An FFT Preconditioning Technique for the Solution of Incompressible Flow with Fractional Step Methods on GPGPU's

by M.Storti, S.Costarelli, R.Paz, L.Dalcin

Centro Internacional de Métodos Computacionales en Ingeniería - CIMEC
INTEC, (CONICET-UNL), Santa Fe, Argentina
mario.storti@gmail.com
http://www.cimec.org.ar/mstorti

((version texstuff-1.0.36-213-g1864495 Wed Apr 11 09:24:51 2012 +0200) (date Wed Apr 11 09:42:15 2012 +0200))

Solution of incompressible Navier-Stokes flows on GPU

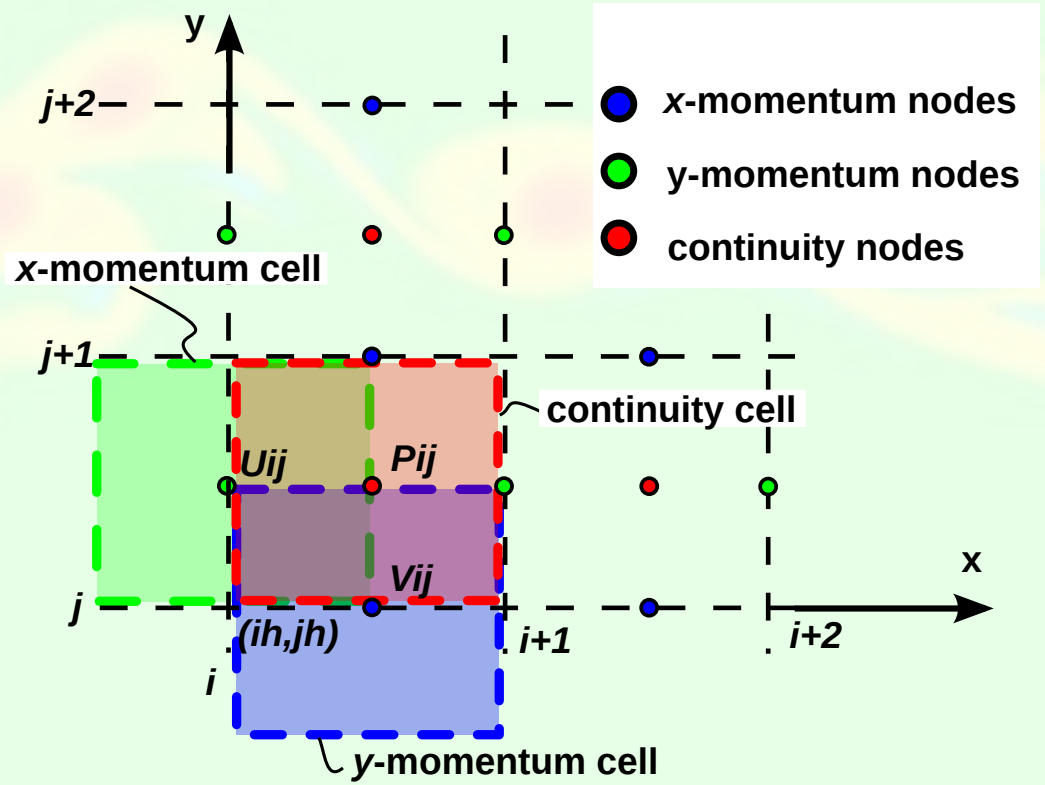
- GPU's are less efficient for algorithms that require access to the *card's (device) global memory*. Shared memory is much faster but usually *scarce* (16K per thread block in the Tesla C1060)  .
- The best algorithms are those that make computations for one cell requiring only information on that cell and their neighbors. These algorithms are classified as *cellular automata (CA)*.
- *Lattice-Boltzmann* and *explicit F⋆M (FDM/FVM/FEM)* fall in this category.
- *Structured meshes* require less data to exchange between cells (e.g. neighbor indices are computed, no stored), and so, they require less shared memory. Also, very fast solvers like *FFT-based* (*Fast Fourier Transform*) or *Geometric Multigrid* are available  .

((version texstuff-1.0.36-213-g1864495 Wed Apr 11 09:24:51 2012 +0200) (date Wed Apr 11 09:42:15 2012 +0200))

Fractional Step Method on structured grids with QUICK

Proposed by *Molemaker et.al. SCA'08: 2008 ACM SIGGRAPH,* Low viscosity flow simulations for animation.

- **Fractional Step Method (a.k.a. pressure segregation)**
- **u, v, w and continuity cells are *staggered*.**
- ***QUICK* advection scheme is used in the predictor stage.**
- **Poisson system is solved with *IOP (Iterated Orthogonal Projection)* (to be described later), on top of *Geometric MultiGrid***



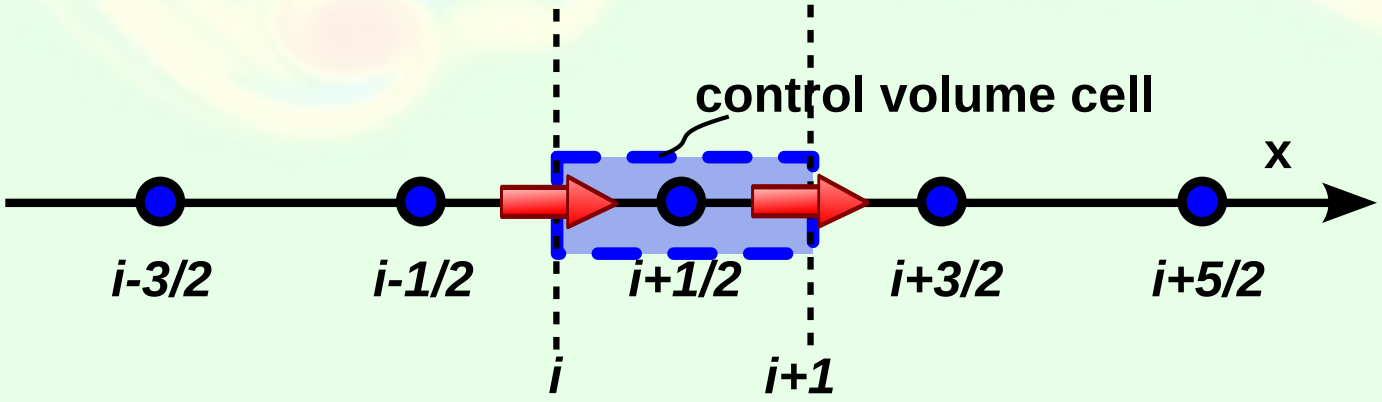
((version texstuff-1.0.36-213-g1864495 Wed Apr 11 09:24:51 2012 +0200) (date Wed Apr 11 09:42:15 2012 +0200))

Quick advection scheme

***1D Scalar advection diffusion*: a= advection velocity, ϕ advected scalar.**

$$\left.\frac{\partial}{\partial x}(a\phi)\right|_{i+1/2} \approx \frac{(a\phi^Q)_{i+1} - (a\phi^Q)_i}{\Delta x},$$

$$\phi_i^Q = \begin{cases} 3/8\,\phi_{i+1/2} + 6/8\,\phi_{i-1/2} - 1/8\,\phi_{i-3/2}, & \text{if } a > 0, \\ 3/8\,u_{i-1/2} + 6/8\,u_{i+1/2} - 1/8\,u_{i+3/2}, & \text{if } a < 0, \end{cases}$$



(launch video khinstab)

((version texstuff-1.0.36-213-g1864495 Wed Apr 11 09:24:51 2012 +0200) (date Wed Apr 11 09:42:15 2012 +0200))

Solution of the Poisson equation on embedded geometries

- Solution of the ***Poisson equation*** is, for large meshes, the more CPU consuming time stage in Fractional-Step like Navier-Stokes solvers.
- One approach for the solution is the ***IOP (Iterated Orthogonal Projection)*** algorithm.
- It is based on solving iteratively the Poisson eq. on the ***whole domain (fluid+solid)***. Solving in the whole domain is fast, because algorithms like Geometric Multigrid or FFT can be used. Also, they are very efficient running on GPU's  .
- However, if we solve in the whole domain, then we can't enforce the boundary condition $(\partial p/\partial n) = 0$ at the solid boundary which, then means the violation of the ***condition of impenetrability at the solid boundary***  .

((version texstuff-1.0.36-213-g1864495 Wed Apr 11 09:24:51 2012 +0200) (date Wed Apr 11 09:42:15 2012 +0200))

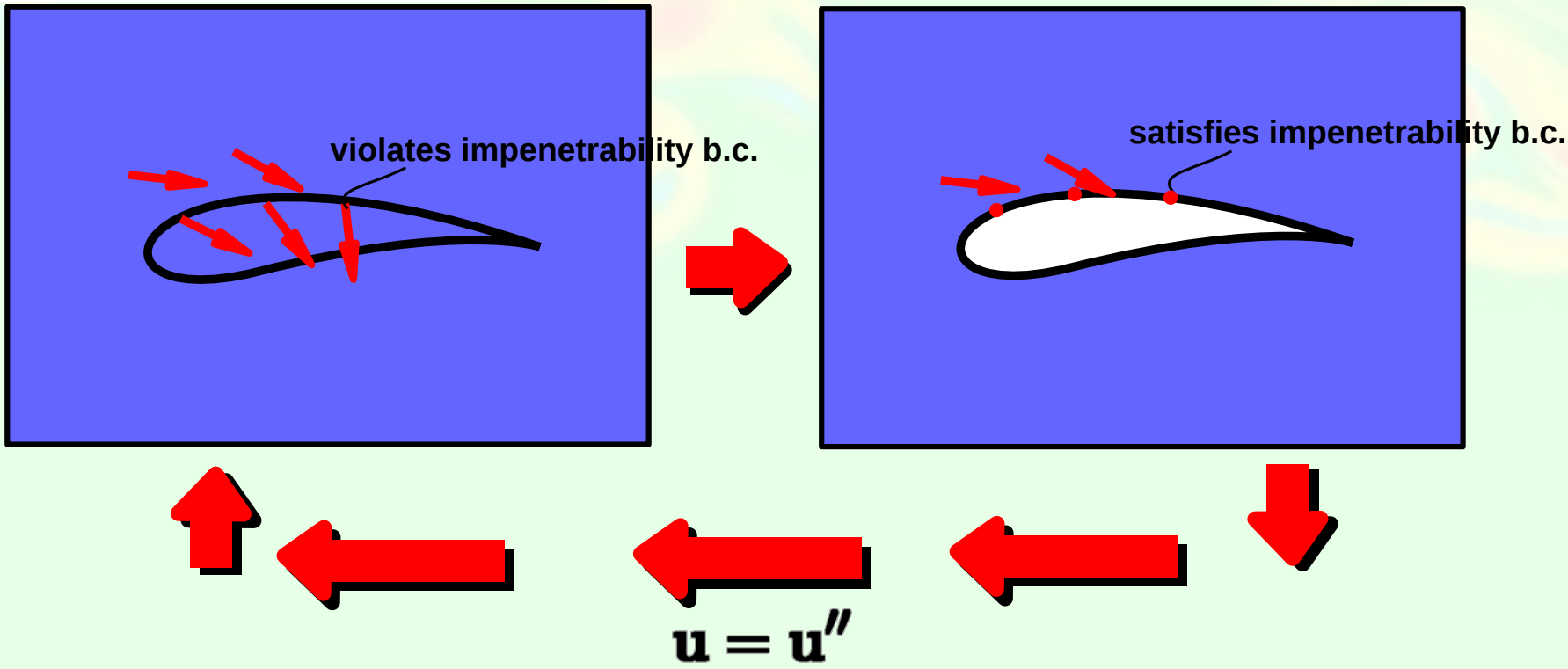
The IOP (Iterated Orthogonal Projection) method

The method is based on succesively solve for the incompressibility condition (on the whole domain: solid+fluid), and impose the boundary condition.

$$\mathbf{u}' = \mathbf{\Pi}_{\mathrm{div}}(\mathbf{u}) \begin{cases} \mathbf{u}' = \mathbf{u} - \nabla P, \\ \Delta P = \nabla \cdot \mathbf{u}, \end{cases}$$

on the whole domain (fluid+solid)

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



((version texstuff-1.0.36-213-g1864495 Wed Apr 11 09:24:51 2012 +0200) (date Wed Apr 11 09:42:15 2012 +0200))

The IOP (Iterated Orthogonal Projection) method (cont.)

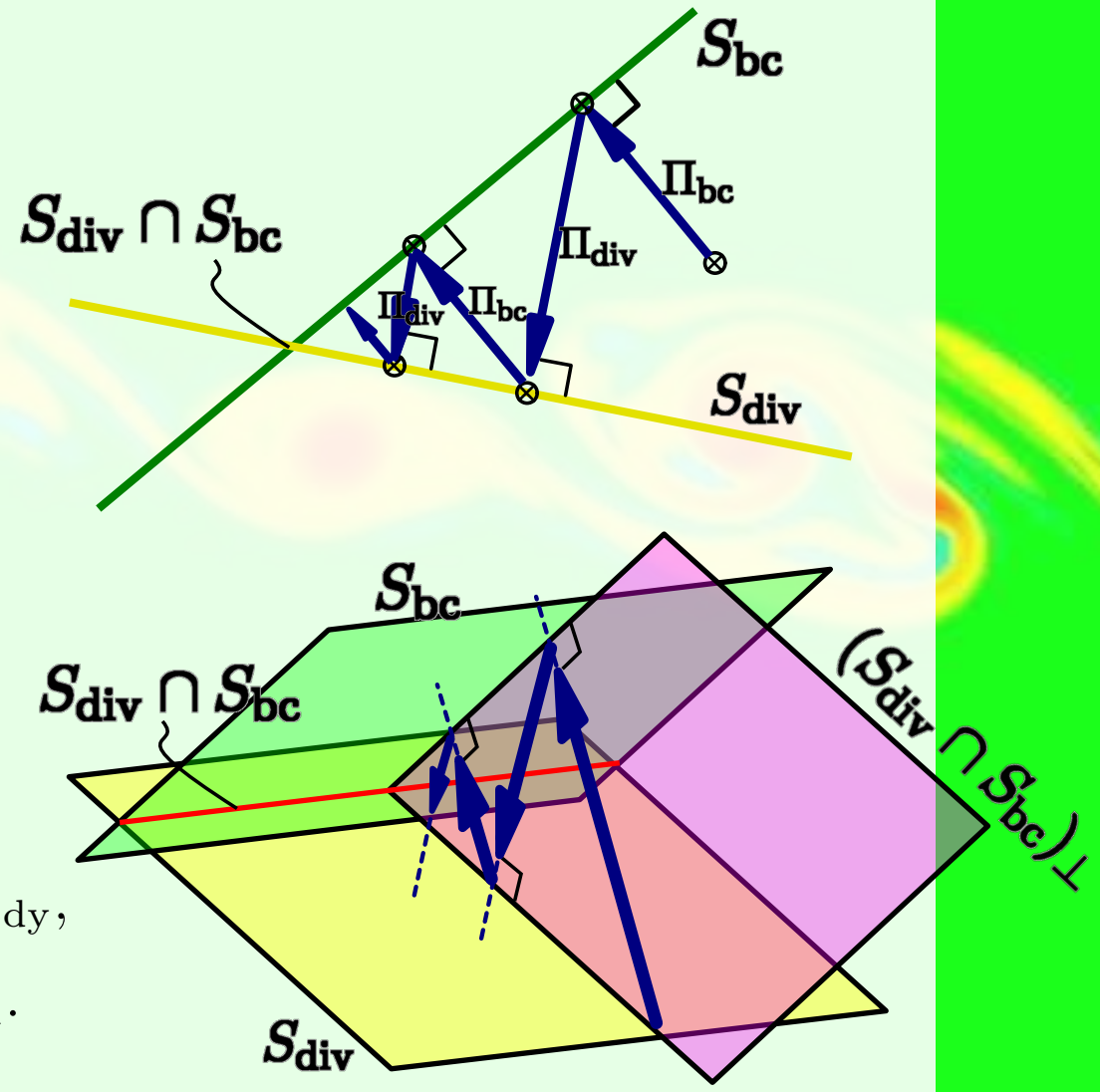
$$\mathbf{w}^{k+1} = \mathbf{\Pi}_{\mathrm{bdy}} \mathbf{\Pi}_{\mathrm{div}} \mathbf{w}^{k}.$$

Projection on the space of *divergence-free* velocity fields:

$$\mathbf{u}' = \mathbf{\Pi}_{\mathrm{div}}(\mathbf{u}) \begin{cases} \mathbf{u}' = \mathbf{u} - \nabla P, \\ \Delta P = \nabla \cdot \mathbf{u}, \end{cases}$$

Projection on the space of velocity fields that satisfy the *impenetrability boundary condition*

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

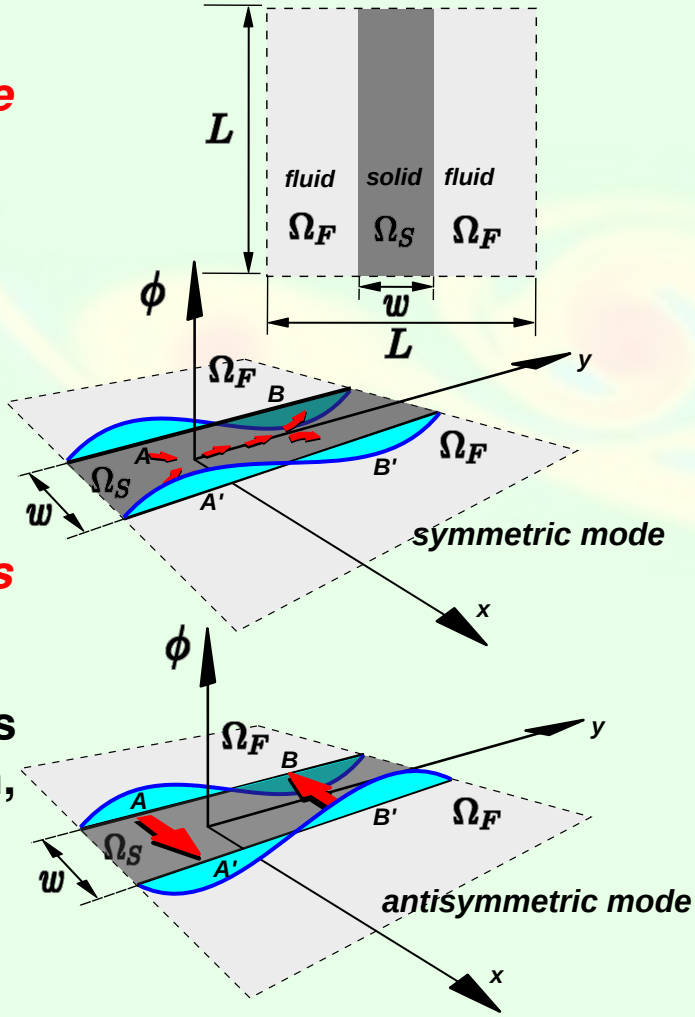


((version texstuff-1.0.36-213-g1864495 Wed Apr 11 09:24:51 2012 +0200) (date Wed Apr 11 09:42:15 2012 +0200))

INTEC UNL CONICET CIMEC

Convergence of IOP

- $\Pi_{\mathrm{bdy}}, \Pi_{\mathrm{div}}$ are orthogonal projection operators on $L_2 \implies$ the algorithm converges, with *linear rate of convergence*  .
- Rate of convergence is $O(1)$, i.e. *NOT depending on refinement*  . For instance for an embedded sphere, the residual is reduced to a factor of 0.1 in 3 iterations. However, the rate of convergence *degrades when thin surfaces are present*  .
- In videogame software, and special effects animation, 3 iterations are usually enough, but *for engineering purposes this is insufficient* and an algorithm with better convergence properties is needed.



((version texstuff-1.0.36-213-g1864495 Wed Apr 11 09:24:51 2012 +0200) (date Wed Apr 11 09:42:15 2012 +0200))

Using IOP/AGP with the FFT transform

- When solving the projection problem $\mathbf{u}' = \mathbf{\Pi}_{\mathrm{div}}(\mathbf{u})$ for IOP or the preconditioning for AGP, we have to solve a ***Poisson problem on the whole (fluid+solid) domain***. This is normally done with a ***Geometric Multigrid*** solver which has a complexity $O(N \log \epsilon)$ (N=nbr of grid cells, ϵ=tolerance). It is an ***iterative solver***.
- On the other hand, FFT solves the same problem in $O(N \log N)$. It is a ***direct solver***.

((version texstuff-1.0.36-213-g1864495 Wed Apr 11 09:24:51 2012 +0200) (date Wed Apr 11 09:42:15 2012 +0200))

Accelerated Global Preconditioning (AGP)

- The IOP algorithm iterates on the ***velocity*** u state.
- A method based on ***pressure*** would be more efficient, and in particular in the GPGPU, due to a better use of the ***shared memory*** .
- In addition, IOP is a stationary method (with linear rate of convergence) . We look for an ***accelerated Krylov space*** algorithm (CG) .
- The proposed ***AGP algorithm*** is to solve the fluid pressure problem with ***PCG (Preconditioned Conjugate Gradient)*** with the solution on the ***whole (fluid+solid) domain***.
- It can be shown that the ***condition number*** of the preconditioned matrix is also $O(1)$.
- It is an ***accelerated method***, so convergence is much better than IOP; for the sphere with three iterations we have a reduction of 1e-3 in the residual (while IOP gives a reduction of 0.1) .
- Conditioning degrades also for ***thin surfaces*** .

Accelerated Global Preconditioning (AGP) (cont.)

To solve:

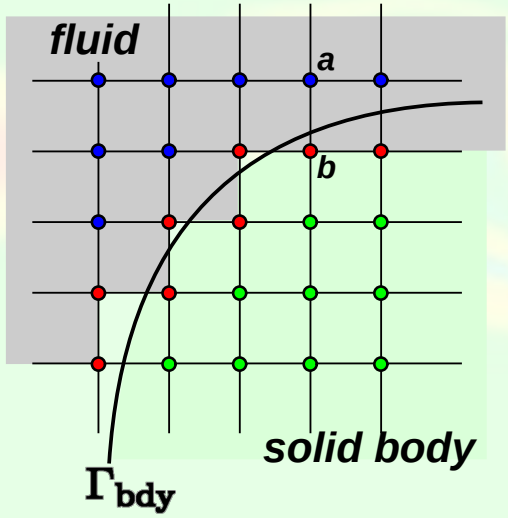
$$\begin{bmatrix} \mathbf{A}_{FF} & \mathbf{A}_{FB} \\ \mathbf{A}_{BF} & \mathbf{A}_{BB} \end{bmatrix} \begin{bmatrix} \mathbf{x}_F \\ \mathbf{x}_B \end{bmatrix} = \begin{bmatrix} \mathbf{b}_F \\ \mathbf{b}_B \end{bmatrix}$$

AGP Preconditioning:

$$\mathbf{P}_{AGP}\mathbf{x}_{FB} = \mathbf{y}_{FB}$$

defined by

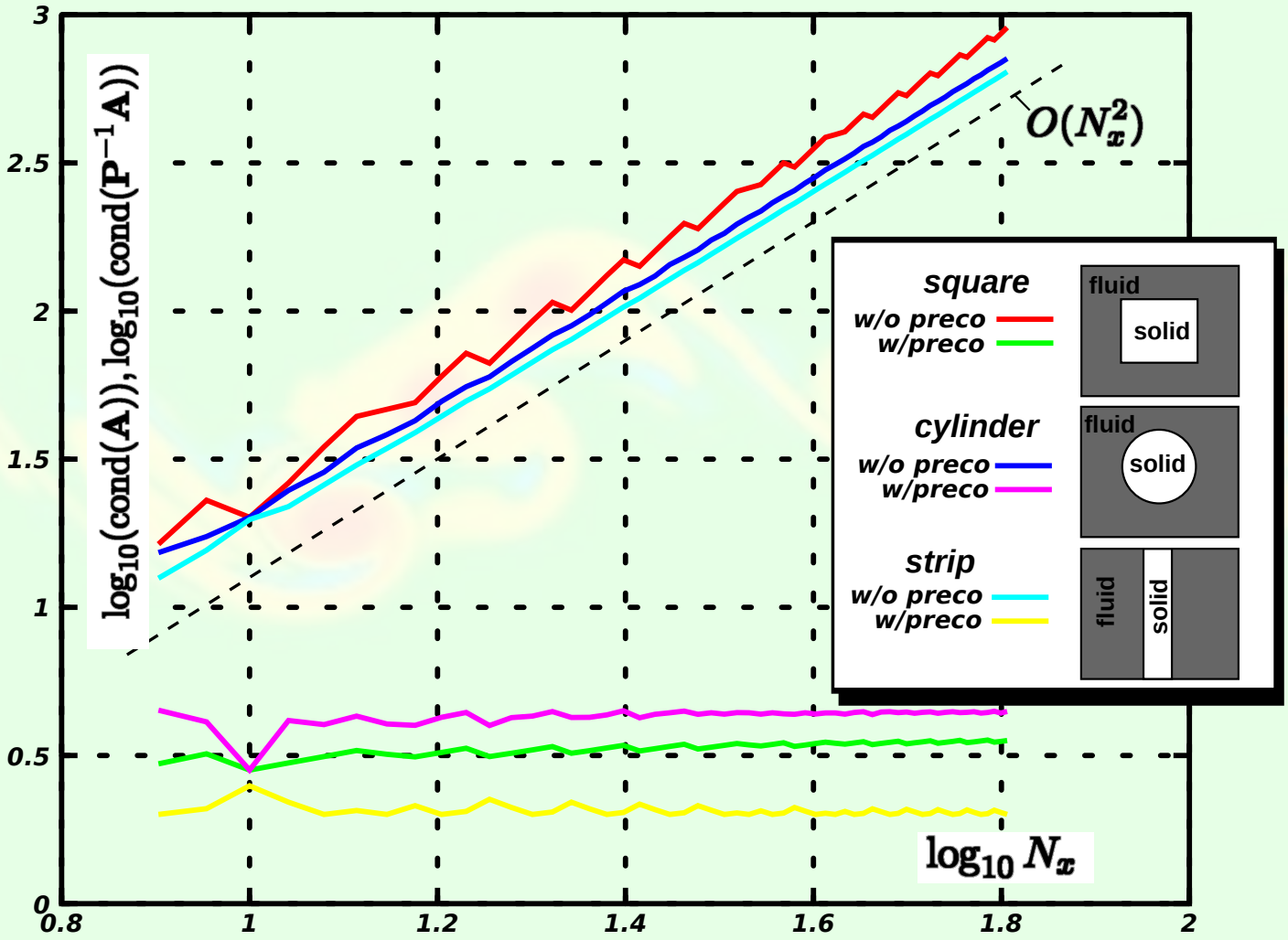
$$\begin{bmatrix} \mathbf{A}_{FF} & \mathbf{A}_{FB} & \mathbf{0} \\ \mathbf{A}_{BF} & \tilde{\mathbf{A}}_{BB} & \mathbf{A}_{BG} \\ \mathbf{0} & \mathbf{A}_{GB} & \mathbf{A}_{GG} \end{bmatrix} \begin{bmatrix} \mathbf{x}_{FB} \\ \mathbf{x}_G \end{bmatrix} = \begin{bmatrix} \mathbf{y}_{FB} \\ \mathbf{0}_G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



- (F) fluid node
- (B) boundary node
- (G) ghost node

((version texstuff-1.0.36-213-g1864495 Wed Apr 11 09:24:51 2012 +0200) (date Wed Apr 11 09:42:15 2012 +0200))

Accelerated Global Preconditioning (AGP) (cont.)



((version texstuff-1.0.36-213-g1864495 Wed Apr 11 09:24:51 2012 +0200) (date Wed Apr 11 09:42:15 2012 +0200))

Spectral decomposition of Stekhlov operators

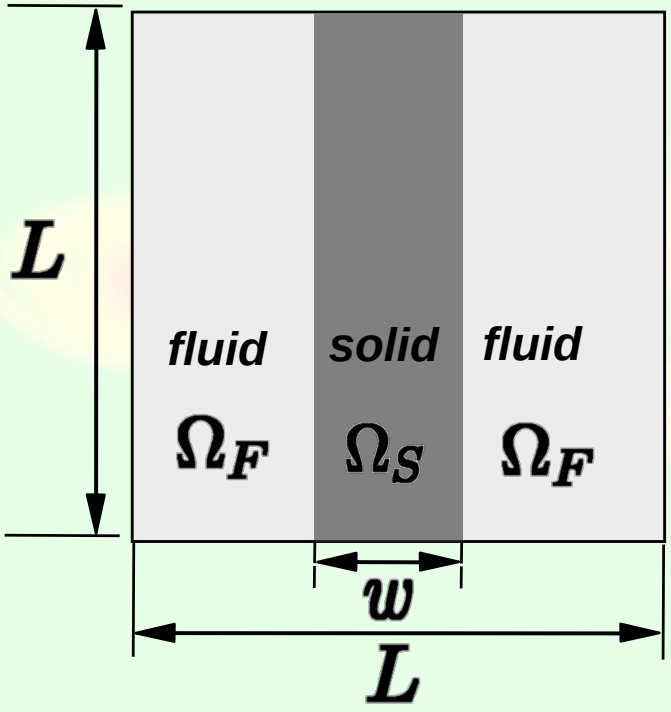
Stekhlov operator $\mathcal{S}_F$ for the fluid domain is defined by: $w = \mathcal{S}_F(v)$, if

$$\Delta\phi = 0, \text{ in } \Omega_F$$

$$\phi_\Gamma = v$$

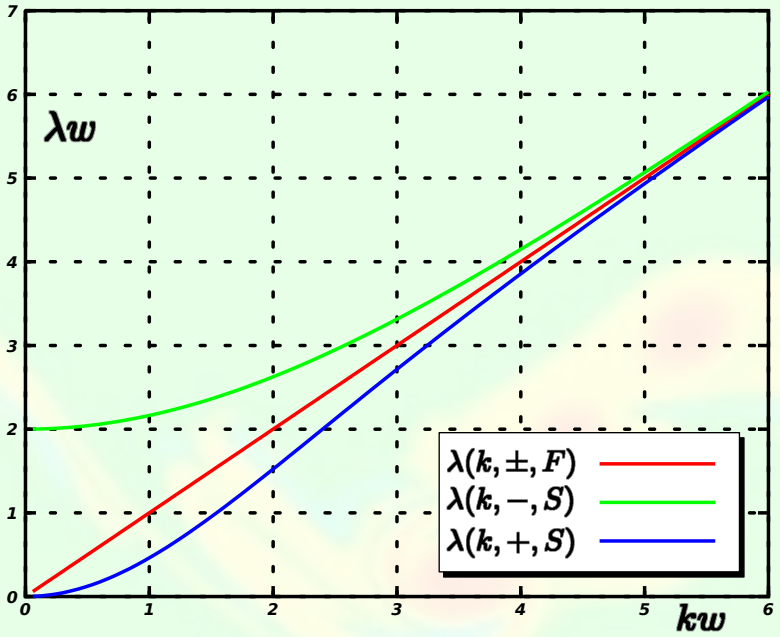
$$\text{then } w = (\partial\phi/\partial n)|_\Gamma$$

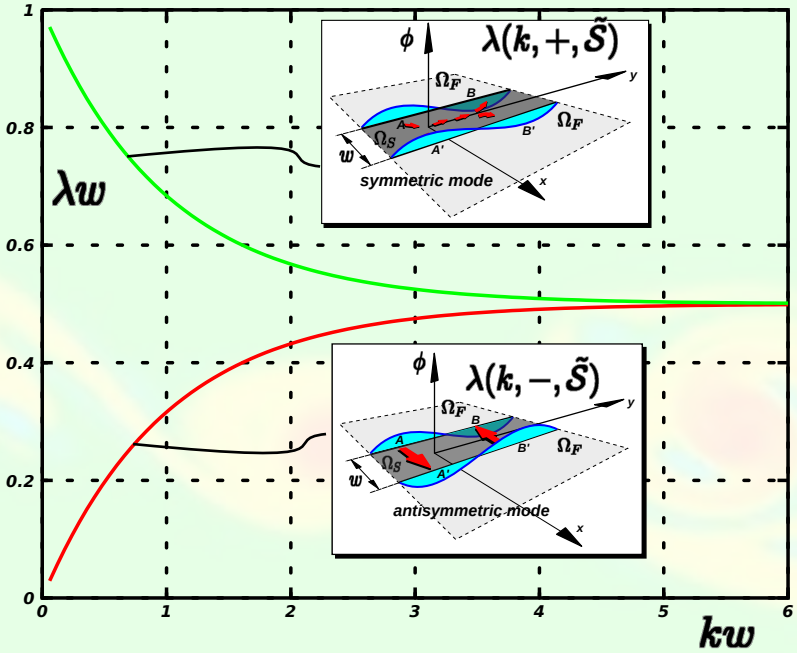
In the same way the Stekhlov operator $\mathcal{S}_S$ for the fluid domain can be defined. It turns out to be that the preconditioned matrix corresponds to $\mathbf{P}^{-1}\mathbf{A} \rightarrow (\mathcal{S}_F + \mathcal{S}_S)^{-1}\mathcal{S}_F$.



((version texstuff-1.0.36-213-g1864495 Wed Apr 11 09:24:51 2012 +0200) (date Wed Apr 11 09:42:15 2012 +0200))

Spectral decomposition of Stekhlov operators (cont.)





$$\lambda(k,\pm,F) = |k|,$$
$$\lambda(k,+,S) = k\tanh(kw/2),$$
$$\lambda(k,-,S) = k\coth(kw/2).$$

$$\lambda(k,\pm,\tilde{S}) = \frac{|k|}{|k| + k\left\{\begin{matrix}\tanh\\ \coth\end{matrix}\right\}(kw/2)},$$
$$\kappa(\tilde{S}) = 1/\lambda_{\min}(\tilde{S}) = L/(\pi w)$$

((version texstuff-1.0.36-213-g1864495 Wed Apr 11 09:24:51 2012 +0200) (date Wed Apr 11 09:42:15 2012 +0200))

FFT Solver

- We have to solve a linear system $\mathbf{A}\mathbf{x} = \mathbf{b}$
- The Discrete Fourier Transform (DFT) is an orthogonal transformation $\tilde{\mathbf{x}} = \mathbf{O}\mathbf{x} = \mathrm{fft}(\mathbf{x})$.
- The inverse transformation $\mathbf{O}^{-1} = \mathbf{O}^T$ is the inverse Fourier Transform $\mathbf{x} = \mathbf{O}^T\tilde{\mathbf{x}} = \mathrm{ifft}(\mathbf{x})$.
- If the operator matrix $\mathbf{A}$ is ***homogeneous*** (i.e. the stencil is the same at all grid points) and the b.c.'s are periodic, then it can be shown that $\mathbf{O}$ diagonalizes $\mathbf{A}$, i.e. $\mathbf{O}\mathbf{A}\mathbf{O}^{-1} = \mathbf{D}$.
- So in the transformed basis the system of equations is diagonal

$$\begin{aligned} (\mathbf{O}\mathbf{A}\mathbf{O}^{-1})\,(\mathbf{O}\mathbf{x}) &= (\mathbf{O}\mathbf{b}), \\ \mathbf{D}\tilde{\mathbf{x}} &= \tilde{\mathbf{b}}, \end{aligned} \tag{1}$$

- For $N = 2^p$ the Fast Fourier Transform (FFT) is an algorithm that computes the DFT (and its inverse) in $O(N\log(N))$ operations.

((version texstuff-1.0.36-213-g1864495 Wed Apr 11 09:24:51 2012 +0200) (date Wed Apr 11 09:42:15 2012 +0200))

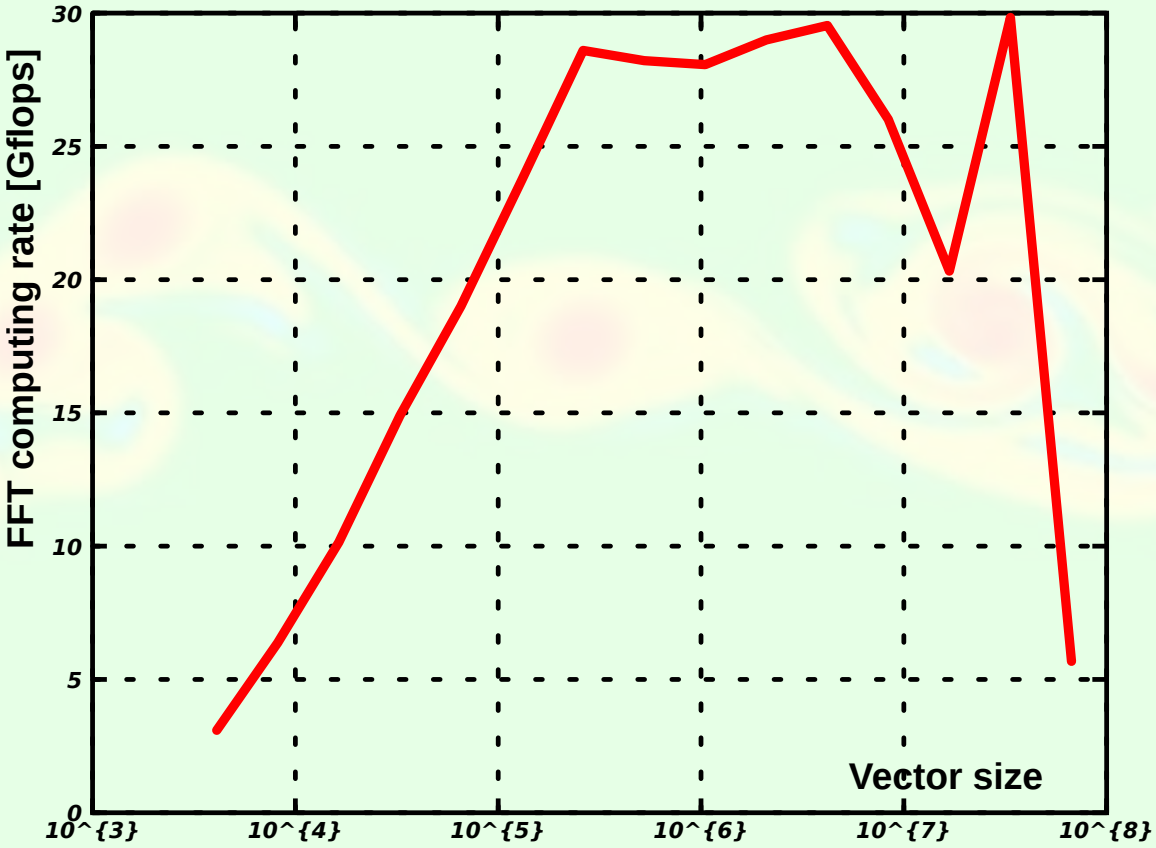
FFT Solver (cont.)

- **So the following algorithm computes the solution of the system in $O(N\log(N))$ ops.**
  - ▷ $\tilde{\mathbf{b}} = \mathrm{fft}(\mathbf{b})$, **(transform r.h.s)**
  - ▷ $\tilde{\mathbf{x}} = \mathbf{D}^{-1}\tilde{\mathbf{b}}$, **(solve diagonal system $O(N)$)**
  - ▷ $\mathbf{x} = \mathrm{ifft}(\tilde{\mathbf{x}})$, **(anti-transform to get the sol. vector)**
- **Total cost: 2 FFT's, plus one element-by-element vector multiply (the reciprocals of the values of the diagonal of $\mathbf{D}$ are precomputed)**
- **In order to precompute the diagonal values of $\mathbf{D}$,**
  - ▷ **We take any vector $\mathbf{z}$ and compute $\mathbf{y} = \mathbf{A}\mathbf{z}$,**
  - ▷ **then transform $\tilde{\mathbf{z}} = \mathrm{fft}(\mathbf{z})$, $\tilde{\mathbf{y}} = \mathrm{fft}(\mathbf{y})$,**
  - ▷ $D_{jj} = y_j/z_j$.

((version texstuff-1.0.36-213-g1864495 Wed Apr 11 09:24:51 2012 +0200) (date Wed Apr 11 09:42:15 2012 +0200))

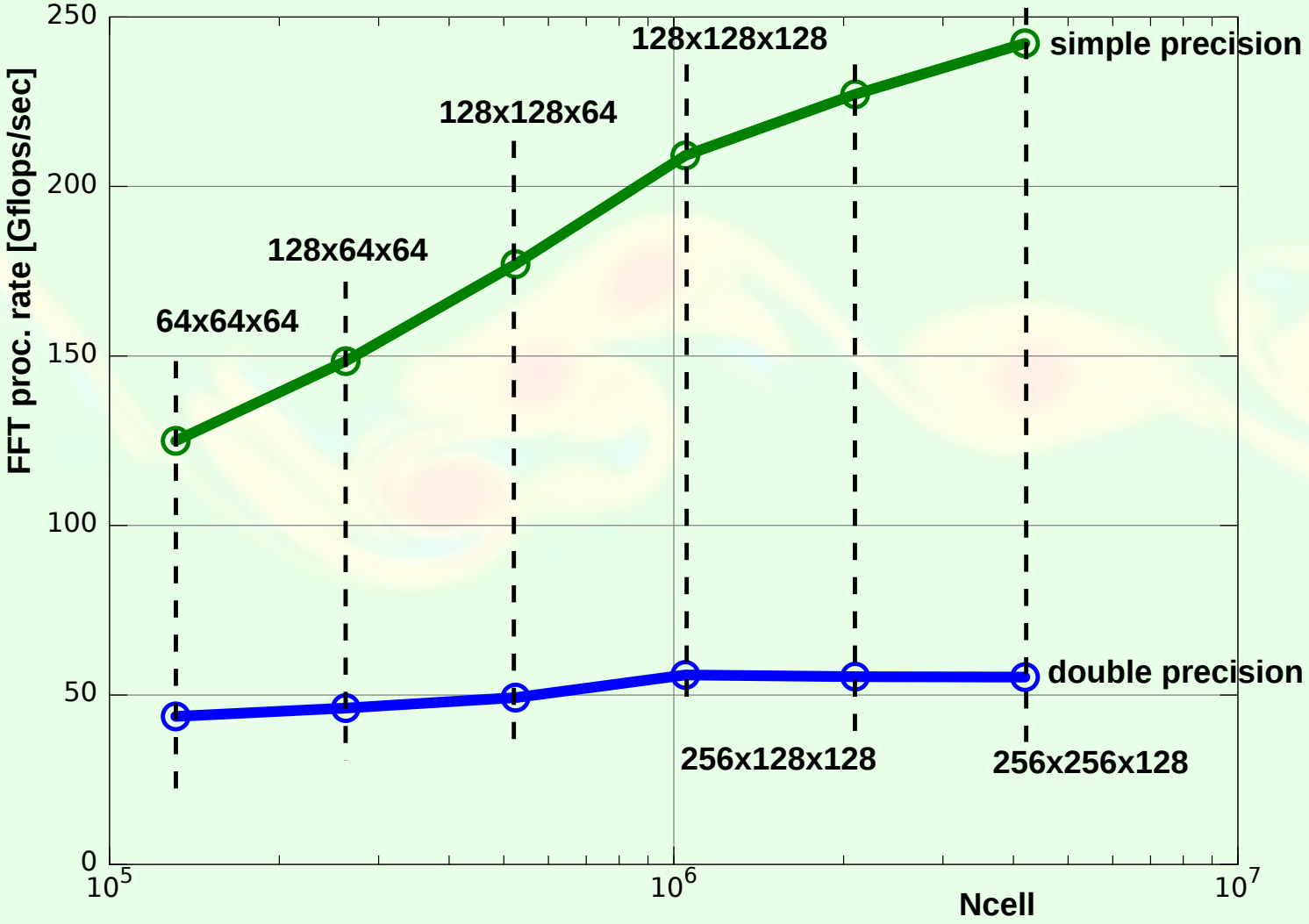
Implementation details on the GPU

- We use the *CUFFT library*.
- Per iteration: 2 FFT's and Poisson residual evaluation. The FFT on the *GPU Tesla C1060* performs at *27 Gflops*, where the operations are counted as $5N\log_2(N)$.



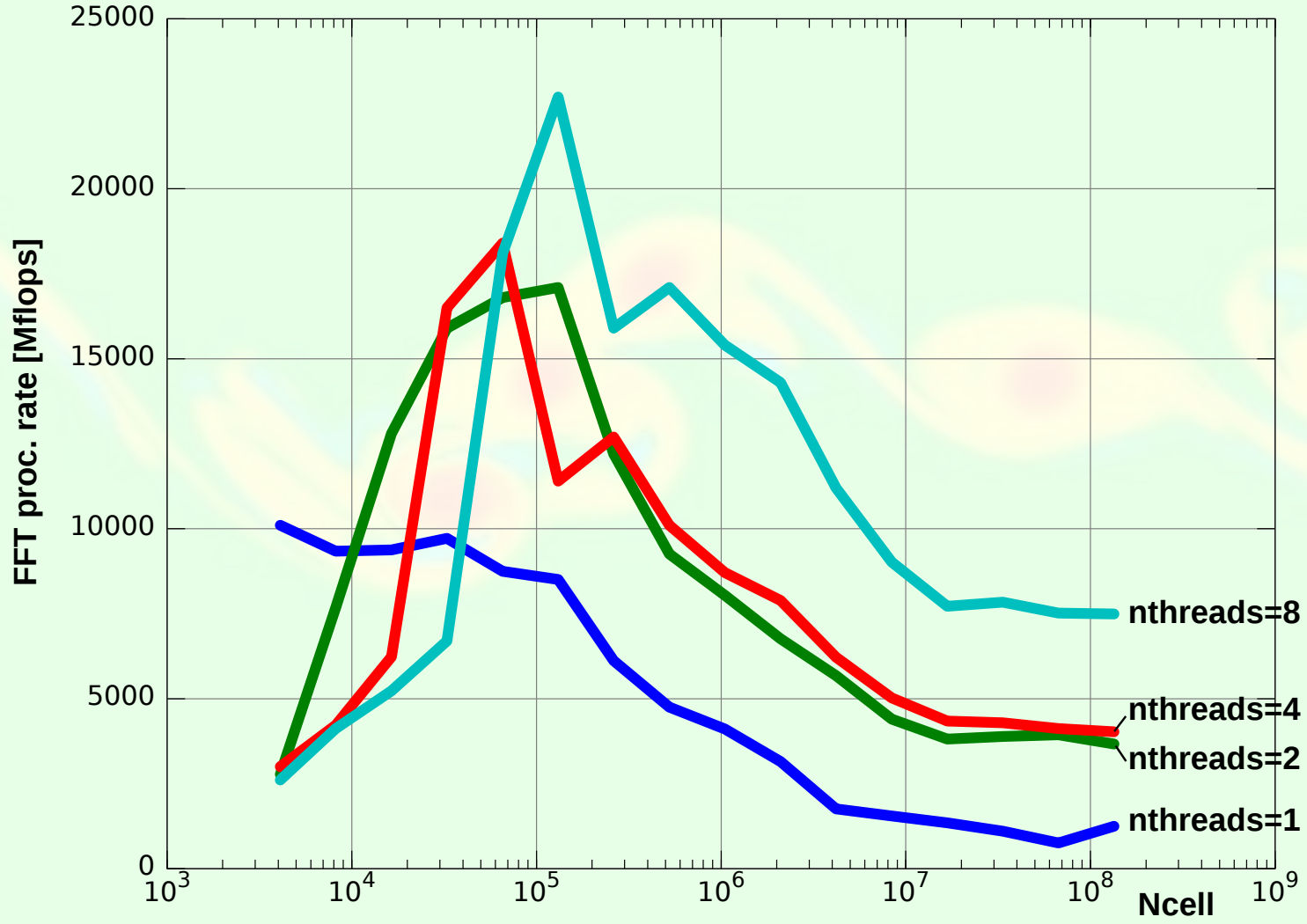
((version texstuff-1.0.36-213-g1864495 Wed Apr 11 09:24:51 2012 +0200) (date Wed Apr 11 09:42:15 2012 +0200))

FFT computing rates in GPGPU. GTX-580



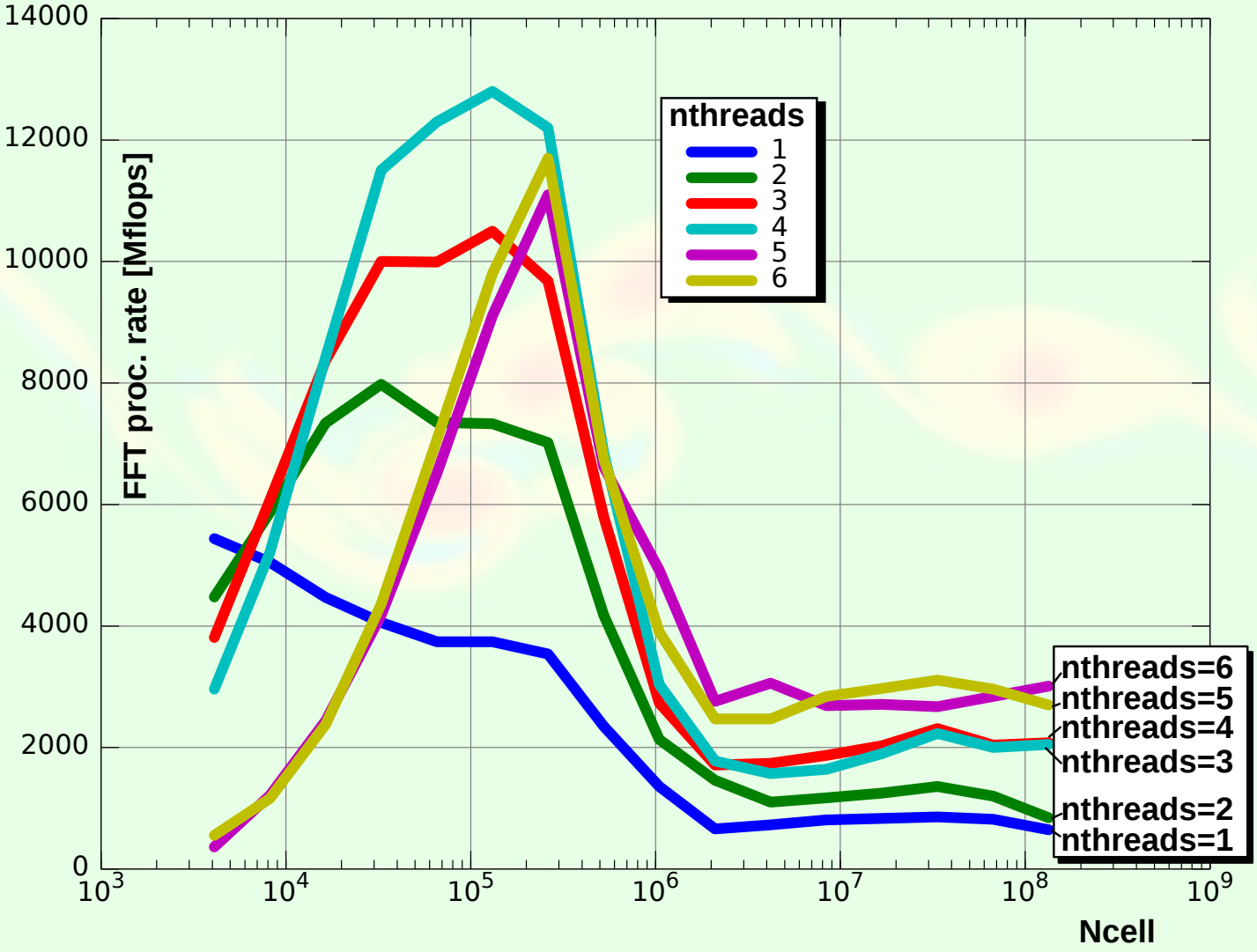
((version texstuff-1.0.36-213-g1864495 Wed Apr 11 09:24:51 2012 +0200) (date Wed Apr 11 09:42:15 2012 +0200))

FFTW (CUFFT) on i7-3820@3.60Ghz (Sandy Bridge)



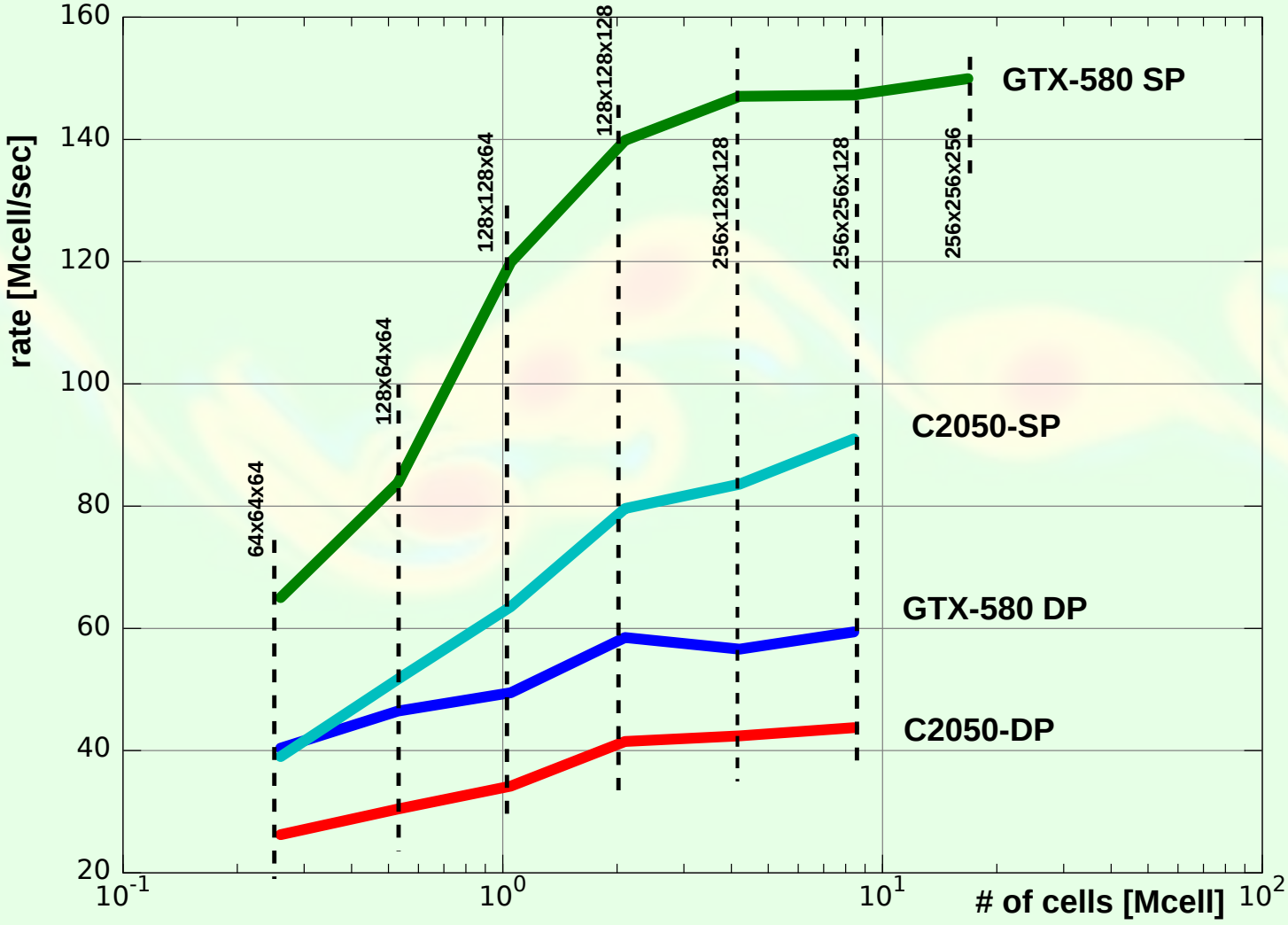
((version texstuff-1.0.36-213-g1864495 Wed Apr 11 09:24:51 2012 +0200) (date Wed Apr 11 09:42:15 2012 +0200))

FFTW (CUFFT) on W3690@3.47Ghz (Nehalem)



((version texstuff-1.0.36-213-g1864495 Wed Apr 11 09:24:51 2012 +0200) (date Wed Apr 11 09:42:15 2012 +0200))

NSFVM Computing rates in GPGPU. Scaling



((version texstuff-1.0.36-213-g1864495 Wed Apr 11 09:24:51 2012 +0200) (date Wed Apr 11 09:42:15 2012 +0200))

NSFVM Computing rates in CPU

- i7-3820@3.60Ghz (Sandy Bridge), 1 core (sequential): 1.7 Mcell/sec
- i7-950@3.07 (Nehalem), 1 core (sequential): 1.51 Mcell/sec
- Cellrates with nthreads$>$1, and W3690@3.47Ghz not available at this time.
- BUT, we expect at most 7 to 10 Mcell/secs, so there is speedup factor of 8 to 10, with respect to the GPGPU (GTX-580, DP).

((version texstuff-1.0.36-213-g1864495 Wed Apr 11 09:24:51 2012 +0200) (date Wed Apr 11 09:42:15 2012 +0200))

NSFVM Computing rates in CPU (cont.)

NSFVM and "Real Time" computing

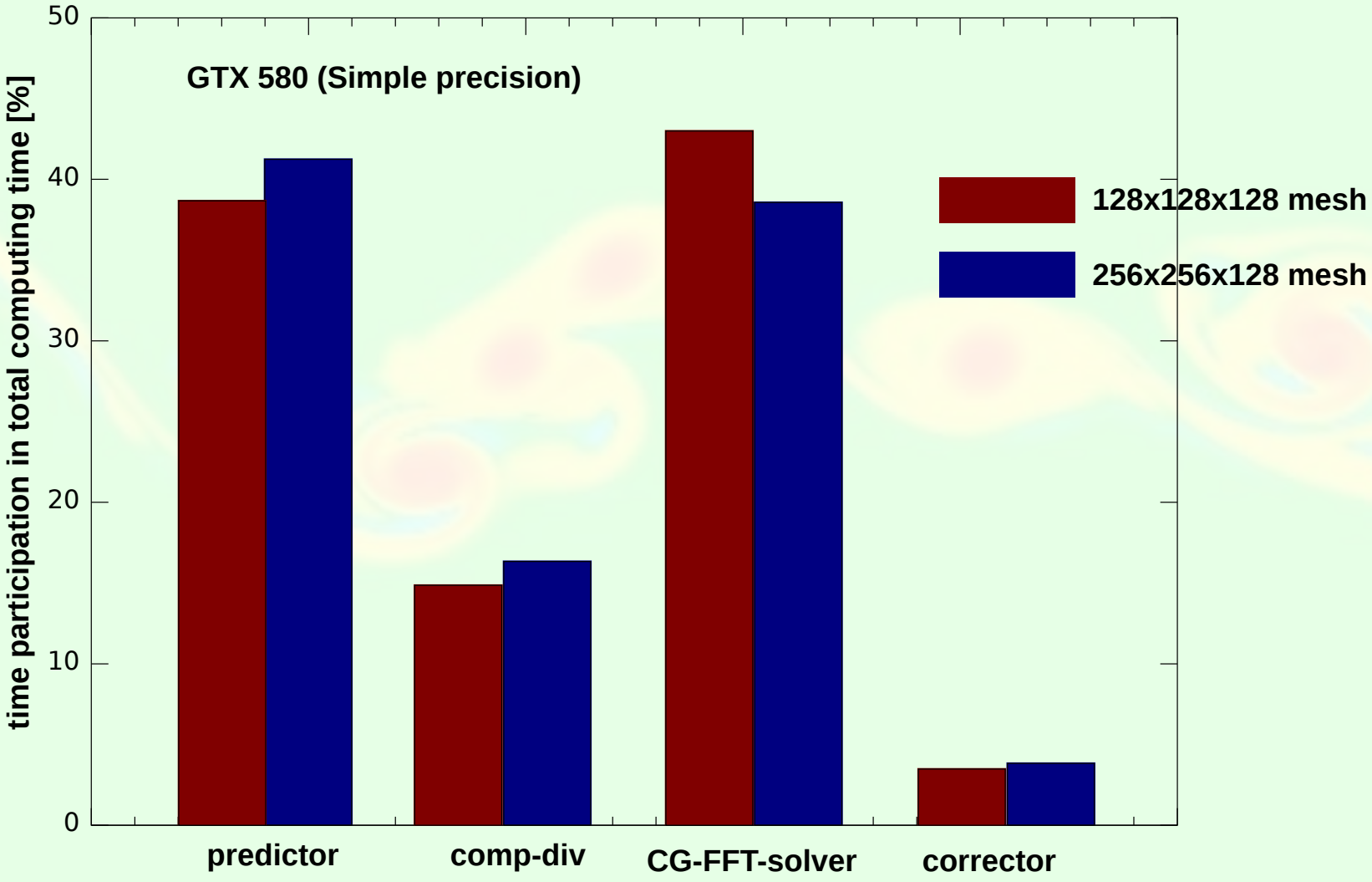
- **For a 128x128x128 mesh ($\approx$ 2Mcell), we have a computing time of 2 Mcell/(140 Mcell/sec) = 0.014 secs/time step.**
- **That means 70 steps/sec.**
- **A von Neumann stability analysis shows that the QUICK stabilization scheme is inconditionally stable if advanced in time with Forward Euler.**
- **With a second order Adams-Bashfort scheme the critical CFL is 0.588.**
- **For NS eqs. the critical CFL has been found to be somewhat lower ($\approx$ 0.5).**
- **If $L = 1, u = 1, h = 1/128, \Delta t = 0.5h/u = 0.004\ [\mathrm{sec}]$, so that we can compute in 1 sec, 0.28 secs of simulation time. We say ST/RT=0.28.**

(launch video nsfvm-bodies)

(launch video kh-instab-128)

((version texstuff-1.0.36-213-g1864495 Wed Apr 11 09:24:51 2012 +0200) (date Wed Apr 11 09:42:15 2012 +0200))

Computing times in GPGPU. Fractional Step components



((version texstuff-1.0.36-213-g1864495 Wed Apr 11 09:24:51 2012 +0200) (date Wed Apr 11 09:42:15 2012 +0200))

LBM and FVM

- This algorithm competes with the popular Lattice Boltzmann Method.
- Both are CA (Cellular Automata) algorithms
- Both are fast (measured in cellrates) on GPGPU's with structured meshes.
- LBM doesn't solve a Poisson equation, so it's partially compressible, and then there is a CFL penalization factor $\propto \mathrm{Mach}_{\mathrm{art}}$.
- Both can be nested refined near surfaces, or other interest zones.
- Higher order treatment of BC's on body surfaces may be better improved in FVM.

((version texstuff-1.0.36-213-g1864495 Wed Apr 11 09:24:51 2012 +0200) (date Wed Apr 11 09:42:15 2012 +0200))

Conclusions

The *Accelerated Global Preconditioning (AGP)* algorithm for the solution of the Poisson equation specially oriented to the solution of Navier-Stokes equations on GPU hardware was presented. It shares some features with the well known *IOP* iteration scheme. As a summary of the comparison between both methods, the following issues may be mentioned

- Both solvers are based on the fact that an efficient preconditioning that consists in solving the Poisson problem 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 with some fast solvers as Multigrid or *FFT*.
- Both solvers have their convergence governed by the spectrum of the $\mathcal{S}^{-1}\mathcal{S}_F$, however

((version texstuff-1.0.36-213-g1864495 Wed Apr 11 09:24:51 2012 +0200) (date Wed Apr 11 09:42:15 2012 +0200))

- ***IOP*** **is a** ***stationary method*** **and its limit rate of convergence is given by**

$$\|\mathbf{r}^{n+1}\| \leq \gamma_{\mathrm{IOP}} \|\mathbf{r}^{n}\|$$
$$\gamma_{\mathrm{IOP}} = 1 - \lambda_{\min}, \tag{2}$$
$$\lambda_{\min} = \min(\mathrm{eig}(\mathcal{S}^{-1}\mathcal{S}_F)).$$

- ***AGP*** **is a preconditioned** ***Krylov space method*** **and its convergence is governed by the condition number of $\mathcal{S}^{-1}\mathcal{S}_F$, i.e.**

$$\kappa(\mathbf{A}^{-1}\mathbf{A}_F) = \frac{1}{\min(\mathrm{eig}(\mathcal{S}^{-1}\mathcal{S}_F))} = \frac{1}{\lambda_{\min}}, \tag{3}$$

- **It has been shown that $\lambda_{\min} = O(1)$, i.e. it** ***does not degrade with refinement*****, so that** ***IOP*** **has a linear convergence with limit rate $O(1)$.**
- **By the same reason, the condition number for** ***AGP does not degrade with refinement*****.**
- ***IOP*** **iterates over both the velocity and pressure fields, whereas** ***AGP*** **iterates only on the pressure vector (which is better for implementation on GPU's).**

((version texstuff-1.0.36-213-g1864495 Wed Apr 11 09:24:51 2012 +0200) (date Wed Apr 11 09:42:15 2012 +0200))

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, Dir: Dr. Sergio Idelsohn).**

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.