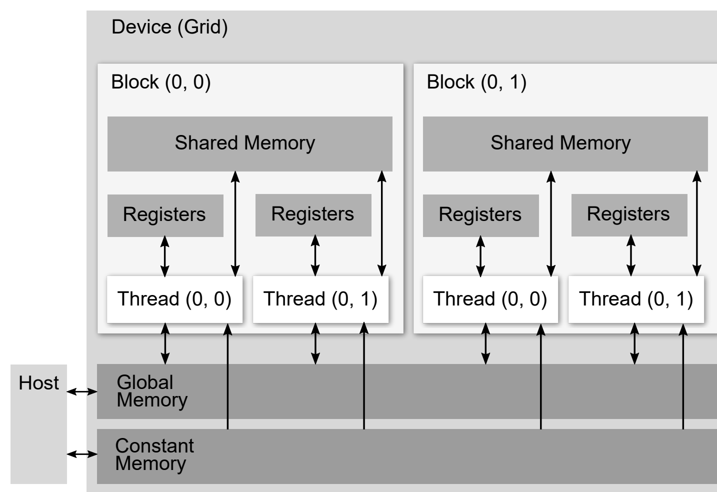


Figure 2: Schematic diagram of internal memories of GPU and data transfer between them.

Registers.

Register memories are the fastest of the GPU and the only ones that has the bandwidth and latency capable of suppling the maximum performance for GPGPU applications. Each kernel can access only 63 register memories, limiting the utilization by GPGPU applications. This value can vary between 63 and 21 depending on the number of threads executed in parallel.

Local.

Local memory is used when the information that ones need to store does not fit inside a register memory.

Shared.

Shared memory can store 16 KB or 48 KB per block of threads and are organized in groups of 32 each one with 32 bits. Ideally, 32 threads can access a shared memory in parallel without losing performance. But unfortunately can occur conflict of access when multiple requests are made by different threads of the same block. This requests can be even to the same address or multiple addresses of the same group. When this happen, the hardware serialize memory

operations, that is, if threads access at the same time the same address of memory, then the requests are executed sequentially, turning the process n times slower.

The biggest challenge in memory utilization of shared types are in its access. It is necessary that the requests are dimensioned so that each thread access its block of information without causing conflict of access and in a single transaction.

Fig. 3 illustrates the access to this memory from CPU.

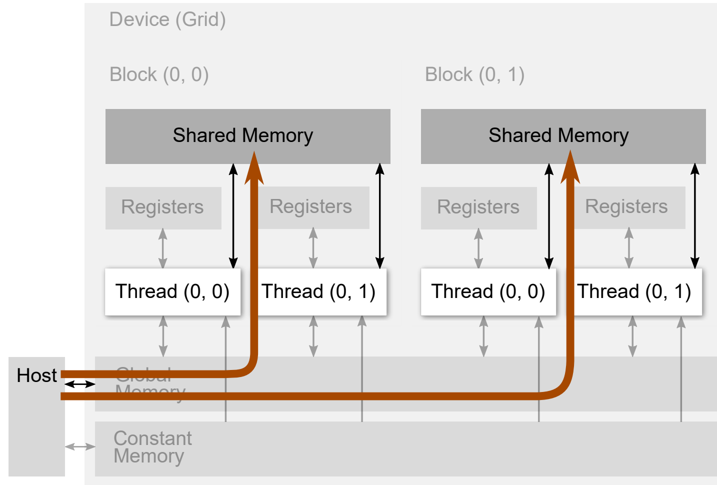


Figure 3: Shared Memory utilizations to increase speedup execution in parallel.

This work used this type of memory. In order to avoid any conflict of access, as described earlier, it was necessary to define which information would be used by assembly of global matrices and how each thread would access this informations. Fig. 4 shows nodal data for a three-dimensional problem, where:

- x, y and z: The coordinates of each node of the finite elements;
- G, L: The global and local degrees of freedom of each node of the finite elements;
- u, v, w: The velocities of each node of the finite elements in x, y and z directions;
- p: The pressure of each node of the finite elements.

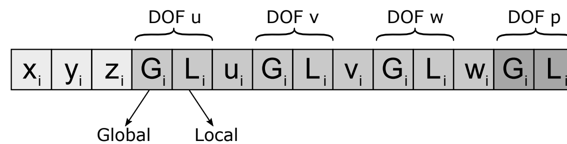


Figure 4: Nodal data for a three-dimensional problem.

Nodal data information are sent to shared memory in contiguous way, so that, only on transaction is necessary for the 21 threads access the informations of the shared memory. This is called memory coalescing. The only access that will be used in a single transaction is the sequential and aligned, thus, is the access that gives the maximum performance.

Fig. 5 illustrates how nodal data are sent to shared memory and then for contiguous access and Fig. 6 presents a schematic diagram (adapted from Cecka *et al.* (2011)) of how assembly is executed in GPU and then in CPU.

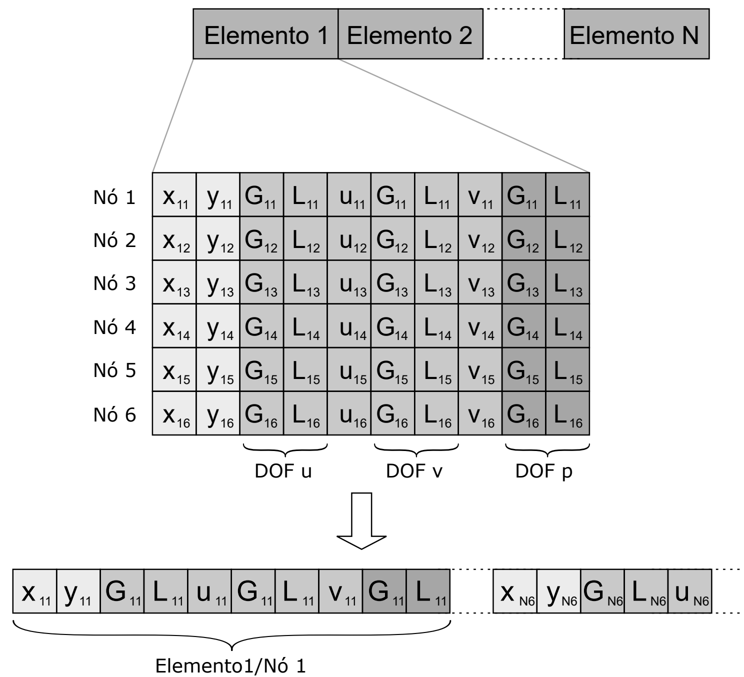


Figure 5: Contiguous access to nodal data.

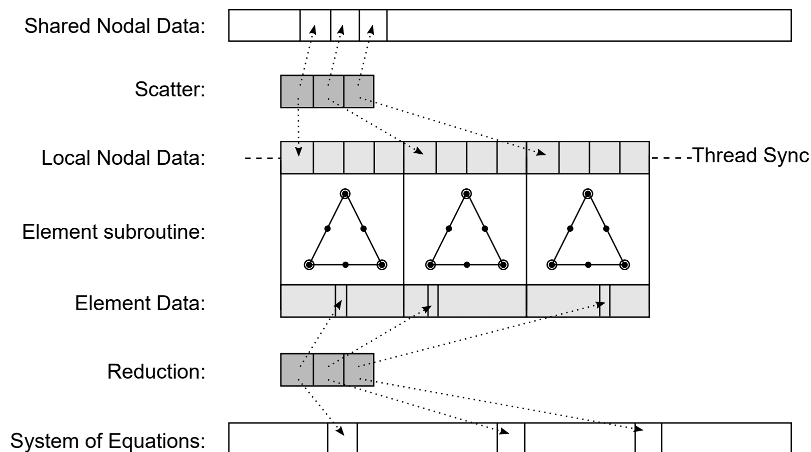


Figure 6: Assembly of global matrices in GPU and then in CPU (adapted from Cecka *et al.* (2011) to be used in this work).

The schematic diagram shown in Fig. 6 is executed as Table 3.

In order to use shared memory it is necessary to fit the nodal data in 16 or 48 KB. Each and every finite element has a certain number of nodes and therefore a certain quantity of bytes. By

Table 3: Algorithm of assembly of global matrices.

Algorithm Assembly of global matrices

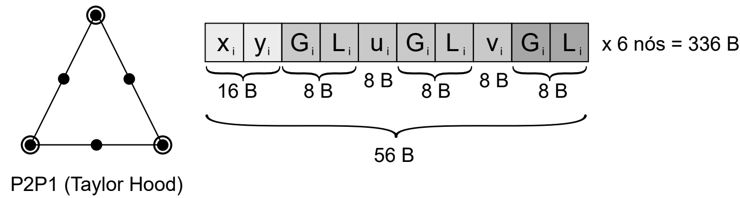
1: **Shared Nodal Data, Scatter and Local Nodal Data**: First the kernel scatter the information from shared memory in a local array so that each thread access data in contiguous way;

2: **Thread Sync**: Execute a synchronization operation that guarantees that each thread access shared information in an aligned and sequential way, keeping all in one transaction;

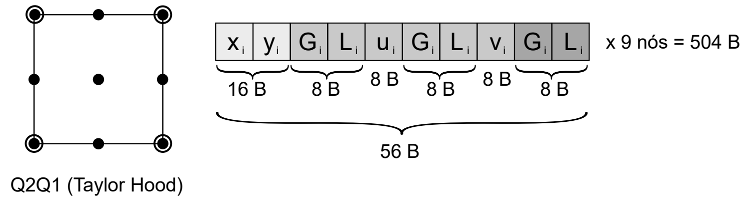
3: **Element subroutine**: Execute finite element subroutines as described in Section 3;

4: **Element Data, Reduction and System of Equations**: Update data in shared memory (i.e. velocities in directions x, y e z and pressure) so that it can be sent to global memory and then to CPU to conclude assembly of global matrices.

using double precision it is necessary 336, 504, 720 e 1944 B for elements of type P2P1, Q2Q1, tetrahedral with 10 nodes and hexahedral with 27 nodes, respectively, as shown in Fig. 7a and 7b.



(a) Number of bytes for a Taylor-Hood P2P1 finite element.



(b) Number of bytes for Taylor-Hood Q2Q1 finite element.

As an example, the maximum number of finite elements of type P2P1 that could be stored in shared memory for fluid dynamics or fluid-structure problems using Immersed Boundary Method are: $49\,152\,\mathrm{B}/336\,\mathrm{B} = 146$ elements. Since each transaction are executed in groups of 32, the ideal situation is to store 128 elements.

Constant.

Constant memory is excellent to store information that are read only and then be sent to all threads that are being executed in GPU. Its limitation is 64 KB.

Global.

The biggest limitation of global memory is its bandwidth. It is really important to have in mind this limitation so that one could develop GPGPU applications that use global memory

only when it is strictly necessary, avoiding transfer of information between CPU and GPU. Certainly there is no way of avoiding access to this memory, but is essential to have in mind its limitations.

Some rules of global memory:

1. Send information and keep it in memory, avoiding sending again to GPU, as long as possible;
2. Execute all kernels;
3. Try to reuse the information stored avoiding bandwidth limitations.

In Farber *et al.* (2011), author presents a series of interesting ways to avoid the bandwidth limitation and other techniques to raise the performance of GPGPU applications.

3 FLUID-STRUCTURE INTERACTION

3.1 Navier Stokes equations

Let Ω_t in $\mathbb{R}^{n_{sd}}$ be the spatial domain with boundary Γ_t at time $t \in (0, T)$. The subscript t indicates the time-dependence of the domain. The Navier-Stokes equations of incompressible flows are written on Ω and $\forall t \in (0, T)$ as

$$\rho\left(\frac{\partial \boldsymbol{u}}{\partial t} + \boldsymbol{u}\cdot\nabla\boldsymbol{u}\right) = \nabla\cdot\boldsymbol{T} + \rho\boldsymbol{b}, \tag{2}$$

$$\nabla\cdot\boldsymbol{u} = 0, \tag{3}$$

where ρ, $\boldsymbol{u}$ and $\boldsymbol{b}$ are the density, velocity and the external force vector, respectively. For a Newtonian fluid the stress $\boldsymbol{T}$ and the strain rate tensors $\epsilon(\boldsymbol{u})$ are defined as

$$\boldsymbol{T} = -p\boldsymbol{I} + 2\mu\boldsymbol{\epsilon}(\boldsymbol{u}) \tag{4}$$

$$\boldsymbol{\epsilon}(\boldsymbol{u}) = \frac{1}{2}\left[\nabla\boldsymbol{u} + (\nabla\boldsymbol{u})^{\mathsf{T}}\right] \tag{5}$$

Here p is the pressure, $\boldsymbol{I}$ is the identity tensor, μ is the viscosity.

The divergence of the stress tensor is written as

$$\nabla\cdot\boldsymbol{T} = -\nabla p + \mu\nabla^2\boldsymbol{u} + \mu\nabla(\nabla\cdot\boldsymbol{u}) \tag{6}$$

Since it is assumed that the flow is incompressible, the last term is zero. Therefore

$$\frac{\partial \boldsymbol{u}}{\partial t} + \boldsymbol{u}\cdot\nabla\boldsymbol{u} = -\nabla p + \nu\nabla^2\boldsymbol{u} + \boldsymbol{b}. \tag{7}$$

Note that all terms are divided by ρ in Eq. 7, originating the fluid kinematic viscosity ν and the kinematic pressure p.

3.2 Boundary and initial conditions

The essential and natural boundary conditions for Eq. (2) are represented as

$$\boldsymbol{u} = \overline{\boldsymbol{u}} \quad \text{on} \quad \Gamma_u, \tag{8}$$

$$(\nu\nabla\boldsymbol{u} - p\boldsymbol{I})\,\boldsymbol{n} = \bar{\boldsymbol{t}} \quad \text{on} \quad \Gamma_t, \tag{9}$$

$$\boldsymbol{u} = \boldsymbol{u}_0 \quad \text{on} \quad \Omega \ \text{ and } \ t = 0. \tag{10}$$

3.3 Weak form

Choosing arbitrary functions $\boldsymbol{w} \in \mathcal{H}_0^1(\Omega)$ and $q \in \mathcal{L}_2(\Omega)$ for the velocities and pressure test functions respectively, we can write the system of partial differential equations described by Eq. (7) in the equivalent integral form as

$$(\boldsymbol{w}, \dot{\boldsymbol{u}})_\Omega + (\boldsymbol{w}, \boldsymbol{u} \cdot \nabla \boldsymbol{u})_\Omega = -(\boldsymbol{w}, \nabla p)_\Omega + (\boldsymbol{w}, \nabla \cdot (2\nu \nabla^s \boldsymbol{u}))_\Omega - (q, \nabla \cdot \mathbf{u})_\Omega + (\boldsymbol{w}, \boldsymbol{b})_\Omega \quad (11)$$

$\forall (\boldsymbol{w}, q)$. Now, integrating by parts the viscous and pressure terms, the weak form of the Navier-Stokes equation for incompressible fluid flow problems can be defined as

$$(\boldsymbol{w}, \dot{\boldsymbol{u}})_\Omega + c(\boldsymbol{u}; \boldsymbol{w}, \boldsymbol{u})_\Omega = -(p, \nabla \cdot \boldsymbol{w})_\Omega - a(\boldsymbol{w}, \boldsymbol{u})_\Omega - (q, \nabla \cdot \boldsymbol{u})_\Omega - \left(\boldsymbol{w}, \bar{\boldsymbol{t}}\right)_{\Gamma_t} + (\boldsymbol{w}, \boldsymbol{b})_\Omega \quad (12)$$

$\forall (\boldsymbol{w}, q)$. Note that the boundary term over Γ vanishes at Γ_u because of the test function. The convective and viscous terms have been written in a compact notation defined by the following bilinear and trilinear forms:

$$a(\boldsymbol{w}, \boldsymbol{u})_\Omega = \int_\Omega \nabla^s \boldsymbol{w} : 2\nu \nabla^s \boldsymbol{u} \, \mathrm{d}\Omega \quad (13)$$

$$c(\boldsymbol{u}; \boldsymbol{w}, \boldsymbol{u})_\Omega = \int_\Omega \boldsymbol{w} \cdot (\boldsymbol{u} \cdot \nabla \boldsymbol{u}) \, \mathrm{d}\Omega \quad (14)$$

3.4 Time integration

The time integration scheme adopted to solve the Navier-Stokes equations is the Newmark method, where the velocity derivative with respect to time at time level $n+1$ can be written as

$$\dot{\boldsymbol{u}}^{n+1} = \frac{1}{\gamma} \frac{(\boldsymbol{u}^{n+1} - \boldsymbol{u}^n)}{\Delta t} - \frac{(1-\gamma)}{\gamma} \dot{\boldsymbol{u}}^n, \quad (15)$$

where $\gamma = 1/2$ is adopted as the Newmark parameter in order to obtain second-order convergence in time. Rewriting the weak form of the Navier-Stokes equation in Eq. (12),

$$\begin{aligned} \{ (\boldsymbol{w}, \boldsymbol{u})_\Omega + \gamma \Delta t \big[c(\boldsymbol{u}; \boldsymbol{w}, \boldsymbol{u})_\Omega + a(\boldsymbol{w}, \boldsymbol{u})_\Omega - (\nabla \cdot \boldsymbol{w}, p)_\Omega + (q, \nabla \cdot \boldsymbol{u})_\Omega - \left(\boldsymbol{w}, \bar{\boldsymbol{t}}\right)_{\Gamma_t} - \\ - (\boldsymbol{w}, \boldsymbol{b})_\Omega \big] \}^{n+1} = (\boldsymbol{w}, \boldsymbol{u}^n)_\Omega + (1-\gamma) \Delta t (\boldsymbol{w}, \dot{\boldsymbol{u}}^n)_\Omega, \forall (\boldsymbol{w}, q). \end{aligned} \quad (16)$$

3.5 Finite Element discretization

This work adopts a mixed formulation so that the fluid finite elements have two unknowns (velocity and pressure) as primitive variables and the Lagrange multipliers are additional unknowns that must be discretized along the interface. The approximation functions for the velocities and pressure, defined locally inside a finite element domain Ω_e, can be defined as

$$\boldsymbol{u}^h(\boldsymbol{x})\big|_{\boldsymbol{x} \in \Omega_e} = \mathbf{N}_u \mathbf{u}_e \quad \text{and} \quad p^h(\boldsymbol{x})\big|_{\boldsymbol{x} \in \Omega_e} = \mathbf{N}_p \mathbf{p}_e \quad (17)$$

respectively. $\mathbf{N}_u$ and $\mathbf{N}_p$ are the shape functions for the velocity and pressure, respectively and $\mathbf{u}_e$ and $\mathbf{p}_e$ its nodal values. The test functions are defined in a similar way by

$$\boldsymbol{w}^h(\boldsymbol{x})\big|_{\boldsymbol{x} \in \Omega_e} = \mathbf{N}_u \mathbf{w}_e \quad \text{and} \quad q^h(\boldsymbol{x})\big|_{\boldsymbol{x} \in \Omega_e} = \mathbf{N}_p \mathbf{q}_e \quad (18)$$

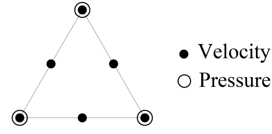


Figure 8: P2P1 Taylor Hood element.

In this work, in order to not violate the LLB compatibility condition for mixed problems, it was used a Taylor-Hood P2P1 element (quadratic interpolation in velocity and linear interpolation in pressure) as illustrated in Fig. 8.

After some algebraic manipulations of the equations demonstrated in previous sections and using the Finite Element discretization just presented we have

$$\begin{cases} \dfrac{\mathbf{M}}{\gamma\Delta t}\mathbf{u}^{n+1} + \left[\mathbf{C}\left(\mathbf{u}^{n+1}\right) + \mathbf{K}\right]\mathbf{u}^{n+1} + \mathbf{G}\mathbf{p}^{n+1} & = \mathbf{f}^{n+1} + \dfrac{\mathbf{M}}{\gamma\Delta t}\mathbf{u}^{n} + \dfrac{1-\gamma}{\gamma}\mathbf{M}\dot{\mathbf{u}}^{n} \\ \mathbf{G}^{\mathsf{T}}\mathbf{u}^{n+1} & = 0 \end{cases} \tag{19}$$

where the local matrices are defined as

$$\mathbf{M}_e = \int_{\Omega_e} \mathbf{N}_u^{\mathsf{T}}\mathbf{N}_u \mathrm{d}\Omega_e \qquad \text{Mass matrix}$$

$$\mathbf{K}_e = \int_{\Omega_e} \mathbf{B}_u^{\mathsf{T}}\nu\mathbf{B}_u \mathrm{d}\Omega_e \qquad \text{Viscous matrix}$$

$$\mathbf{C}_e = \int_{\Omega_e} \mathbf{N}_u^{\mathsf{T}}\left[(\mathbf{N}_{u,i}\mathbf{u}_e)\otimes\mathbf{e}_i\right]\mathbf{N}_u \mathrm{d}\Omega_e \qquad \text{Convective matrix}$$

$$\mathbf{G}_e = -\int_{\Omega_e} (\nabla\cdot\mathbf{N}_u)^{\mathsf{T}}\mathbf{N}_p \mathrm{d}\Omega_e \qquad \text{Gradient operator}$$

$$\mathbf{G}_e^{\mathsf{T}} \qquad \text{Divergent operator}$$

$$\mathbf{f}_e = \int_{\Gamma_{te}} \mathbf{N}_u^{\mathsf{T}}\bar{\mathbf{t}}\mathrm{d}\Gamma_{te} + \int_{\Omega_e} \mathbf{N}_u^{\mathsf{T}}\mathbf{b}\mathrm{d}\Omega_e \qquad \text{Field forces and boundary conditions}$$

3.6 Interface

The strategy adopted in this work is to use embedded interfaces in FSI simulations to compute the fluid flow variables from a Eulerian fixed mesh. The fluid mesh is defined over the fluid domain and extends totally into the structural domain or into some portion of it, as shown in Fig. 9.

For that reason, the structure's wet surface, in general, does not match the fluid grid nodes, hence the fluid velocities at the interface must be weakly enforced. We define a domain that contains the fluid Ω^f and the structural Ω^s (Fig. 10). Γ^i is the interface between the fluid and the structure domains and $\boldsymbol{n}^f$ and $\boldsymbol{n}^s$ their respective unit outward normal vectors.

Now, to ensure the fluid velocity compatibility at the interface, we must have, for a non-slip interface type,

$$\boldsymbol{u}^f = \dot{\boldsymbol{d}}^s = \overline{\boldsymbol{u}}^i \qquad \forall\boldsymbol{x}\in\Gamma^i, \tag{20}$$

$$\boldsymbol{T}^f\boldsymbol{n}^f = -\boldsymbol{T}^s\boldsymbol{n}^s \qquad \forall\boldsymbol{x}\in\Gamma^i \tag{21}$$

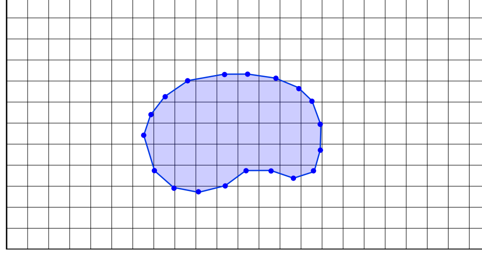


Figure 9: Typical fluid mesh for embedded interface approach (Gomes *et al.*, 2013).

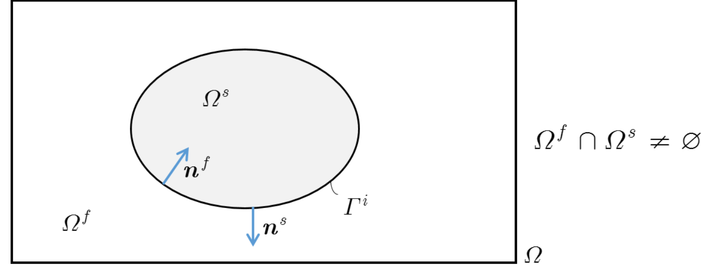


Figure 10: Fluid and structural domains (Gomes *et al.*, 2013).

where Eq. 20 is the kinematic interface condition, $\dot{\boldsymbol{d}}$ is the structure velocity and Eq. 21 represents the dynamic condition. The superscripts on these equations distinguish the variables form the different domains (fluid and structure).

The imposition of Eq. 20 in the fluid problem can be done using Lagrange multipliers. Therefore, it is possible to define a functional given by

$$\Pi = \left(\boldsymbol{\lambda}, \boldsymbol{u}^f - \dot{\boldsymbol{d}}^s\right)_{\Gamma^i} \tag{22}$$

where λ represents the Lagrange multiplier of condition Eq. 20 and constitutes an additional variable field for the problem. The Lagrange multiplier can be identified as a traction acting along Γ^i. The variational form of this function, considering the structure movement to be known *a priori* at the time of solving the fluid problem, is

$$\delta\Pi = \left(\delta\boldsymbol{\lambda}, \boldsymbol{u}^f - \dot{\boldsymbol{d}}^s\right)_{\Gamma^i} + \left(\boldsymbol{\lambda}, \delta\boldsymbol{u}^f\right)_{\Gamma^i}. \tag{23}$$

These terms must be added to the weak form presented in Eq. 16. Thus we have

$$\begin{aligned} \Big\{ (\boldsymbol{w}, \boldsymbol{u})_{\Omega^f} + \gamma\Delta t\big[c\,(\boldsymbol{u}; \boldsymbol{w}, \boldsymbol{u})_{\Omega^f} + a\,(\boldsymbol{w}, \boldsymbol{u})_{\Omega^f} - (\nabla\cdot\boldsymbol{w}, p)_{\Omega^f} + (q, \nabla\cdot\boldsymbol{u})_{\Omega^f} - \\ - \left(\boldsymbol{w}, \overline{\boldsymbol{t}}\right)_{\Gamma_t} - \left(\delta\boldsymbol{\lambda}, \boldsymbol{u} - \overline{\boldsymbol{u}}^i\right)_{\Gamma^i} - (\boldsymbol{w}, \boldsymbol{\lambda})_{\Gamma^i} - (\boldsymbol{w}, \boldsymbol{b})_{\Omega^f}\big]\Big\}^{n+1} = \\ = (\boldsymbol{w}, \boldsymbol{u}^n)_{\Omega^f} + (1-\gamma)\,\Delta t\,(\boldsymbol{w}, \dot{\boldsymbol{u}}^n)_{\Omega^f}, \ \forall\,(\delta\boldsymbol{\lambda}, \boldsymbol{w}, q) \end{aligned} \tag{24}$$

where the convective and viscous terms defined in Eq. 12 are restricted to Γ^i.

3.7 Discretization of the interface

The discretization of the interface can be done in several ways. The most intuitive and adopted technique is to define nodes at the intersections between the interface and the fluid

elements. These nodes are then used to define the Lagrange multipliers' nodal values, and some appropriate approximating functions are used to interpolate the values along the interface. In our simulations, the adoption of low-order finite elements has shown a high sensibility to numerical instabilities in the Lagrange multiplier results.

In order to address this problem, we propose an alternative technique to discretize the interface independently of the fluid mesh (Gomes *et al.*, 2015). Additionally, discontinuous piecewise constant shape functions were adopted to simplify the implementation. The schematic discretization of the interface is shown in Fig. 11, where the structured mesh consists of quadrilateral fluid elements.

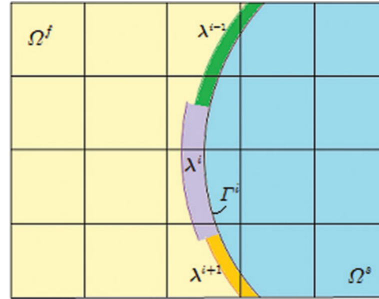


Figure 11: Discretization of the interface between wet surface and the fluid (Gomes *et al.*, 2013).

The approximating and test functions of the Lagrange multipliers can be defined as

$$\boldsymbol{\lambda}^h(\boldsymbol{x})\big|_{\boldsymbol{x}\in\Gamma_e^i} = \mathbf{N}_\lambda \boldsymbol{\lambda}_e \quad \text{and} \quad \delta\boldsymbol{\lambda}^h(\boldsymbol{x})\big|_{\boldsymbol{x}\in\Gamma_e^i} = \mathbf{N}_\lambda \delta\boldsymbol{\lambda}_e \tag{25}$$

$\mathbf{N}_\lambda$ is the shape function of the Lagrange multipliers.

4 POLYGON INTEGRATION

In this section it is presented the main topics of polygon integration theory applied in this work. The complete theory one can find in (Sudhakar *et al.*, 2014).

Consider the integration of polynomial function $\mathcal{F}$ in domain $\mathcal{R} \subset \mathbb{R}^n$ like:

$$I_{\mathcal{R}} = \int_{\mathcal{R}} \mathcal{F} \mathrm{d}\mathcal{R} \tag{26}$$

Let $\mathcal{R}$ be the region in $\mathbb{R}^n$ delimited by closed surface $\mathcal{S}$ (in present work Ω_s and can be seen in Fig. 10). Let $\hat{\mathrm{n}}$ be the vector normal to external surface of $\mathcal{R}$ in $\mathcal{S}$, so, the divergent theorem defines that for any vector $\mathbf{F}$ defined in $\mathcal{R}$, like

$$\int_{\mathcal{R}} \nabla \cdot \mathbf{F} \mathrm{d}\mathcal{R} = \int_{\mathcal{S}} \mathbf{F} \cdot \hat{\mathrm{n}} \mathrm{d}\mathcal{S} \tag{27}$$

It can be defined the vector $\mathbf{F}$ as being

$$\mathbf{F} = \mathcal{G}(\mathbf{x})\,\hat{i} + 0\hat{j} + 0\hat{k} \tag{28}$$

where $\hat{i}$, $\hat{j}$ and $\hat{k}$ are the normal vectors in directions x, y and z, respectively.

Table 4: Algorithm of polygon integration

Algorithm Integration in {polygon} [polyhedron]	
1:	Identify and store {sides} [faces] of cut polygon;
2:	Define {line} [plane] of reference;
3:	Delete {sides} [faces] that are over {line} [plane] of reference;
4:	Distribute main Gauss points over each {side} [face] of polygon that has normal component different from zero;
5:	For each main Gauss point over {side} [face] project vector with coordinates $\mathbf{X}_i$ at {line} [plane] of reference
5.1:	Distribute internal Gauss points between $\mathbf{X}_i$ and κ_i
5.2:	For each internal Gauss point χ_i calculate $\mathcal{G}\left(\mathbf{X}_i\right) = \mathcal{G}\left(\mathbf{X}_i\right) + \lvert\mathbf{J}\rvert\mathcal{F}\left(\chi_{i,j}\right)\mathrm{w}_{i,j}$, where $\lvert\mathbf{J}\rvert$ is the Jacobian of projected line;
6:	Calculate $I_{\mathcal{R}} = I_{\mathcal{R}} + \lvert\mathbf{J}\rvert\mathcal{G}\left(\mathbf{X}_i\right)\mathrm{n}_x\left(\mathbf{X}_i\right)\mathrm{W}_i$, where $\lvert\mathbf{J}\rvert$ is the Jacobian of polygon site with normal component different from zero.

Therefore

$$\mathcal{G}\left(\mathbf{x}\right) = \int_{\kappa}^{\mathbf{x}} \mathcal{F}\mathrm{d}x \tag{29}$$

where κ is a reference point over integration line.

Substituting in Eq. 27 and 28 in 29, one gets

$$\int_{\mathcal{R}} \mathcal{F}\mathrm{d}\mathcal{R} = \int_{\mathcal{S}} \mathcal{G}\left(\mathbf{x}\right) n_x \mathrm{d}\mathcal{S} \tag{30}$$

where n_x is the vector component $\hat{\mathbf{n}}$ in direction x.

The integration of the cut polygon is, therefore, executed like Table 4.

5 NUMERICAL SIMULATIONS

This section presents the results of numerical simulations. In this work it was developed a computational code implementing the theory described in previous sections and the assembly method executed in GPU as described in Section 2.3.

The numerical simulations was executed in two phases, one sequential and one parallelized in GPU. This way we could test and validate the software even for computers without GPU and with computers that has graphic board that could execute CUDA.

The setup used for the performance analysis and the comparison between the sequential and parallel executions was composed of an NVIDIA GeForce GTX 750 processor (driver version: 364.72) installed in PCI Express 3.0 bus of a Intel Core i5 3.20 GHz with 16 GB of RAM and 64 bits architecture running Windows 10 Professional. The graphic board has 4 multiprocessors, 512 CUDA cores, 1024 MB of DDR5 memory with 80.16 GB/s of bandwidth. CUDA version is 5.0.

For all simulations was first executed a sensitivity analysis with the total execution time of assembly of global matrices, $\mathbf{K}$, $\mathbf{G}$, $\mathbf{M}$, $\mathbf{C}$ and $\mathbf{C_t}$, the solution of linear system and the simulation with different number of finite elements. This way was possible to check the quantity of finite elements that we could benefit from GPU utilization. The times presented for the parallel code include kernels calls for assembly of global matrices, but do not include the time spent at CPU-GPU data transfer.

It is necessary to find the best combination of blocks and threads that generates the least computational cost and the best performance. Exhaustive experiments was executed in order to find the relation and best configuration:

$$\#\text{Blocks} = \frac{(N + \#Threads - 1)}{\#Threads} \tag{31}$$

where

- #Blocks: Number of blocks executed in parallel;
- N: Number of finite elements in the problem;
- #Threads: Number of threads executed per block – This number is related to the number of bytes that are stored in shared memory..

6 FLOW PAST CYLINDER

This is another classical example and used as validation case even for Fluid Dynamics or Fluid-Structure Interaction codes. In order to verify and validate GPU code, the parameters found in Schaffer *et al.* (1996) will be used once this reference presents the results of different research groups.

In this section will be presented one example with a non-conforming mesh using Immersed Boundary Method. The drag and lift coefficients were calculated so that we could get a quantitative evaluation, software validation e result comparison (i.e. Gomes *et al.* (2013) and Schaffer *et al.* (1996) – from all groups available, this work used the results from Bänsch, E. et al. from Univ. Freiburg, Inst. für Angewandte Mathematik since they solved the Fluid Dynamics problem using Finite Element with an unstructured mesh and Taylor-Hood P2P1 elements..

Fig. 12 describes the geometry and boundary conditions of the following examples.

6.1 Non-conforming mesh

Sensitivity analysis

For the sensitivity analysis of this simulation it was used meshes with different number of Taylor-Hood P2P1 finite elements (108, 518, 1040, 5104, 10816, 20856 e 41292 finite elements).

Fig. 13 shows the comparison between sequential and parallel execution of the total time of assembly of global matrices and the reached speedup. Fig. 14 shows the comparison between sequential and parallel execution of the total time of simulation and the reached speedup.

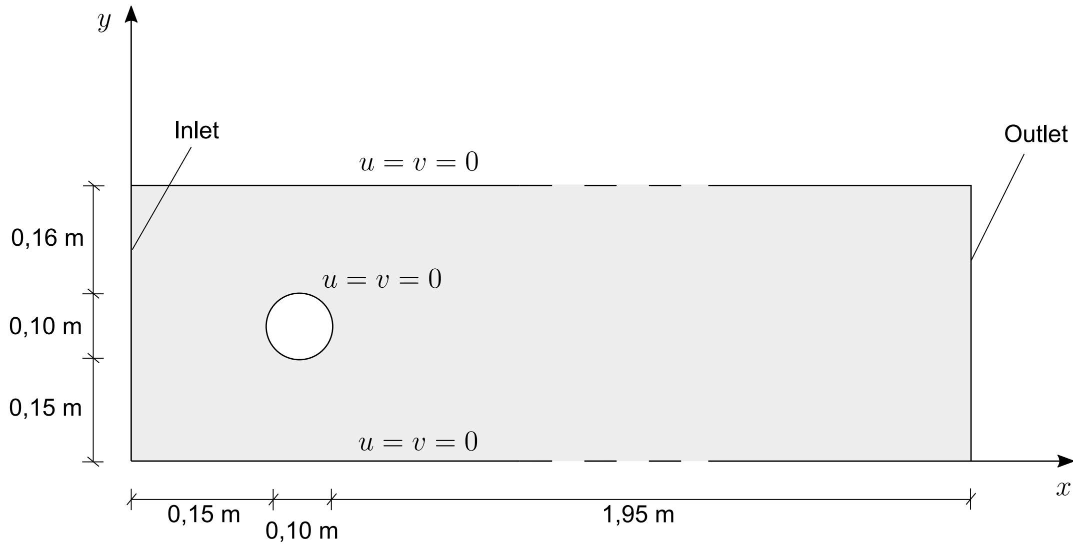


Figure 12: Geometry.

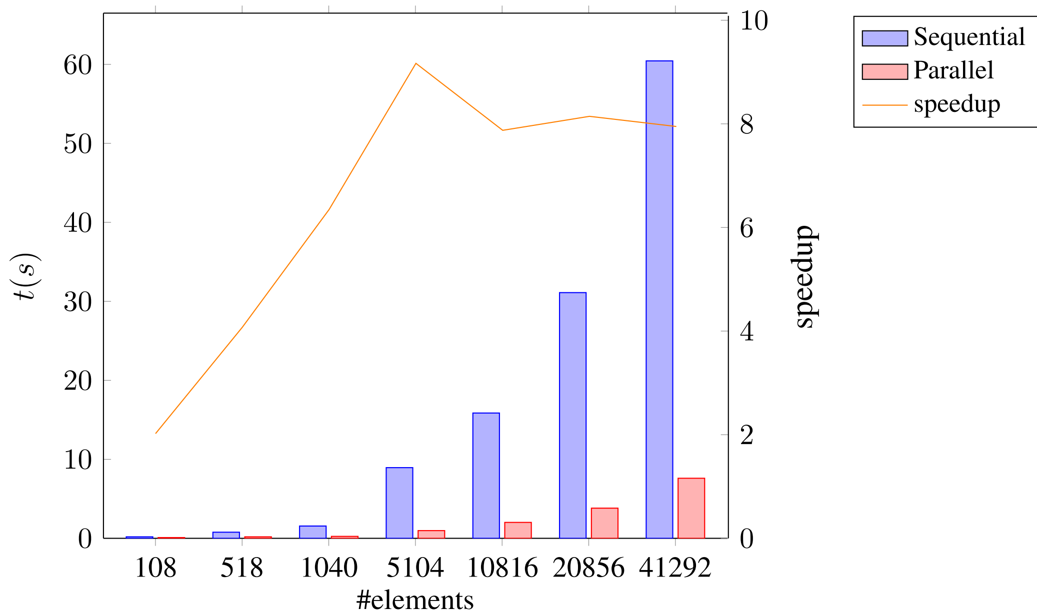


Figure 13: Total time comparison for assembly of global matrices and reached speedup.

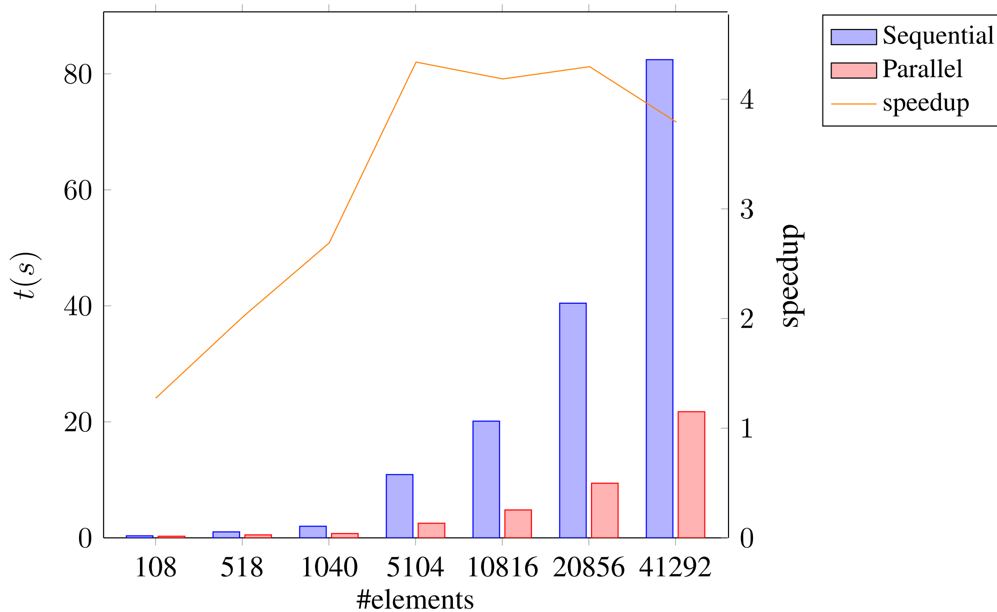


Figure 14: Total simulation time comparison and reached speedup.

The comparative results show a considerable reduction in total processing time, with an speedup of approximately 9 for the assembly of global matrices for mesh with 5104 finite elements. The speedup curve, shows a reduction in its inclination and a stabilization over 5000 elements. So that, above this value, increasing the number of elements does not obtain a significant speedup, as one can see for more coarser meshes.

The total solution time of this problem shows a speedup of approximately 4.5 for the mesh with 5104 finite elements. This is due to the fact that the solution of linear equations is executed in a sequential way, using PARDISO package (Kusmin *et al.*, 2013), . Above 12000 elements, there is a decrease in its speedup.

One can see that with a more coarser mesh there is little advantage in using parallel processing instead of sequential. The reason for that is related to the time necessary to GPU starts its parallelization process (i.e. initialization of blocks of threads, threads and other internal processes).

Results

A mesh with 42176 Taylor-Hood P2P1 finite elements and 84973 nodes was used and is presented in Fig. 15.

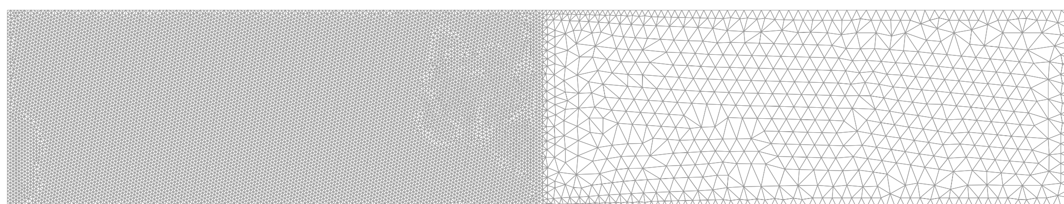


Figure 15: Mesh with 42176 Taylor-Hood P2P1 finite elements and 84973 nodes.

Analytical solution – Circumference.

There are two ways that can be used to generate the structure domain with circumference sahpe. Whether with a finite element mesh or using an alnalytical equation. The way used in this work is through and analytical equation.

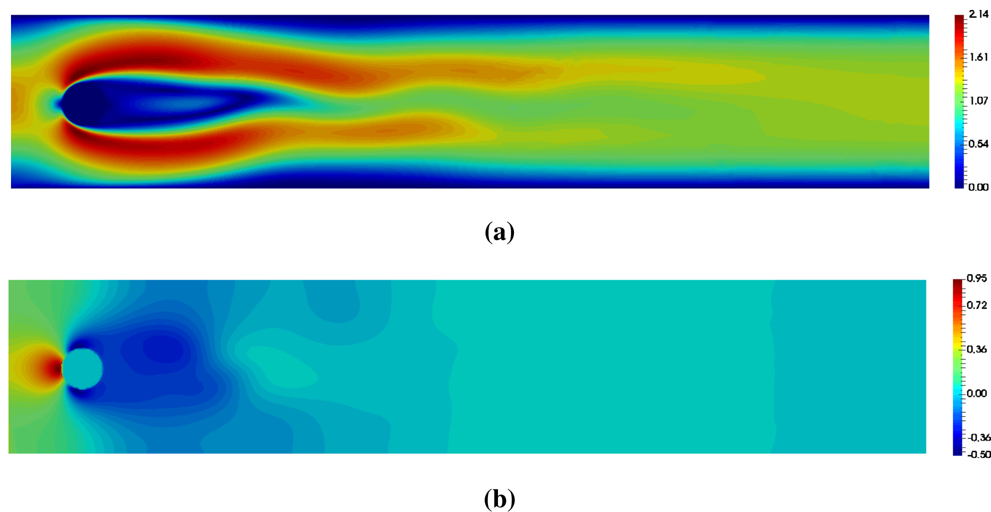
Transient case.

The inflow velocity is given by

$$U\,(0,y,t) = 4U_m y\,(H-y)\,/H^2,\ V=0$$

with $U_m = 1,5m/s$. Reynolds number, for this velocity and duct configuration is given by $Re = 100$.

Fig. 16 and 17 show the results of velocity field in $\mathrm{m/s}$ and pressure in Pa for this configuration for different instants of time.



(a)

(b)

Figure 16: (a) Velocity field and (b) pressure for $t = 2.5\,\mathrm{s}$.

The quantitative results are presented in Table 5.

Table 5: Maximum drag and lift coefficients comparison for transient case.

	c_{Lmax}	c_{Dmax}
Present work with 180 Lagrange Multipliers	0.9103	3.2267
Gomes *et al.* (2013) with 170 Lagrange Multipliers	1.0369	3.2450
Bänsch, E. et al. (1996)	1.0060	3.2240

Performance analysis

Fig. 19 shows the comparison between sequential and parallel execution of total time for assembly of global matrices and simuation and reached speedup.

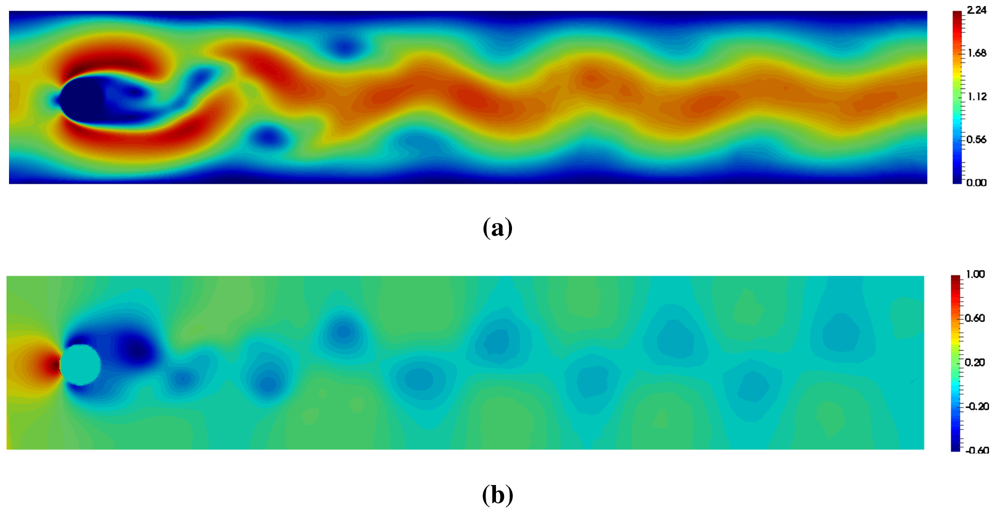


Figure 17: (a) Velocity field and (b) pressure for $t = 6.0\,\mathrm{s}$.

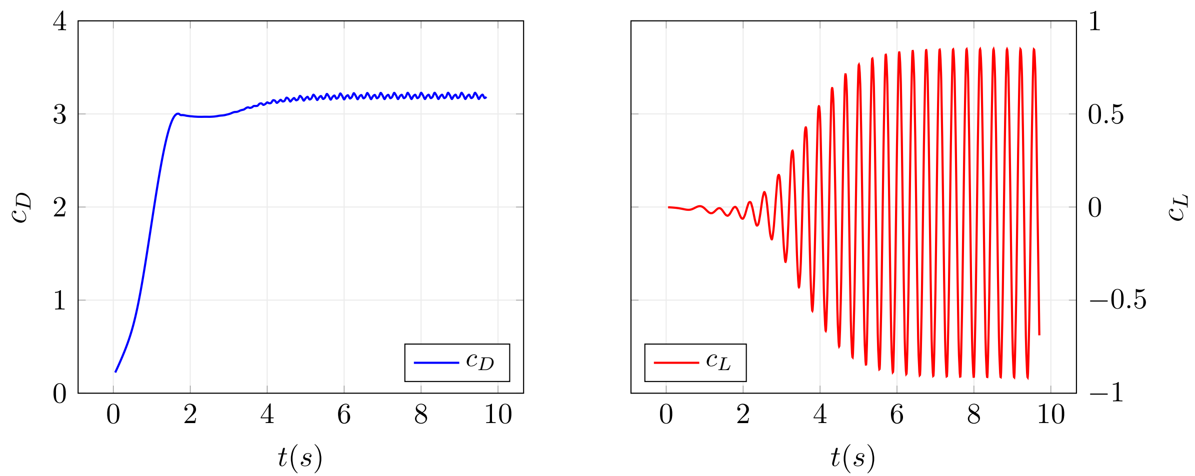


Figure 18: Drag and lift coefficients over time.

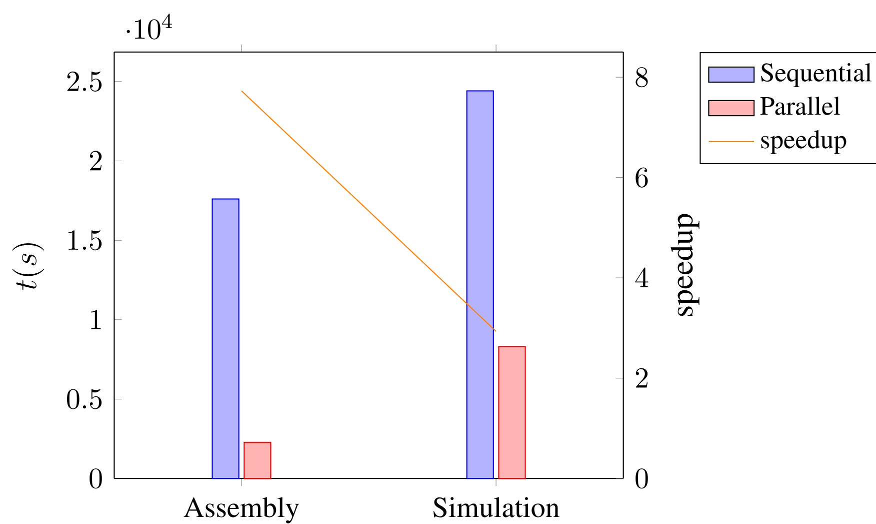


Figure 19: Total time comparison for assembly of global matrices and reached speedup.

The comparative results show a considerable reduction in processing time with a speedup of approximately 8 for assembly of global matrices and approximately 3 for the total simulation time. These values agrees with sensitivity analysis, even with speedups lower than expected. This is due to the fact that the mesh has refinements near the cylinder, increasing then, the total processing time.

The coefficient drag and lift results corroborate with the results found in Bänsch, E. et al. (1996). The size and geometry of domain has huge impact on these coefficients and should be carefully chosen in order to have the results as those shown in this paper.

7 CONCLUSION

This work has presented a computational implementation code for the solution of Navier-Stokes equation for two-dimensional problems, whether stationary or transient, using Finite Elements with Immersed Boundary Method. Taylor-Hood P2P1 finite elements stabilized for LBB condition were used in all meshes. It was developed an algorithm in C++ language for sequential and parallel execution in such way that it could solve Fluid Dynamics and Fluid-Structure Interaction problems in computers with graphic boards that has the power of GPU and CUDA and in computers without graphic parallelization potential. Some examples were simulated in order to validate and assess their performance when comparing sequential and parallel executions. Some conclusions can be pointed out:

- Developed computational code is robust for laminar flows and easily extensible for other problems different from fluids (i.e. structural problems);
- There is a considerable speedup of approximately 10 in assembly of global matrices and 4 for total solution time in comparison with a sequential code, using only one graphic card;
- The bottleneck (to not get greater speedups) is in time for the solution of linear equations, depending exclusively on available RAM memory and CPU processing power (PARDISO);
- A common bottleneck when using GPUs for solving scientific problems is in the data transfer between CPU and GPU. In this work it was presented a novel and power way of using shared memory so that access to GPU memory is made in a contiguous way increasing speedups in assembly of global matrices.

REFERENCES

Amdahl, G. M., 1967. Validity of the Single Processor Approach to Achieving Large-Scale Computing Capabilities.

Cecka, C., Lew, A. J., Darve, E., 2011. Assembly of Finite Element Methods on graphics processors. *International Journal for Numerical Methods in Engineering*, vol. 85, n. 5, pp. 640–669.

Couto, L. F. M., 2016. *Arquitetura de computação paralela aplicada a problemas de dinâmica dos fluidos e interação fluido-estrutura*. Universidade de São Paulo.

Farber, R., 2011. *CUDA application design and development*. Elsevier.

Gamnitzer, P., 2010. *Residual-based variational multiscale methods for turbulent flows and fluid-structure interaction*. Ph.D. Technischen Universitat Munchen.

Gomes, H. C., 2013. *Método dos Elementos Finitos com Fronteiras Imersas aplicado a problemas de dinâmica dos fluidos e interação fluido-estrutura*. Ph. D.: Universidade de São Paulo.

Gomes, H. C., Pimenta, P. M., 2015. Embedded interface with discontinuous Lagrange multipliers for fluid-structure interaction analysis. *International Journal for Computational Methods in Engineering Science and Mechanics*, vol. 16, pp. 98–111.

Lieu, T., Farhat, C., Lesoinne, M., 2006. Reduced-order fluid/structure modeling of a complete aircraft configuration. *Computer Methods in Applied Mechanics and Engineering*, vol. 195, n. 41-43, pp. 5730–5742.

Schaffer, M., Turek, S., Durst, F., Krause, E., Rannacher, R., 1996. Benchmark computations of laminar flow around a cylinder. *Flow Simulation with High-Performance Computers II*, vol. 48, pp. 547–566.

Sudhakar, Y., De Almeida, J. P. M., Wall, W. A., 2014. An accurate, robust, and easytoimplement method for integration over arbitrary polyhedra: Application to embedded interface methods. *Journal of Computational Physics*, vol. 273, pp. 393–415.

Tezduyar, T. E., Sathe, S., Senga, M., Aureli, L., 2005. Finite element modeling of fluidstructure interactions with spacetime and advanced mesh update techniques. Zilina, Slovakia.