(This is a sample cover image for this issue. The actual cover is not yet available at this time.)

Computers & Fluids 74 (2013) 44–57

Contents lists available at SciVerse ScienceDirect

Computers & Fluids

journal homepage: www.elsevier.com/locate/compfluid

A FFT preconditioning technique for the solution of incompressible flow on GPUs

Mario A. Storti [a,*], Rodrigo R. Paz [a], Lisandro D. Dalcin [a], Santiago D. Costarelli [a], Sergio R. Idelsohn [a,b,c]

[a] Centro Internacional de Métodos Computacionales en Ingeniería (CIMEC), INTEC, CONICET-UNL, Santa Fe, Argentina
[b] Institució Catalana de Recerca i Estudis Avançats (ICREA), Barcelona, Spain
[c] International Center for Numerical Methods in Engineering (CIMNE), Technical University of Catalonia (UPC), Gran Capitán s/n, 08034 Barcelona, Spain

ARTICLE INFO

Article history:
Received 20 August 2012
Received in revised form 6 December 2012
Accepted 30 December 2012
Available online 9 January 2013

Keywords:
Graphics processing units
Incompressible Navier–Stokes
Poisson equation

ABSTRACT

Graphic processing units have received much attention in last years. Compute-intensive algorithms operating on multidimensional arrays that have nearest neighbor dependency and/or exploit data locality can achieve massive speedups. Simulation of problems modeled by time-dependent Partial Differential Equations by using explicit time-stepping methods on structured grids is an instance of such GPU-friendly algorithms. Solvers for transient incompressible fluid flow cannot be developed in a fully explicit manner due to the incompressibility constraint. Segregated algorithms like the fractional step method require the solution of a Poisson problem for the pressure field at each time level. This stage is usually the most time-consuming one. This work discuss a solver for the pressure problem in applications using immersed boundary techniques in order to account for moving solid bodies. This solver is based on standard Conjugate Gradients iterations and depends on the availability of a fast Poisson solver on the whole domain to define a preconditioner. We provide a theoretical and numerical evidence on the advantages of our approach versus classical techniques based on fixed point iterations such as the Iterated Orthogonal Projection method.

1. Introduction

Graphics Processing Units (GPUs) are computer co-processors used in desktop computers and workstations to off-load the renderization of complex graphics from the main processor (CPU). They have evolved to complex systems containing many processing units, a large amount of on-board memory and a computing power in the order of teraflops. They are instances of massively parallel architectures and Single Instruction Multiple Data (SIMD) paradigms.

Recently, GPUs are becoming increasingly popular among scientists and engineers for High Performance Computing (HPC) applications [1–3,6,10,14,15,17–19,25,27]. This tendency motivated GPU manufacturers to develop General Purpose Graphics Processing Units (GPGPUs) targeting the HPC market.

In the pursuit of more realistic visualization algorithms for video games and special effects, solving Partial Differential Equations (PDEs) has become a necessary ingredient [4,5,12,24,29]. Numerical schemes employed in these applications usually sacrifice accuracy for speed, resulting in very fast implementations when comparing to engineering codes.

The resolution of Computational Fluid Dynamics (CFD) problems on GPUs requires specialized algorithms due to the particular hardware architecture of these devices. Algorithms that fall in the category of Cellular Automata (CA) are the best fitted for GPUs. For instance, explicit Finite Volume or Finite Element methods, jointly with *immersed boundary* techniques [28] to represent solid bodies, can be used on structured cartesian meshes. In the case of incompressible flows, it is not possible to develop a purely explicit algorithm, due to the essentially non-local nature of the incompressibility condition.

Segregated algorithms solve an implicit Poisson equation for the pressure field, being this stage the most time-consuming in the solution procedure. Using fast Poisson solvers like Multigrid (MG) or Fast Fourier Transform (FFT) is tempting but treating moving solid bodies becomes cumbersome in the case of MG or unsuitable for FFT. To surpass these difficulties, Molemaker et al. proposed in [17] the Iterated Orthogonal Projection (IOP) method which requires a series of projections on the complete grid (fluid and solid) to enforce the incompressibility and boundary conditions.

In this work we propose an alternative to IOP, that we call Accelerated Global Preconditioning (AGP). The solver is based on using a Preconditioned Conjugate Gradients (PCGs) algorithm, so that, it is an accelerated iterative method in contrast to the station-

* Corresponding author.
E-mail address: mario.storti@gmail.com (M.A. Storti).

0045-7930/$ - see front matter © 2013 Elsevier Ltd. All rights reserved.
http://dx.doi.org/10.1016/j.compfluid.2012.12.019

ary scheme used in IOP. In addition, AGP method iterates only on pressure, whereas IOP iterates on both pressure and velocity.

The remainder of this article is organized as follows. Section 2 describes the IOP solver and the QUICK scheme for advection terms. Also, the rate of convergence of IOP is studied. Section 3 introduces the Accelerated Global Preconditioning solver providing a theoretical evidence on the advantages of our approach versus classical techniques based on fixed point iterations such as the IOP method. The numerical performance of the method is studied in Section 5. Concluding remarks are given in Section 6.

2. The Iterated Orthogonal Projection (IOP) solver

The Navier–Stokes governing equations for an incompressible, laminar, constant viscosity fluid are (see Fig. 1)

$$\begin{aligned}
&\frac{\partial u_i}{\partial t}+\frac{\partial (u_j u_i)}{\partial x_j}=-\frac{1}{\rho}\frac{\partial p}{\partial x_i}+\nu\Delta u_i+f_i, \quad \text{in } \Omega_{\text{fluid}},\\
&\frac{\partial u_j}{\partial x_j}=0, \quad \text{in } \Omega_{\text{fluid}},\\
&\mathbf{u}=\mathbf{u}_{\text{bdy}}, \quad \text{at } \Gamma_{\text{bdy}},\\
&\text{periodic B.C's}, \quad \text{at } \Gamma_\infty,\\
&p=0, \quad \text{at } \mathbf{x}_0,
\end{aligned} \tag{1}$$

where u_i are the components of velocity, ρ density, p is pressure, Δ is the Laplace operator, x_j are the spatial coordinates, f_i a gravity field, and t is time. Einstein's summation convention on repeated indices is assumed. Periodic boundary conditions are imposed on the far boundary

$$\Gamma_\infty=\Gamma_x^+\cup\Gamma_x^-\cup\Gamma_y^+\cup\Gamma_y^-\cup\Gamma_z^+\cup\Gamma_z^-, \tag{2}$$

i.e.

$$\begin{aligned}
&\mathbf{u}_{\Gamma_x^+}=\mathbf{u}_{\Gamma_x^-},\\
&p_{\Gamma_x^+}=p_{\Gamma_x^-},
\end{aligned} \tag{3}$$

(and similar expressions for y and z). Also, pressure is defined up to a constant.

The numerical scheme is based on a Fractional Step-like solver, using the Quadratic Upstream Interpolation for Convection Kinematics (QUICK, see [16]) on a staggered grid. QUICK is an advection scheme with very low numerical dissipation and is well suited for structured finite difference schemes.

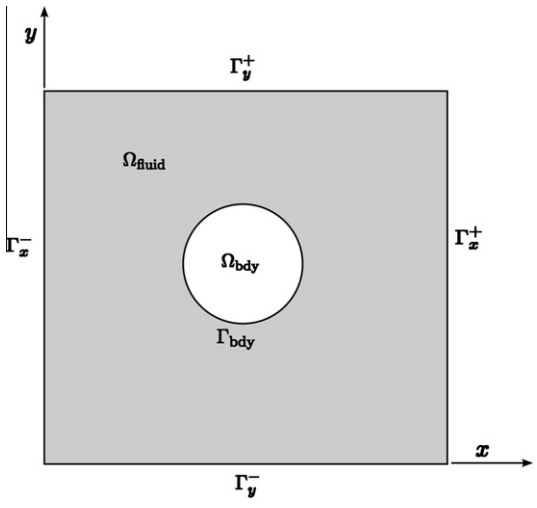

Fig. 1. Geometrical description of problem.

The rectangular box $\Omega=\{0\leqslant x/L_x,y/L_y,z/L_z\leqslant 1\}$ is discretized with $N_x\times N_y\times N_z$ continuity cells. The mesh size $h=L_j/N_j$ is assumed to be the same for all the spatial directions (see Fig. 2). The mesh is *staggered*, so that cells for the discrete balance of continuity and each of the momentum equations do not coincide. The continuity cells are centered around $\mathbf{x}=(i+1/2,j+1/2,k+1/2)h$ positions, and pressure values are assumed to be positioned at the center of these cells. The cells for x-momentum are shifted $h/2$ in the x direction, i.e. they are centered at $\mathbf{x}=(i,j+1/2,k+1/2)h$, and similarly, *mutatis mutandis*, for the other directions. Internal solid bodies will be treated as embedded and the details will be discussed later. The QUICK scheme can accommodate general boundary conditions, but in this article slip or non-slip will be represented in terms of thin solid bodies covering the boundary of the domain, due to the use of the FFT solver.

2.1. The predictor step: QUICK advection scheme

In the first stage, the velocity field $\mathbf{u}^n$ is advanced to an intermediate state $\mathbf{u}^{n+1,p}$

$$\frac{u_i^{n+1,p}-u_i^n}{\Delta t}=\frac{\partial}{\partial x_j}\left(u_j^n u_i^n\right)+f_i, \tag{4}$$

where the superindex p stands for *predictor*. This predicted field may be not divergence free, so that it is corrected via a Poisson stage (or IOP, which is equivalent) to be explained later. The QUICK implementation of the predictor stage is discussed in this section.

Let's consider the x component of the balance equation. For the discretization of the x-momentum balance the corresponding x-momentum cell is used, which is shifted $h/2$ in the x direction, as described before. The discrete equation is obtained by a Finite Volume Method (FVM) approach, i.e. by performing a momentum balance on the cell as

$$\Omega^c\left(\frac{u_x^{n+1,p}(\mathbf{x})-u_x^n(\mathbf{x})}{\Delta t}\right)_{(i,j+1/2,k+1/2)}=M^n_{x,(i,j+1/2,k+1/2)}+\Omega^c f_{x,(i,j+1/2,k+1/2)}. \tag{5}$$

where Ω^c is the cell volume. Note that all terms are evaluated at the center of the x-momentum cells. Discretization of temporal and external force field terms are straightforward. The nonlinear convection term is evaluated as

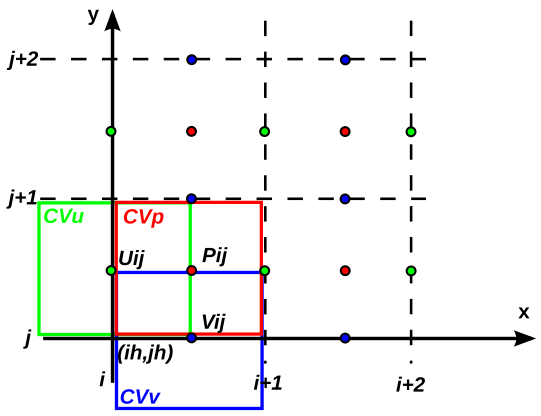

Fig. 2. Staggered scheme (2D). Continuity cells (CVp, in red) are centered at the pressure nodes $\mathbf{x}=(i,j,k)h$. x-momentum cells (CVu, in green) are centered at the u nodes $\mathbf{x}=(i,j+1/2,k+1/2)h$, and y-momentum cells (CVv, in blue) are centered at the v nodes $\mathbf{x}=(i+1/2,j,k+1/2)h$. (For interpretation of the references to color in this figure legend, the reader is referred to the web version of this article.)

$$M_{x,(i,j+1/2,k+1/2)} = S_x[(u^c u^Q)_{i+1/2,j+1/2,k+1/2} - (u^c u^Q)_{i-1/2,j+1/2,k+1/2}] + S_y[(v^c u^Q)_{i,j+1,k+1/2} - (v^c u^Q)_{i,j,k+1/2}] + S_z[(w^c u^Q)_{i,j+1/2,k+1} - (w^c u^Q)_{i,j+1/2,k}], \quad (6)$$

where S_x is the area of the cell faces perpendicular to the x-axis, and so on. The superscripts c and Q stand for *centered* and QUICK respectively, and the superindex n has been dropped since all quantities are evaluated in t^n. Each term represents the flux of momentum through a cell face. Each contribution involves the product of the velocity normal to the surface (which is approximated with a *centered* expression) and the velocity component which is being advected (which is approximated with a QUICK-upwinded expression). In Eq. (6) the advected component is always u (since the x-momentum is considered) whereas the normal velocity may be u, v, or w, depending on the face of the cell to be considered. Note that in the first term, different approximations (centered or QUICK) are used for the same component u depending on whether it is used as a normal velocity or an advected component.

The centered approximations are

$$u^c_{i+1/2,j+1/2,k+1/2} = 1/2(u_{i,j+1/2,k+1/2} + u_{i+1,j+1/2,k+1/2}),$$
$$v^c_{i,j,k+1/2} = 1/2(v_{i-1/2,j,k+1/2} + v_{i+1/2,j,k+1/2}), \quad (7)$$
$$w^c_{i,j+1/2,k} = 1/2(w_{i-1/2,j+1/2,k} + w_{i+1/2,j+1/2,k}).$$

Note that the right hand sides involve u values in the center of the corresponding x-momentum cells whereas the QUICK-upwinded approximations are

$$u^Q_{i-1/2,j+1/2,k+1/2} = \begin{cases} (c_0 u_i + c_1 u_{i-1} + c_2 u_{i-2})_{j+1/2,k+1/2}, \\ \quad \text{if } u^c_{i-1/2,j+1/2,k+1/2} > 0, \\ (c_0 u_{i-1} + c_1 u_i + c_2 u_{i+1})_{j+1/2,k+1/2}, \\ \quad \text{if } u^c_{i-1/2,j+1/2,k+1/2} < 0, \end{cases}$$

$$u^Q_{i,j,k+1/2} = \begin{cases} (c_0 u_{j+1/2} + c_1 u_{j-1/2} + c_2 u_{j-3/2})_{i,k+1/2}, & \text{if } v^c_{i,j,k+1/2} > 0, \\ (c_0 u_{j-1/2} + c_1 u_{j+1/2} + c_2 u_{j+3/2})_{i,k+1/2}, & \text{if } v^c_{i,j,k+1/2} < 0, \end{cases}$$

$$u^Q_{i,j+1/2,k} = \begin{cases} (c_0 u_{k+1/2} + c_1 u_{k-1/2} + c_2 u_{k-3/2})_{i,j+1/2}, & \text{if } w^c_{i,j+1/2,k} > 0, \\ (c_0 u_{k-1/2} + c_1 u_{k+1/2} + c_2 u_{k+3/2})_{i,j+1/2}, & \text{if } w^c_{i,j+1/2,k} < 0. \end{cases} \quad (8)$$

The coefficients c_j are $c_0 = 3/8$, $c_1 = 6/8$, $c_1 = -1/8$. They are the basis of QUICK and guarantee that the upwinded approximations are precise to third order.

The advection step is applied to the whole domain $\Omega = \Omega_{\text{bdy}} \cup \Omega_{\text{fluid}}$, independently of the position of the cell (inside the body, boundary, or fluid). If some of the involved cell values fall outside the fluid domain, they are obtained from interior values via the periodic boundary conditions, i.e. all indices i,j,k are assumed to be cyclic *modulo* N_x, N_y, N_z.

2.2. The projection step in FSM

Once the predicted field $\mathbf{u}^{n+1,p}(\mathbf{x})$ is computed, it may not satisfy the divergence condition (second line in Eq. (1)), neither the boundary conditions

$$\mathbf{u}^{n+1,p} = \mathbf{u}_{\text{bdy}}, \quad \text{at } \Gamma_{\text{bdy}}, \quad (9)$$

where $\mathbf{u}_{\text{bdy}}$ is the velocity of the body. In the standard Fractional Step method these conditions are enforced by computing a Poisson stage

$$\mathbf{u}^{n+1} = \mathbf{u}^{n+1,p} - \nabla P, \quad (10)$$

where $P = (\Delta t/\rho)p$ and p is pressure. P is computed through the following Poisson equation

$$\Delta P = \nabla \cdot \mathbf{u}^{n+1,p}, \quad \text{in } \Omega_{\text{fluid}},$$
$$\left.\frac{\partial p}{\partial n}\right|_{\Gamma_{\text{bdy}}} = (\mathbf{u}_{\text{bdy}} - \mathbf{u}^{n+1,p}) \cdot \hat{\mathbf{n}}. \quad (11)$$

where $\hat{\mathbf{n}}$ is the unit vector normal to Γ_{bdy} pointing towards the fluid. An alternative form to enforce these conditions is the Iterated Orthogonal Projection (IOP) method (see [17]). The idea behind IOP is that as the mesh is structured and cartesian, there are fast solvers (as Multigrid (MG) or Fast Fourier Transform (FFT)) that may solve the Poisson equation very efficiently, provided there are no *holes* (i.e. bodies) in the domain. Given a non-solenoidal vector field $\mathbf{u}$, the *orthogonal projection* operator $\mathbf{\Pi}_{\text{div}}$ defined by

$$\mathbf{u}' = \mathbf{\Pi}_{\text{div}}(\mathbf{u}) \quad \Rightarrow \quad \begin{cases} \mathbf{u}' = \mathbf{u} - \nabla P, \\ \Delta P = \nabla \cdot \mathbf{u}, \end{cases} \quad \text{in } \Omega. \quad (12)$$

projects orthogonally with respect to the L_2 norm, onto the subspace of solenoidal fields S_{div}. Note that the Poisson equation is solved in the whole domain, so that the projected velocity $\mathbf{u}'$ may be nonzero in Ω_{bdy}. $\mathbf{u}'$ is then projected onto the subspace S_{bdy} of velocity fields that satisfy the solid boundary condition

$$\mathbf{u}'' = \mathbf{\Pi}_{\text{bdy}}(\mathbf{u}') \quad \Rightarrow \quad \begin{cases} \mathbf{u}'' = \mathbf{u}_{\text{bdy}}, & \text{in } \Omega_{\text{bdy}}, \\ \mathbf{u}'' = \mathbf{u}', & \text{in } \Omega_{\text{fluid}}. \end{cases} \quad (13)$$

In the case that the solid body is moving then $\mathbf{u}_{\text{bdy}} \neq 0$ and then S_{bdy} is an *affine subspace*. It is easy to see that $\mathbf{\Pi}_{\text{bdy}}$ is also an orthogonal projection operator with respect to L_2. Of course, if the $\mathbf{u}''$ velocity field satisfies also the continuity condition (i.e. $\mathbf{u}'' \in S_{\text{div}}$) then $\mathbf{u}'' \in S_{\text{div}} \cap S_{\text{bdy}}$, and the algorithm stops, i.e. $\mathbf{u}'' = \mathbf{u}^{n+1}$. In the general case a sequence $\mathbf{w}^k$ with $\mathbf{w}^0 = \mathbf{u}^{n+1,p}$ and $\mathbf{w}^\infty = \mathbf{u}^{n+1}$ is generated via the successive application of the projection operators $\mathbf{\Pi}_{\text{div}}$ and $\mathbf{\Pi}_{\text{bdy}}$,

$$\mathbf{w}^{k+1} = \mathbf{\Pi}_{\text{bdy}}\mathbf{\Pi}_{\text{div}}\mathbf{w}^k. \quad (14)$$

It can be shown that the sequence converges [17], *provided that the projection operators are orthogonal in the same norm*, as it is the case here. The geometric interpretation of the algorithm is shown in Fig. 3.

2.3. Rate of convergence of IOP

The convergence of IOP is related to the eigenvalue spectrum of the combined operator $\mathbf{G} = \mathbf{\Pi}_{\text{bdy}}\mathbf{\Pi}_{\text{div}}$. If an eigenfunction of that operator can be found, i.e.

$$\mathbf{G}\bar{\mathbf{v}} = \gamma\bar{\mathbf{v}}, \quad (15)$$

then the IOP sequence for that velocity field will be

$$\bar{\mathbf{v}}, \gamma\bar{\mathbf{v}}, \gamma^2\bar{\mathbf{v}}, \ldots, \gamma^k\bar{\mathbf{v}}, \ldots \quad (16)$$

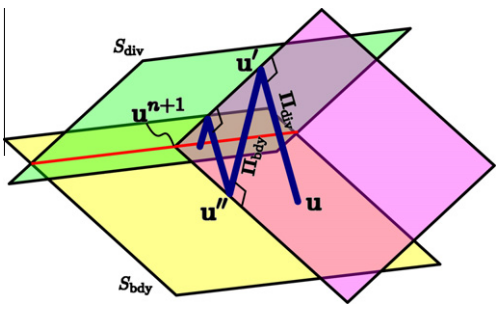

Fig. 3. Geometric interpretation of the convergence of IOP. Starting at a velocity field $\mathbf{u}$ the $\mathbf{\Pi}_{\text{div}}$ operator projects it to $\mathbf{u}'$ int the space of solenoidal velocity fields, but perhaps violating the boundary condition. Then the $\mathbf{\Pi}_{\text{bdy}}$ projects $\mathbf{u}'$ on $\mathbf{u}''$ in the space of velocity fields that satisfy the boundary conditions. It can be shown that the generated sequence is convergent, provided that both projection operators are orthogonal in the same norm.

Such an eigenfunction can be found in a particular geometry and sheds light on the convergence of IOP and the behavior of the rate of convergence with respect to refinement and the geometrical characteristics of the domain, as for instance aspect ratio of the immersed bodies.

Consider a 2D problem in $\Omega = [0,L] \times [0,L]$, and a solid body which is a vertical strip of width b centered at $x = L/2$, i.e.

$$\Omega_{\text{bdy}} = \{|x - L/2| < b/2\}. \tag{17}$$

Consider now the following function

$$\phi(\mathbf{x}) = \begin{cases} \frac{\sinh(kx)}{\sinh(kL_f)} \sin(ky), & \text{for } x < x_-, \\ -\frac{\sinh(k(x-L/2))}{\sinh(kb/2)} \sin(ky), & \text{for } x_- < x < x_+, \\ -\frac{\sinh(k(L-x))}{\sinh(kL_f)} \sin(ky), & \text{for } x > x_+. \end{cases} \tag{18}$$

where $x_\pm = (L \pm b)/2$ are the vertical boundaries of the solid domain, $L_f = (L - b)/2$ is the half length of fluid domain, and $k = 2\pi/L$ is the wave number. This function is shown at Figs. 4 and 5. By construction, $\Delta\phi = 0$ except at the boundary $\Gamma_{\text{bdy}} = \{|x - L/2| = b/2\}$.

Let's start with the following velocity field

$$\mathbf{u} = \begin{cases} \nabla\phi, & \text{in } \Omega_{\text{fluid}}, \\ 0, & \text{in } \Omega_{\text{bdy}}. \end{cases} \tag{19}$$

By construction $\nabla \cdot \mathbf{u} = 0$ in Ω_{fluid}, Ω_{bdy}. At the interface Γ_{bdy}, the divergence can be computed in the sense of distributions and

$$\begin{aligned} \nabla \cdot \mathbf{u} &= (u|_{(x_-)^+} - u|_{(x_-)^-})\delta(x - x_-) + (u|_{(x_+)^+} - u|_{(x_+)^-})\delta(x \\ &\quad - x_-), \\ &= -\left.\frac{\partial\phi}{\partial x}\right|_{(x_-)^-} \delta(x - x_-) + \left.\frac{\partial\phi}{\partial x}\right|_{(x_+)^+} \delta(x - x_+), \\ &= k \coth(kL_f) \ \sin(ky)[-\delta(x - x_-) + \delta(x - x_+)], \end{aligned} \tag{20}$$

where δ is Dirac's δ distribution.

Next, the second equation of (12) must be solved for P. The result is that P is a scalar multiple of ϕ, i.e. $P = c\phi$. It results that

$$\begin{aligned} \Delta P &= c\left(\left.\frac{\partial\phi}{\partial x}\right|_{(x_-)^+} - \left.\frac{\partial\phi}{\partial x}\right|_{(x_-)^-}\right)\delta(x - x_-) + c\left(\left.\frac{\partial\phi}{\partial x}\right|_{(x_+)^+} - \left.\frac{\partial\phi}{\partial x}\right|_{(x_+)^-}\right)\delta(x - x_+), \\ &= ck[\coth(kb/2) + \coth(kL_f)] \ \sin(ky)[-\delta(x - x_-) + \delta(x - x_+)], \end{aligned} \tag{21}$$

and then,

$$c = \frac{\coth(kL_f)}{\coth(kb/2) + \coth(kL_f)}. \tag{22}$$

Applying the correction $-\nabla P$ to $\mathbf{u}$ (the first line in Eq. (12),

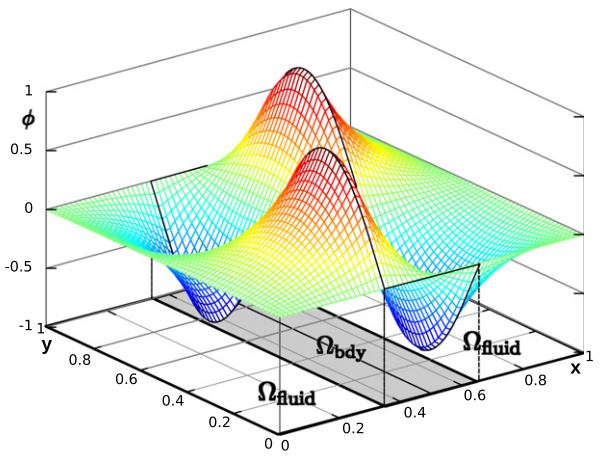

Fig. 4. Eigenmode with slowest convergence rate for IOP.

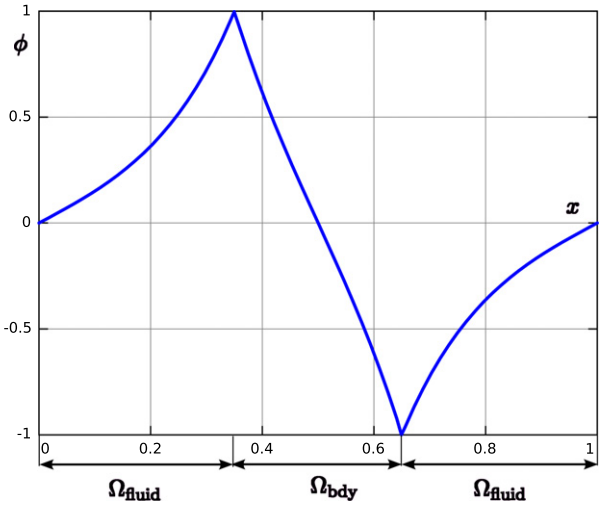

Fig. 5. Eigenmode with slowest convergence rate for IOP. Cut at $y = 0.25$.

$$\mathbf{u}' = \mathbf{u} - \nabla P = \begin{cases} (1 - c) \ \nabla\phi, & \text{in } \Omega_{\text{fluid}}, \\ c \ \nabla\phi, & \text{in } \Omega_{\text{bdy}}. \end{cases} \tag{23}$$

Finally, the application of the $\mathbf{\Pi}_{\text{bdy}}$ consists in simply setting to zero the velocity field in Ω_{bdy}, so that

$$\begin{aligned} \mathbf{u}'' &= \mathbf{\Pi}_{\text{bdy}}\mathbf{u}', \\ &= \begin{cases} (1 - c) \ \nabla\phi, & \text{in } \Omega_{\text{fluid}}, \\ 0, & \text{in } \Omega_{\text{bdy}}. \end{cases} \end{aligned} \tag{24}$$

Comparing (24) with (19) shows that $\mathbf{u}$ is an eigenvalue of $\mathbf{G} = \mathbf{\Pi}_{\text{bdy}}\mathbf{\Pi}_{\text{div}}$ with eigenvalue

$$\gamma = 1 - c = 1 - \frac{\tanh(kb/2)}{\tanh(kb/2) + \tanh(kL_f)}. \tag{25}$$

As expected, the amplification factor γ results to be $0 < \gamma < 1$, otherwise the scheme would be non-convergent.

A family of eigenfunctions can be constructed in the same way, simply replacing the wave number k by higher frequency ones, with the restriction that an integer number of wavelengths must be present in the y direction, i.e.

$$k_m = \frac{2\pi m}{L}, \quad m = 1, 2, \ldots \tag{26}$$

and the corresponding amplification factors are

$$\gamma_{\text{as},m} = \frac{\tanh(k_m L_f)}{\tanh(k_m b/2) + \tanh(k_m L_f)}. \tag{27}$$

Of course, they fall all in the range $0 < \gamma_m < 1$. It can be shown also that for $b < L/2$ the γ_m is a decreasing sequence $\gamma_{m+1} < \gamma_m$, so the mode with slowest rate of convergence (highest γ) is that one corresponding to k_1, i.e. Eq. (25).

Note that the modes given by (18) are *antisymmetric* (hence the "as" subscript) with respect to the center of the strip, i.e. $\phi(x - L/2) = -\phi(L/2 - x)$. There is another branch of eigenvalues corresponding to symmetric modes, in that case the amplification factors are

$$\gamma_{\text{s},m} = \frac{\tanh(k_m b/2)}{\tanh(k_m b/2) + \tanh(k_m L_f)}. \tag{28}$$

where the "s" subindex stands for *symmetric*. For $b < L/2$ it can be shown that for this branch of amplification factors $0 < \gamma_{\text{s},m} < 1/2$, so the slowest rate of convergence is still the first antisymmetric mode.

Also, it can be shown that the set of symmetric and antisymmetric modes together represent a complete set of eigenfunctions for the Stekhlov operators [21,22,26], and then the convergence rate given by (25) is not an upper bound but rather the spectral radius of the amplification matrix **G** and hence the best estimate for the rate of convergence of the algorithm.

2.4. Convergence in the discrete case

The rate of convergence given in (25) corresponds to the continuum case. However, discretization affects mostly the high frequency (large k_m) modes, but as it has been shown the global rate of convergence is governed by the slowest mode, which is slightly affected by refinement. In fact, as the mesh is refined ($h \to 0$) the convergence rate approaches the value of the continuum (25). This means that *the rate of convergence for IOP does not degrade with refinement*.

2.5. Convergence and aspect ratio

Note that for high aspect ratio (i.e. $L/b \to \infty$) the amplification factor $\gamma \to 1$, i.e. convergence degrades. In fact for $b \ll L$ the amplification factor is

$$\begin{aligned} \gamma &\approx 1 - \frac{\pi b/L}{\tanh(2\pi)}, \\ &\approx 1 - 3.14\frac{b}{L}, \end{aligned} \tag{29}$$

i.e., the rate of convergence r (as number of iterations per order of magnitude) is

$$\begin{aligned} &\gamma^r = 1/10, \\ &r = -\frac{\log 10}{\log \gamma} \approx 0.37\frac{L}{b}[\text{iter/OM}]. \end{aligned} \tag{30}$$

which is proportional to the aspect ratio and *iter/OM* means *"iterations per order of magnitude"*.

This result has been confirmed with numerical experiments for other geometries as well. The physical explanation is that IOP has good convergence when $\mathbf{\Pi}_{\text{div}}$ is close to the solution of the Poisson problem *on the fluid domain only* (recall that $\mathbf{\Pi}_{\text{div}}$ solves the Poisson problem in the whole domain $\Omega_{\text{fluid}} + \Omega_{\text{bdy}}$). Conversely, convergence is poor when the inclusion of the solid body domain Ω_{bdy} distorts too much the solution and this is just what happens when the aspect ratio is large for a mode like (18). Note that the sources on opposite sides of the strip have different sign and then a large (spurious) flow is generated inside the body.

3. The Accelerated Global Preconditioning

The algorithm proposed in this paper is based in the IOP, in the sense of using a fast solver on the whole mesh, but the main difference is that this global solution is used as a preconditioner for the Preconditioned Conjugate Gradient (PCG) method. Recall that the PCG method is an *accelerated* convergence method, i.e. the rate of convergence increases during iteration, hence the name Accelerated Global Preconditioning (AGP) (see [13]).

Consider a situation like that in Fig. 6, with a solid body described by the boundary Γ_{bdy}. This is embedded in a structured grid of constant mesh size h. In the traditional Fractional Step Method a Poisson problem is solved *outside* the body. Recall that pressure nodes are at the center of the continuity cells (marked with dashed lines in the figure). In order to construct an approximation to the Poisson equation a Finite Element (FEM) mesh is considered where the pressure nodes are at the corner of the finite elements (marked as solid lines in the figure). Note that the cell mesh is *dual* to the continuity cell mesh, i.e. the center of the continuity cells (which are the pressure nodes) are at semi-integer positions $(i+1/2, j+1/2, k+1/2)h$ coincident with the corner of the finite elements and, *vice versa*, the center of the finite elements are then at full integer positions (ih, jh, kh). The center of the finite elements are computed and it is checked whether the element falls inside or outside the body. In this way the body is approximated by a *staircase geometry* as is shown in gray in the figure. (This degrades de convergence to $O(h)$ instead of $O(h^2)$ and can be fixed by more sofisticated techniques as the Immersed Boundary Method [23], but this will be not discussed here.) As it is usual in FEM discretizations the imposition of the homogeneous Neumann condition is done by simply assembling only those elements that are in the fluid part. The other elements that are not in gray are *ghost elements* and are not assembled for the solution of the Poisson problem. Only the pressure in the nodes connected to some element that is assembled are relevant, i.e. those that are marked in blue and red. Those that are marked in green are *ghost* and are not computed. From those that are computed, the set that are surrounded completely by computed elements (and then are not connected to ghost elements) are classified as *interior to the fluid*, (subindex F) and the rest are classified as *boundary* (subindex B, filled in red in the figure). So the Poisson problem is

$$\mathbf{A}\mathbf{x} = \mathbf{b}, \tag{31}$$

and the splitting of nodes induces a matrix splitting like this

$$\begin{bmatrix} \mathbf{A}_{FF} & \mathbf{A}_{FB} \\ \mathbf{A}_{BF} & \mathbf{A}_{BB} \end{bmatrix} \begin{bmatrix} \mathbf{x}_F \\ \mathbf{x}_B \end{bmatrix} = \begin{bmatrix} \mathbf{b}_F \\ \mathbf{b}_B \end{bmatrix} \tag{32}$$

In the following, the preconditioning operator P will be described. First consider the whole matrix for the Laplace operator $\widetilde{\mathbf{P}}$, i.e. assembling over fluid and ghost cells for F, B, and G nodes. Note that a symbol different from **A** is used for this matrix since it is assembled on a different set of element/cells. The preconditioning is then defined formally as $\mathbf{y}_{FB} = \mathbf{P}\mathbf{x}_{FB}$, where $\mathbf{x}_{FB}$ is the solution of

$$\begin{bmatrix} \widetilde{\mathbf{P}}_{FF} & \widetilde{\mathbf{P}}_{FB} & \widetilde{\mathbf{P}}_{FG} \\ \widetilde{\mathbf{P}}_{BF} & \widetilde{\mathbf{P}}_{BB} & \widetilde{\mathbf{P}}_{BG} \\ \widetilde{\mathbf{P}}_{GF} & \widetilde{\mathbf{P}}_{GB} & \widetilde{\mathbf{P}}_{GG} \end{bmatrix} \begin{bmatrix} \mathbf{x}_{FB} \\ \mathbf{x}_G \end{bmatrix} = \begin{bmatrix} \mathbf{y}_{FB} \\ \mathbf{0}_G \end{bmatrix}. \tag{33}$$

However it can be seen that

- $\widetilde{\mathbf{P}}_{FF} = \mathbf{A}_{FF}$ since the F nodes are those for which all neighbor elements are assembled in the Poisson problem.
- $\widetilde{\mathbf{P}}_{FB} = \mathbf{A}_{FB}$, and $\widetilde{\mathbf{P}}_{BF} = \mathbf{A}_{BF}$ since for instance, such a coefficient would link nodes as a and b in the figure. This coefficient comes from the assembly of all the elements that are connected to a *and* b, but since a is an F node, it means that all elements connected to a are assembled.
- $\widetilde{\mathbf{P}}_{FG} = \widetilde{\mathbf{P}}_{GF} = 0$ since F nodes are only connected to fluid elements and G are only connected to ghost elements, so that they cannot share an element.

So

$$\begin{bmatrix} \mathbf{A}_{FF} & \mathbf{A}_{FB} & \mathbf{0} \\ \mathbf{A}_{BF} & \widetilde{\mathbf{P}}_{BB} & \widetilde{\mathbf{P}}_{BG} \\ \mathbf{0} & \widetilde{\mathbf{P}}_{GB} & \widetilde{\mathbf{P}}_{GG} \end{bmatrix} \begin{bmatrix} \mathbf{x}_{FB} \\ \mathbf{x}_G \end{bmatrix} = \begin{bmatrix} \mathbf{y}_{FB} \\ \mathbf{0}_G \end{bmatrix}. \tag{34}$$

$\mathbf{x}_G$ can be eliminated from the bottom line, and then the following equations is obtained

$$\begin{bmatrix} \mathbf{A}_{FF} & \mathbf{A}_{FB} \\ \mathbf{A}_{BF} & \widetilde{\mathbf{P}}_{BB} - \widetilde{\mathbf{P}}_{BG}\widetilde{\mathbf{P}}_{GG}^{-1}\widetilde{\mathbf{P}}_{GB} \end{bmatrix} \mathbf{x}_{FB} = \mathbf{y}_{FB}. \tag{35}$$

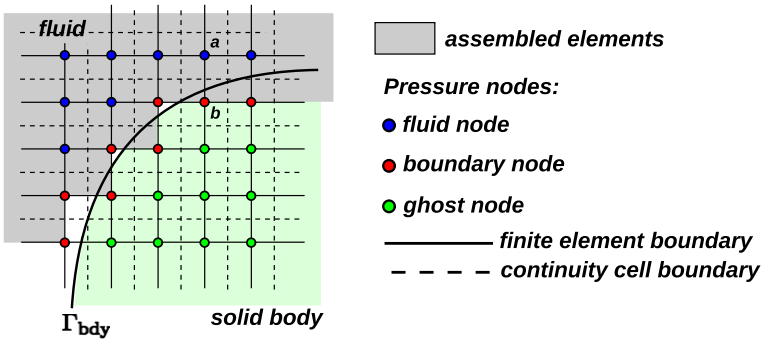

Fig. 6. Description of nodes and elements used in the AGP.

This allows to obtain an explicit expression for the preconditioning matrix

$$\mathbf{P} = \begin{bmatrix} \mathbf{A}_{FF} & \mathbf{A}_{FB} \\ \mathbf{A}_{BF} & \widetilde{\mathbf{P}}_{BB} - \widetilde{\mathbf{P}}_{BG}\widetilde{\mathbf{P}}_{GG}^{-1}\widetilde{\mathbf{P}}_{GB} \end{bmatrix}. \tag{36}$$

A first consequence of this expression is that most eigenvalues of the preconditioned matrix will be 1. Consider the subspace of all vectors $\mathbf{x}$ such that the boundary component B is null, i.e. the non-null entries are only on F nodes, then

$$\begin{aligned} &\mathbf{A}\mathbf{x} = \mathbf{P}\mathbf{x}, \\ &\mathbf{P}^{-1}\mathbf{A}\mathbf{x} = \mathbf{x}, \end{aligned} \tag{37}$$

so that $\mathbf{x}$ is an eigenvector with eigenvalue 1.

The proposed technique is named Accelerated Global Preconditioning (AGP) because the solution of (33) is done on an infinite mesh with periodic boundary conditions so that it can be solved via FFT transform, which is very efficient, but the whole analysis does not depend on how the solution of this system is done; i.e. it may be obtained by Multigrid iteration as well.

3.1. Convergence of AGP

Note that in the IOP method after the first application of $\mathbf{u}' = \mathbf{\Pi}_{\text{div}}\mathbf{u}$ the velocity field $\mathbf{u}'$ is solenoidal everywhere. After the $\mathbf{u}'' = \mathbf{\Pi}_{\text{bdy}}\mathbf{u}'$ step, the field $\mathbf{u}''$ is solenoidal at both Ω_{fluid} and Ω_{bdy} (it is zero if the body is stationary, and a rigid motion if it is in movement) but not at the interface. In the second application of $\mathbf{\Pi}_{\text{div}}$ the right hand side in Eq. (12) is zero everywhere, except for a possible concentrated source term at the interface, i.e. a Dirac's δ.

Something similar happens for AGP, after the first iteration of the PCG the right hand side for the preconditioning step (33) is zero everywhere, except at the boundary nodes. In Section 3 the AGP was introduced in the discrete version; the continuum counterpart will be used now for assessing its convergence properties. Both the Poisson problem (31) and the preconditioner are written as mappings between surface values at Γ_{bdy} and solenoidal pressure fields that satisfy the Laplace equation everywhere, except at the interface.

First, the Poisson equation in the fluid domain Ω_{fluid} is written in abstract form as

$$\mathcal{A}(\psi) = g, \tag{38}$$

where g is a source term in Γ_{bdy} and ψ a function defined on Ω_{fluid} that satisfies

$$\begin{cases} \Delta\psi = 0, & \text{in } \Omega_{\text{fluid}}, \\ \nabla\psi \cdot \hat{\mathbf{n}} = -g, & \text{at } \Gamma_{\text{bdy}}, \end{cases} \tag{39}$$

where $\hat{\mathbf{n}}$ is the normal to Γ_{bdy} pointing into the fluid.

Next, the preconditioner is written in abstract form as

$$\mathcal{P}(\psi) = g, \tag{40}$$

where g is also a source term on Γ_{bdy} and ψ is defined on Ω_{fluid} and is obtained from the following problem. Let ψ' defined in $\Omega = \Omega_{\text{fluid}} \cup \Omega_{\text{bdy}}$, satisfying

$$\begin{cases} \Delta\psi' = 0, & \text{in } \Omega_{\text{fluid}} \text{ and } \Omega_{\text{bdy}}, \\ [(\nabla\psi')^{+} - (\nabla\psi')^{-}] \cdot \hat{\mathbf{n}} = -g, & \text{at } \Gamma_{\text{bdy}}, \end{cases} \tag{41}$$

where $(\nabla\psi')^{+}$ is evaluated on the side of the fluid region and $(\nabla\psi')^{-}$ from the body region. Now, the AGP algorithm can be put in the continuum case as solving (38) with PCG, using (40) as a preconditioner. Then, rates of convergence for the AGP an be estimated in terms of the condition number of the preconditioned operator

$$\kappa = \text{cond}(\mathcal{P}^{-1}\mathcal{A}). \tag{42}$$

The condition number κ will be computed for the strip problem used before (see Section 2.3) for assessing the convergence of IOP.

The eigenfunctions of the preconditioned problem should satisfy

$$\mathcal{P}^{-1}\mathcal{A}\psi = \lambda\psi. \tag{43}$$

If we define $g = \mathcal{A}\psi$, then it is equivalent to

$$\begin{aligned} &\mathcal{P}\psi = (1/\lambda)g, \\ &\mathcal{A}\psi = g. \end{aligned} \tag{44}$$

and it can be rewritten as

$$\begin{aligned} &\Delta\psi' = 0, && \text{in } \Omega_{\text{fluid}}, \Omega_{\text{bdy}}, \\ &\tfrac{\partial\psi'}{\partial n} = -g, && \text{at } \Gamma_{\text{bdy}}, \\ &\left(\tfrac{\partial\psi'}{\partial n}\right)^{+} - \left(\tfrac{\partial\psi'}{\partial n}\right)^{-} = -(1/\lambda)g, && \text{at } \Gamma_{\text{bdy}}, \end{aligned} \tag{45}$$

and set ψ to the restriction of ψ' to Ω_{fluid}. It can be shown that the function ϕ defined in (18) satisfies this set of equations. Effectively ϕ by construction satisfies the first line of (45) and the second and third lines give

$$\begin{aligned} &g = k_m \ \coth(k_m L_f) \ \sin(k_m y), \\ &\frac{1}{\lambda}g = k_m[\coth(k_m L_f) + \coth(k_m b/2)] \ \sin(k_m y), \end{aligned} \tag{46}$$

so it gives,

$$\begin{aligned} &g = k_m \ \coth(k_m L_f) \ \sin(k_m y), \\ &\lambda_{\text{as},m} = \frac{\coth(k_m L_f)}{\coth(k_m L_f) + \coth(k_m b/2)}. \end{aligned} \tag{47}$$

Again, it can be shown that for $b < L/2$ λ_m is a monotonically increasing sequence, and in addition $\lambda_\infty = 1/2$. The subindex "as" stands for *antisymmetric*. Now for the symmetric modes it can be shown that

$$\lambda_{s,m} = \frac{\tanh(k_m L_f)}{\tanh(k_m L_f) + \tanh(k_m b/2)}, \tag{48}$$

and that $\lambda_{s,m}$ is monotonically decreasing and $1 = \lambda_{s,1} > \lambda_{s,m} > \lambda_{s,\infty} = 1/2$. Then, the highest eigenvalue is $\lambda_{s,1} = 1$, and the lowest is $\lambda_{as,m}$. Then, the condition number is

$$\kappa = \frac{\lambda_{s,1}}{\lambda_{as,1}}, = \frac{\tanh(\pi b/L) + \tanh(2\pi L_f/L)}{\tanh(\pi b/L)}. \tag{49}$$

Again, this approximation holds also in the discrete case, since both the maximum and minimum eigenvalues have low frequency. Following the same arguments in Section 2.4, it can be shown that *the condition number of the preconditioned AGP operator and hence the rate of convergence for PCG (with AGP) does not degrade under refinement.*

3.2. High aspect ratio limit

For a high aspect ratio strip ($L/b \gg 1$) the limit is

$$\begin{aligned} \kappa &\approx \frac{\tanh(2\pi)}{\pi}\,\frac{L}{b}, \\ &\approx 0.32\,\frac{L}{b}, \end{aligned} \tag{50}$$

3.3. Spectrum of AGP operator and IOP convergence

It can be shown that the amplification factors for IOP (see equations (27) and (28)) are related to the eigenvalues of the AGP method by the simple relation

$$\gamma_m = 1 - \lambda_m, \tag{51}$$

so that the slowest rate of convergence (the γ_m closer to 1) corresponds to $m = 1$, i.e.

$$\begin{aligned} &\gamma = 1 - \lambda_{\min}, \\ &r \approx \log 10 \frac{1}{\lambda_{\min}}, \quad \text{for } \lambda_{\min} \ll 1. \end{aligned} \tag{52}$$

4. Comparison of IOP and AGP

The differences and similitudes of both methods are summarized here:

- Both solvers are based on the fact that the Poisson equation on the fluid domain can be approximated by solving on the global domain (fluid + solid). Of course, this represents more computational work than solving the problem only in the fluid, but this can be faster in a structured mesh using some fast solvers such as Multigrid or FFT.
- Both solvers have their convergence governed by the spectrum of the AGP preconditioned operator $\mathcal{P}^{-1}\mathcal{A}$, more precisely on its lowest eigenvalue $\lambda_{\min}$.
- It has been shown that for a fixed geometry $\lambda_{\min} = O(1)$, i.e. it *does not degrade with refinement*, so that IOP has a linear convergence with limit rate $O(1)$, i.e. it does not degrade with refinement.
- By the same reason, the condition number for AGP *does not degrade with refinement.*
- IOP is a *stationary method* and its limit rate of convergence is given by

$$\begin{aligned} &\|\mathbf{r}^{n+1}\| \leqslant \gamma \|\mathbf{r}^n\| \\ &\gamma = 1 - \lambda_{\min}, \\ &\lambda_{\min} = \frac{\tanh(kb/2) + \tanh(kL_f)}{\tanh(kL_f)}, \quad \text{(strip of aspect ratio } L/b), \\ &\gamma \approx 1 - 3.14\,\frac{b}{L}, \quad (L/b \gg 1). \end{aligned} \tag{53}$$

- AGP is an *accelerated method* (PCG) and the convergence for AGP can be assessed from the condition number of the preconditioned operator, which is

$$\begin{aligned} \kappa(\mathcal{P}^{-1}\mathcal{A}) &= \frac{1}{\lambda_{\min}}, \\ &= \frac{\tanh(kL_f)}{\tanh(kb/2) + \tanh(kL_f)}, \text{(strip of aspect ratio } L/b), \\ &\approx 0.32\,\frac{L}{b} \qquad (L/b \gg 1). \end{aligned} \tag{54}$$

- In fact, it can be shown that the iterates for both IOP and AGP can be put as polynomials on the preconditioned operator $\mathcal{P}^{-1}\mathcal{A}$, and due to the *minimization property characteristic of Krylov space methods* like CG (see [13]) the convergence of AGP is always better than that of IOP.
- IOP iterates over both the velocity and pressure fields, whereas AGP iterates only on the pressure vector (which is better for implementation on GPUs architectures, since reduces memory access).
- As the minimum eigenvalue is proportional to the reciprocal of the aspect ratio (i.e. $\lambda_{\min} \propto b/L$) the convergence of both algorithms degrade for high aspect ratio bodies. In the case of IOP, the rate of convergence is proportional to L/b. In the case of AGP the condition number of the preconditioned operator $\kappa(\mathcal{P}^{-1}\mathcal{A})$ is proportional to L/b. Due to the estimates of rate of convergence for CG as compared to stationary methods, it is expected that the convergence rates of AGP will be comparatively much better than that for IOP for geometries with high aspect ratio bodies.

4.1. Solving the Poisson equation with the FFT

Both IOP and AGP are based on the fact that a fast solver in the whole domain (solid + fluid) exists. There are at least two possibilities: MG and FFT. The second has been chosen in this work, and the basis of this component of the algorithm will be given here.

The linear system to be solved is denoted as

$$\mathbf{A}\mathbf{x} = \mathbf{b}. \tag{55}$$

Let $\tilde{\mathbf{x}} = \mathbf{O}\mathbf{x}$ denote the application of the Discrete Fourier Transform (DFT) to a vector $\mathbf{x}$. It can be shown that $\mathbf{O}$ is an orthogonal matrix (i.e. $\mathbf{O}^T\mathbf{O} = \mathbf{I}$), where $(\cdot)^T$ denotes transpose. By applying the transformation to (55) the transformed equation is obtained

$$(\mathbf{O}\mathbf{A}\mathbf{O}^T)(\mathbf{O}\mathbf{x}) = (\mathbf{O}\mathbf{b}). \tag{56}$$

It can be shown that the transformed system is diagonal (i.e. $\mathbf{O}\mathbf{A}\mathbf{O}^T = \mathbf{D}$, with $\mathbf{D}$ a diagonal matrix) provided that the matrix $\mathbf{A}$ is invariant under translations, i.e. the stencil of the operator is the same for all the cells of the mesh. Also, the boundary conditions must be periodic (but this restriction will be removed below, see Section 5.1.3).

Now consider the following algorithm that computes the solution of the linear system.

- Compute the transform of the right hand side: $\tilde{\mathbf{b}} = \mathbf{O}\mathbf{b}$.
- Solve the diagonal system in the transformed basis $\tilde{\mathbf{x}} = \mathbf{D}^{-1}\tilde{\mathbf{b}}$.
- Obtain the antitransformed solution vector by applying the inverse DFT: $\mathbf{x} = \mathbf{O}^T\,\tilde{\mathbf{x}}$.

The total operation count for this algorithm is two DFT's, plus one element-by-element vector multiply (the reciprocals of the values of the diagonal of $\mathbf{D}$ are precomputed), For N a power of 2 (i.e. $N = 2^p$) the Fast Fourier Transform (FFT) is an algorithm that

computes the DFT (and its inverse) in $O(N\log(N))$ operations, then the cost of the algorithm is $O(N\log(N))$.

Another possibility is Multigrid (MG), which is a *stationary iterative* method with cost $O(N)$ *for a given tolerance*, i.e. its cost is $O(N\log(\epsilon))$, where ϵ is the tolerance used as stopping criterion. On the other hand the FFT solver is a *direct method* with operation count $O(N\log(N))$ to solve the linear system to machine precision.

5. Numerical experiments

5.1. Convergence of IOP and AGP. Condition number of AGP

5.1.1. Convergence of IOP iteration

Figs. 7 and 8 show the convergence of IOP iteration for a $16\times16\times16$ and $64\times64\times64$ meshes on a computational domain which is a cube of unit side $L=1$. The body domain is a sphere of radius 0.3, i.e. $\Omega_{\text{bdy}}=\{\|\mathbf{x}\|\leqslant R=0.3\}$. As can be seen the IOP iteration curves exhibit the *typical linear rate of convergence of stationary methods*. The convergence for both meshes start at a high convergence rate of less than 3 iter/OM (iterations per order of magnitude), and then they switch to a slower convergence of 13.7 iter/OM for the 16^3 mesh and 12.3 iter/OM for the fine 64^3 mesh. However note that *the rates of convergence are independent of mesh refinement*.

5.1.2. Condition number for AGP does not degrade with refinement

The condition number of matrices for the Poisson problem have been computed with and without preconditioning (see Fig. 9).

- In the experiments the number of cells N_x along x ranges from 8 to 64.
- The Poisson problem is computed selecting the quadrangles whose center fall outside the body problem.
- In all cases the domain is the unit square with periodic boundary conditions.
- The bodies considered are: cylinder of radius 0.2, a vertical strip of width 0.5, and a square of side 0.5.
- The condition numbers are computed with Octave `cond ()` function.

Note that in all cases the nonpreconditioned matrix condition number grows as $O(N_x^2)$, whereas *with the preconditioning it remains constant*.

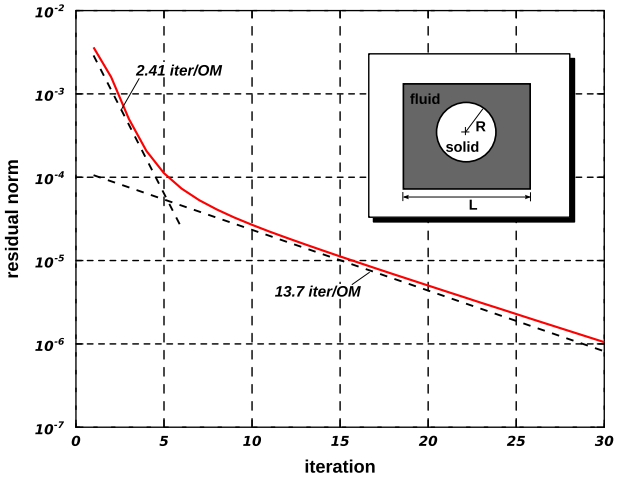

Fig. 7. Convergence of IOP loop for a sphere of $R=0.3$ in a cube of $L=1$, with a coarse mesh of $16\times16\times16$.

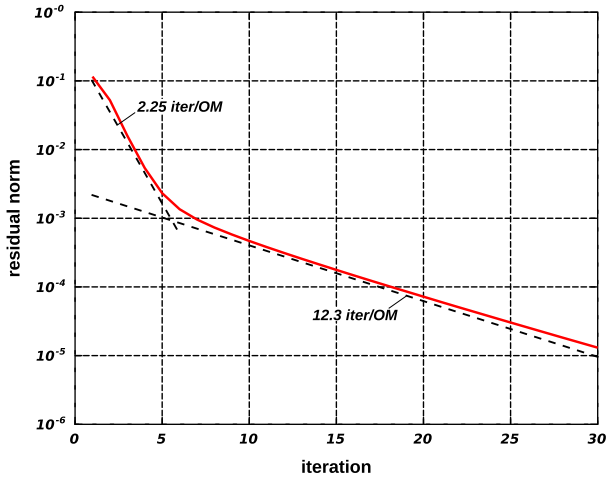

Fig. 8. Convergence of IOP loop for a sphere of $R=0.3$ in a cube of $L=1$, with a fine mesh of $64\times64\times64$.

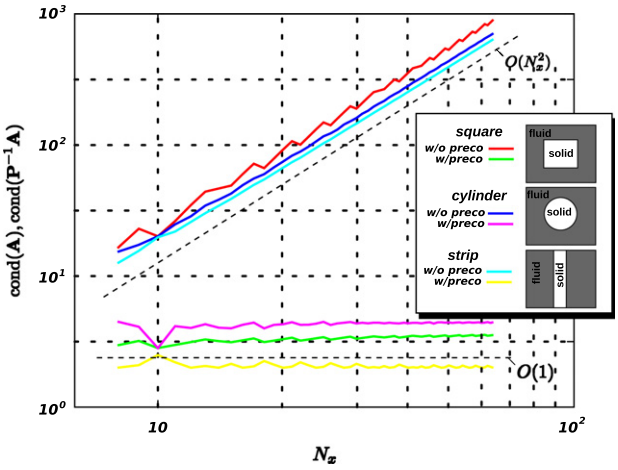

Fig. 9. Condition number of Poisson problem with and without the AGP preconditioning.

5.1.3. Bodies with large aspect ratio

Note that both IOP and AGP preconditioning are based on the inclusion of the solid domain in the computation of the pressure Poisson equation. As it have been shown, this causes convergence to degrade when objects with large aspect ratio are present in the domain. Consider the case where the fluid occupies the *interior* of a square

$$\Omega_{\text{fluid}}=\{(x,y)/\max(|x-L/2|,|y-L/2|)<L/2-b\} \tag{57}$$

where b is the width of the wall (see Fig. 10).

Two cases are considered, a fixed wall width of thickness $b=0.05L$, and the case $b=h$, i.e. the width of the wall is of just one cell. So as the mesh is refined, the aspect ratio of the wall increases. The condition number for the problem with and without preconditioning are shown.

In the case of a fixed value $b=0.05$ the condition number of the preconditioned case is bounded, whereas for the case of $b=h$ the condition number increases with aspect ratio. If no preconditioning is used the condition number increases as $O\left(N_x^2\right)$.

This case is of practical interest because it is a workaround to solve problems with solid boundary conditions. In its simplest form the FFT solver requires periodic boundary conditions in order to be applied. Other common boundary conditions like homoge-

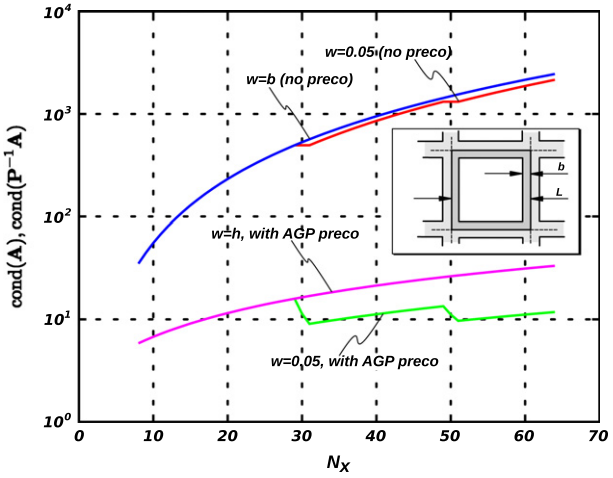

Fig. 10. Condition number for Poisson problem on a square, with and without AGP preconditioning.

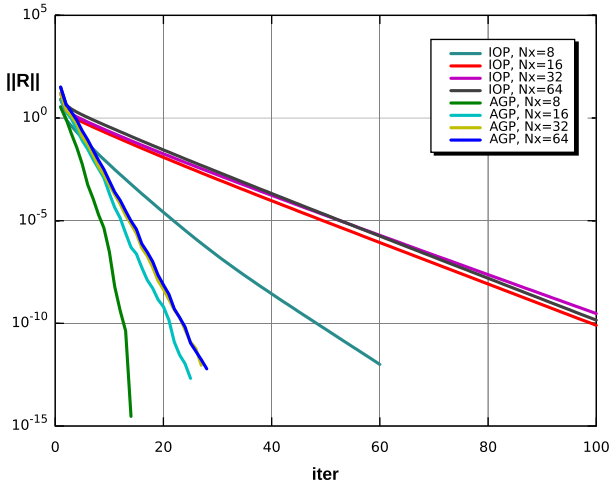

Fig. 11. Convergence histories for a 2D problem with a circular body of radius $R = 0.3$. Convergence is shown for both AGP and IOP, and several refinements. (N_x is the number of cells per side).

neous Dirichlet and Neumann boundary conditions can be also implemented using different flavors of the FFT, but if all combinations are to be considered (i.e. Dirichlet on some sides of the box, and Neumann on the others) it requires a tedious programming of all the cases and dispatch to the appropriate FFT routine.

But a solid boundary condition can be represent also by simply putting a thin layer of solid at the boundary. However, this can be inefficient if the additional work represented by the layer is significant. This numerical example shows that a layer as thin as a 5% of the side length of the box can be used, with very little degradation of the condition number. In such a case the increase in the computational time is not significant, approximately 30%, because the layer covers all the 6 sides of the box, but it can be shown that smaller widths b can be used for finer meshes. In the numerical results shown in following sections with closed cavities the width of the solid layer is 2.5% of the domain length. In such a case the computation overhead due to the wall layer is only 15%.

5.1.4. Convergence histories for IOP and AGP compared

In Fig. 11 the convergence histories for AGP and IOP in a 2D problem, with a circular body of radius $R = 0.3$, and several degrees of refinement $N_x = 8, 16, 32, 64$, where N_x is the number of cells per side are shown.

It is observed that the convergence histories tend to a fixed rate of convergence as the mesh is refined, in fact the convergence histories are almost the same for $N_x = 32$ and 64. This verifies the estimates discussed in Sections 2.4 and 3.1.

The rate of convergence is much higher for AGP. Note that if higher (weaker) tolerances (for instance 10^{-3}) are used then the convergence of both methods is similar. This is acceptable for non-critical applications like video-game and special effects, but usually not for engineering computations. If lower (stronger) tolerances (let's say 10^{-6}) are enforced then the difference is substantial.

5.2. Computational efficiency on GPU hardware

5.2.1. Computing times of FFT on GPU and CPU hardware

As it has been discussed, for large problems the most consuming time component of the algorithm are the two FFT applications per AGP iteration (same for IOP), so the efficiency of the available libraries will be assessed.

The GPU implementation was coded using the Compute Unified Device Architecture (CUDA) from Nvidia [7,20] (release 4.2, V0.2.1221). CUDA comes with an efficient FFT implementation called the CUFFT library. On the other hand, for CPU the Fastest Fourier Transform in the West (FFTW) [8,9] (release 3.1–2) library was used. The computing rates for this two libraries on the Nvidia GTX-580 GPUs, and processors Intel i7-3820@3.60 GHz, and Intel W3690@3.47 Ghz are shown in Figs. 12–14. The computing rate in Gflops is computed as

$$\text{rate[Gflops]} = 10^{-9} \times \frac{2N_v \log_2(N_v)}{\text{elapsed time [s]}} \tag{58}$$

where $N_v = N_x \cdot N_y \cdot N_z/2$ is the total size of the complex vector to be transformed. Note that this is half the number of cells, since using the *R2C* (for *Real to Complex*) flavor of the FFT the number of operations can be reduced by a half. The computing rate for the GTX-580 is near 240 Gflops in simple precision for meshes of $256 \times 128 \times 128$ (8 million cells, 4 million cells in the complex vector). For double precision the rate drops by almost a factor of 4. Note that previous boards *not* in the Tesla family had a typical speed relation of 8:1 from simple to double precision, so this ratio 4:1 signifies an improvement for the GTX-580. Typical boards on the Tesla family have a speed ratio of 2:1.

On the other hand the fastest CPU processor tested is the Sandy Bridge i7-3820 which (multi-threaded in its 6 cores) peaks at 20

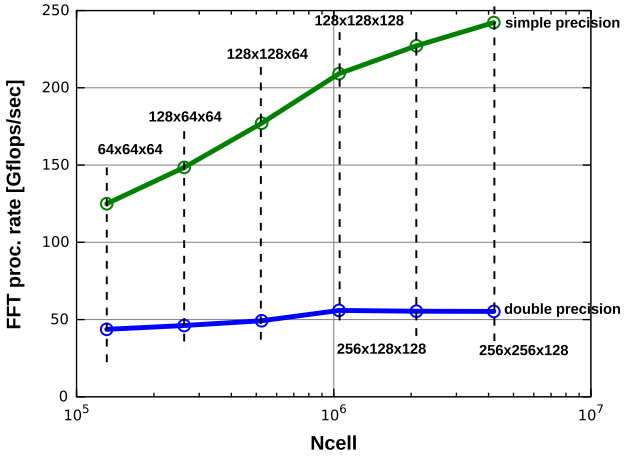

Fig. 12. Computing rates for the CUFFT implementation on the Nvidia GTX-580 GPU.

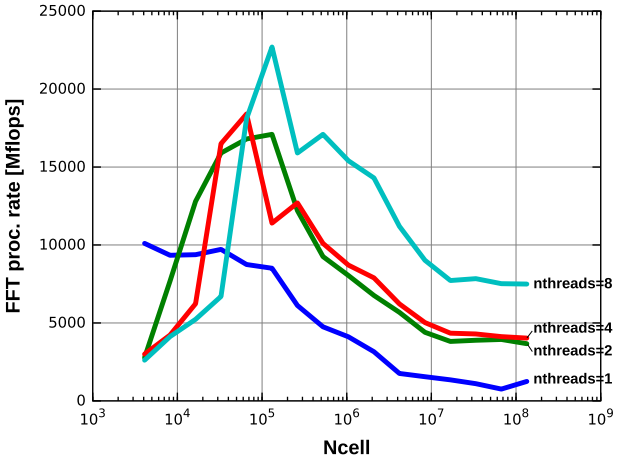

Fig. 13. Computing rates for the FFTW (SMP) implementation on the Intel i7-3820 (Sandy Bridge) CPU (double precision).

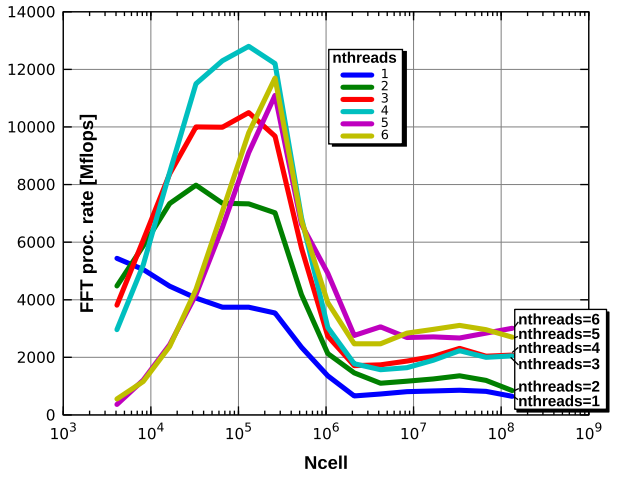

Fig. 14. Computing rates for the FFTW (SMP) implementation on the Intel W3690 (Nehalem) CPU (double precision).

Gflops for vectors of size $O(10^5)$. However this performance drops at almost 8 Gflops for large vectors, when the vector does not fit in the processor's cache. So, in double precision for large vectors there is a speedup of a factor 8 between the FFT on the GPU board and the CPU.

Note also that, in contrast with the deterioration in performance of the CPU's, the computing rate of the CUFFT in simple precision seams to steadily increase as the vector length increases, whereas the double precision shows a small increase in performance. This means that it is likely that the performance will be kept for boards with a larger device memory (the GTX-580 has 3 GB RAM) allowing for larger computations in a single device.

5.2.2. Computing rates

In Fig. 15 the computing rates in Mcell/s for the code presented in this article on an Nvidia GTX-580 GPU, and a Nvidia Tesla C2050, with single (SP) and double precision (DP), are shown. In DP for large meshes it reaches a rate of 60 Mcell/s. As a reference, the same algorithm was implemented in CPU using the GNU `g++` compiler (with optimization flags `-O3 -funroll-loops`), obtaining a rate on one core of the Intel i7-3820@3.47 GHz (Sandy Bridge) of 1.7 Mcell/s. Assuming perfect scalability a maximum of 6.8

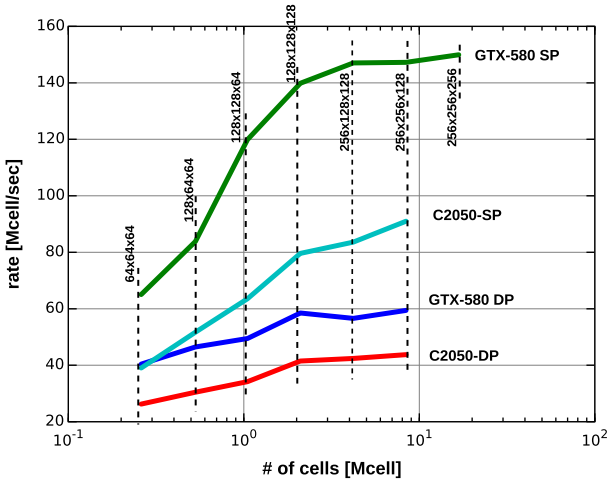

Fig. 15. Computing rates in Mcell/s for the algorithm presented in this paper in an Nvidia GTX-580 GPU, and a Nvidia Tesla C2050, with single (SP) and double precision (DP).

Mcell/s at most would be reached using the four cores of the i7-3820, which translates in a speedup of at least 7:1 for the GPU over this CPU.

5.3. Real time computing

Many applications in engineering need Real Time Computing (RTC), i.e. to have a code fast enough such that $T_{comp} \leqslant T_{sim}$, where T_{comp} is the computing time needed for simulating T_{sim} seconds of the physical problem. For instance this is the case in applications where the computations are needed for take some action back on the physical process, as in control or disaster management.

The approach presented here allows to do RTC in moderately large meshes. Consider for instance a mesh of 128^3 cells ($\approx$2 Mcell). The computing time on the GTX-580 in SP is (see Fig. 15) 140 Mcell/s, so each time step takes approximately 2 Mcell/(140 Mcell/s) = 0.014 s per time step, i.e. 70 steps per second can be computed.

A von Neumann stability analysis shows that the QUICK stabilization scheme is inconditionally unstable if advanced in time with Forward Euler. With a second order Adams-Bashfort scheme the critical Courant-Friedrichs-Lewy (CFL) number is CFL < 0.588 for an scalar advection problem, and for Navier–Stokes (at high Reynolds numbers) it is somewhat lower, CFL < 0.5. If $L = 1$ (m), and maximum velocity $u = 1$ (m/s), mesh step is $h = 1/128$ (m), then the critical time step is $\Delta t = 0.5h/u = 0.004$ (s), so that $T_{sim} = 70\Delta t = 0.28$ (s) can be computed in $T_{comp} = 1$ (s) of computing time. It means that for such a mesh the computations go 1:4 slower than the physical process. Other approach to RTC is to circumvent the restriction of CFL < 1 characteristic of explicit methods [11].

5.4. Flow simulations

Numerical simulations of several flows involving moving bodies are shown in Figs. 16–20. In all cases (except for the case of the example in Section 5.4.3) the flows represent a body moving inside a square or cubic cavity of length side 1 (m). In order to circumvent the restriction of periodic boundaries intrinsic to the FFT solver, a thin layer (2.5% of the square or cubic domain side length) is defined as a fixed body. In all cases the color corresponds to $\log_{10}(|\boldsymbol{\omega}|)$, i.e. the absolute magnitude of the vorticity vector $\boldsymbol{\omega} = \nabla \times \mathbf{u}$ in logarithmic scale. This quantity helps in the visualization of boundary layers, since the magnitude of vorticity has variations of several orders of magnitude there at high Reynolds

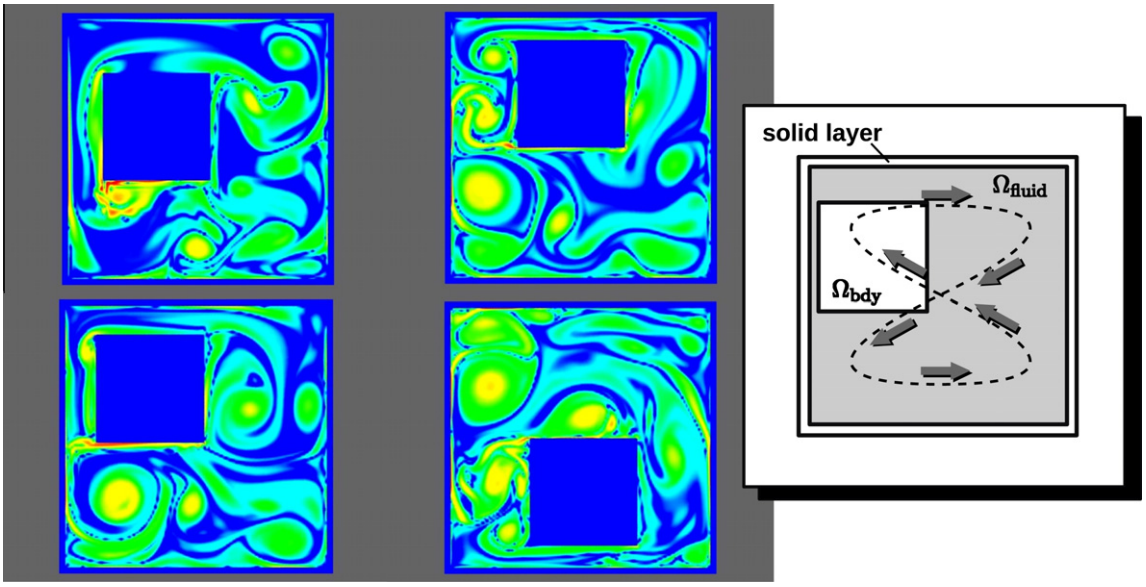

Fig. 16. Colormap of $\log_{10}(|\omega|)$ for a square of side $L_s = 0.4$ (m) moving in a square domain of side $L = 1$ (m). The square moves forming a Lissajous 8-shaped curve. (For interpretation of the references to color in this figure legend, the reader is referred to the web version of this article.)

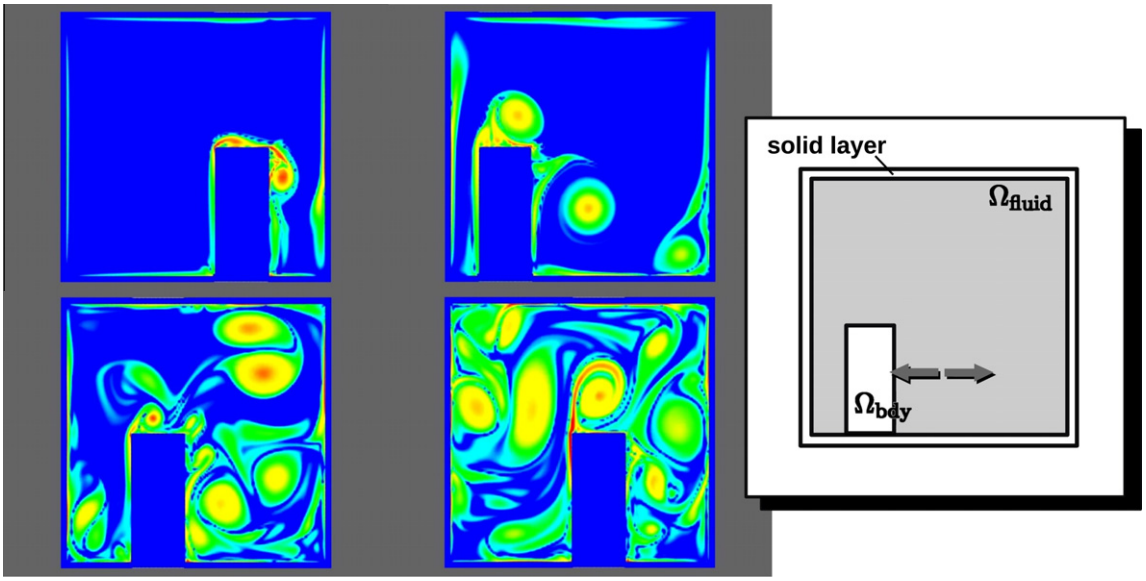

Fig. 17. Colormap of $\log_{10}(|\omega|)$ for a rectangle sliding on the bottom of the domain. (For interpretation of the references to color in this figure legend, the reader is referred to the web version of this article.)

numbers. In 2D cases the mesh was 128×128 and in 3D cases $128 \times 128 \times 128$. In all cases the side of the domain (square in 2D, cube in 3D) was $L = 1$ (m) and kinematic viscosity was $\nu = 6.33 \times 10^{-5}$ (m^2/s).

5.4.1. Square moving in curved trajectory

The body is a square of side $L_s = 0.4$ (m), and the center of the body (x_c, y_c) describes an 8-shaped Lissajous curve, described by

$$x_c = \frac{L}{2} + A\cos(2\omega t), y_c = \frac{L}{2} + A\cos\left(\frac{\pi}{2} + \omega t\right), \quad \omega = 1\ (\mathrm{s}^{-1}),\ A = 0.2\ (\mathrm{m}) \tag{59}$$

As the body displaces fluid high levels of vorticity can be observed at the vertices. As the simulation progresses large vortices remain rotating in the fluid with long filamentary vorticity layers that are a characteristic 2D feature (they are unstable in 3D).

5.4.2. Moving rectangular obstacle

The body is a rectangle of height $H = 0.5$ (m) and width $W = 0.2$ (m). An harmonic horizontal displacement as follows

$$x_c = (L/2) + A\cos(\omega t), \quad \omega = 1\ (\mathrm{s}^{-1}),\ A = 0.3\ (\mathrm{m}), \tag{60}$$

is imposed. As the body displaces fluid a large concentration of vorticity is observed in the upper corner of the body, with characteristic trailing filamentary vortex layers that detach from the corners.

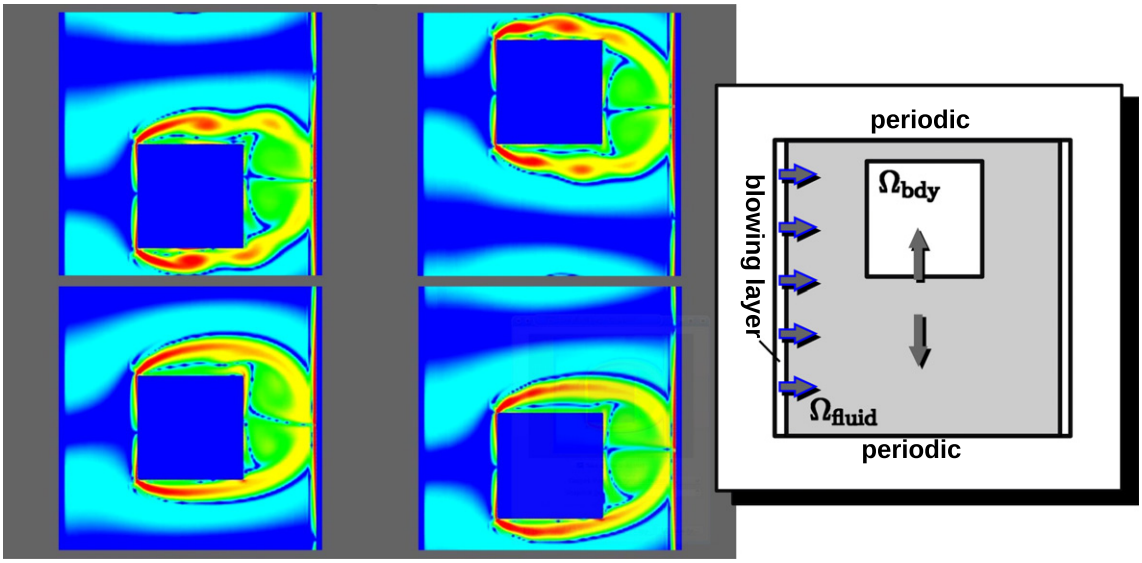

Fig. 18. Colormap of $\log_{10}(|\omega|)$ for a square body performing harmonic motion in the vertical direction with a cross flow in the horizontal direction. (For interpretation of the references to color in this figure legend, the reader is referred to the web version of this article.)

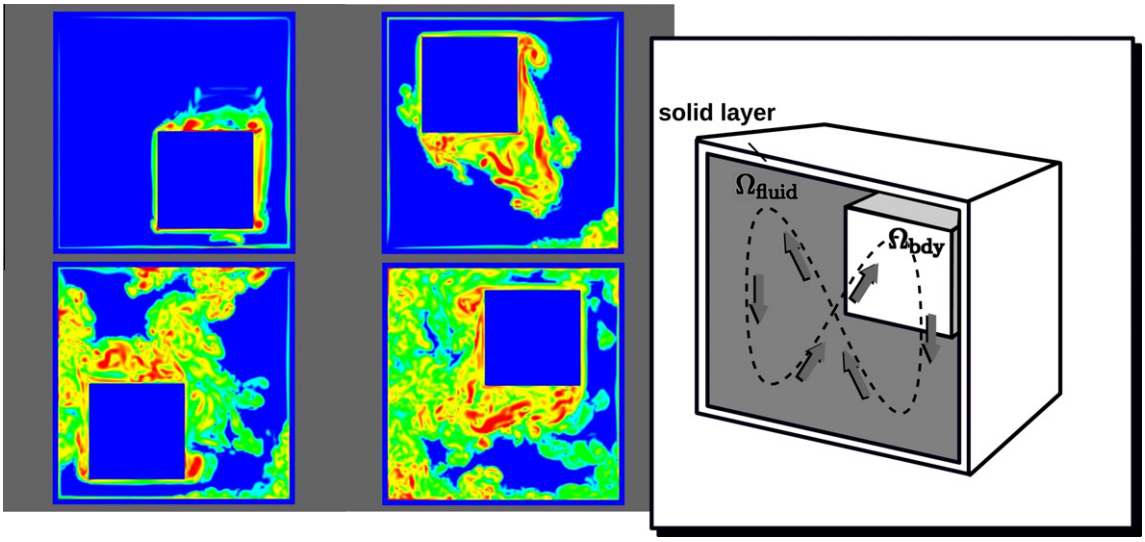

Fig. 19. Colormap of $\log_{10}(|\omega|)$ for a cube moving in a Lissajous 8-shaped curve. (For interpretation of the references to color in this figure legend, the reader is referred to the web version of this article.)

5.4.3. Square moving vertically with mean horizontal flow

In this example the exterior boundary of the computational domain is not at rest, but rather it is intended to generate a mean flow that impinges on the body. This freestream flow is obtained with a layer of width 0.025 (m) at the left and right sides were a positive x velocity of $u = 1$ (m/s) is imposed. Periodic boundary conditions are imposed in the vertical y direction. The body is a square of side $L_s = 0.4$ (m), the center of the body (x_c, y_c) is centered in the x direction and experiences an harmonic vertical movement

$$\begin{aligned} &y_c = (L/2) + A\cos(\omega t), \\ &\omega = 0.5\ (\mathrm{s}^{-1}),\ A = 0.2\ (\mathrm{m}). \end{aligned} \tag{61}$$

An accelerating boundary layer is formed at the left side facing the fluid stream. The boundary layer accelerate towards the corners and detach there. If the vertical movement were at a constant velocity then the flow would be equivalent to a fixed body with an impinging stream at an angle of attack. A notable feature of the flow is that when the body reaches the extreme positions in the y direction the vortex layers become unstable and start shedding vortices, whereas when the body is moving the vortex layer stabilizes.

5.4.4. Moving cube

This is a 3D case. The center (x_c, y_c, z_c) of a cube of side $L_s = 0.4$ (m) is describing a Lissajous 8-shaped figure in the $z = 0.66$ (m) plane, as follows

$$\begin{aligned} &x_c = L/2 + A\cos(\omega t), \\ &y_c = L/2 + A\cos\left(\frac{\pi}{2} + 2\omega t\right), \\ &z_c = 0.66\ (\mathrm{m}), \\ &\omega = 2\ (\mathrm{s}^{-1}),\ A = 0.4\ (\mathrm{m}) \end{aligned} \tag{62}$$

This is similar to the case Section 5.4.1 but 3D. The large filamentary vortex layers are no more present, but instead there is a large amount of small eddies characteristic of a 3D flow.

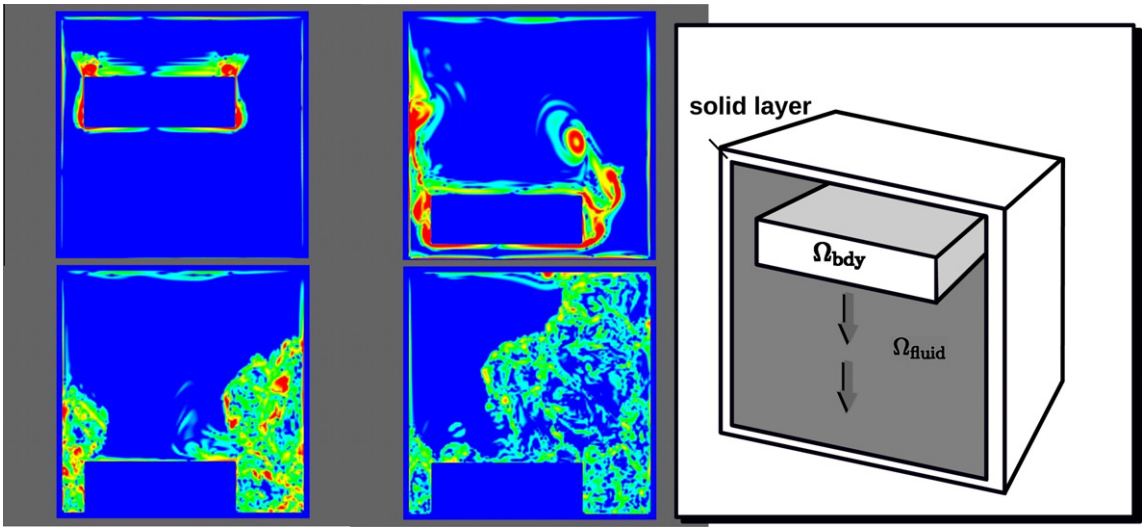

Fig. 20. Colormap of $\log_{10}(|\omega|)$ for a falling block. (For interpretation of the references to color in this figure legend, the reader is referred to the web version of this article.)

5.4.5. Falling block

The body is a parallelepiped block of dimensions $L_x = L_z = 0.6$ (m), $L_y = 0.2$ (m). The center of the body is initially at $(x_c, y_c, z_c) = (0.4125, 0.95, 0.5)$ [m] and starts falling vertically with a velocity of 1 (m/s). As the body falls it displaces a large quantity of fluid that forms a turbulent region expanding from both sides of the block.

6. Conclusions

We presented a new method called Accelerated Global Preconditioning for solving the incompressible Navier–Stokes equations with moving bodies. The algorithm is based on a pressure segregated, staggered grid, Finite Volume formulation and uses a FFT solver for preconditioning the CG solution of the Poisson problem. Theoretical estimates of the condition number of the preconditioned Poisson problem are given, and several numerical examples are presented validating these estimates. The algorithm is specially suited for implementation on GPU hardware. The condition number of the preconditioned Poisson equation does not degrade with refinement. The algorithm allows computing 3D problems in real time on moderately large meshes for many problems of practical interest in the area of Computational Fluid Dynamics.

Acknowledgments

This work has received financial support from Consejo Nacional de Investigaciones Científicas y Técnicas (CONICET, Argentina, PIP 5271/05), Universidad Nacional del Litoral (UNL, Argentina, Grant CAI + D 2009-65/334), Agencia Nacional de Promoción Científica y Tecnológica (ANPCyT, Argentina, grants PICT-1506/2006, PICT-1141/2007, PICT-0270/2008), and European Research Council (ERC) Advanced Grant, Real Time Computational Mechanics Techniques for Multi-Fluid Problems (REALTIME, Reference: ERC-2009-AdG). The authors made extensive use of *Free Software* as GNU/Linux OS, GCC/G++ compilers, Octave, and *Open Source* software as VTK among many others. In addition, many ideas from these packages have been inspiring to them.

References

[1] Adams S, Payne J, Boppana R. Finite difference time domain (FDTD) simulations using graphics processors. HPCMP users group conference, 0:334–338; 2007.
[2] Bell N, Garland M. Implementing sparse matrix-vector multiplication on throughput-oriented processors. In: SC '09: proceedings of the conference on high performance computing networking. Storage and analysis. New York, NY, USA: ACM; 2009. p. 1–11.
[3] Corrigan A, Camelli F, Löhner R, Wallin J. Running unstructured grid-based CFD solvers on modern graphics hardware. Int J Numer Methods Fluids 2011;66(2):221–9.
[4] Crane K, Llamas I, Tariq S. Real-time simulation and rendering of 3D fluids. GPU Gems 2007;3:633–75.
[5] Elcott S, Tong Y, Kanso E, Schröder P, Desbrun M. Stable, circulation-preserving, simplicial fluids. In: SIGGRAPH Asia '08: ACM SIGGRAPH ASIA 2008 courses. New York, NY, USA: ACM; 2008. p. 1–11.
[6] Elsen E, LeGresley P, Darve E. Large calculation of the flow over a hypersonic vehicle using a GPU. J Comput Phys 2008;227(24):10148–61.
[7] Farber R. CUDA application design and development. Morgan Kaufmann; 2011.
[8] Frigo M, Johnson S. FFTW: an adaptive software architecture for the FFT. In: Acoustics, speech and signal processing, 1998. Proceedings of the 1998 IEEE international conference on, vol. 3. IEEE; 1998. p. 1381–4.
[9] Frigo M, Johnson S. FFTW: fastest fourier transform in the west. In: Astrophysics source code library, record ascl: 1201.015, vol. 1; 2012. p. 01015.
[10] Goddeke D, Strzodka R, Mohd-Yusof J, McCormick P, Wobker H, Becker C, et al. Using GPUs to improve multigrid solver performance on a cluster. Int J Comput Sci Eng 2008;4(1):36–55.
[11] Idelsohn S, Nigro N, Limache A, Oñate E. Large time-step explicit integration method for solving problems with dominant convection. Comput Methods Appl Mech Eng 2012;217–220:168–85.
[12] Irving G, Guendelman E, Losasso F, Fedkiw R. Efficient simulation of large bodies of water by coupling two and three dimensional techniques. ACM Trans Graph 2006;25(3):805–11.
[13] Kelley C. Iterative methods for linear and nonlinear equations. Society for Industrial Mathematics; 1995.
[14] Klöckner A, Warburton T, Bridge J, Hesthaven J. Nodal discontinuous Galerkin methods on graphics processors. J Comput Phys 2009;228(21):7863–82.
[15] Lastra M, Mantas JM, Urena C, Castro MJ, García-Rodríguez JA. Simulation of shallow-water systems using graphics processing units. Math Comput Simul 2009;80(3):598–618.
[16] Leonard B. A stable and accurate convective modelling procedure based on quadratic upstream interpolation. Comput Methods Appl Mech Eng 1979;19(1):59–98.
[17] Molemaker J, Cohen JM, Patel S, Noh J. Low viscosity flow simulations for animation. In: SCA '08: proceedings of the 2008 ACM SIGGRAPH/Eurographics symposium on computer animation, Aire-la-Ville, Switzerland, Switzerland; 2008. Eurographics Association. p. 9–18.
[18] Mossaiby F, Rossi R, Dadvand P, Idelsohn S. OpenCL-based implementation of an unstructured edge-based finite element convection–diffusion solver on graphics hardware. Int J Numer Methods Eng 2012;89:1635–51.
[19] Mullen P, Crane K, Pavlov D, Tong Y, Desbrun M. Energy-preserving integrators for fluid animation. In: SIGGRAPH '09: ACM SIGGRAPH 2009 papers, New York, NY, USA. ACM; 2009. pp. 1–8.
[20] Nickolls J, Buck I, Garland M, Skadron K. Scalable parallel programming with CUDA. ACM Queue 2008;6(2):40–53.
[21] Paz R, Nigro N, Storti M. On the efficiency and quality of numerical solutions in CFD problems using the interface strip preconditioner for domain decomposition. Int J Numer Methods Fluids 2006;52:89–118.
[22] Paz R, Storti M. An interface strip preconditioner for domain decomposition methods application to hydrology. Int J Numer Methods Eng 2005;62(13):1873–94.

[23] Peskin C. The immersed boundary method. Acta Numer 2002;11(0):479–517.
[24] Rinaldi P, García Bauza C, Vénere M, Clausse A. Paralelización de autómatas celulares de aguas superficiales sobre placas gráficas. In: Cardona A, Storti M, Zuppa C, editors. Mecánica Computacional, vol. XXVII; 2008. p. 2943–57.
[25] Ryoo S, Rodrigues C, Baghsorkhi S, Stone S, Kirk D, Hwu W. Optimization principles and application performance evaluation of a multithreaded GPU using CUDA. In: Proceedings of the 13th ACM SIGPLAN symposium on principles and practice of parallel programming. ACM; 2008. p. 73–82.
[26] Storti M, Dalcfn L, Paz R, Yommi A, Sonzogni V, Nigro N. A preconditioner for the Schur complement matrix. Adv Eng Software 2006;37:754–62.
[27] Thibault JC, Senocak I. CUDA implementation of a Navier–Stokes solver on multi-GPU desktop platforms for incompressible flows. In: AIAA, editor. 47th AIAA aerospace sciences meeting including the new horizons forum and aerospace exposition (Disc 1); 2009. p. 1–15.
[28] Wang X, Wang C, Zhang L. Semi-implicit formulation of the immersed finite element method. Comput Mech 2012;49:421–30.
[29] Wu E, Liu Y, Liu X. An improved study of real-time fluid simulation on GPU: research articles. Comput Animat Virtual Worlds 2004;15(3–4):139–46.