dNami : génération de code et différentiation automatique dans un environnement HPC

Nicolas Alferez

Collaborations : Prof. Jean-Christophe Robinet
Alessandro Franchini (PhD student)
Donato Variale (PhD student)

Dr Emile Touber
Dr Stephan Winn

Dr Ivan Mary

La génération de code : un changement de paradigme

Exemple de système d'EDPs modélisant une grande classe de problèmes physiques :

$$\begin{cases} \dfrac{\partial \overrightarrow{LHS\left(\overrightarrow{q}\right)}}{\partial t} = \overrightarrow{RHS}\left(\overrightarrow{q}\right) \\ \overrightarrow{q} : \mathbb{R}^4 \to \mathbb{R}^n \\ (x, y, z, t) \mapsto \overrightarrow{q}\,(x, y, z, t) \\ + \quad \textbf{Boundary and Initial Conditions} \end{cases} \tag{1}$$

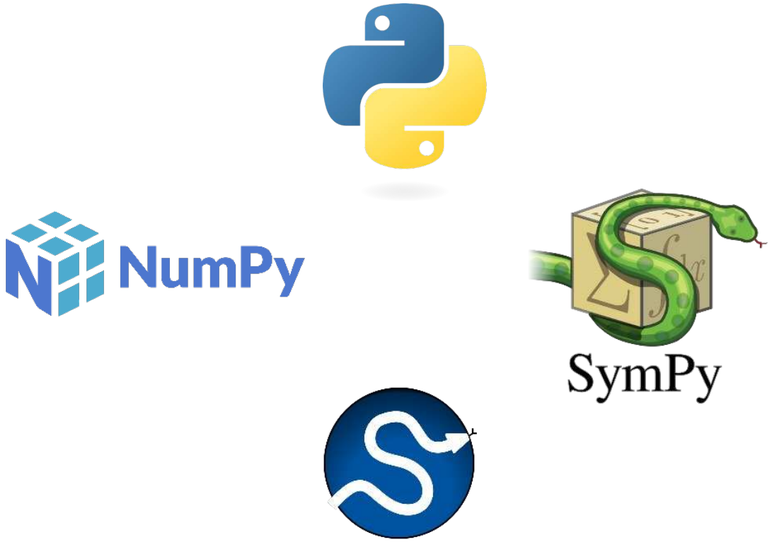
Enjeux :

- montée en complexité sur $\overrightarrow{LHS}\left(\overrightarrow{q}\right)$ et $\overrightarrow{RHS}\left(\overrightarrow{q}\right)$ avec les modèles physiques

- prise en compte de géométries et conditions aux limites complexes (BC)

La génération de code : un changement de paradigme

Problème (1) avec un nombre modéré de dégrées de libertés spatio-temporels :



Avantages

- Efficacité de développement
- Large communauté scientifique
- Solutions éprouvées et validées

Inconvénients :

- passage à l'échelle difficile pour les problèmes de grandes tailles.
- contrôle partiel sur les méthodes numériques.

La génération de code : un changement de paradigme

Problème (1) avec un grand nombre de dégrées de libertés spatio-temporels : contraintes HPC

I. Développement de code « maison» avec une implémentation de **bas niveau** (C ou Fortran en général) :

Problème physique continu

$$\begin{cases} \dfrac{\partial \overrightarrow{LHS}\left(\overrightarrow{q}\right)}{\partial t} = \overrightarrow{RHS}\left(\overrightarrow{q}\right) \\ \overrightarrow{q} : \mathbb{R}^4 \rightarrow \mathbb{R}^n \\ (x,y,z,t) \mapsto \overrightarrow{q}\,(x,y,z,t) \\ + \quad \text{Boundary and Initial Conditions} \end{cases} \quad (1)$$

Discrétisation

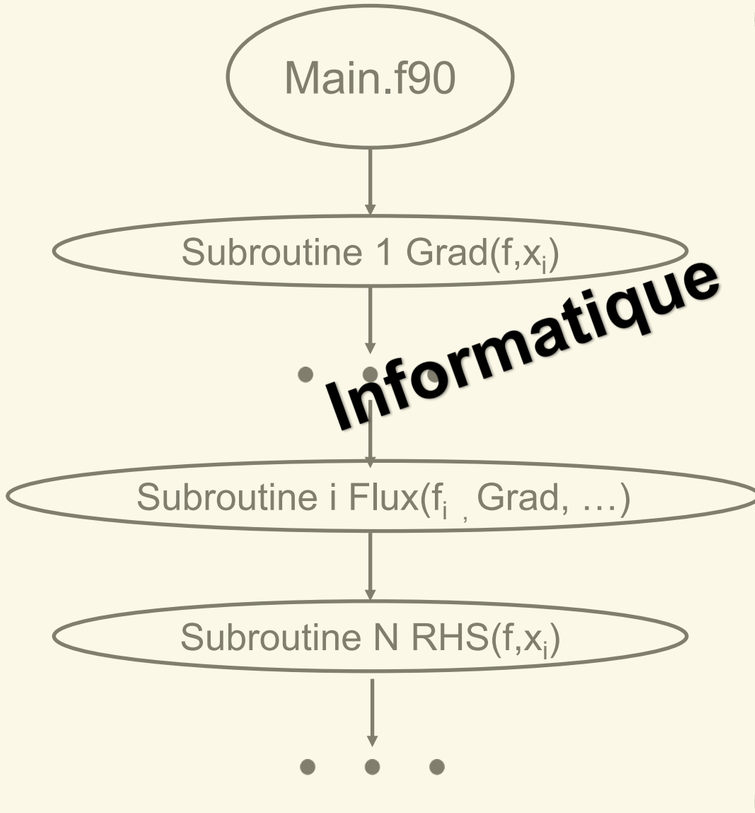
$$\frac{\partial f}{\partial x_i} \approx \sum_k a_k f\,(i+k)$$

$$\overrightarrow{F}_{i+1/2}\left(f_{i-n}, \cdots, f_{i+n}\right)$$

+ schéma d'intégration temporelle

Physique + Mathématiques Appliquées

Implémentation



La génération de code : un changement de paradigme

Problème (1) avec un grand nombre de dégrées de libertés spatio-temporels : contraintes HPC

I. Développement de code « maison» avec une implémentation de **bas niveau** (C ou Fortran en général) :

Problème physique continu → Discrétisation → Implémentation

Avantages

- Contrôle sur les méthodes numériques
- Maîtrise de l'ensemble de la méthode d'intégration (important pour les calculs instationnaires)
- Possibilité d'implémentation de méthodes ad hoc adaptées à un problème physique donné (exemple calcul propagation acoustique)

Inconvénients

- Programmation modulaire peut être fastidieuse

La génération de code : un changement de paradigme

Problème

I. Déve éral) :

Prob

- Contr
- Maîtri
  d'intég
- Possi
  hoc a
  (exem

lieuse

La génération de code : un changement de paradigme

Problème (1) avec un grand nombre de dégrées de libertés spatio-temporels : contraintes HPC

I. Développement de code « maison» avec une implémentation de **bas niveau** (C ou Fortran en général) :

Problème physique continu → Discrétisation → Implémentation

Avantages

- Contrôle sur les méthodes numériques
- Maîtrise de l'ensemble de la méthode d'intégration
- Possibilité d'implémentation de méthodes ad-hoc adaptées à un problème physique donné (exemple calcul acoustique)

Inconvénients

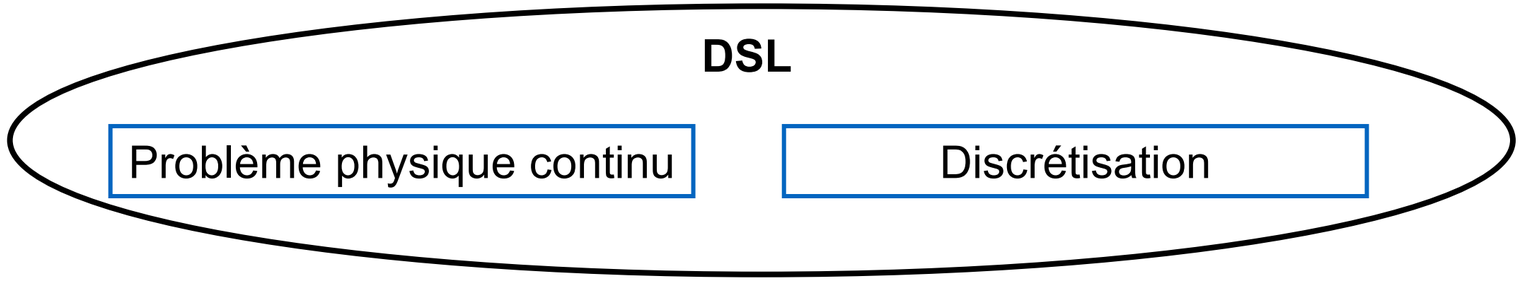
- Programmation modulaire peut être fastidieuse
- Implémentation non flexible, limitée à un choix de problème (1)
- … et « efficace » pour un choix de hardware.
- Demande des compétences en calcul scientifique aux spécialistes du domaine physique.
- Programmation modulaire contraignante pour le HPC

La génération de code : un changement de paradigme

Problème (1) avec un grand nombre de dégrées de libertés spatio-temporels : contraintes HPC

II. Utilisation de « domain specific language » (DSL), implémentation de **haut niveau** (souvent Python) :

Domain Specific Language : par opposition au General Propose Language, c'est un langage de programmation spécialisé dans la programmation d'une catégorie donnée d'application. Exemple : HTML, Latex



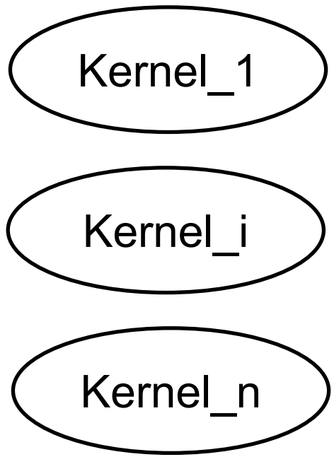
On utilise les mots du langage (DSL) pour créer les boucles, déclarer des variables…

Nombreux exemples applicables à la résolution des équations de NS :

OpenSBLI (**Southampton**), HTR (**Stanford**), Saiph (**BSC**), Devito (**Imperial College**), FEniCSx (**Cambridge**), …

Implémentation

« backend » générée **automatiquement**.



portage efficace « gratuit » vers

CPU/GPU

Scalabilité

La génération de code : un changement de paradigme

Problème (1) avec un grand nombre de dégrées de libertés spatio-temporels : contraintes HPC

II. Utilisation de « domain specific language » (DSL), implémentation de **haut niveau** (souvent Python) :

Avantages

- Développement très efficace.
- Portage effectué par les informaticiens. Support automatique des nouvelles architectures matérielles.
- Très bonne efficacité HPC.
- Flexibilité sur le choix du problème (1) mais pas toujours…

Inconvénients

- Méthodes numériques souvent prédéfinies.
- Code source finale souvent « illisible »
- Nécessite d'apprendre un nouveau langage (DSL)
- Solver souvent attaché à un choix de problème (1)
- Difficultés de débogage.

La génération de code : un changement de paradigme

Problème (1) avec un grand nombre de dégrées de libertés spatio-temporels : contraintes HPC

III. Utilisation de librairies performantes pour l'algèbre linéaire (SLEPc, PETc,…).

Non décrit ici, mais dNami est interfacé avec PETc pour la résolution de problèmes aux valeurs propres de grandes tailles (stabilité globale).

IV. Jit compilation : julia

Semble prometteur. Peu d'exemple de solver HPC pour les problèmes de grandes tailles, mais théoriquement adapté ?

dNami: un DSL pour le calcul multi block en maillages structurés

Objectifs :

- Problème (1) le plus général possible. Application aux équations de Navier-Stokes compressibles, avec fermeture thermodynamique complexe (gaz non-idéaux, transports multiespèces, déséquilibre thermochimique).
- Choix arbitraire de méthodes numériques pour la discrétisation spatiale.
- Permettre une prise en main facile (code généré lisible, langage simple et possibilité de développer « à la main »).

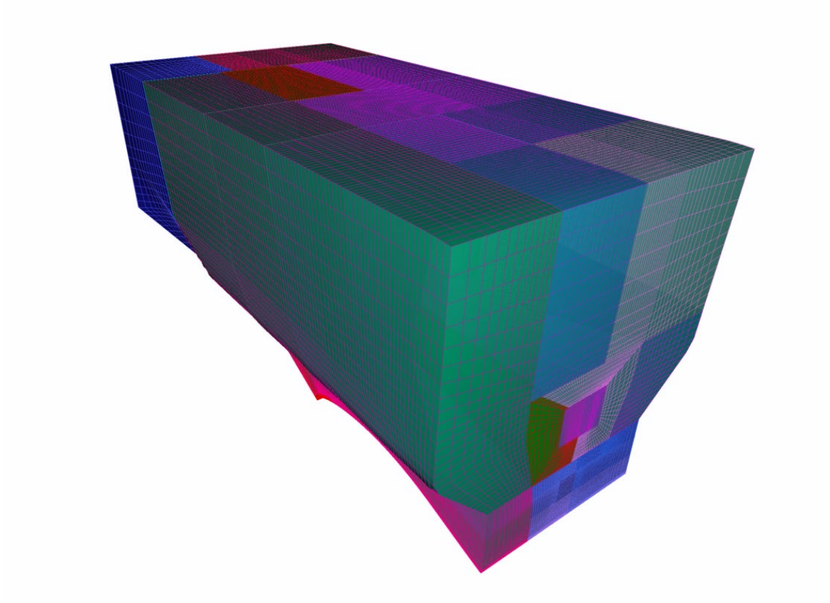
Limitations :

- Maillages structurés multi block.
- Avancée en temps explicite (pour l'instant).

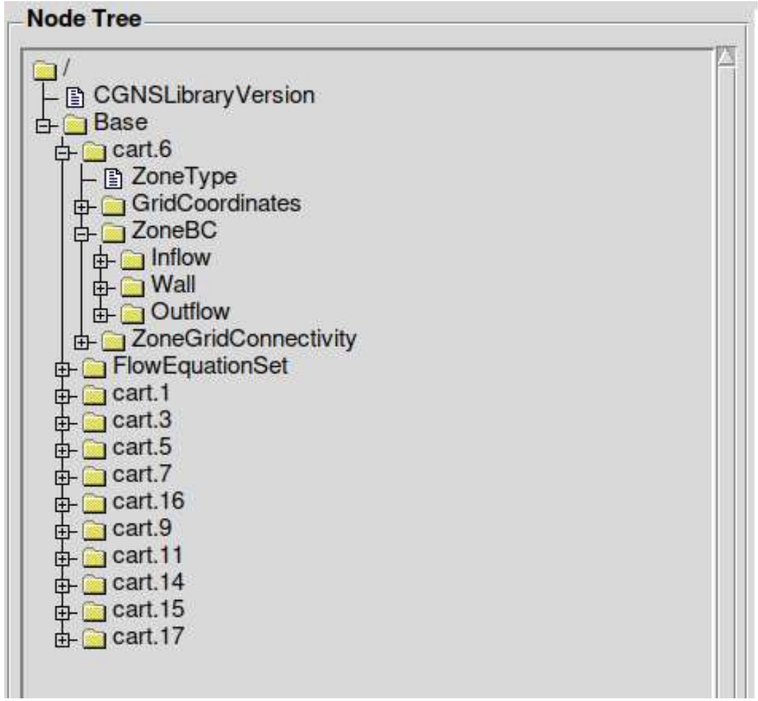
Intégration dans le framework CASSIOPEE (ONERA)

(http://elsa.onera.fr/Cassiopee/)

“Cassiopee (CFD Advanced Set of Services In an Open Python EnvironmEnt) is a set of Python modules providing advanced services for preparing CFD computations and post-processing CFD solutions.”



Automatic work partitioning



Complies with the CGNS standard.

CGNS is **a data model** enabling the **description** of a numerical **simulation** case. CGNS is now **widely used in the CFD community** for exchanging data between the various software of a simulation pipeline (**mesh generators**, **flow/structure/thermal solvers**, **post-processing tools**).

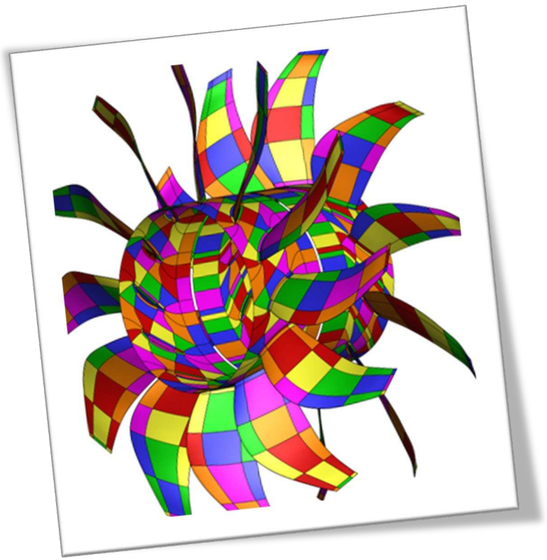
Intégration dans le framework CASSIOPEE (ONERA)

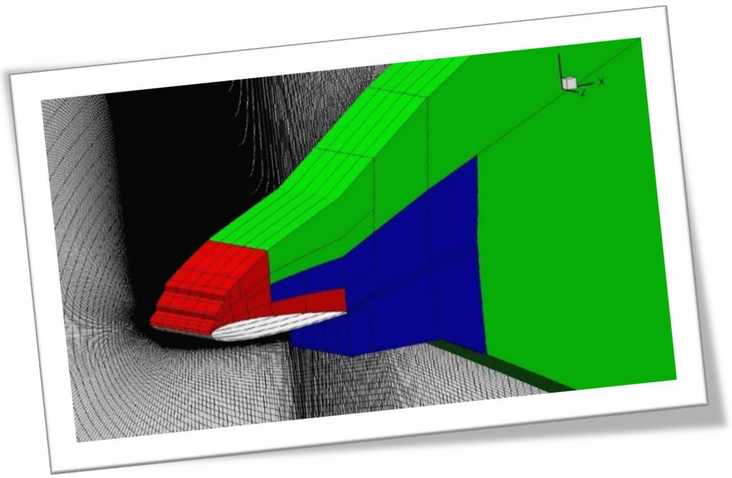
(http://elsa.onera.fr/Cassiopee/)

CASSIOPEE

Python/CGNS

Connector Connectivity, Transfers, …	**Co-processing**	**Mesh generation**	**dNami**





CGNS/Python standard : pas de copie entre modules

Environnement Python (script, interopérabilités)

Compatible HPC: arbre distribué (MPI)

dNami : démo de fonctionnalités

(https://github.com/dNamiLab/dNami)

2D Navier-Stokes equations for an Ideal Gaz

$$\begin{cases} \dfrac{\partial}{\partial t}\begin{pmatrix}\rho\\ \rho u\\ \rho v\\ \rho e_t\end{pmatrix}+\dfrac{\partial}{\partial x}\begin{pmatrix}\rho u\\ \rho u^2+p\\ \rho uv\\ u(\rho e_t+p)\end{pmatrix}+\dfrac{\partial}{\partial y}\begin{pmatrix}\rho v\\ \rho uv\\ \rho v^2+p\\ v(\rho e_t+p)\end{pmatrix}-\dfrac{\partial}{\partial x}\begin{pmatrix}0\\ \tau_{xx}\\ \tau_{xy}\\ u\tau_{xx}+v\tau_{xy}+q_x\end{pmatrix}-\dfrac{\partial}{\partial y}\begin{pmatrix}0\\ \tau_{xy}\\ \tau_{yy}\\ u\tau_{xy}+v\tau_{yy}+q_y\end{pmatrix}=\mathbf{0} \\ p=(\gamma-1)\,\rho \mathrm{e} \qquad \mathrm{e}=e_t-0.5\times\left(u^2+v^2\right) \end{cases}$$

Les équations sont renseignées à l'aide de chaines de caractères et d'un langage simple. Possibilité d'écrire tout mot du langage Fortran (sin,cos, sign,…).

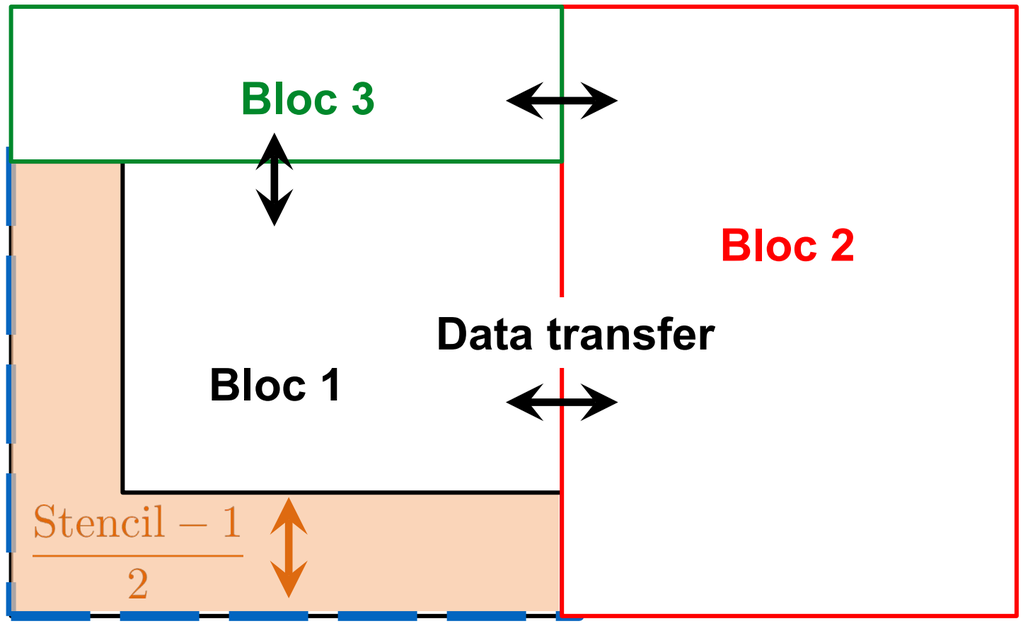
$$\frac{\partial u}{\partial x} \Longrightarrow \text{« [u]_1x »} \qquad u\left(i+1,j,k\right) \Longrightarrow \text{« u _ip1jk »}$$

dNami : spécification des BCs

- Conditions aux limites physiques : règles de spécification aussi générales que celles du RHS. BCs sur « q » ou « RHS ».
- Conditions aux limites numériques :

BCs physiques :
equations sur « q » ou « RHS »

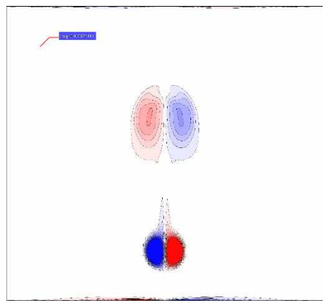
Schémas centrés d'ordres réduits ou schémas décentrés au choix.

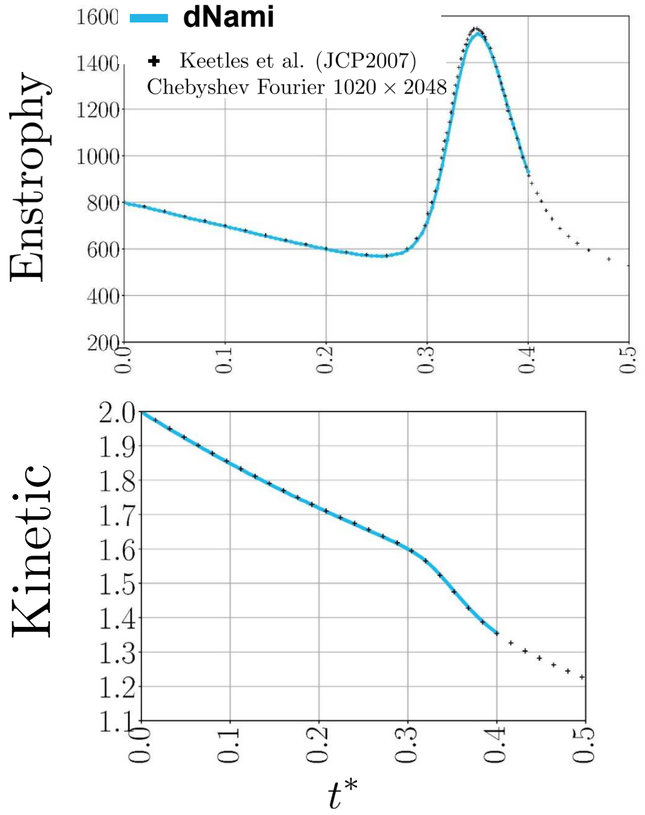


+
utilisation de Sympy pour les décompositions le long des caractéristiques

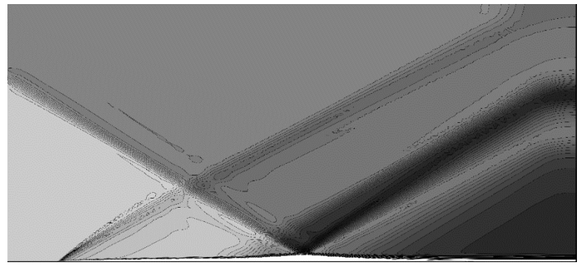
dNami : quelques cas test NS compressible et RANS

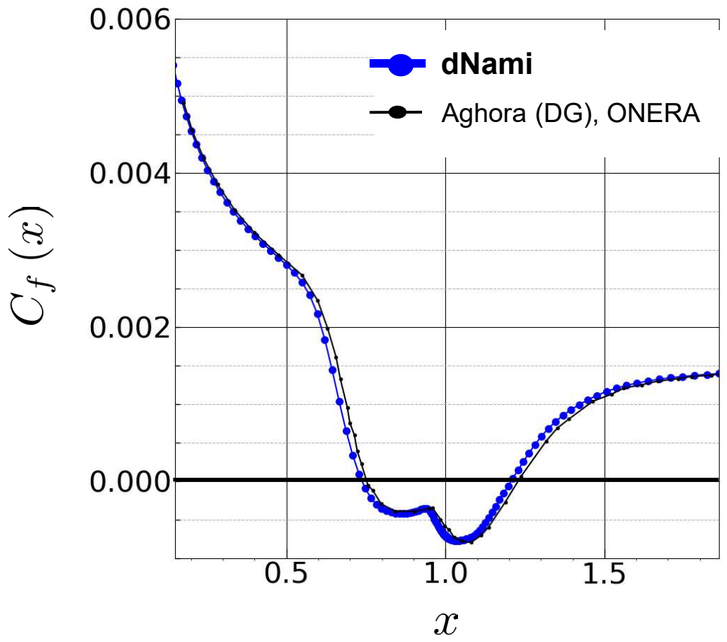
Vortex dipole



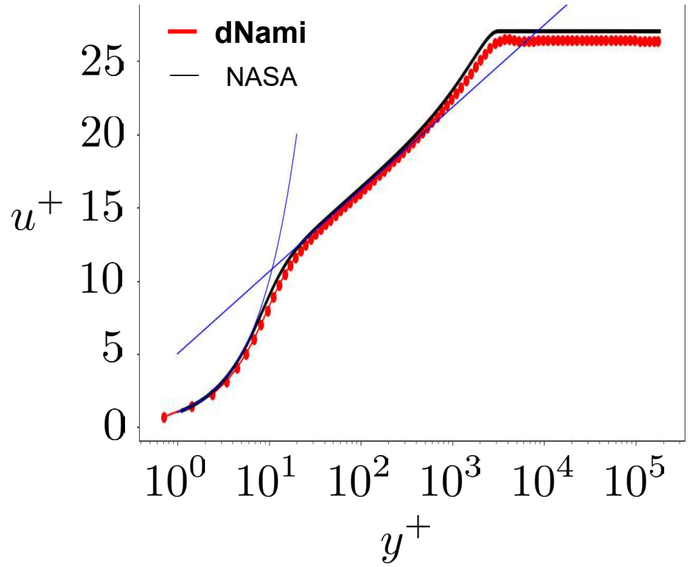


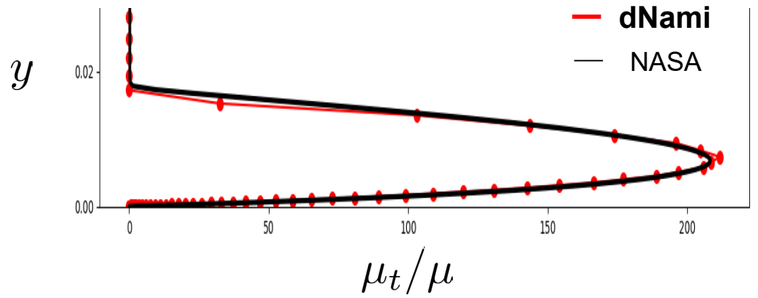
SWBLI – Degrez case





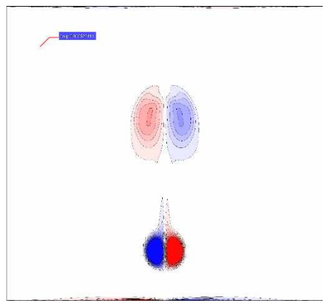
Couche limite turbulente (RANS – Spalart Allmaras)



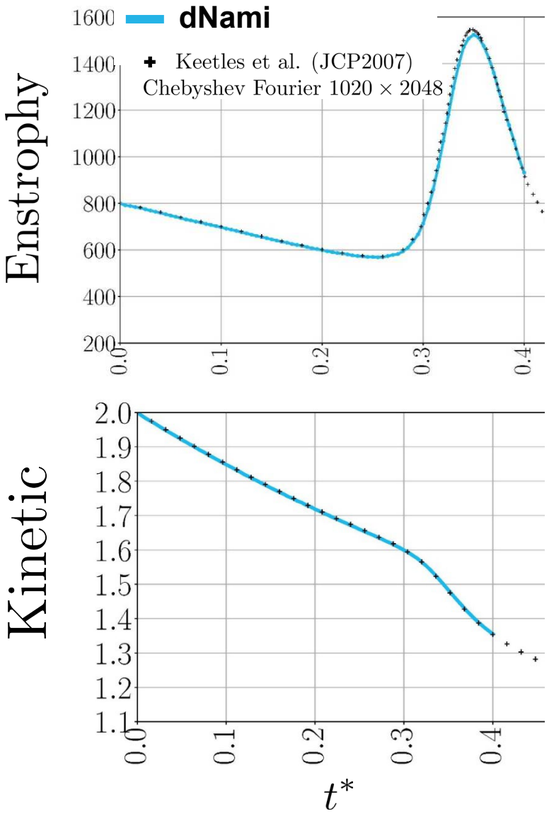


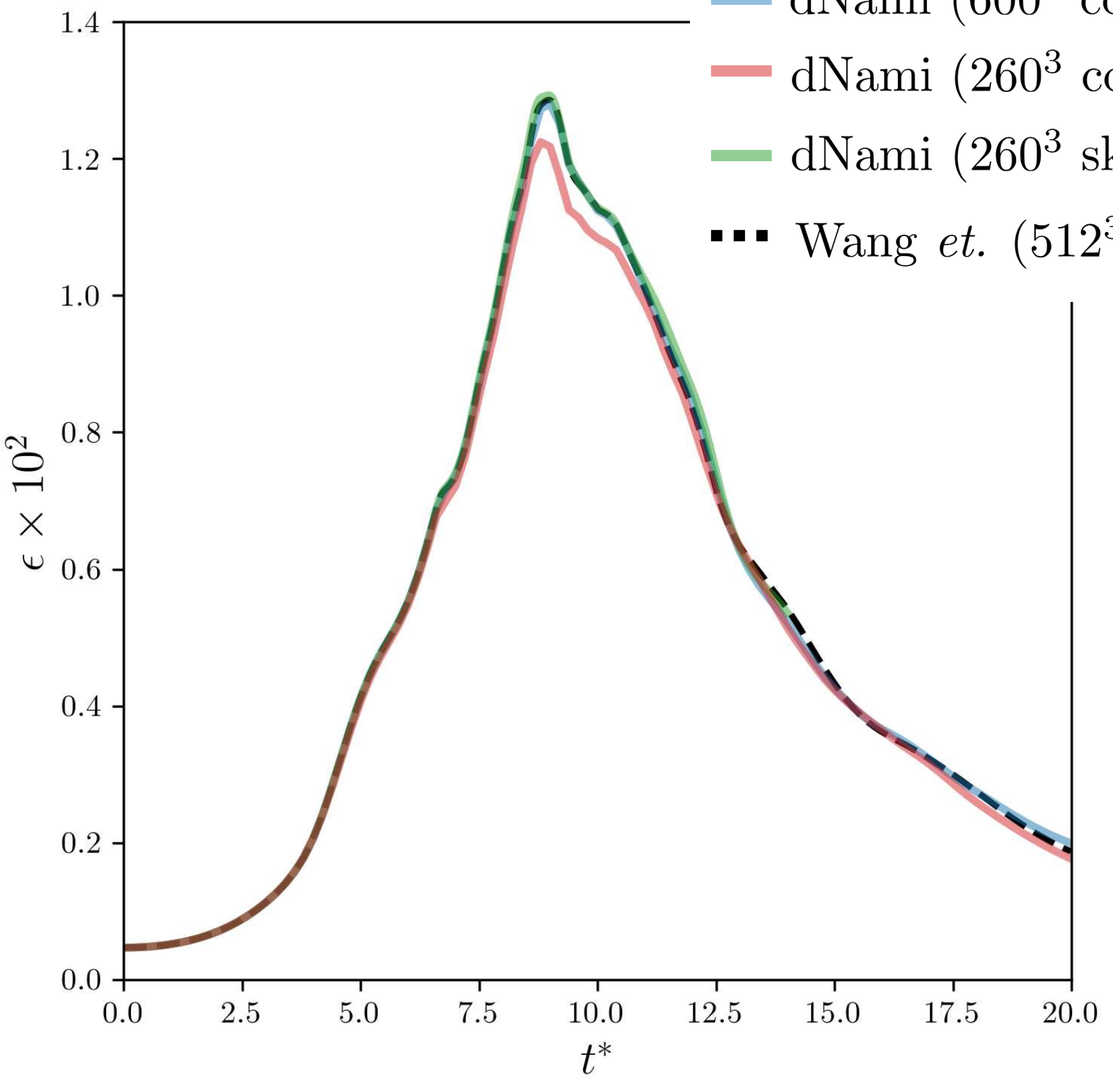
dNami : quelques cas test NS compressible et RANS

Vortex dipole

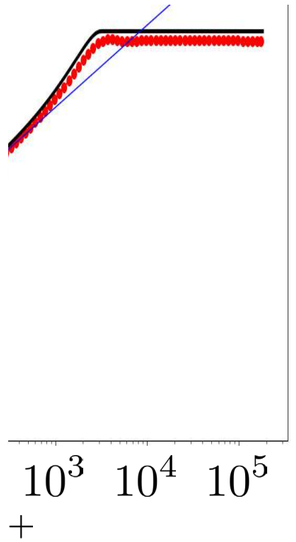


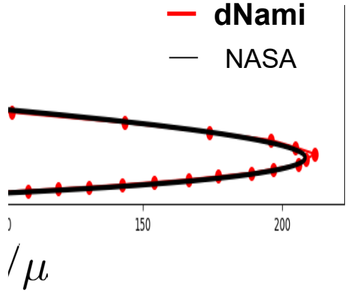
3D TGV





turbulente
rt Allmaras)

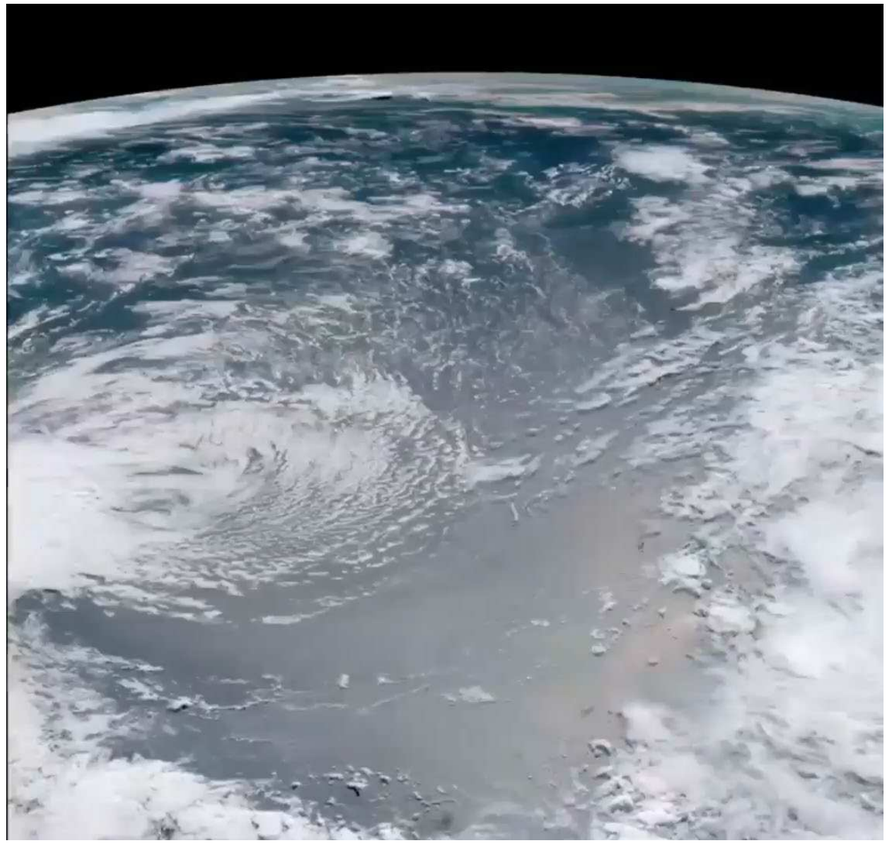




dNami : application géophysique (Hunga-Tonga tsunami)

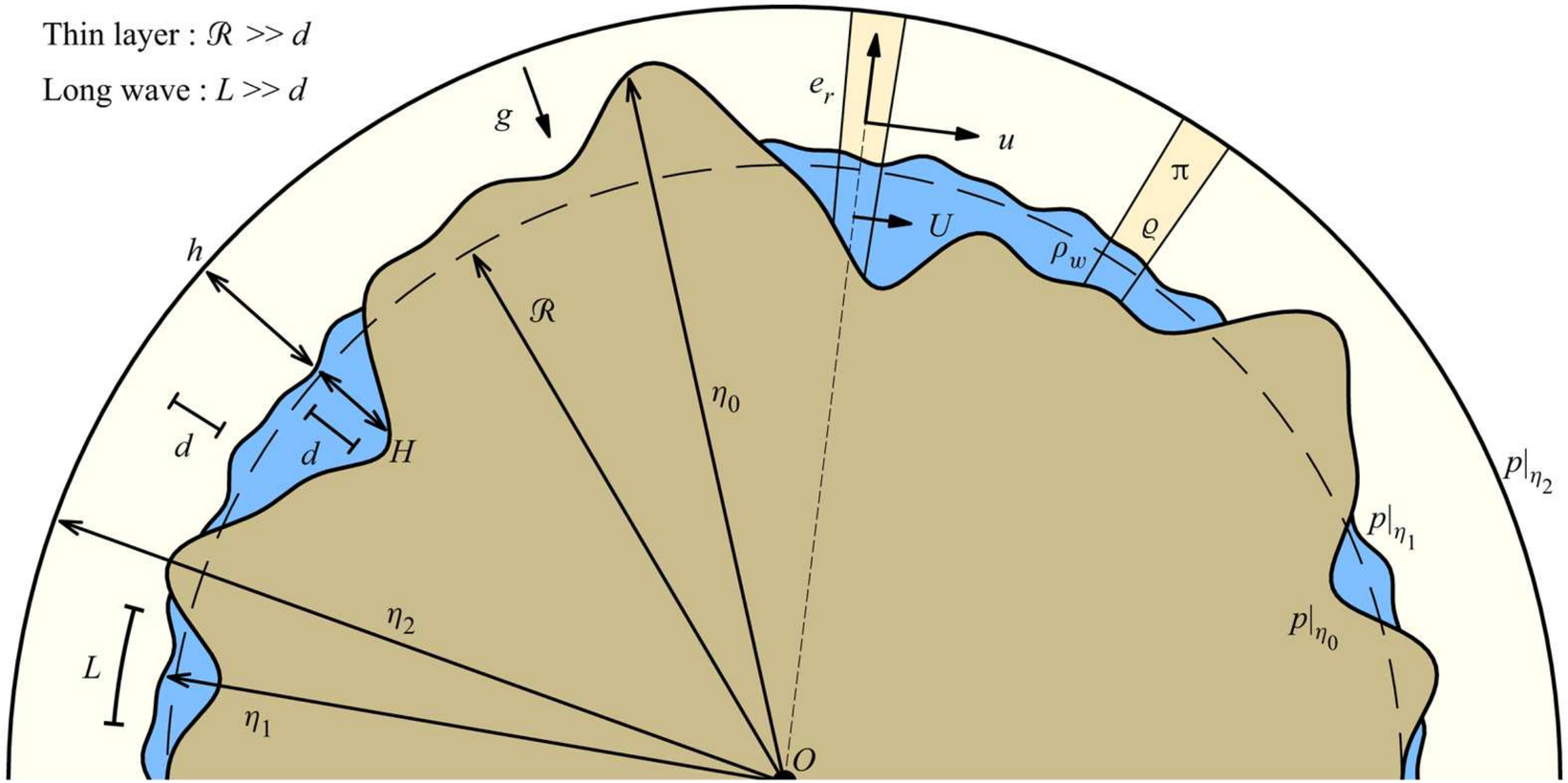
15 juillet 2022 : éruption du volcan Hunga-Tonga

- Forçage atmosphérique + tsunami
- Dysfonctionnement des systèmes d'alerte, notamment sur les côtes japonaises.



dNami : application géophysique (Hunga-Tonga tsunami)

Proposer un nouveau modèle simple d'équations de Saint-Venant couplées avec un modèle de propagation atmosphérique (Euler 2D).



dNami : application géophysique (Hunga-Tonga tsunami)

Écriture des équations au niveau continu du modèle couplé ⟹ **Écritures des dictionnaires de symboles pour dNami** ⟹ **Génération du solver** ⟹ **Exécution du solver**

Temps long — **Temps long**

```
#GenerateRHS:
append_Rhs(Adv, 5, 4, rhsname, vnamesrc_Adv, update=False, rhs=rhs, stored=True)
append_Rhs(S, 5, 4, rhsname, vnamesrc_S, update=True, rhs=rhs, stored=False)
append_Rhs(C, 5, 4, rhsname, vnamesrc_C, update=True, rhs=rhs, stored=False)
append_Rhs(Fss, 5, 4, rhsname, vnamesrc_Fss, update=True, rhs=rhs, stored=False)

#GenerateFilters:
genFilter(13, 8, len(varsolved), rhs=rhs)
```

Winn, S. D., et al. "Two-way coupled long-wave isentropic ocean-atmosphere dynamics." *Journal of Fluid Mechanics* 959 (2023)

15-jan-2022
UT 04:00

himawari 8 (jma)
goes 16 & 17 (noaa)
meteosat 8 & 11 (eumetsat)

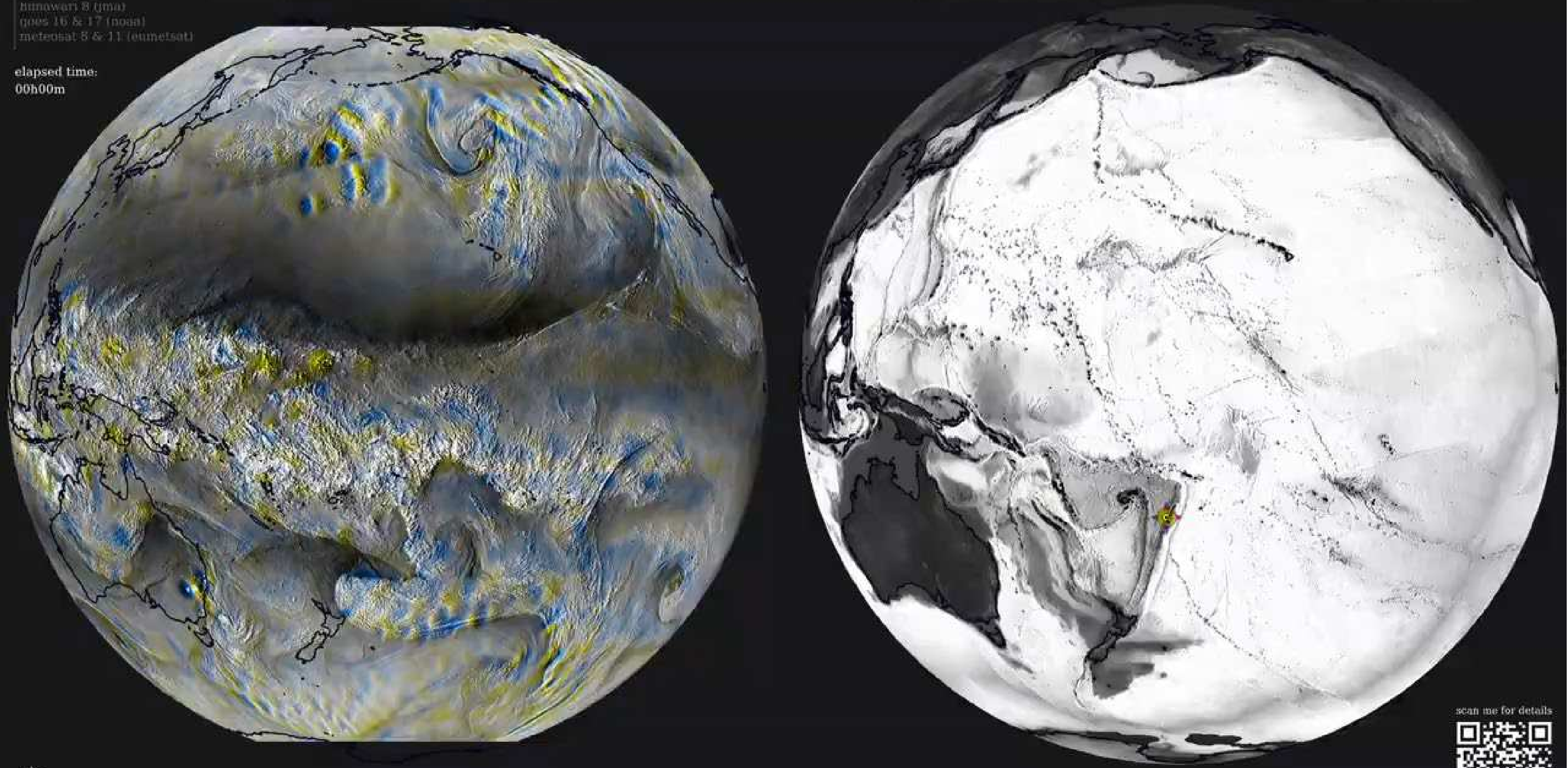
elapsed time:
00h00m

Satellite observations

brightness temperature ~8km above ground

Coupled atmosphere-ocean model

sea-surface displacements, main atmospheric pressure wave & topography



(C) S2T Unit

210 222 235 248 260
instantaneous values [K]

-20 -10 0 10 20
band-pass filtered variations in ~30-minute intervals [mK]

1 2 4 6 8 10
maximum absolute sea-level variation [cm]

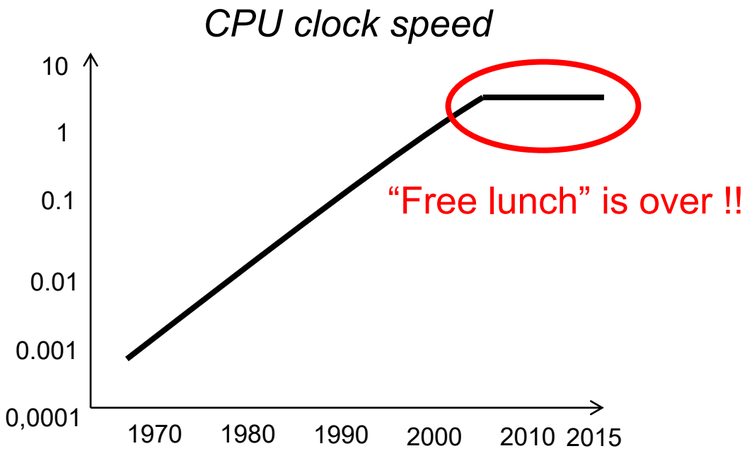
-5 0 5
layer-averaged pressure fluctuation [Pa]

scan me for details

Calcul Haute Performance : cas d’étude

Apport de l’approche Domain Specific Language

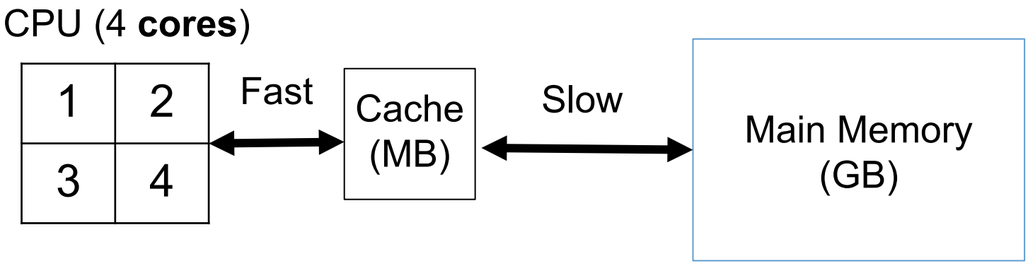
Contraintes HPC imposées par les architectures modernes



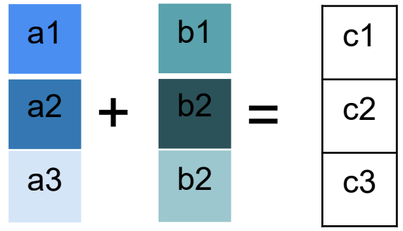
CPU	Cores	Vector (DP)	Speedup
Nehalem (2009)	4	2	8
Haswell (2014)	18	4	72
Broadwell (2016)	24	4	96
Skylake - …. (2017-2023)	28	8	224

CPU and GPU are divided in smaller units (Cores) to perform operations in //

MIMD :



SIMD :



Each Cores can performs **Vectors** operations

- Limit memory transfers
- Efficiently distribute among Cores
- Efficient vectorization

≡

Code modernization (both CPU/GPU)

Optimisation de code : où aller et quand s'arrêter?

Algorithmes développés pour le calcul scientifique :

utilisent des **données** + effectuent des **opérations arithmétiques**

Bytes (Octets) ← ; → FLop (Floating-Point Operation)

- Si **beaucoup** de Bytes pour **peu** de FLop → Transferts de données à l'exécution
- Si **beaucoup** de FLop pour **peu** de Bytes → Opérations arithmétiques à l'exécution

} Dépendant de **l'architecture matérielle**

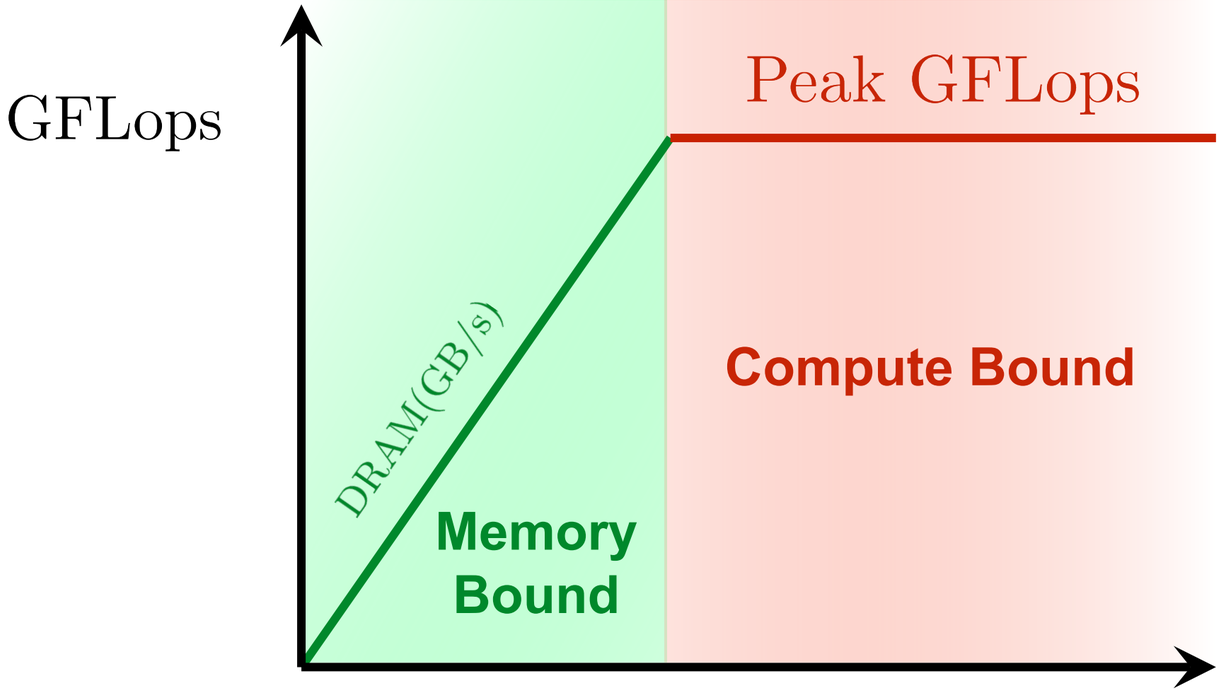
→ Orienter les actions de développement à l'aide du model « **Roofline** »

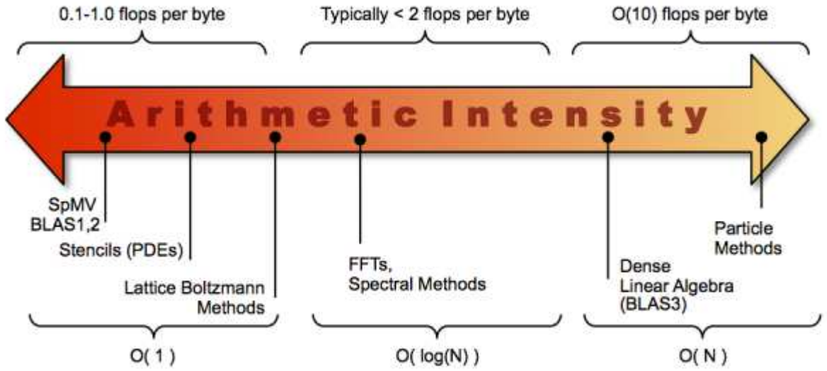
Optimisation de code : où aller et quand s'arrêter?

« Roofline » performance model (University of California, Berkeley):

$$\mathrm{AI} = \frac{\mathrm{FLop}}{\mathrm{Bytes}}$$: Arithmetic Intensity

GFLops : GFLop per second





S. Williams, et al., *The Roofline Model: A Pedagogical Tool for Auto-tuning Kernels on Multicore Architectures*, Hot Chips 20, August 10, 2008,

dNami : cas pratique sur les équations de NS compressibles

Avantage de l'approche DSL

- Contrôle fin et simple sur l'écriture des boucles.
- Possibilité d'écrire les équations sans stockage intermédiaire (difficile à réaliser pour NS3D à la main).

Cas d'étude : écoulement de TGV

Maillage : 52 millions de points (100^3 par cœurs sur 52 cœurs)

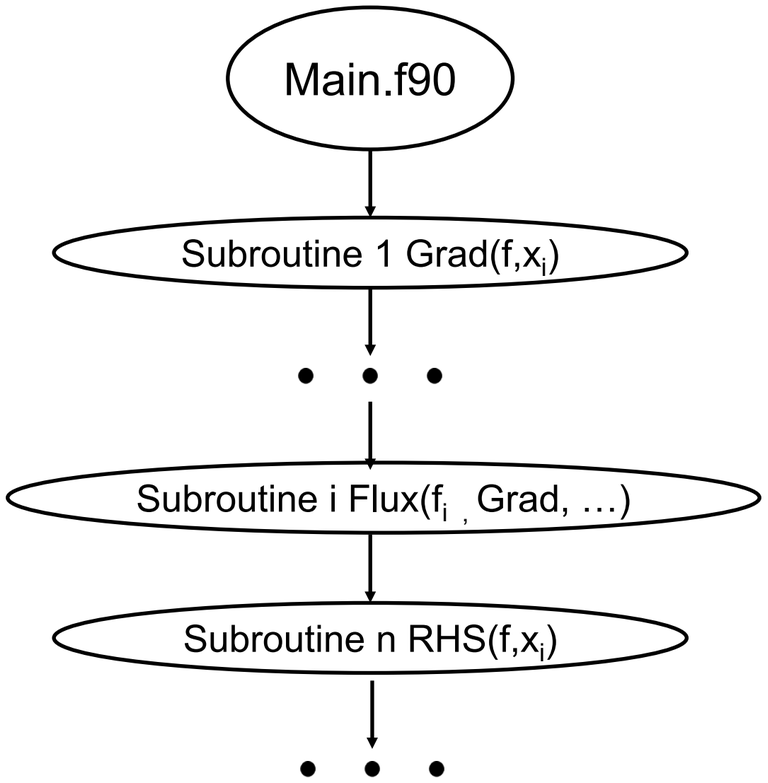
Hardware : Workstation 52 cœurs (Intel(R) Xeon(R) Gold 6230R)

Méthodes numériques : FDo4 (stencil 5 points), RK3 + Filtre explicite sur 11 points (ordre 10) tous les 5 pas de temps

20 pas de temps

dNami : cas pratique sur les équations de NS compressibles

1er cas : le code écrit « à la main » :



Code très modulaire avec très peu de factorisation (cas limite).

Voir fichiers de données .py

dNami : cas pratique sur les équations de NS compressibles

1er cas : le code écrit « à la main » :

RK3 + Filter + RHS :
(no transfer MPI)

$$\mathrm{T}_{\mathrm{elapse}} = 58\ \mathrm{s} \qquad \mathrm{T}_{\mathrm{reduced}} = \frac{\mathrm{T}_{\mathrm{elapse}}}{100^3 \times 3 \times 20} = 0.97\ \mu\mathrm{s}$$

Peut-on aller plus vite ?

2ème cas : le code factorisé par dNami

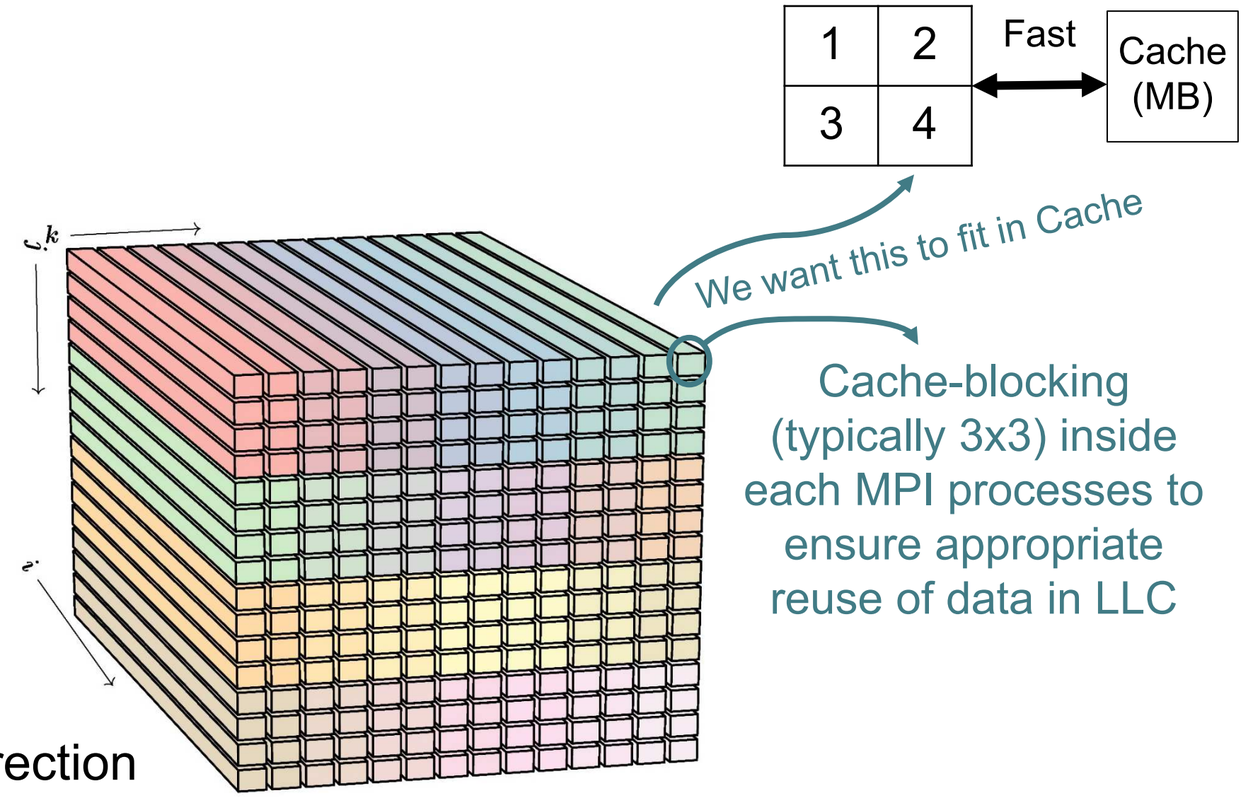
Calcul du RHS dans une seule boucle (par direction). Aucune variable intermédiaire stockée en mémoire (grad(u), flux…)

dNami : cas pratique sur les équations de NS compressibles

2ème cas : le code factorisé par dNami

Calcul du RHS dans une seule boucle (par direction). Aucune variable intermédiaire stockée en mémoire (grad(u), flux…)

Cache-blocking technique :



dNami : cas pratique sur les équations de NS compressibles

2ème cas : le code factorisé par dNami

RK3 + Filter + RHS :
(no transfer MPI)

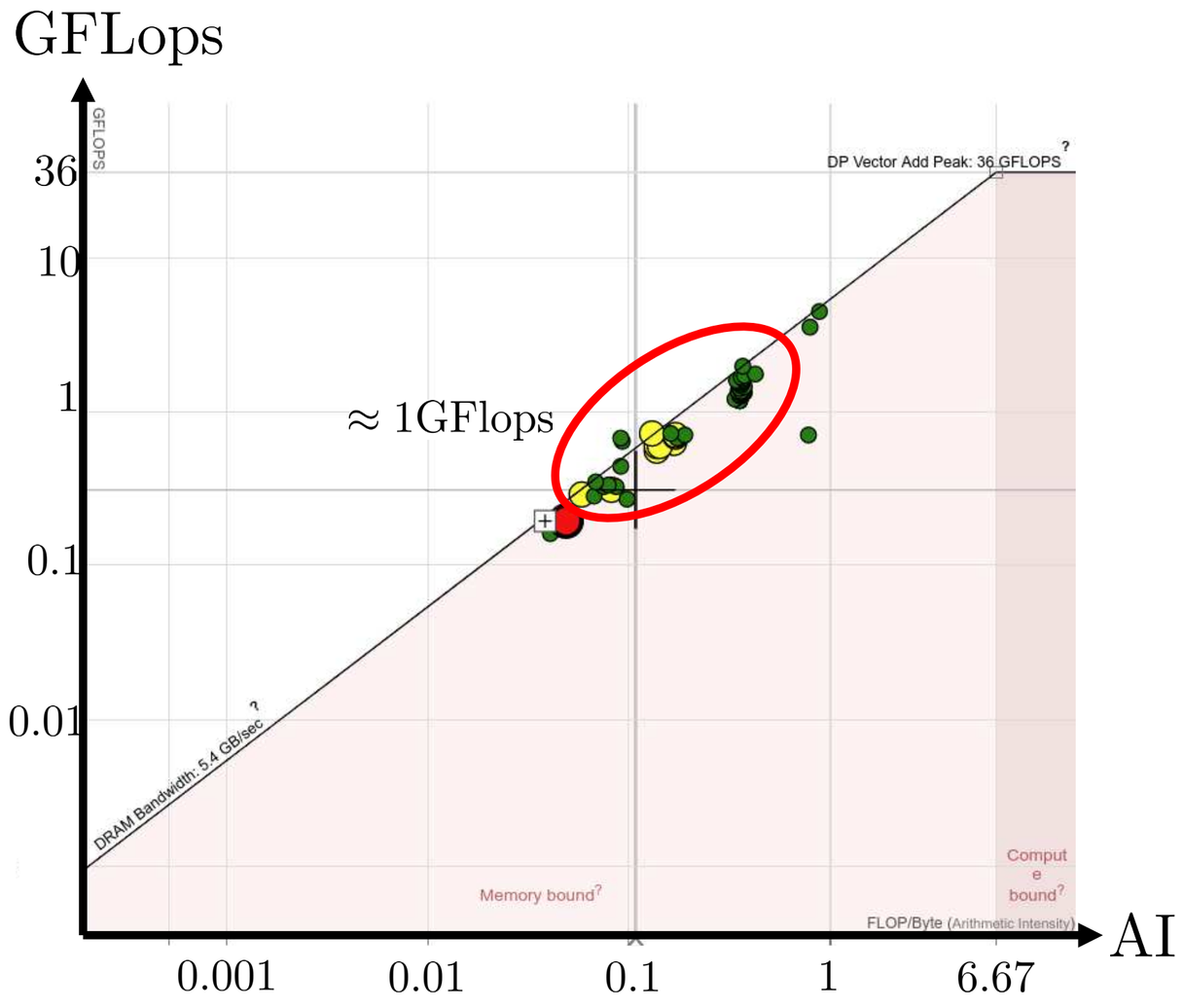
$$\mathrm{T}_{\mathrm{elapse}} = 9 \ \mathrm{s} \qquad \mathrm{T}_{\mathrm{reduced}} = \frac{\mathrm{T}_{\mathrm{elapse}}}{100^3 \times 3 \times 20} = 0.15 \ \mu\mathrm{s}$$

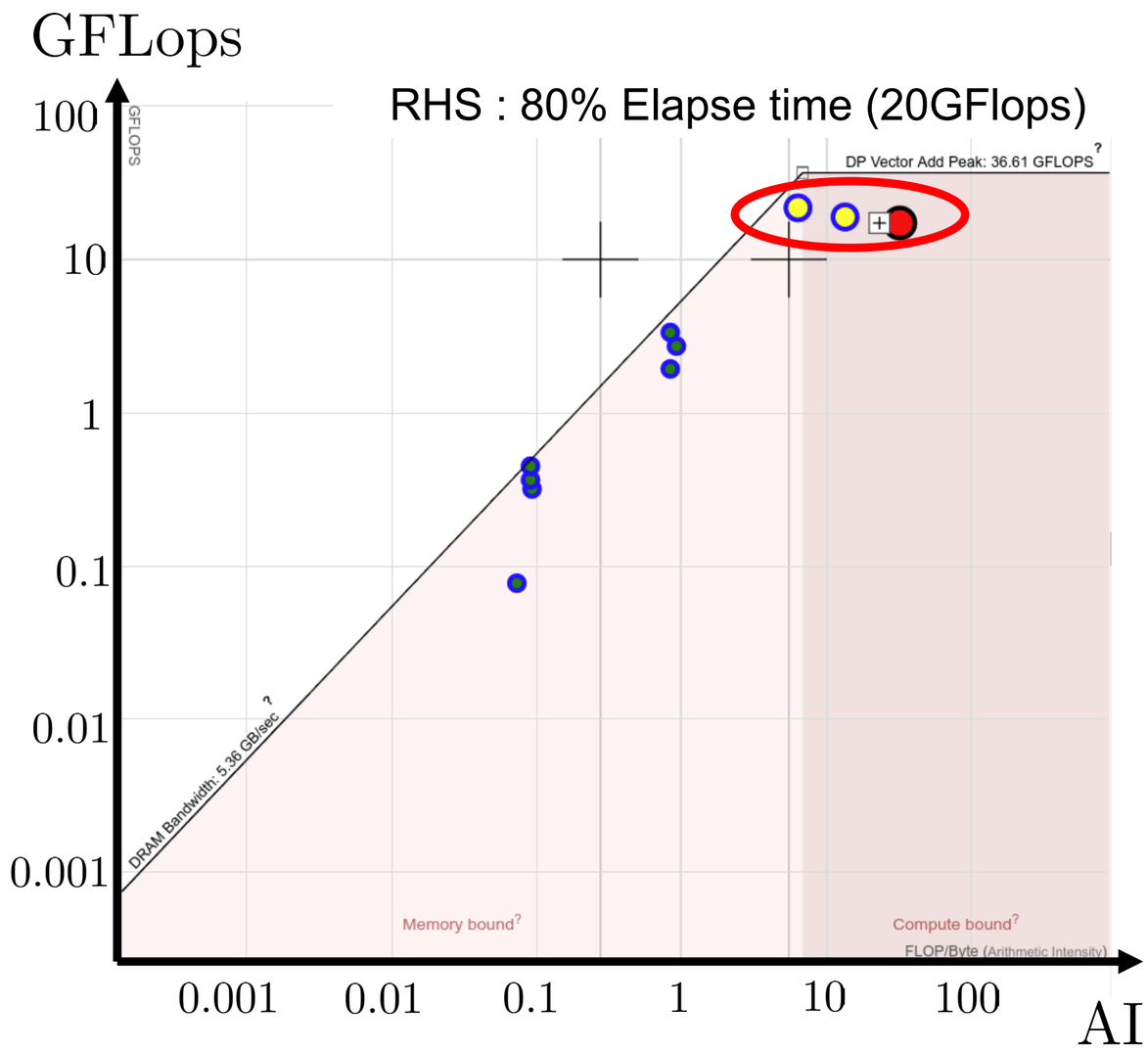
6.5 speedup !

Que s'est-il passé ?

dNami : cas pratique sur les équations de NS compressibles

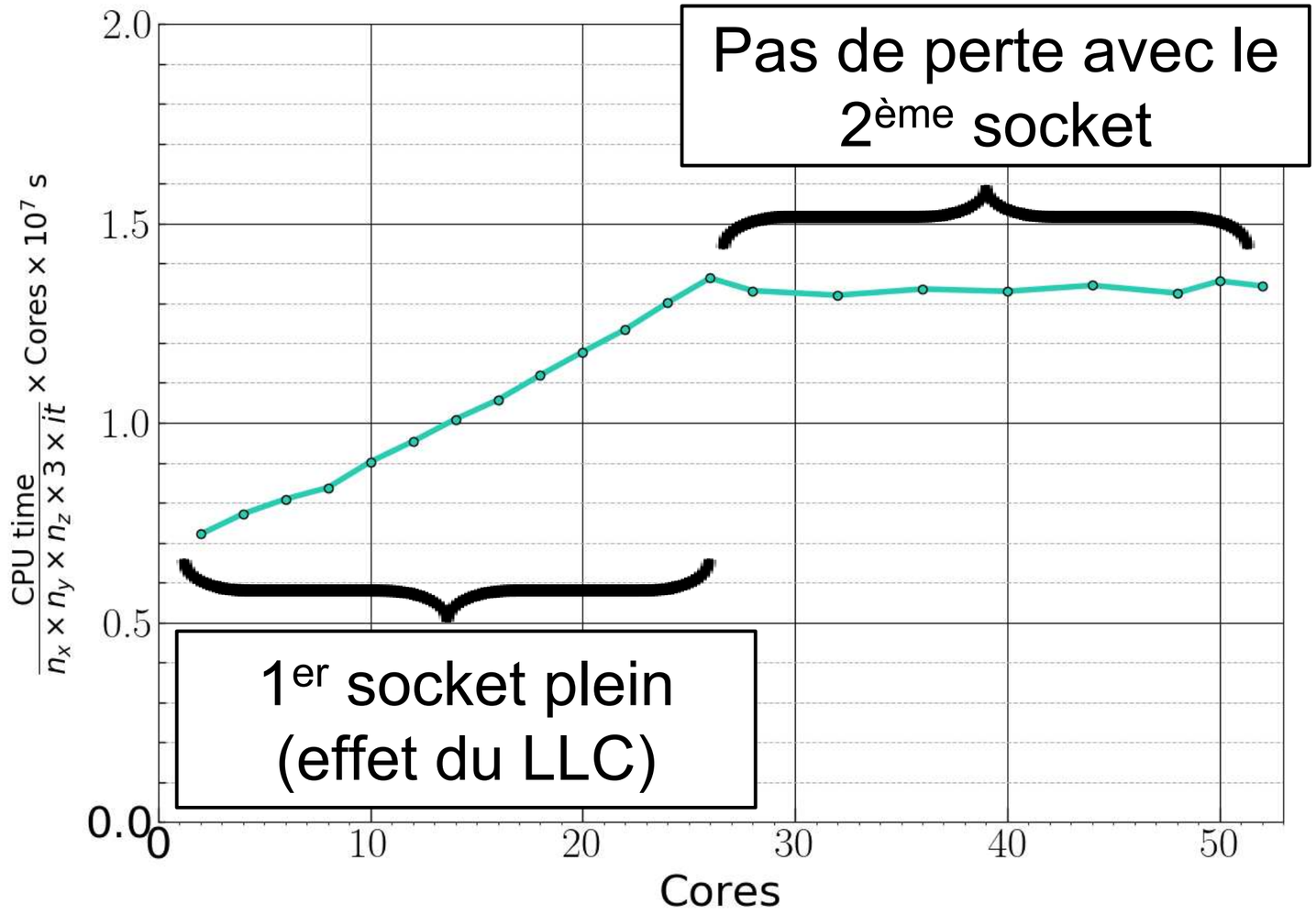
Analyse à l'aide des « Rooflines » :





dNami : cas pratique sur les équations de NS compressibles

Courbe de scalabilité faible :



Quelques messages « à emporter » :

- La difficulté principale imposée par les architectures HPC modernes est la scalabilité « intranoeud ». Nécessite d'exploiter pleinement tous les niveaux de parallélismes tout en gérant les transferts mémoires.
- Factoriser l'algorithme et faire attention aux accès mémoire est capital.
- Une approche à base de DSL permet d'atteindre des niveaux de performances difficilement, voire inatteignable « à la main ».
- Modernisation des codes => gros gains de performance possibles.

Génération de code et différentiation automatique

Application aux calculs de modes globaux

Travaux de thèse d'Alessandro Franchini

Génération automatique du code « linéarisé »

Objectif : « discretize then differentiate »

$$\begin{cases} \dfrac{\partial \overrightarrow{q}}{\partial t} = \overrightarrow{RHS}\left(\overrightarrow{q}\right) \\ \overrightarrow{q} : \mathbb{R}^4 \to \mathbb{R}^n \\ (x, y, z, t) \mapsto \overrightarrow{q}\,(x, y, z, t) \end{cases} \quad (1)$$

+ Boundary and Initial Conditions

$$\Longrightarrow$$

$$\begin{cases} \dfrac{\partial \overrightarrow{q'}}{\partial t} = \dfrac{\partial \overrightarrow{\overline{\mathrm{RHS}}}}{\partial \overrightarrow{q}} \odot \overrightarrow{q'} \\ \overrightarrow{q'} : \mathbb{R}^4 \to \mathbb{R}^n \\ (x, y, z, t) \mapsto \overrightarrow{q}\,(x, y, z, t) \end{cases} \quad (2)$$

+ Linearised Boundary Conditions

Solveur non linéaire HPC généré automatiquement par dNami

Solveur linéaire HPC généré automatique par dNami ?

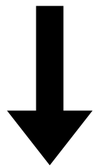
Génération automatique du code « linéarisé »

Méthode : couplage dNami avec Tapenade (INRIA)

Automatic Differentiation tool that computes **tangent** or **adjoint** derivatives.

Inputs:

1. Routine to differentiate
2. Dependent and Independent variables



Generation of differentiated code

original: T,U ↦ e	tangent mode: T,Ṫ,U,U̇ ↦ e,ė
	ė2 = 0.0
e2 = 0.0	e2 = 0.0
do i=1,n	do i=1,n
	ė1 = Ṫ(i)-U̇(i)
e1 = T(i)-U(i)	e1 = T(i)-U(i)
	ė2 = ė2 + 2.0*e1*ė1
e2 = e2 + e1*e1	e2 = e2 + e1*e1
end do	end do
	ė = 0.5*ė2/SQRT(e2)
e = SQRT(e2)	e = SQRT(e2)

Example of differentiated pseudo-code

Hascoet and Pascual, *Trans. Math. Softw.* **39**, (2013).

Génération automatique du code « linéarisé »

Exemple avec l'équation de Burgers 1D :

$$\frac{\partial u}{\partial t} + \frac{\partial u^2}{\partial x} - \nu \frac{\partial^2 u}{\partial x^2} = 0$$

```
dim = 1 # Spatial Dimension

coefficients = {'nu' : 1}

varname = { 'u' : 1}

eqn = {'u' : ' 0.5 * [  u ** 2 ]_1x  \
             - nu * [ u ]_2xx '}
```

```
d1_srcu_dx_0_im1jk = q(i-1,indvars(1))**2
d1_srcu_dx_0_ip1jk = q(i+1,indvars(1))**2
d1_srcu_dx_0_ijk = -&
          0.5_wp*d1_srcu_dx_0_im1jk+&
          0.5_wp*d1_srcu_dx_0_ip1jk
d1_srcu_dx_0_ijk = d1_srcu_dx_0_ijk*param_float(1)
d2_srcu_dx_0_ijk_im1jk = q(i-1,indvars(1))
d2_srcu_dx_0_ijk_ip0jk = q(i+0,indvars(1))
d2_srcu_dx_0_ijk_ip1jk = q(i+1,indvars(1))
d2_srcu_dx_0_ijk = 1.0_wp*d2_srcu_dx_0_ijk_im1jk-&
          2.0_wp*d2_srcu_dx_0_ijk_ip0jk+&
          1.0_wp*d2_srcu_dx_0_ijk_ip1jk
d2_srcu_dx_0_ijk = d2_srcu_dx_0_ijk*param_float(1)*param_float(1)

rhs(i,indvars(1)) =   -  ( 0.5*d1_srcu_dx_0_ijk-&
                    param_float(1 + 5)*d2_srcu_dx_0_ijk )
```

Mathematical expression

Symbolic Python

FORTRAN generated scheme

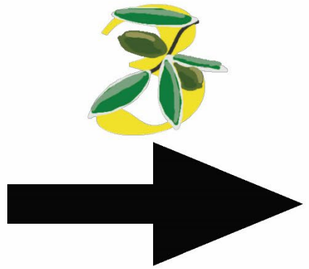
Génération automatique du code « linéarisé »

Exemple avec l'équation de Burgers 1D :

```
d1_srcu_dx_0_im1jk = q(i-1,indvars(1))**2
d1_srcu_dx_0_ip1jk = q(i+1,indvars(1))**2
d1_srcu_dx_0_ijk = -&
          0.5_wp*d1_srcu_dx_0_im1jk+&
          0.5_wp*d1_srcu_dx_0_ip1jk
d1_srcu_dx_0_ijk = d1_srcu_dx_0_ijk*param_float(1)
d2_srcu_dx_0_ijk_im1jk = q(i-1,indvars(1))
d2_srcu_dx_0_ijk_ip0jk = q(i+0,indvars(1))
d2_srcu_dx_0_ijk_ip1jk = q(i+1,indvars(1))
d2_srcu_dx_0_ijk = 1.0_wp*d2_srcu_dx_0_ijk_im1jk-&
          2.0_wp*d2_srcu_dx_0_ijk_ip0jk+&
          1.0_wp*d2_srcu_dx_0_ijk_ip1jk
d2_srcu_dx_0_ijk = d2_srcu_dx_0_ijk*param_float(1)*param_float(1)

rhs(i,indvars(1)) =    -  ( 0.5*d1_srcu_dx_0_ijk-&
                    param_float(1 + 5)*d2_srcu_dx_0_ijk )
```

FORTRAN generated scheme

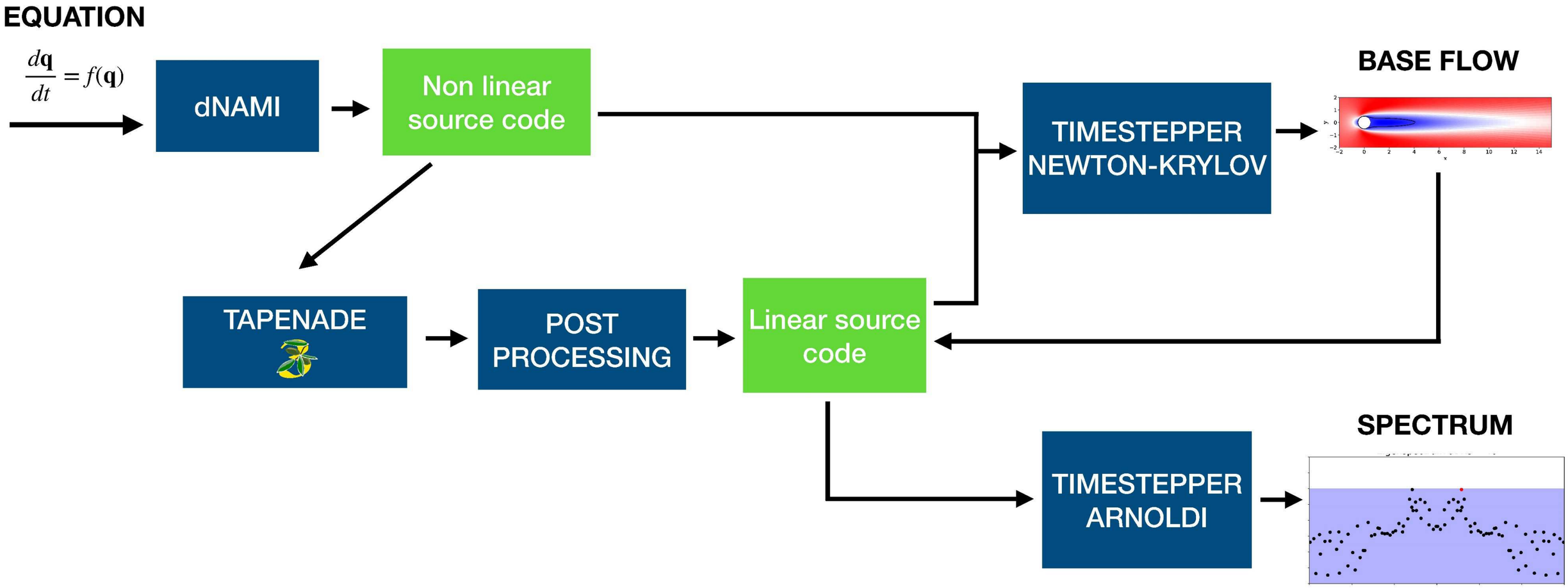


```
d1_srcu_dx_0_im1jkd = 2*q(i-1, indvars(1))*qd(i-1, indvars(1))
d1_srcu_dx_0_ip1jkd = 2*q(i+1, indvars(1))*qd(i+1, indvars(1))
d1_srcu_dx_0_ijkd = 0.5_wp*d1_srcu_dx_0_ip1jkd -
                    0.5_wp*d1_srcu_dx_0_im1jkd
d1_srcu_dx_0_ijkd = param_float(1)*d1_srcu_dx_0_ijkd
d2_srcu_dx_0_ijk_im1jkd = qd(i-1, indvars(1))
d2_srcu_dx_0_ijk_ip0jkd = qd(i+0, indvars(1))
d2_srcu_dx_0_ijk_ip1jkd = qd(i+1, indvars(1))
d2_srcu_dx_0_ijkd = d2_srcu_dx_0_ijk_im1jkd -
                  2.0_wp*d2_srcu_dx_0_ijk_ip0jkd
                  + d2_srcu_dx_0_ijk_ip1jkd
d2_srcu_dx_0_ijkd = param_float(1)**2*d2_srcu_dx_0_ijkd

rhsd(i, indvars(1)) = param_float(1+5)*d2_srcu_dx_0_ijkd
                      - 0.5*d1_srcu_dx_0_ijkd
```

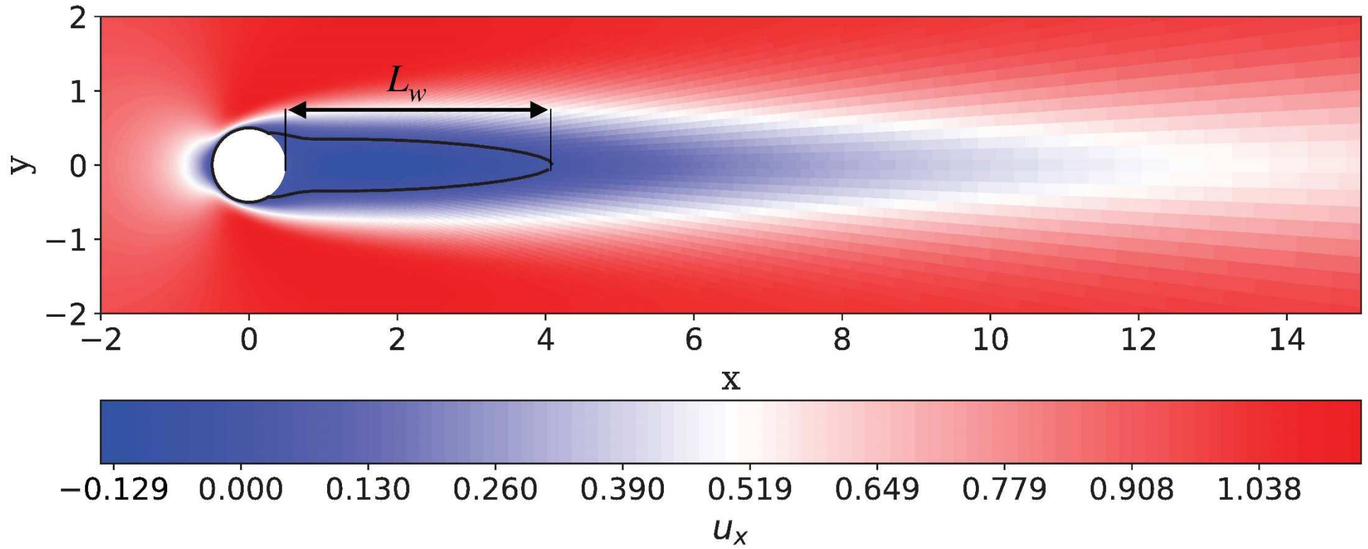
FORTRAN differentiated code

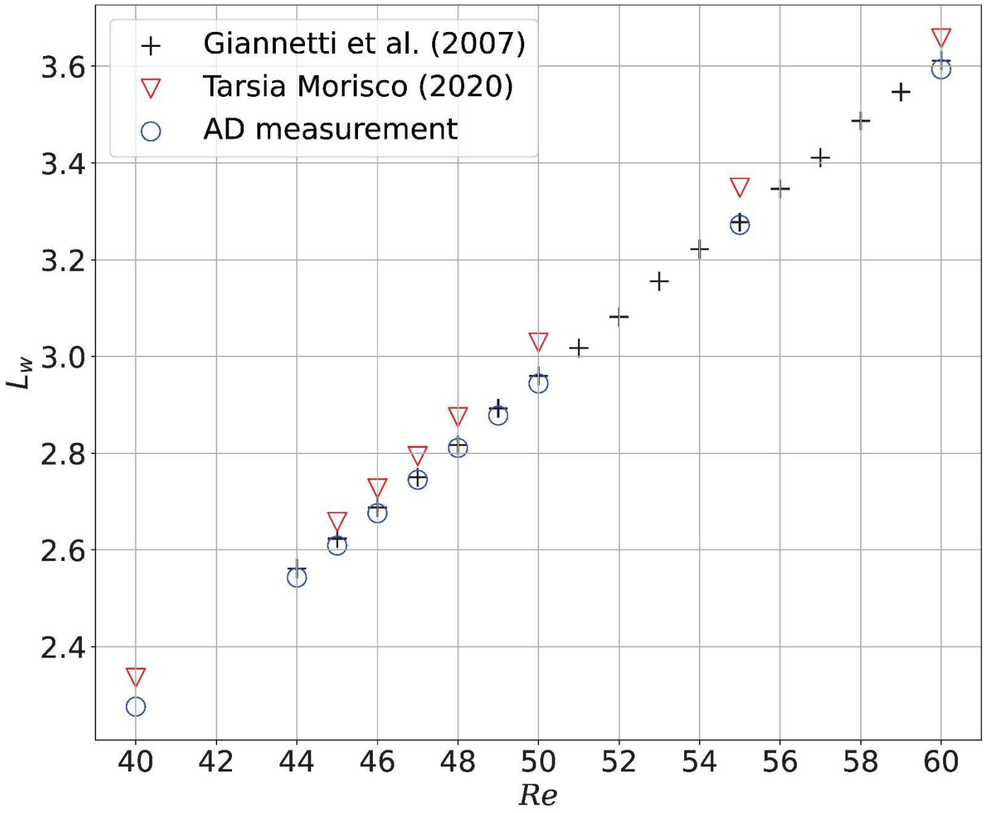
Application aux calculs des modes de stabilité globale



Application aux calculs des modes de stabilité globale

Validation : modes globaux derrière un cylindre 2D





Calcul du point fixe

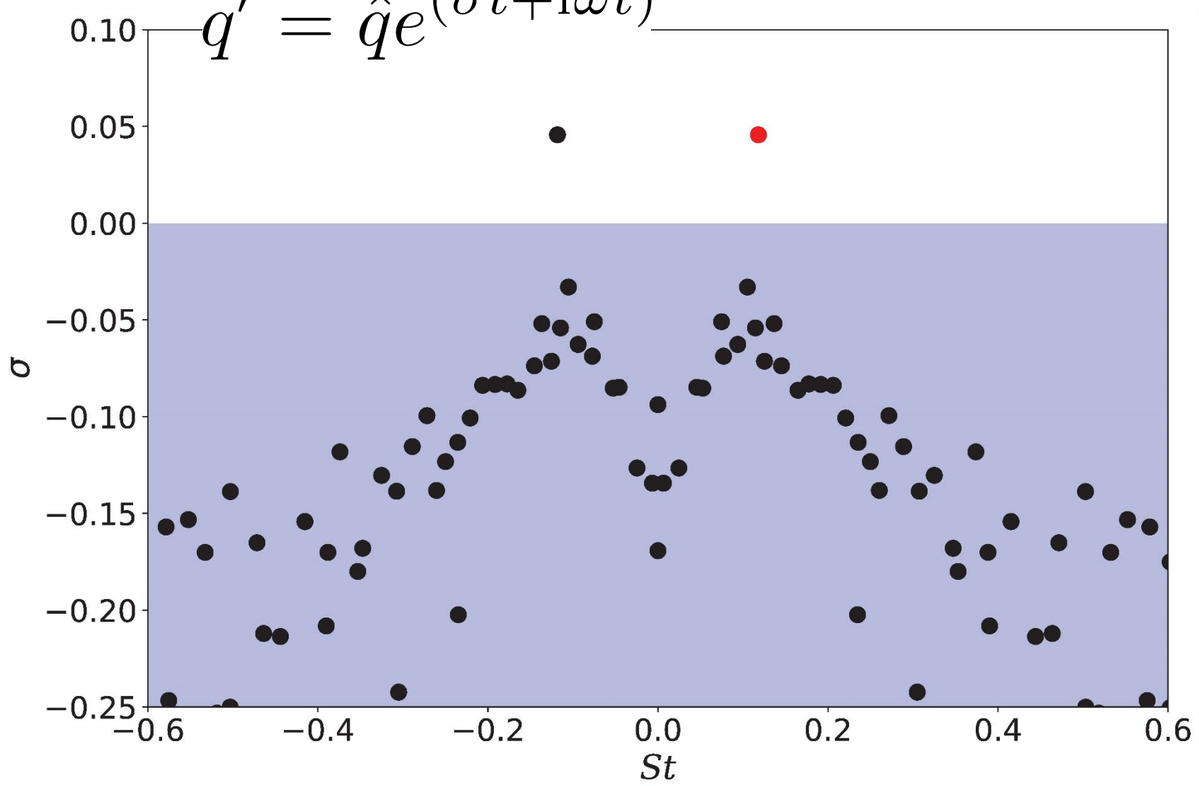
GIANNETTI, F., & LUCHINI, P. (2007). Structural sensitivity of the first instability of the cylinder wake.

COSIMO TARSIA MORISCO. (2020) Nonlinear dynamics and linear stability analysis of over-expanded nozzle flows.

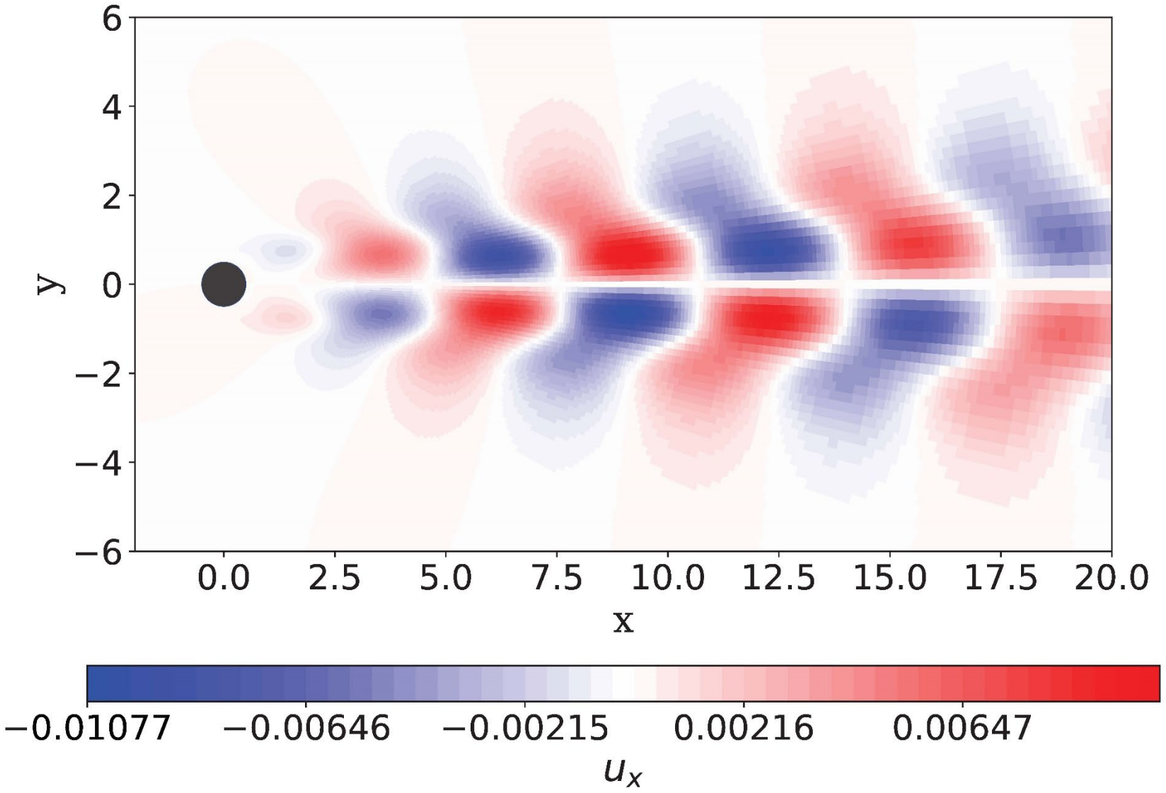
Application aux calculs des modes de stabilité globale

Validation : modes globaux derrière un cylindre 2D

$$q' = \hat{q}e^{(\sigma t + \mathrm{i}\omega t)}$$



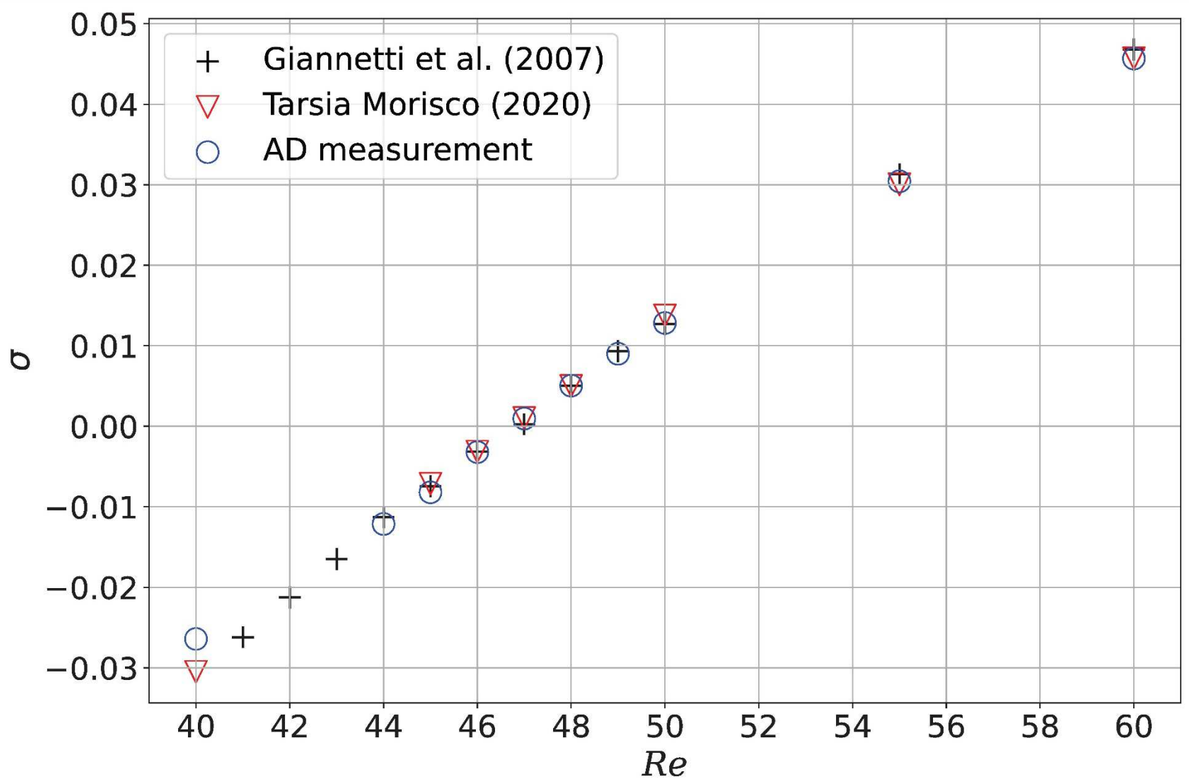
Single unstable mode at $(\sigma, St) = (0.030, 0.119)$ at $Re = 60$



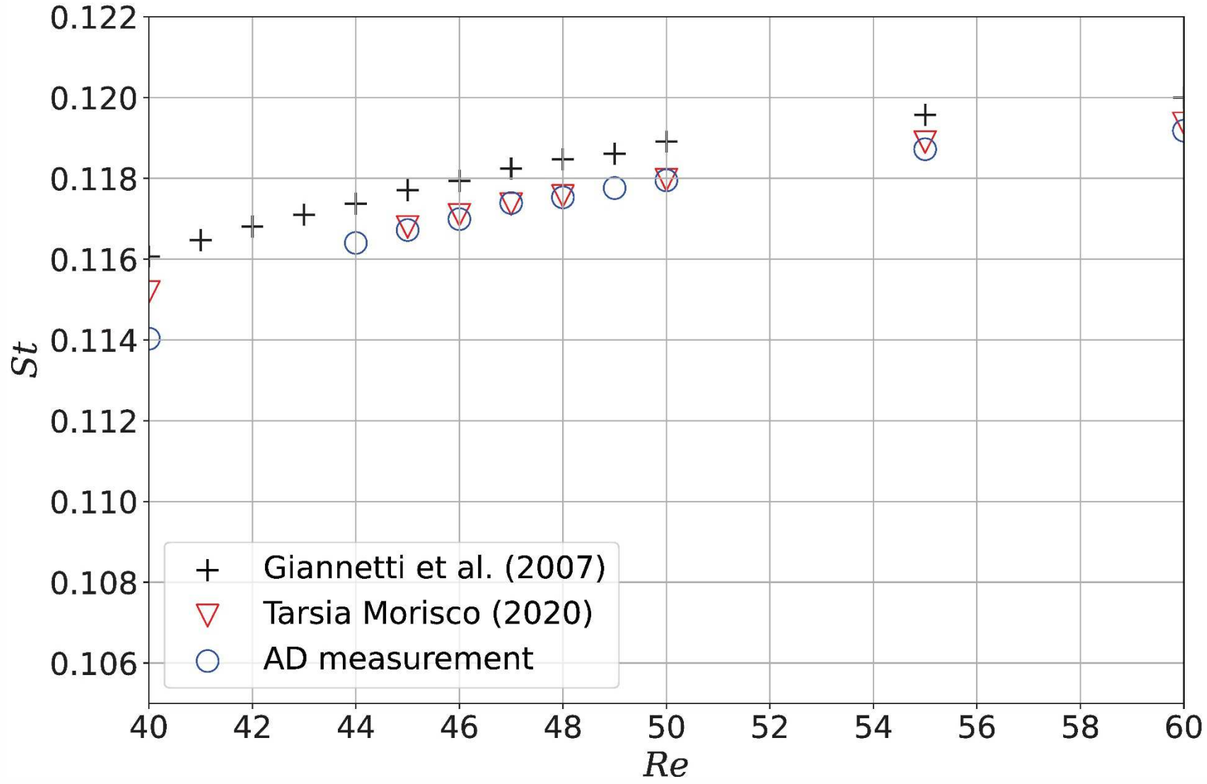
Streamwise velocity component mode u_x for global unstable mode

Application aux calculs des modes de stabilité globale

Validation : modes globaux derrière un cylindre 2D



Growth rate (σ) for different Reynolds numbers: $Re_c \in (46,47)$

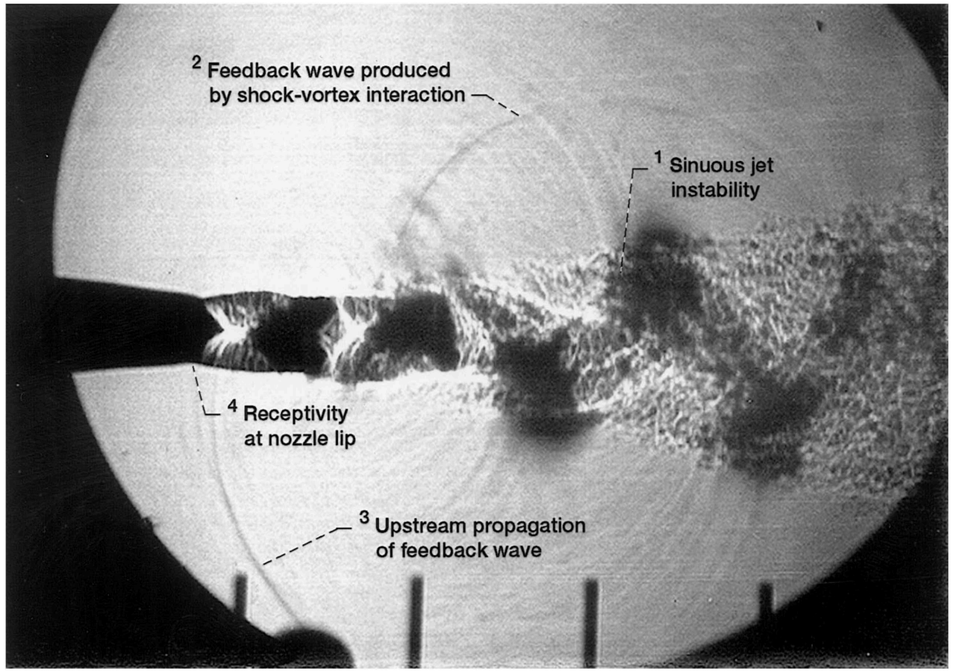


Strouhal number (St) for different Reynolds numbers (Re)

Application aux calculs des modes de stabilité globale

Étude de l'acoustique de jet sous-détendu par des méthodes de stabilités : « screech »

- Upstream acoustic wave in supersonic under expanded jets
- Very loud discrete frequency
- Can cause fatigue and damage aircrafts



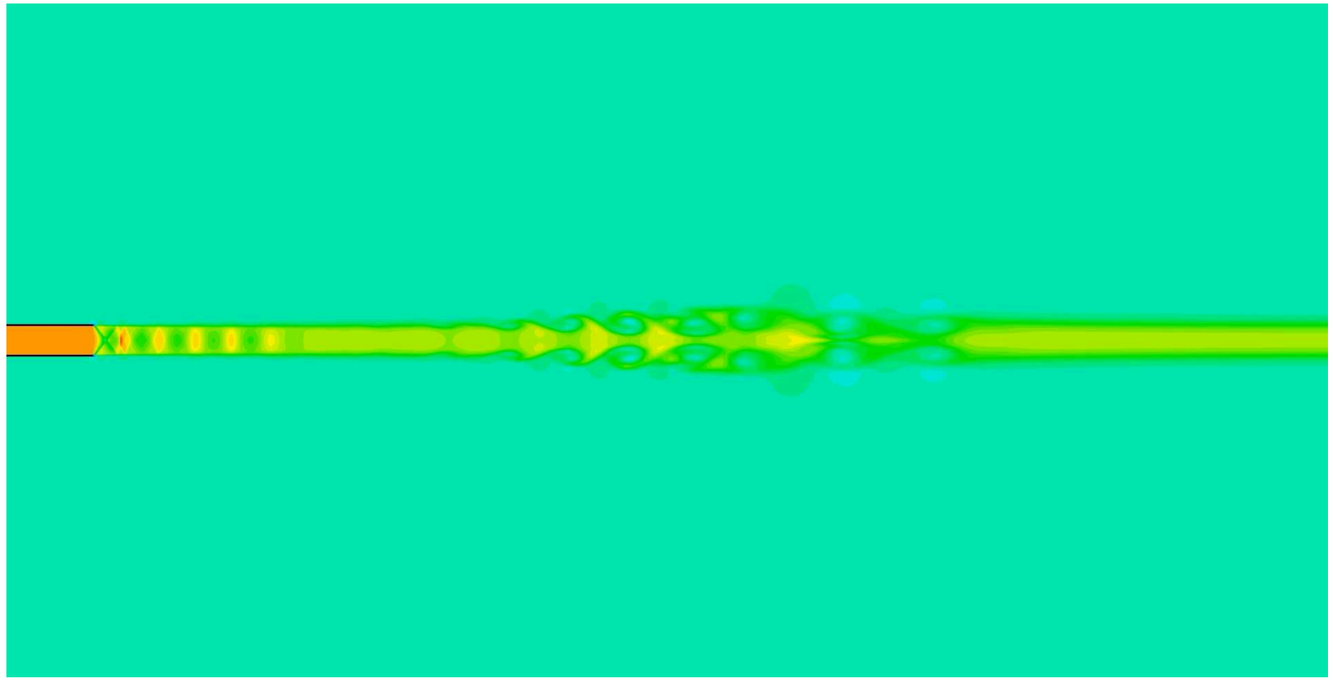
Ganesh Raman, **Advances in understanding supersonic jet screech: review and perspective**, Progress in Aerospace Sciences, Volume 34, Issues 1–2,

Application aux calculs des modes de stabilité globale

Étude de l'acoustique de jet sous-détendu par des méthodes de stabilités : « screech »

Calcul du champ de base :

- Timestepper Newton-Krylov method
- Convergence : $\epsilon < 10^{-6}$



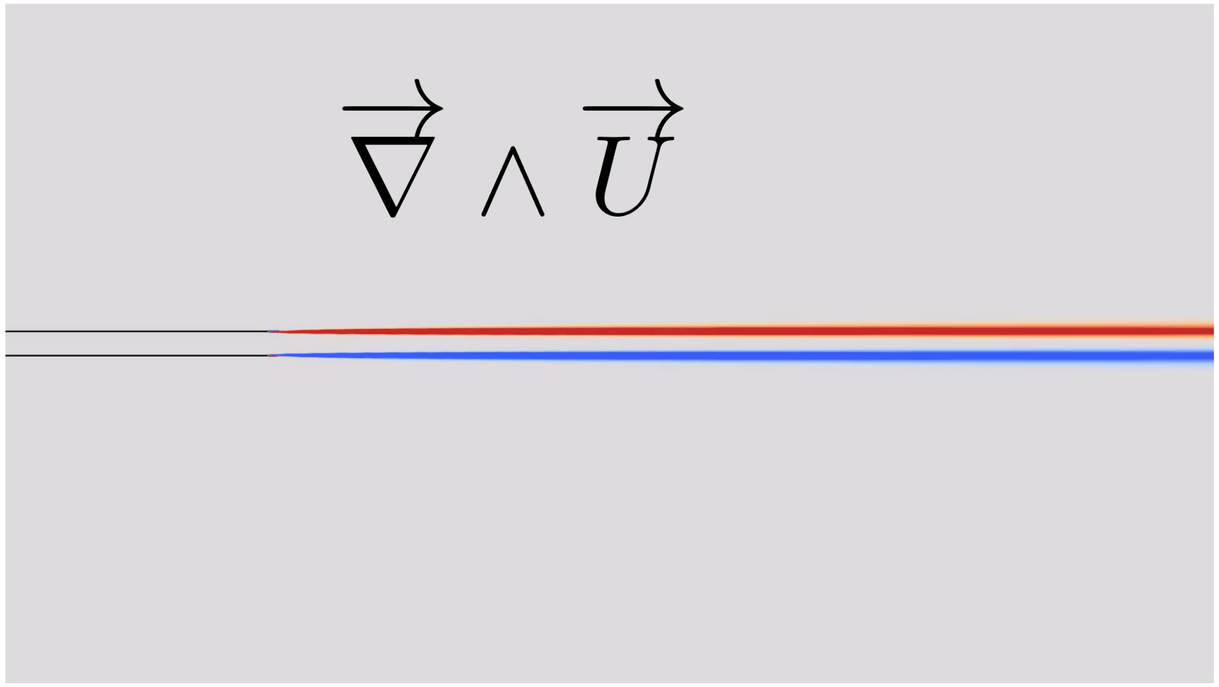
Sequence of images showing the convergence of a Newton method starting from an unsteady solution

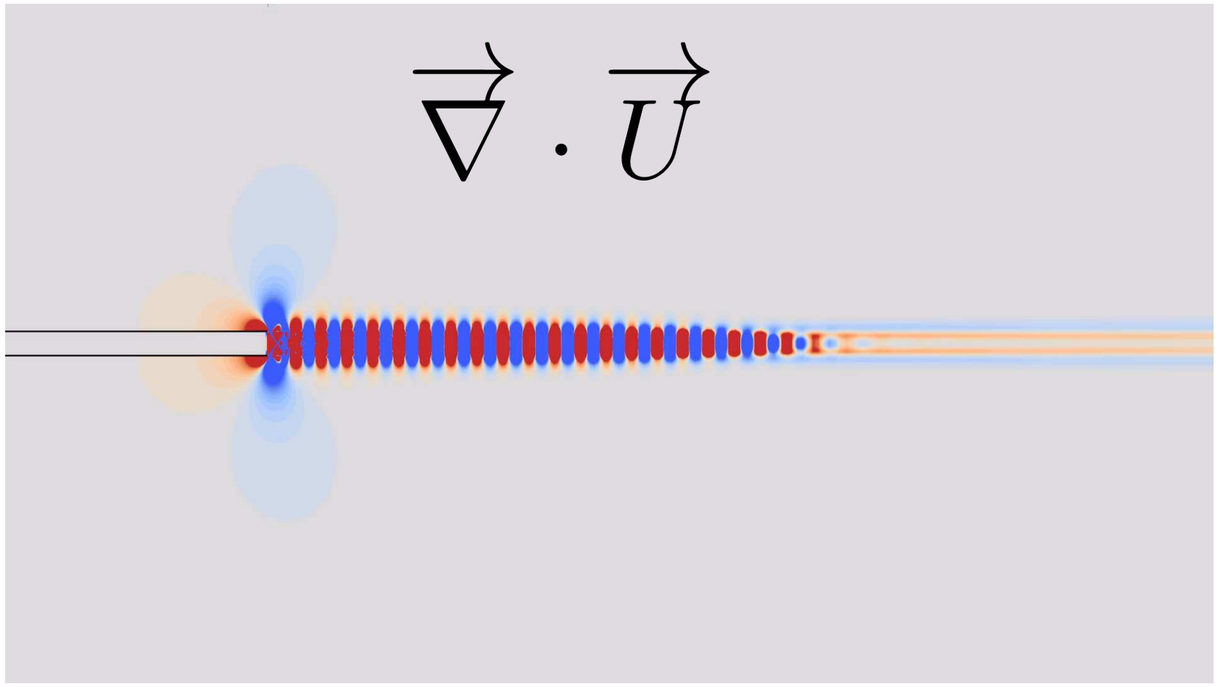
Frantz, R. , Loiseau, J.C. & Robinet, J.C.. (2023) Applied Mechanics Reviews

Application aux calculs des modes de stabilité globale

Étude de l'acoustique de jet sous-détendu par des méthodes de stabilités : « screech »

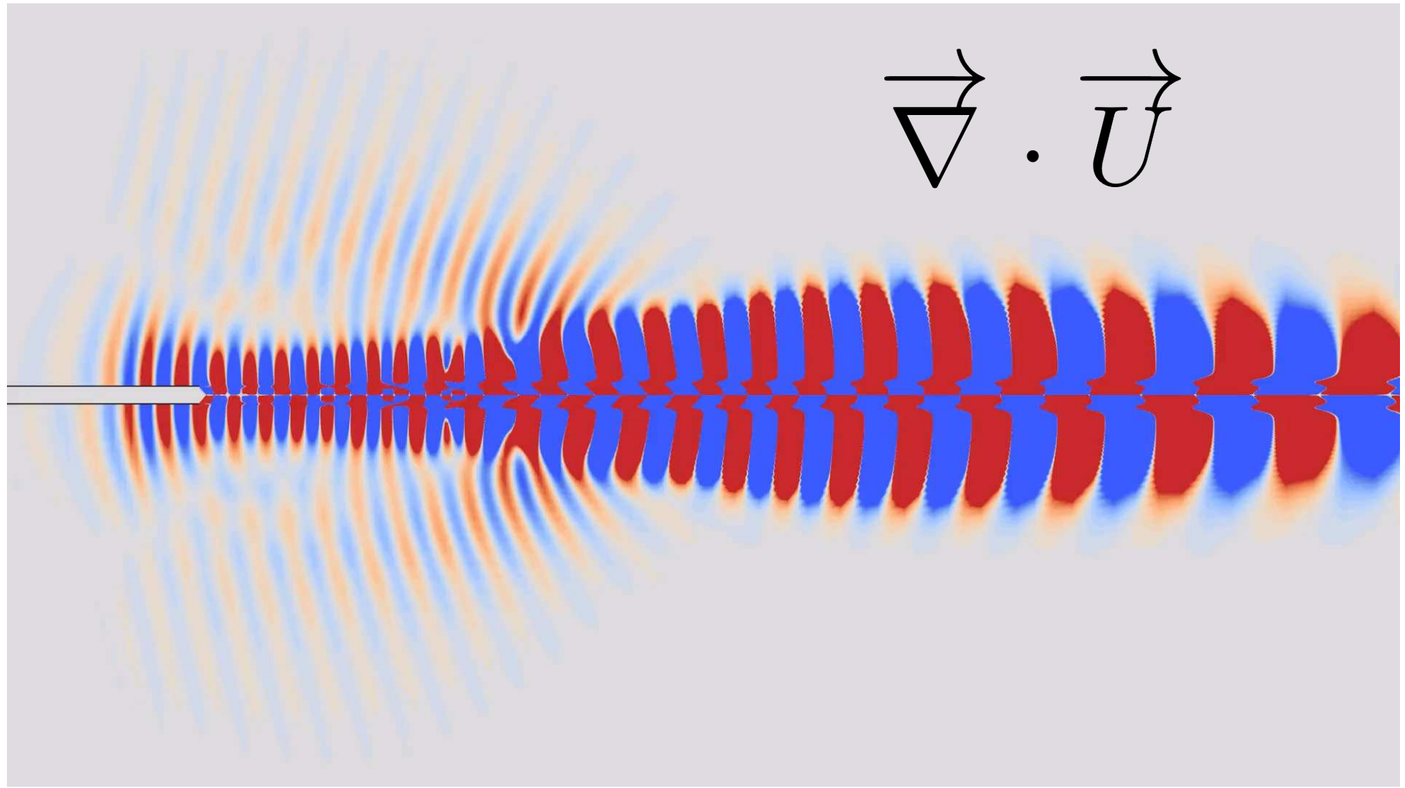
Solution non linéaire





Application aux calculs des modes de stabilité globale

Étude de l'acoustique de jet sous-détendu par des méthodes de stabilités : « screech »



The time evolution of the most unstable global mode captures « screech » emission

Merci pour votre attention