Numerical modelling and High Performance Computing for sediment flows: Part one

Jean-Baptiste Keck

Thursday 22nd November, 2018

Table of contents

Coastal sedimentary processes

Physics of particle-laden fresh water flows above salted water:

River delta of Irrawady, Myanmar (ESA)

How to build a high performance fluid solver ?

Navier-Stokes equations

Variables of interest

- We consider an incompressible fluid with the following properties:
  1. A constant viscosity μ (resistance to deformation by shear stress).
  2. A constant density ρ (mass per unit volume).

- In 2D, we are interested in solving the following physical fields:
  1. The fluid velocity $\boldsymbol{u}(\boldsymbol{x}, t) = \begin{bmatrix} u_x(\boldsymbol{x}, t) \\ u_y(\boldsymbol{x}, t) \\ 0 \end{bmatrix}$
  2. The fluid vorticity $\boldsymbol{\omega}(\boldsymbol{x}, t) = \begin{bmatrix} 0 \\ 0 \\ \omega_z(\boldsymbol{x}, t) \end{bmatrix}$
  3. The internal pressure of the fluid $P(\boldsymbol{x}, t)$.

We define those variables on a spatial domain Ω with boundaries $\partial\Omega$.
i.e. $\forall\ \boldsymbol{x} = (x, y) \in \Omega$ which is a rectangular domain $[0, L_x] \times [0, L_y]$.

Variables of interest

- We consider an incompressible fluid with the following properties:
  1. A constant viscosity μ (resistance to deformation by shear stress).
  2. A constant density ρ (mass per unit volume).

- In 3D, we are interested in solving the following physical fields:
  1. The fluid velocity $\boldsymbol{u}(\boldsymbol{x}, t) = \begin{bmatrix} u_x(\boldsymbol{x}, t) \\ u_y(\boldsymbol{x}, t) \\ u_z(\boldsymbol{x}, t) \end{bmatrix}$
  2. The fluid vorticity $\boldsymbol{\omega}(\boldsymbol{x}, t) = \begin{bmatrix} \omega_x(\boldsymbol{x}, t) \\ \omega_y(\boldsymbol{x}, t) \\ \omega_z(\boldsymbol{x}, t) \end{bmatrix}$
  3. The internal pressure of the fluid $P(\boldsymbol{x}, t)$.

We define those variables on a spatial domain Ω with boundaries $\partial\Omega$.
i.e. $\forall\ \boldsymbol{x} = (x, y, z) \in \Omega$ which is a cuboid domain $[0, L_x] \times [0, L_y] \times [0, L_z]$.

Variables of interest - velocity and vorticity

- The vorticity is a pseudovector field that describes the local spinning motion of the fluid near some point in the whole domain.
- You can directly compute the vorticity $\boldsymbol{\omega}$ by taking the curl of the velocity $\boldsymbol{u}$. Given $\nabla = \left(\frac{\partial}{\partial x}, \frac{\partial}{\partial y}, \frac{\partial}{\partial z}\right)$ we have $\boldsymbol{\omega} = \nabla \times \boldsymbol{u}$.
- In 3D, this is 3-components vector field (like the velocity):

$$\begin{bmatrix}\omega_x \\ \omega_y \\ \omega_z\end{bmatrix} = \begin{bmatrix}\frac{\partial}{\partial x} \\ \frac{\partial}{\partial y} \\ \frac{\partial}{\partial z}\end{bmatrix} \times \begin{bmatrix}u_x \\ u_y \\ u_z\end{bmatrix} = \begin{bmatrix}\frac{\partial u_z}{\partial y} - \frac{\partial u_y}{\partial z} \\ \frac{\partial u_x}{\partial z} - \frac{\partial u_z}{\partial x} \\ \frac{\partial u_y}{\partial x} - \frac{\partial u_x}{\partial y}\end{bmatrix}$$

Variables of interest - velocity and vorticity

- The vorticity is a pseudovector field that describes the local spinning motion of the fluid near some point in the whole domain.
- You can directly compute the vorticity $\boldsymbol{\omega}$ by taking the curl of the velocity $\boldsymbol{u}$. Given $\nabla = \left(\frac{\partial}{\partial x}, \frac{\partial}{\partial y}, \frac{\partial}{\partial z}\right)$ we have $\boldsymbol{\omega} = \nabla \times \boldsymbol{u}$.
- In 2D, this is a scalar field (only one component):

$$\begin{bmatrix} \omega_x \\ \omega_y \\ \omega_z \end{bmatrix} = \begin{bmatrix} \frac{\partial}{\partial x} \\ \frac{\partial}{\partial y} \\ \cancel{\frac{\partial}{\partial z}} \end{bmatrix} \times \begin{bmatrix} u_x \\ u_y \\ \cancel{u_z} \end{bmatrix} = \begin{bmatrix} \cancel{\frac{\partial u_z}{\partial y}} - \cancel{\frac{\partial u_y}{\partial z}} \\ \cancel{\frac{\partial u_x}{\partial z}} - \cancel{\frac{\partial u_z}{\partial x}} \\ \frac{\partial u_y}{\partial x} - \frac{\partial u_x}{\partial y} \end{bmatrix}$$

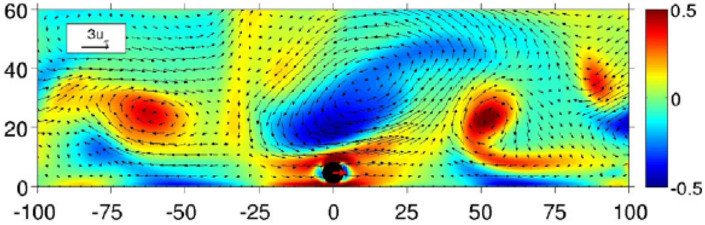
Variables of interest - velocity and vorticity

- The vorticity is a pseudovector field that describes the local spinning motion of the fluid near some point in the whole domain.
- You can directly compute the vorticity $\boldsymbol{\omega}$ by taking the curl of the velocity $\boldsymbol{u}$. Given $\nabla = \left(\frac{\partial}{\partial x}, \frac{\partial}{\partial y}, \frac{\partial}{\partial z}\right)$ we have $\boldsymbol{\omega} = \nabla \times \boldsymbol{u}$.
- In 2D, this is a scalar field (only one component):

$$\begin{bmatrix}\omega_x \\ \omega_y \\ \omega_z\end{bmatrix} = \begin{bmatrix}\frac{\partial}{\partial x} \\ \frac{\partial}{\partial y} \\ 0\end{bmatrix} \times \begin{bmatrix}u_x \\ u_y \\ 0\end{bmatrix} = \begin{bmatrix}0 \\ 0 \\ \frac{\partial u_y}{\partial x} - \frac{\partial u_x}{\partial y}\end{bmatrix}$$

Variables of interest - velocity and vorticity

- The vorticity is a pseudovector field that describes the local spinning motion of the fluid near some point in the whole domain.
- You can directly compute the vorticity $\boldsymbol{\omega}$ by taking the curl of the velocity $\boldsymbol{u}$. Given $\nabla = \left(\frac{\partial}{\partial x}, \frac{\partial}{\partial y}, \frac{\partial}{\partial z}\right)$ we have $\boldsymbol{\omega} = \nabla \times \boldsymbol{u}$.
- In 2D, this is a scalar field (only one component):



Example of 2D velocity and vorticity fields

Navier-Stokes - conservation of mass

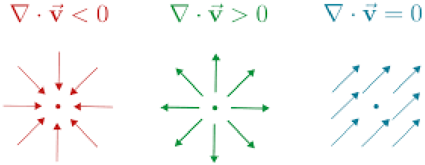
- The first equation relates to the conservation of the mass:

$$\frac{\partial \rho}{\partial t} + \nabla \cdot (\rho \boldsymbol{u}) = 0$$

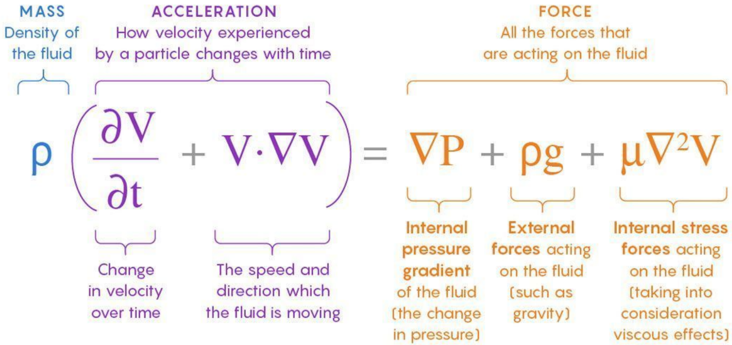
- With a constant density ρ this equations reduce to:

$$\nabla \cdot \boldsymbol{u} = \frac{\partial u_x}{\partial x} + \frac{\partial u_y}{\partial y} + \frac{\partial u_z}{\partial z} = 0$$

- You can see the divergence as a scalar field representing the quantity of a vector field's source at each point:



Navier-Stokes - conservation of momentum



Navier-Stokes Equations

Describe the flow of incompressible fluids.

Navier-Stokes - velocity-vorticity formulation

- What if we do not want to solve the pressure field ?

$$\boxed{\begin{aligned} \nabla \cdot \boldsymbol{u} &= 0 \\ \rho \left[\frac{\partial \boldsymbol{u}}{\partial t} + (\boldsymbol{u} \cdot \nabla)\, \boldsymbol{u} \right] &= \mu \Delta \boldsymbol{u} - \boldsymbol{\nabla} P + \rho \boldsymbol{g} \end{aligned}}$$

- Just apply the curl operator to the second equation:

$$\boxed{\begin{aligned} \nabla \cdot \boldsymbol{u} &= 0 \\ \nabla \times \left(\rho \left[\frac{\partial \boldsymbol{u}}{\partial t} + (\boldsymbol{u} \cdot \nabla)\, \boldsymbol{u} \right] \right) &= \nabla \times \left(\mu \Delta \boldsymbol{u} - \boldsymbol{\nabla} P + \rho \boldsymbol{g} \right) \end{aligned}}$$

- In 3D $(\boldsymbol{u}, \boldsymbol{\omega})$ formulation has 6 unknowns vs. 4 for $(\boldsymbol{u}, P)$ formulation.

Navier-Stokes - velocity-vorticity formulation

- What if we do not want to solve the pressure field ?

$$\boxed{\begin{gathered}\nabla \cdot \boldsymbol{u} = 0 \\ \rho \left[\frac{\partial \boldsymbol{u}}{\partial t} + (\boldsymbol{u} \cdot \nabla)\, \boldsymbol{u} \right] = \mu \Delta \boldsymbol{u} - \boldsymbol{\nabla} P + \rho \boldsymbol{g}\end{gathered}}$$

- Just apply the curl operator to the second equation:

$$\boxed{\begin{gathered}\nabla \cdot \boldsymbol{u} = 0 \\ \rho \left[\frac{\partial \boldsymbol{\omega}}{\partial t} + (\boldsymbol{u} \cdot \nabla) \boldsymbol{\omega} - (\boldsymbol{\omega} \cdot \nabla) \boldsymbol{u} \right] = \mu \Delta \boldsymbol{\omega} - 0 + \rho (\nabla \times \boldsymbol{g})\end{gathered}}$$

- In 3D $(\boldsymbol{u}, \boldsymbol{\omega})$ formulation has 6 unknowns vs. 4 for $(\boldsymbol{u}, P)$ formulation.

Navier-Stokes - velocity-vorticity formulation

- What if we do not want to solve the pressure field ?

$$\boxed{\begin{gathered}\nabla \cdot \boldsymbol{u} = 0 \\ \rho \left[\frac{\partial \boldsymbol{u}}{\partial t} + (\boldsymbol{u} \cdot \nabla)\, \boldsymbol{u}\right] = \mu \Delta \boldsymbol{u} - \boldsymbol{\nabla} P + \rho \boldsymbol{g}\end{gathered}}$$

- Just apply the curl operator to the second equation:

$$\boxed{\begin{gathered}\nabla \cdot \boldsymbol{u} = 0 \\ \frac{\partial \boldsymbol{\omega}}{\partial t} + (\boldsymbol{u} \cdot \nabla)\boldsymbol{\omega} - (\boldsymbol{\omega} \cdot \nabla)\boldsymbol{u} = \frac{\mu}{\rho}\Delta \boldsymbol{\omega} + \nabla \times \boldsymbol{g}\end{gathered}}$$

- In 3D $(\boldsymbol{u}, \boldsymbol{\omega})$ formulation has 6 unknowns vs. 4 for $(\boldsymbol{u}, P)$ formulation.

Navier-Stokes - velocity-vorticity formulation

- What if we do not want to solve the pressure field ?

$$\nabla \cdot \boldsymbol{u} = 0$$
$$\rho \left[\frac{\partial \boldsymbol{u}}{\partial t} + (\boldsymbol{u} \cdot \nabla)\, \boldsymbol{u}\right] = \mu \Delta \boldsymbol{u} - \boldsymbol{\nabla} P + \rho \boldsymbol{g}$$

- Just apply the curl operator to the second equation:

$$\nabla \cdot \boldsymbol{u} = 0$$
$$\frac{\partial \boldsymbol{\omega}}{\partial t} + (\boldsymbol{u} \cdot \nabla)\boldsymbol{\omega} - (\boldsymbol{\omega} \cdot \nabla)\boldsymbol{u} = \nu \Delta \boldsymbol{\omega} + \nabla \times \boldsymbol{g}$$

- In 3D $(\boldsymbol{u}, \boldsymbol{\omega})$ formulation has 6 unknowns vs. 4 for $(\boldsymbol{u}, P)$ formulation.

Navier-Stokes - velocity-vorticity formulation

- What if we do not want to solve the pressure field ?

$$\boxed{\begin{gathered}\nabla \cdot \boldsymbol{u} = 0 \\ \rho \left[\frac{\partial \boldsymbol{u}}{\partial t} + (\boldsymbol{u} \cdot \nabla)\, \boldsymbol{u}\right] = \mu \Delta \boldsymbol{u} - \boldsymbol{\nabla} P + \rho \boldsymbol{g}\end{gathered}}$$

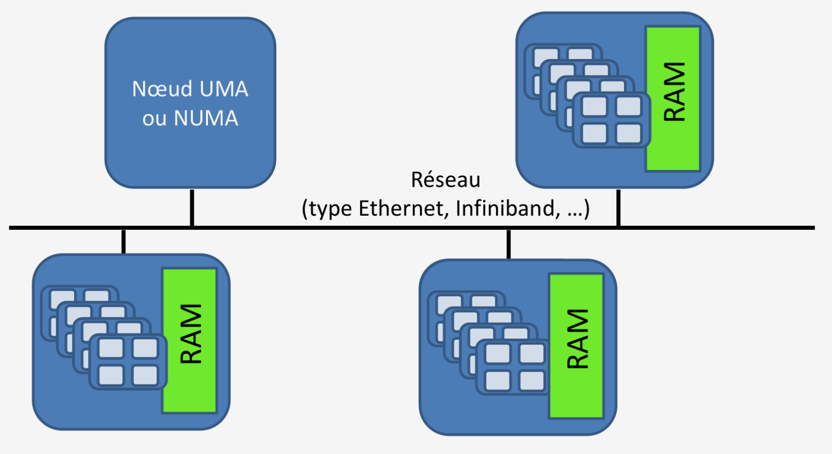
- Just apply the curl operator to the second equation:

$$\boxed{\begin{gathered}\nabla \cdot \boldsymbol{u} = 0 \\ \frac{\partial \boldsymbol{\omega}}{\partial t} + (\boldsymbol{u} \cdot \nabla)\boldsymbol{\omega} = (\boldsymbol{\omega} \cdot \nabla)\boldsymbol{u} + \nu \Delta \boldsymbol{\omega} + \nabla \times \boldsymbol{g}\end{gathered}}$$

- In 3D $(\boldsymbol{u}, \boldsymbol{\omega})$ formulation has 6 unknowns vs. 4 for $(\boldsymbol{u}, P)$ formulation.

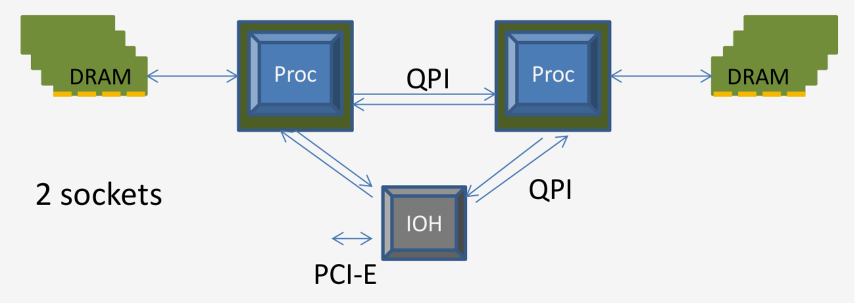
Target hardware

Global view of a compute cluster



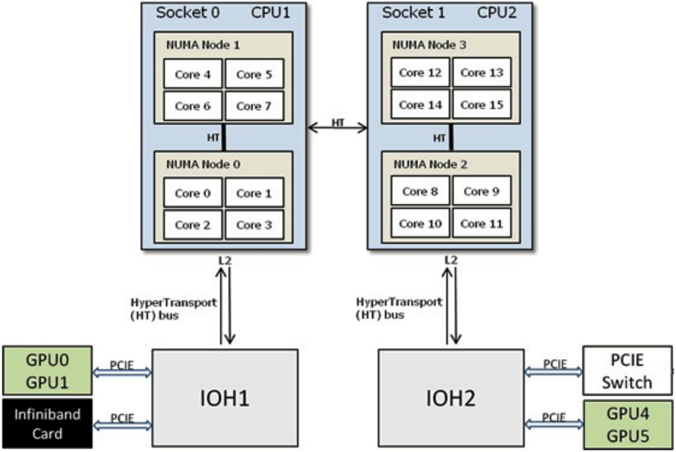
A single compute node

- Each compute node has its own processor(s) and memory banks:

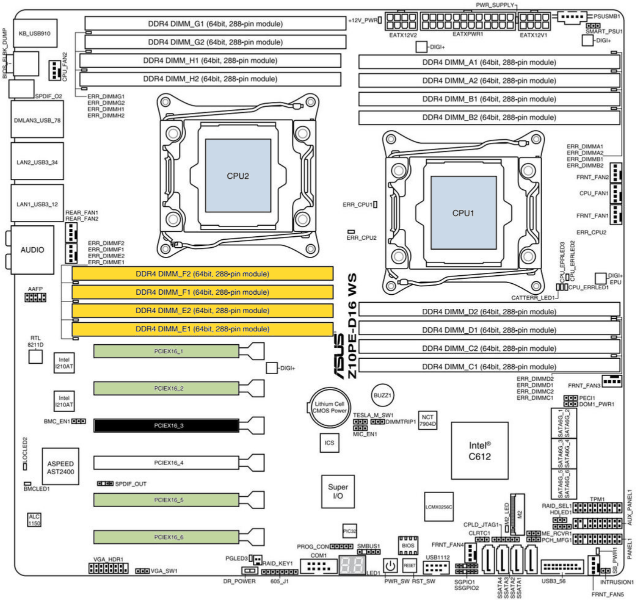


- On conventional architectures you can have 1 to 8 sockets (CPUs).

Zoom on a dual socket compute node



Dual socket motherboard



Accelerators and coprocessors - server grade GPUs

Nvidia Tesla P100

AMD Firepro W9100

This range of products is dedicated to the double precision compute tasks (no graphical tasks). Here FP64 = 1/2 FP32.

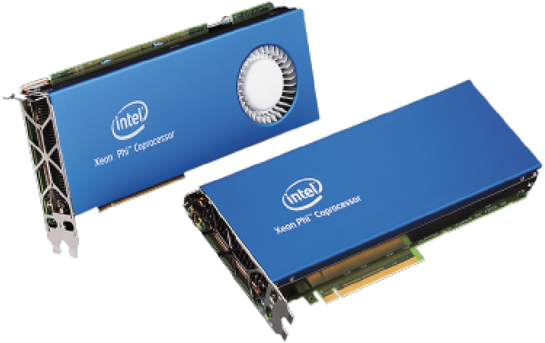
Accelerators and coprocessors - gaming GPUs

Nvidia Titan Xp

AMD RX Vega 64

This range of products is dedicated to the single precision compute tasks (mostly graphical tasks). Here FP64 $= \alpha$ FP32 where $\alpha \in [1/32, 1/4]$.

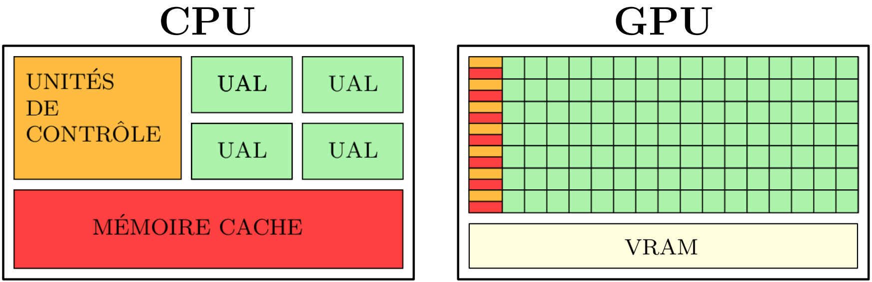
Accelerators and coprocessors - Intel's response to GPUs



MIC (Many Integrated Core) coprocessor - Xeon-Phi (72 cores)

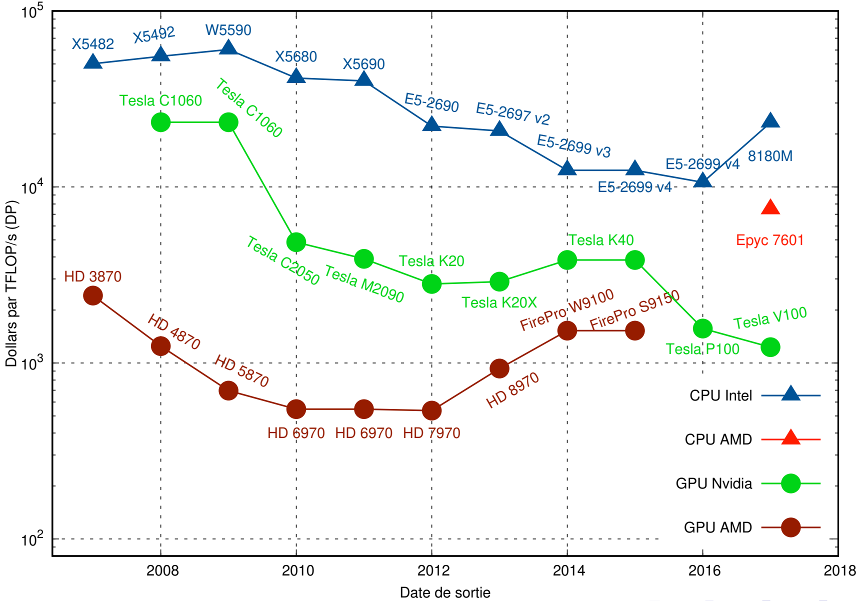
Those cards have been discontinued by Intel since 2017.
Here FP64 $= 1/2$ FP32.

CPU vs GPU



- More compute cores (but operating in groups) Less cache per core
- Embedded memory

Advantage of coprocessors



Programming models for coprocessors

- OpenCL (Open Computing Language): Framework for writing programs that execute across heterogeneous hardware (CPU, GPU, MIC, FPGA, DSP). This is not synonym of **performance portability**
  1. OpenCL 1.x: Based on C99 (2008).
  2. OpenCL 2.x: Based on a subset of C++14 (2013).
- CUDA (Compute Unified Device Architecture): Nvidia only (C++14)
- OpenMP (Open Multi-Processing): Since v.4.0 (2013) you can offload compute task to accelerators (pragma based approach).
- OpenACC (Open Accelerators): Like OpenMP but for accelerators.

	Intel		**AMD**		**Nvidia**
	CPU	MIC	CPU	GPU	**GPU**
OpenCL	2.1	1.2	2.0	1.2	1.2
CUDA	-	-	-	-	10.0
OpenMP	gcc/icc	gcc/icc	gcc	gcc	gcc
OpenACC	-	-	-	-	gcc/pgcc

The HySoP library

Discretization of the variables

- Variable are discretized on regular cartesian grids (with ghosts):

$$\Omega = [0, L_x] \times [0, L_y] \times [0, L_z]$$

$$N = N_x \times N_y \times N_z$$

$$\boldsymbol{dx} = [dx, dy, dz] = \left[\frac{L_x}{(N_x - 1)}, \frac{L_y}{(N_y - 1)}, \frac{L_z}{(N_z - 1)}\right]$$

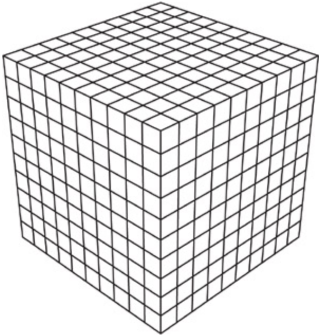
$$F_{ijk}(t) = F(idx, jdy, kdz, t)$$

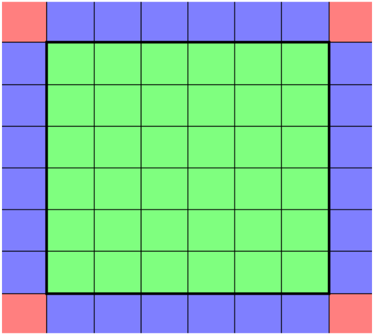
$$\forall (i, j, k) \in [\![0, N_x - 1]\!] \times [\![0, N_y - 1]\!] \times [\![0, N_z - 1]\!]$$

- This allows us to use efficient methods like:
  1. Finite differences
  2. Spectral methods

Discretization of the variables

- Variable are discretized on regular cartesian grids (with ghosts):

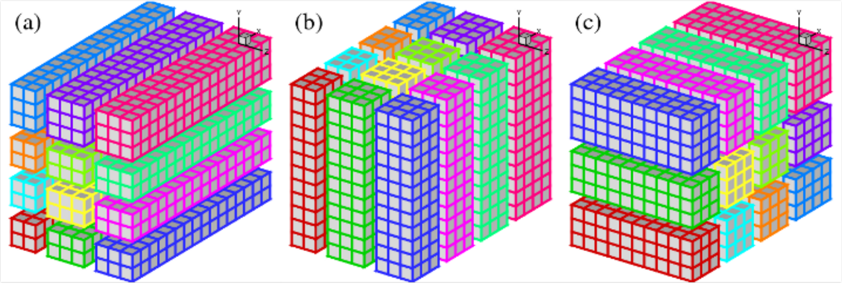




- This allows us to use efficient methods like:
  1. Finite differences
  2. Spectral methods

Discretization of the variables

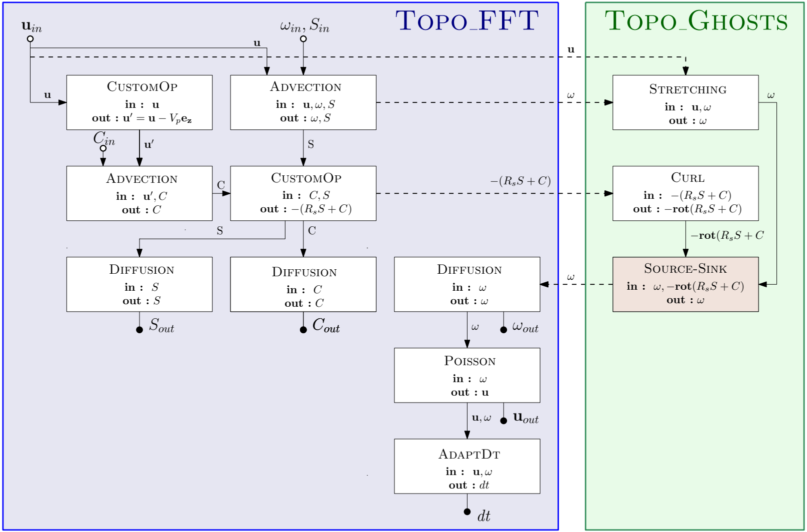
- All the variable are distributed on the compute nodes:



- One variable may have many different topologies depending on operator constraints.
- We use `MPI` (Message Passing Interface) for inter node communication.

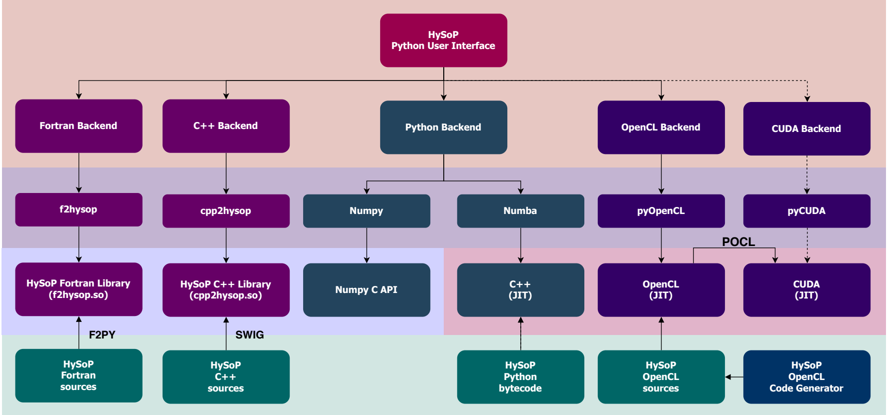
Building a problem

After specifying variables, the user builds a DAG of operators:



HySoP backends

- The user has to choose operators and the backend it will run on:



- Currently we have `Python`, `C++`, `Fortran` and `OpenCL` support.

Operator splitting

Operator splitting

- The idea is to split the Navier-Stokes equations in $(\boldsymbol{u}, \boldsymbol{\omega})$ formulation.
- Independant operators are easier to implement, test and debug !

$$\boxed{\frac{\partial \boldsymbol{\omega}}{\partial t} + (\boldsymbol{u} \cdot \nabla)\boldsymbol{\omega} = (\boldsymbol{\omega} \cdot \nabla)\boldsymbol{u} + \nu \Delta \boldsymbol{\omega} + \nabla \times \boldsymbol{g}}$$

- We split the momentum equations as the following:
  1. Transport: $\dfrac{\partial \boldsymbol{\omega}}{\partial t} + (\boldsymbol{u} \cdot \nabla)\boldsymbol{\omega} = 0$
  2. Stretching: $\dfrac{\partial \boldsymbol{\omega}}{\partial t} = (\boldsymbol{\omega} \cdot \nabla)\boldsymbol{u}$ `[this term only appears in 3D]`
  3. Diffusion: $\dfrac{\partial \boldsymbol{\omega}}{\partial t} = \nu \Delta \boldsymbol{\omega}$
  4. Ext. forces: $\dfrac{\partial \boldsymbol{\omega}}{\partial t} = \nabla \times \boldsymbol{g}$

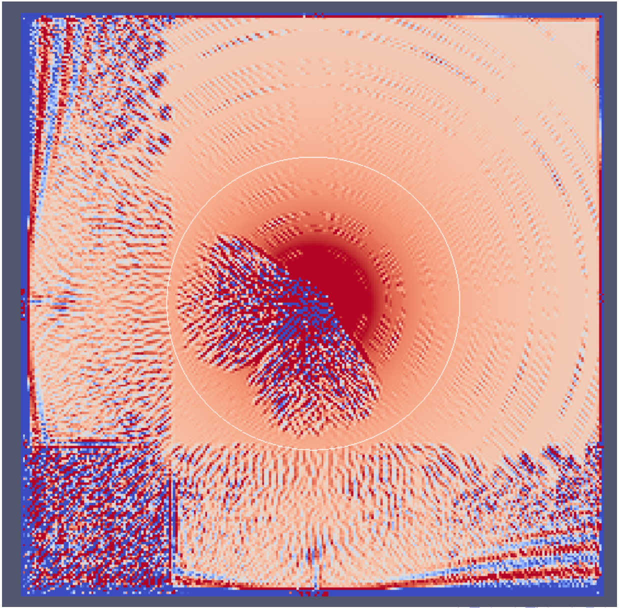
Operator splitting

- The idea is to split the Navier-Stokes equations in $(\boldsymbol{u}, \boldsymbol{\omega})$ formulation.
- Independant operators are easier to implement, test and debug !

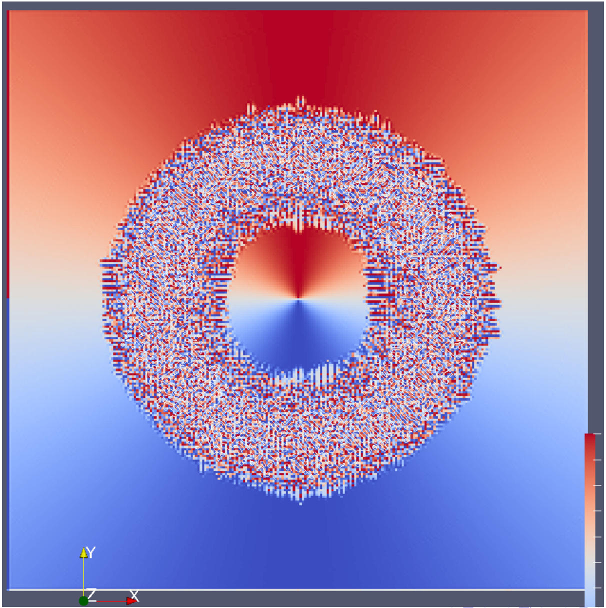
$$\boxed{\frac{\partial \boldsymbol{\omega}}{\partial t} + (\boldsymbol{u} \cdot \nabla)\boldsymbol{\omega} = (\boldsymbol{\omega} \cdot \nabla)\boldsymbol{u} + \nu \Delta \boldsymbol{\omega} + \nabla \times \boldsymbol{g}}$$

- Each discretized operator forces the timestep to be small enough:
  1. Transport: $\Delta t_{adv} < \dfrac{\mathtt{LCFL}}{|\boldsymbol{\omega}|_\infty}$ with $LCFL < 1$.
  2. Stretching: $\Delta t_{str} < \dfrac{RK}{max_i \sum_j \left| \dfrac{\partial \boldsymbol{u}_i}{\partial \boldsymbol{x}_j} \right|}$
  3. Diffusion: $\Delta t_{diff} < \dfrac{RK}{\nu} dx^2$
  4. Ext. forces: Δt_{fext} depends on the force field
- $\Delta_t = min(\Delta t_{adv}, \Delta t_{str}, \Delta t_{diff}, \Delta t_{fext})$.

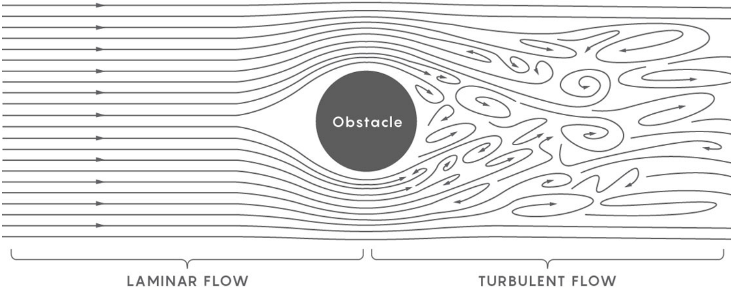
What happens if those conditions are not met ?



What happens if those conditions are not met ?



What about immersed boundaries ?



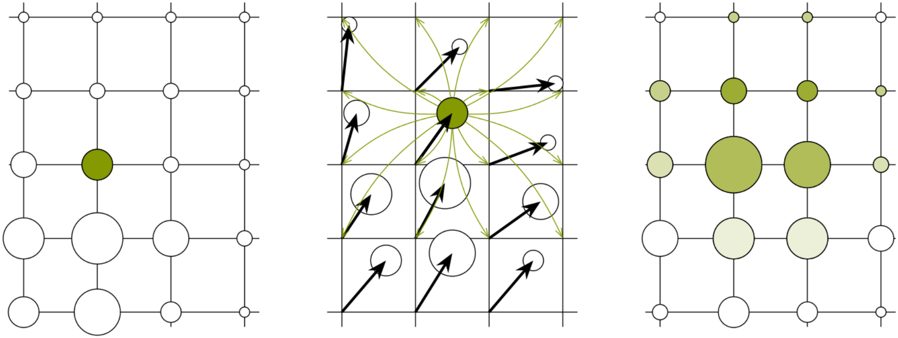
- We just introduce a penalization term to correct the velocity:

$$\frac{\partial u(\boldsymbol{x}, t)}{\partial t} = \chi_s(\boldsymbol{x})\lambda(\boldsymbol{u}_d - \boldsymbol{u}(\boldsymbol{x}, t))$$

$$\chi_s(\boldsymbol{x}) = \begin{cases} 1 \textit{ if } \boldsymbol{x} \in S \\ 0 \textit{ if } \boldsymbol{x} \notin S \end{cases}$$

Remeshed particles methods

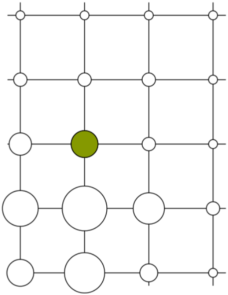
Directional operator splitting example: advection



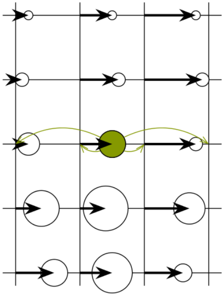
(a) Instant t^n **(b)** Advection et remaillage **(c)** Instant t^{n+1}

Remeshed particles methods - directional splitting

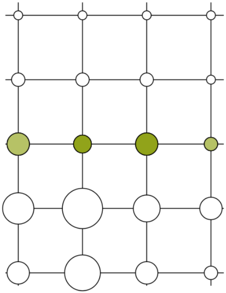
2D advection-remesh using directional splitting (1st axis)



(a) Instant t^n



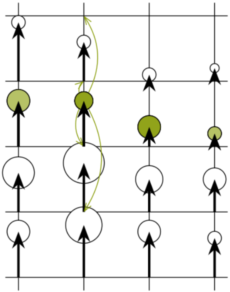
(b) Advection et remaillage, direction X



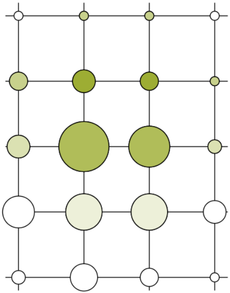
(c) État intermédiaire

Remeshed particles methods - directional splitting

2D advection-remesh using directional splitting (2nd axis)

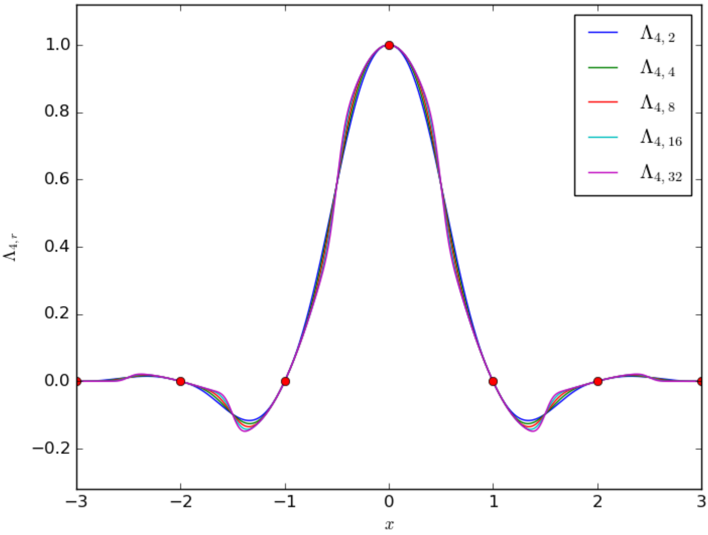


(d) Advection et remaillage, direction Y



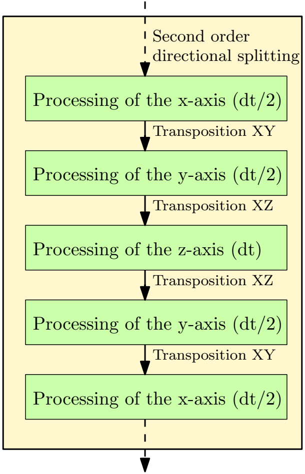
(e) Instant t^{n+1}

Remeshing kernels as obtained with [3]



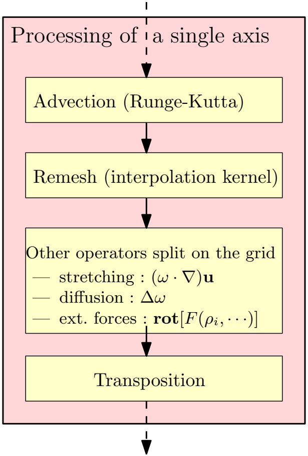
Generalized directional splitting (Strang 2nd order)

- Generalized directional splitting to all splittable operators.
- Between each directional advection-remesh step the data is **transposed** to ensure contiguous memory accesses on the accelerator:



Generalized directional splitting (Strang 2nd order)

- Generalized directional splitting to all splittable operators.
- Between each directional advection-remesh step the data is **transposed** to ensure contiguous memory accesses on the accelerator:



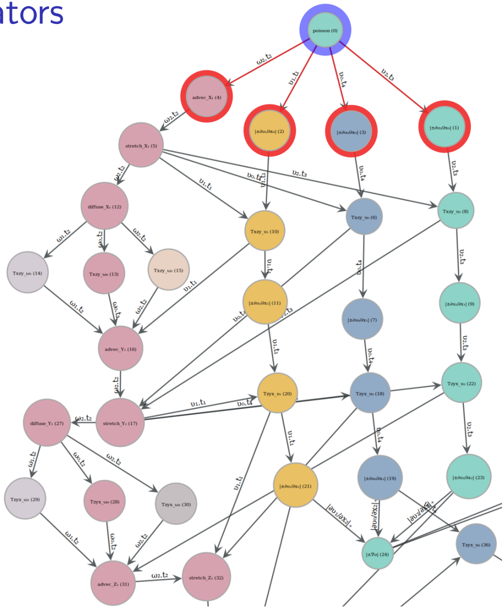
Example of directional splitting for the stretching

- Conservative form: $\dfrac{\partial \boldsymbol{\omega}}{\partial t} = \mathbf{div}\left[\boldsymbol{u} \otimes \boldsymbol{\omega}\right] = \boldsymbol{u} \underbrace{(\nabla \cdot \boldsymbol{\omega})}_{0} + (\boldsymbol{\omega} \cdot \nabla)\boldsymbol{u}.$

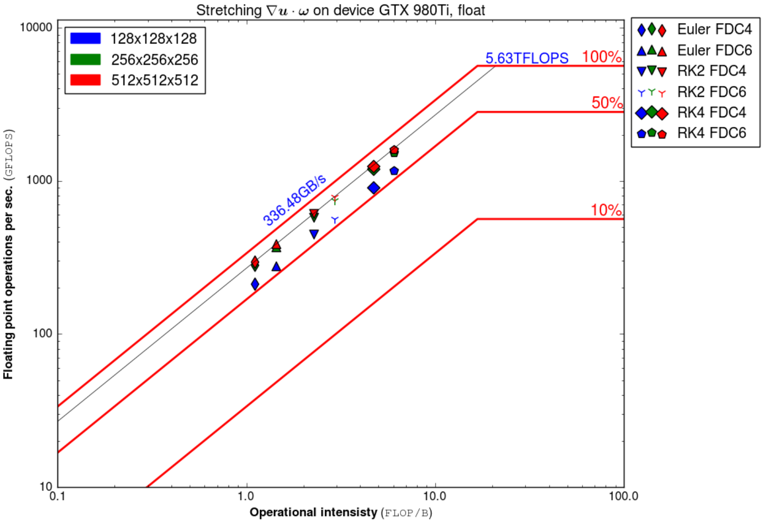
$$\boxed{\frac{\partial \boldsymbol{\omega}}{\partial t} = \mathbf{div}\left[\boldsymbol{u} \otimes \boldsymbol{\omega}\right] = \mathbf{div}\begin{bmatrix} \boldsymbol{u}_x\boldsymbol{\omega}_x & \boldsymbol{u}_x\boldsymbol{\omega}_y & \boldsymbol{u}_x\boldsymbol{\omega}_z \\ \boldsymbol{u}_y\boldsymbol{\omega}_x & \boldsymbol{u}_y\boldsymbol{\omega}_y & \boldsymbol{u}_y\boldsymbol{\omega}_z \\ \boldsymbol{u}_z\boldsymbol{\omega}_x & \boldsymbol{u}_z\boldsymbol{\omega}_y & \boldsymbol{u}_z\boldsymbol{\omega}_z \end{bmatrix}}$$

- Directional splitting (3 operators $\rightarrow$ 9 operators)
  1. Splitting axe x: $\dfrac{\partial \boldsymbol{\omega}}{\partial t} = \dfrac{\partial}{\partial x}\left[\boldsymbol{\omega}_x \boldsymbol{u}\right]$
  2. Splitting axe y: $\dfrac{\partial \boldsymbol{\omega}}{\partial t} = \dfrac{\partial}{\partial y}\left[\boldsymbol{\omega}_y \boldsymbol{u}\right]$
  3. Splitting axe z: $\dfrac{\partial \boldsymbol{\omega}}{\partial t} = \dfrac{\partial}{\partial z}\left[\boldsymbol{\omega}_z \boldsymbol{u}\right]$

DAG of operators



Performances obtained for the stretching operator



Conclusion and perspectives

Conclusion

- OpenCL is an interesting tool for HPC on heterogeneous compute platforms.
- However you have to ensure the portability of the performances.
- Splitting the Navier-Stokes in many subproblems is the key for simplicity and a natural framework for task parallelisation.
- You can already compute nice simulations with a single compute node (even without GPU).

Future developments

- Implement the multi-scale approach and MPI FFT-based solvers (global transposition of memory accross all processes).
- Full in-core simulation on multiple GPUs should enable high spatial resolution simulations.
- Full release of HySoP v2.0 to the public in 2019.

References

[1] G. Balarac et al. "Multi-scale Problems, High Performance Computing and Hybrid Numerical Methods". In: *Mathematics for Industry* (2014), pp. 245–255.

[2] P. Burns and E. Meiburg. "Sediment-laden fresh water above salt water: nonlinear simulations". In: *Journal of Fluid Mechanics* 762 (Nov. 2014), pp. 156–195.

[3] Georges-Henri Cottet et al. "High order Semi-Lagrangian particle methods for transport equations: numerical analysis and implementation issues". In: *ESAIM: Mathematical Modelling and Numerical Analysis* 48.4 (July 2014), pp. 1029–1060.

Thanks for your attention !
Any questions ?