Python multi-domaines

Loïc Gouarin

Université Paris 13, Villetaneuse, laboratoire LAGA

4 décembre 2008

Plan

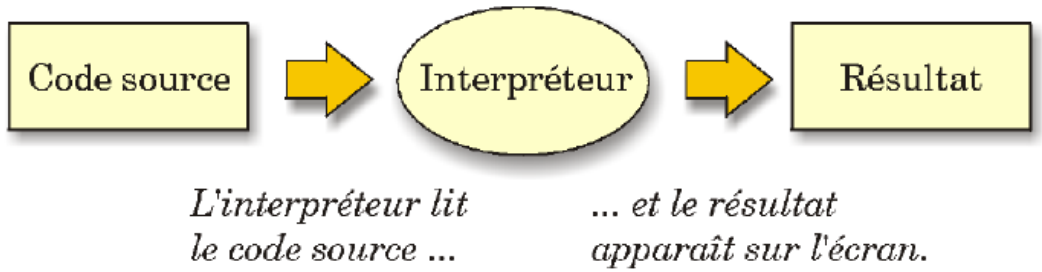
Le langage Python

1. développé en 1989 par Guido van Rossum
2. open-source
3. portable
4. orienté objet
5. dynamique
6. extensible
7. support pour l'intégration d'autres langages

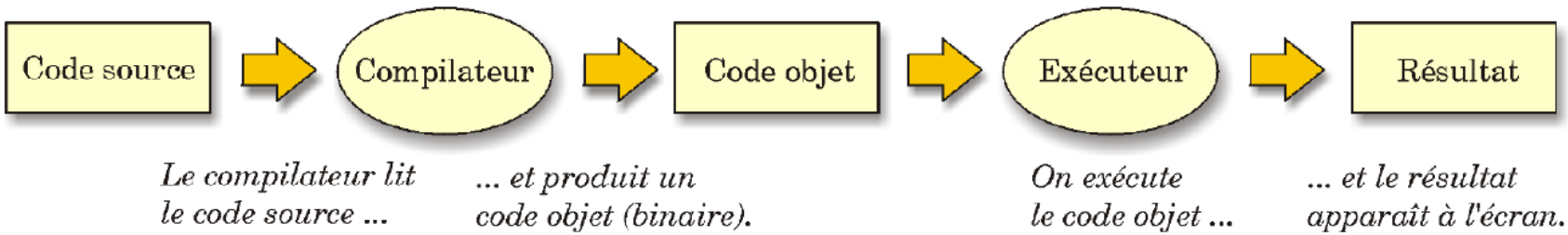
Comment faire fonctionner mon code source ?

Il existe 2 techniques principales pour effectuer la traduction en langage machine de mon code source :

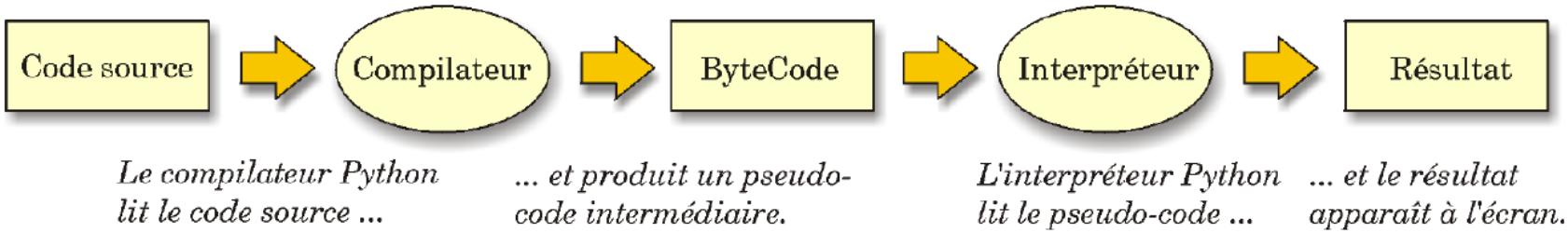
- Interprétation



- Compilation



Et Python ?

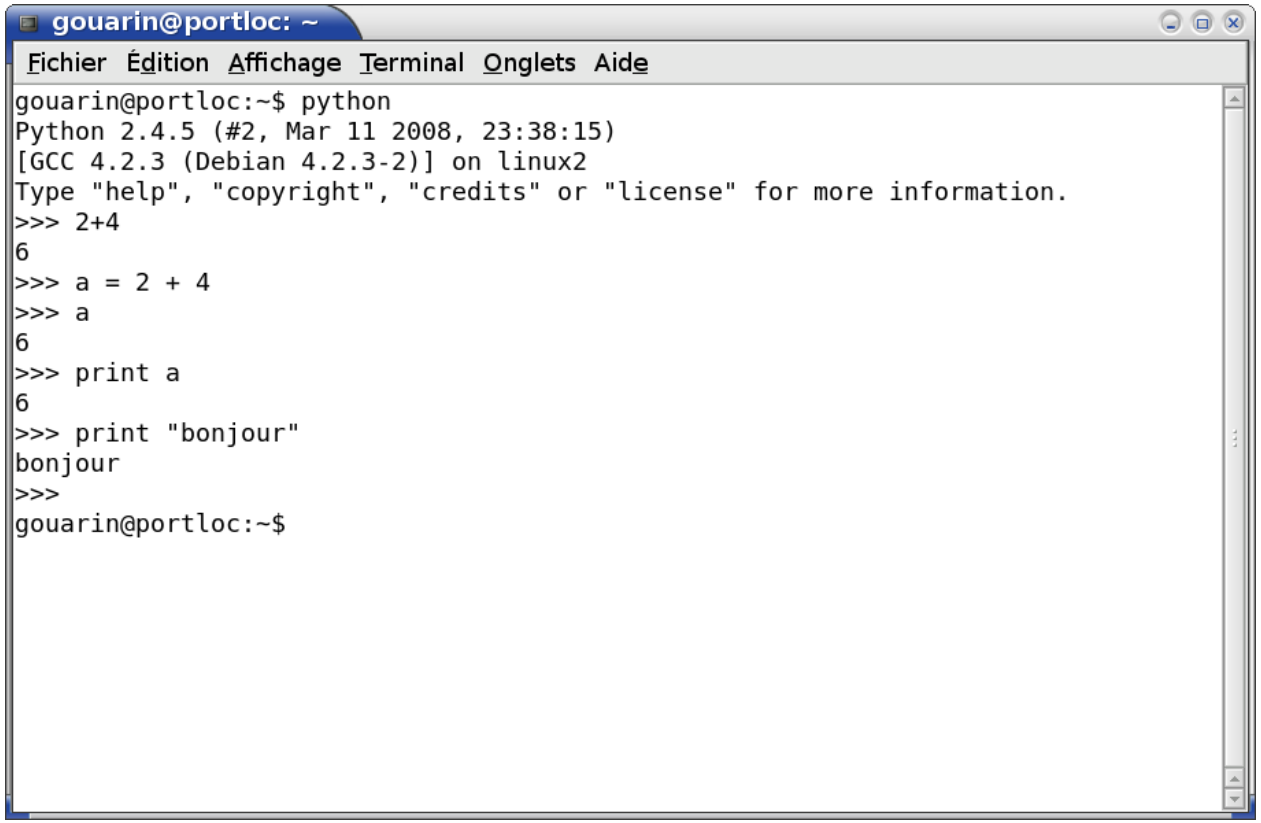


Avantages :

- interpréteur permettant de tester n'importe quel petit bout de code,
- compilation transparentes,
- interprétation du *bytecode* satisfaisante pour un bon nombre de programmes,
- *bytecode* portable.

L'interpréteur

Sous Linux



Que peut-on faire avec Python ?

- **web**
  Django, TurboGears, Zope, Plone, ...
- **bases de données**
  MySQL, PostgrSQL, Oracle, ...
- **réseaux**
  TwistedMatrix, PyRO, ...
- **Gui**
  Gtk, Qt, Tcl/Tk, WxWidgets
- **représentation graphique**
  gnuplot, matplotlib, VTK, ...
- **calcul scientifique**
  numpy, scipy, sage, ...
- **...**

Pourquoi utiliser Python pour le calcul scientifique ?

- peut être appris en quelques jours
- permet de faire des tests rapides
- alternative à Matlab, Octave, Scilab, ...
- parallélisation (MPI, PVM)
- tourne sur la plupart des plateformes
- très bon support pour le calcul scientifique
- très bon garbage collector

Le module numpy

- outils performants pour la manipulation de tableaux à N dimensions
- fonctions basiques en algèbre linéaire
- fonctions basiques pour les transformées de Fourier
- outils pour intégrer du code Fortran
- outils pour intégrer du code C/C++
- outils d'installation avec support tests unitaires

Comparaison numpy, Matlab, C

on veut calculer

$u = 100 \exp\left(-100(x-0.5)^2\right)$ avec $x \in [0,1]$.

en Python

```
from numpy import *
x = linspace(0.,1.,100)
u=100.*exp(-100.*(x-.5)**2)
```

en Matlab

```
x = linspace(0.,1.,100)
u=100.*exp(-100.*(x-.5).^2)
```

Comparaison numpy, Matlab, C

en C

```
#include <stdio.h>
#include <math.h>
#define N 100

int main(void)
{
    int i;
    double dx=1./(N-1);
    double x=0., u[N];

    for (i=0;i<N;i++)
        {
            u[i]=100.*exp(-100.*(x-.5)*(x-.5));
            x += dx;
         }
  return 0;
}
```

Le module scipy

- reprend l'ensemble des outils de numpy
- algorithmes sur les transformées de Fourier discrètes
- outils pour l'intégration
- outils pour l'interpolation
- algèbre linéaire
- outils pour l'optimisation
- manipulation d'images
- outils pour les matrices creuses
- outils pour le traitement du signal

Résolution d'une EDP

On cherche à résoudre en multi domaines

$$\begin{cases} \partial_t u(\boldsymbol{x},t) = \Delta u(\boldsymbol{x},t) \quad \text{pour} \quad \boldsymbol{x} \in \Omega = [0,1]\times[0,1] \quad \text{et} \quad t \geq 0 \\ u(\boldsymbol{x},.) = 0 \quad \text{sur} \quad \partial\Omega \quad \text{et} \quad t \geq 0 \\ u(\boldsymbol{x},0) = 100e^{-100\left[(x-0.5)^2+(y-0.5)^2\right]} \quad \text{pour} \quad \boldsymbol{x} \in \Omega \end{cases}$$

De quoi a-t-on besoin pour résoudre l'EDP en parallèle ?

1. savoir résoudre l'équation sur un domaine
   - quel schéma spatial utiliser ?
   - quel schéma en temps utiliser ?
   - quel solveur utiliser ?
2. outils de communications pour la parallélisation
3. savoir quoi faire entre les sous-domaines
   - quelles méthodes utiliser ?
   - quelles données échanger ?

Résolution du problème mono domaine

Le cas explicite

- Euler explicite en temps
- Schéma aux différences finies centré à 5 points

Le cas implicite

- Euler implicite en temps
- Schéma aux différences finies centré à 5 points

Shéma à l'intérieur

Euler explicite + différences finies

$$u_{i,j}^{n+1} = u_{i,j}^{n} + \frac{\Delta t}{\Delta x^2}\left(u_{i-1,j}^{n} - 2u_{i,j}^{n} + u_{i+1,j}^{n}\right) + \frac{\Delta t}{\Delta y^2}\left(u_{i,j-1}^{n} - 2u_{i,j}^{n} + u_{i,j+1}^{n}\right)$$

Remarque : pas besoin de stocker de matrice

Comment calculer u^{n+1} avec Python ?

Pur Python

```
def chaleurWithLoop(up,u,dx,dy,dt,nu):
    nx=u.shape[0]
    ny=u.shape[1]

    cx=dt*nu/dx/dx
    cy=dt*nu/dy/dy

    for i in xrange(1,nx-1):
        for j in xrange(1,ny-1):
            up[i,j]=u[i,j] \
                    +cx*(u[i+1,j]-2.*u[i,j]+u[i-1,j]) \
                    +cy*(u[i,j+1]-2.*u[i,j]+u[i,j-1])
```

temps CPU normalisé : 1058

Python et numpy

```
def chaleurVectorized(up,u,dx,dy,dt,nu):
    nx=u.shape[0]
    ny=u.shape[1]

    cx=dt*nu/dx/dx
    cy=dt*nu/dy/dy

    up[1:-1,1:-1]=u[1:-1,1:-1] \
          + cx*(u[2:,1:-1]-2.*u[1:-1,1:-1]+u[:-2,1:-1]) \
          + cy*(u[1:-1,2:]-2.*u[1:-1,1:-1]+u[1:-1,:-2])
```

temps CPU normalisé : 13

Python et f2py

```
SUBROUTINE chaleurFortran(up, u, nx, ny, dx, dy, dt, nu)
  implicit none
  integer :: nx, ny, i, j
  real*8  :: up(0:nx, 0:ny), u(0:nx, 0:ny)
  real*8  :: dx, dy, dt, nu
  real*8  :: cx, cy
!f2py intent(in) u
!f2py intent(in, out) up

  cx=dt*nu/dx/dx
  cy=dt*nu/dy/dy

  DO j=1,nx-1
     DO i=1,ny-1
        up(i,j)=u(i,j) &
             +cx*(u(i+1,j)-2.*u(i,j)+u(i-1,j)) &
             +cy*(u(i,j+1)-2.*u(i,j)+u(i,j-1))
     END DO
  END DO

END SUBROUTINE chaleurFortran
```

temps CPU normalisé : 1.0

Python et C API

- C API permet de passer des objets Python à un code C ou C++
- la documentation est fournie avec Python
- vous faites tout vous même
- vous pouvez faire beaucoup plus qu'avec f2py

temps CPU normalisé : 1.0

Autres possibilités d'optimisation

- psyco
- weave
- swig
- programme C + f2py

Code mono domaine

```
from numpy import *
from chaleurC import chaleurC
from chaleur import bc

nx, ny, nbStep, nu = 101, 101, 100, 10.
dx = 1./(nx-1)
dy = 1./(ny-1)
dt = dx**2/(4.*nu)

X,Y = meshgrid(linspace(0.,1.,nx), linspace(0.,1.,ny))

u = 100.*exp(-100.*((X-.5)**2+(Y-.5)**2))
up = u.copy()

for t in xrange(nbStep):
    up = chaleurC(up,u,dx,dy,dt,nu)
    bc([1,1,1,1],up)
    u[:] = up[:]
```

Quels sont les outils pour la parallélisation ?

1. Threads
2. PVM
   - pypvm
   - pynpvm
3. MPI
   - pyPar
   - pyMPI
   - mpi4py
4. openMP ?
5. GPU
   - pystream

MPI et python

Un exemple simple

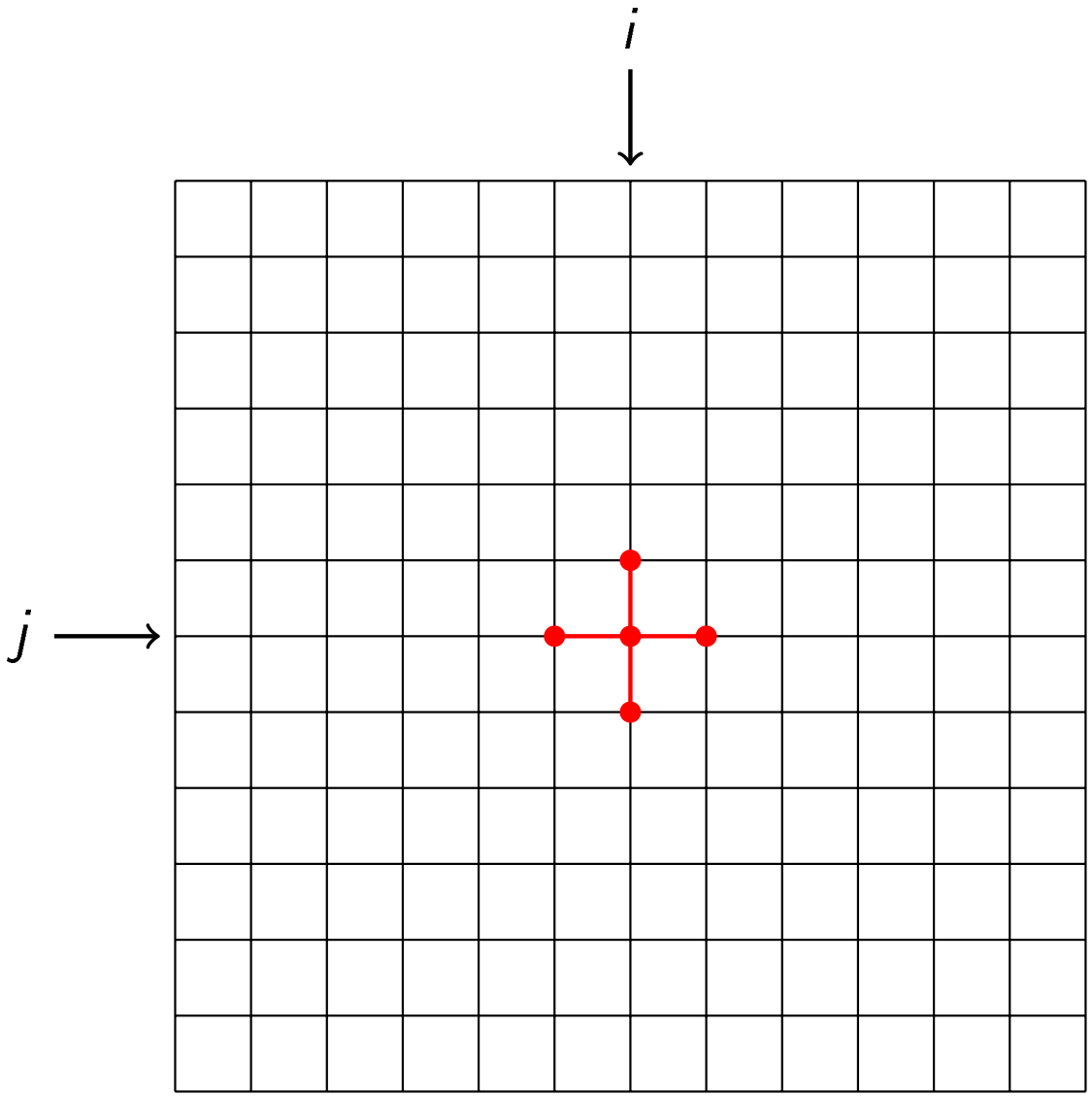
```
import mpi4py.MPI as mpi
from numpy import array

class point:
    def __init__(self,id,x,y):
        self.id = id
        self.x = x
        self.y = y

    def __str__(self):
        s = "coordonnees du point "+str(self.id)+" :\n"
        s += "x : "+str(self.x)+" , y : "+str(self.y)+"\n"
        return s
```

MPI et python

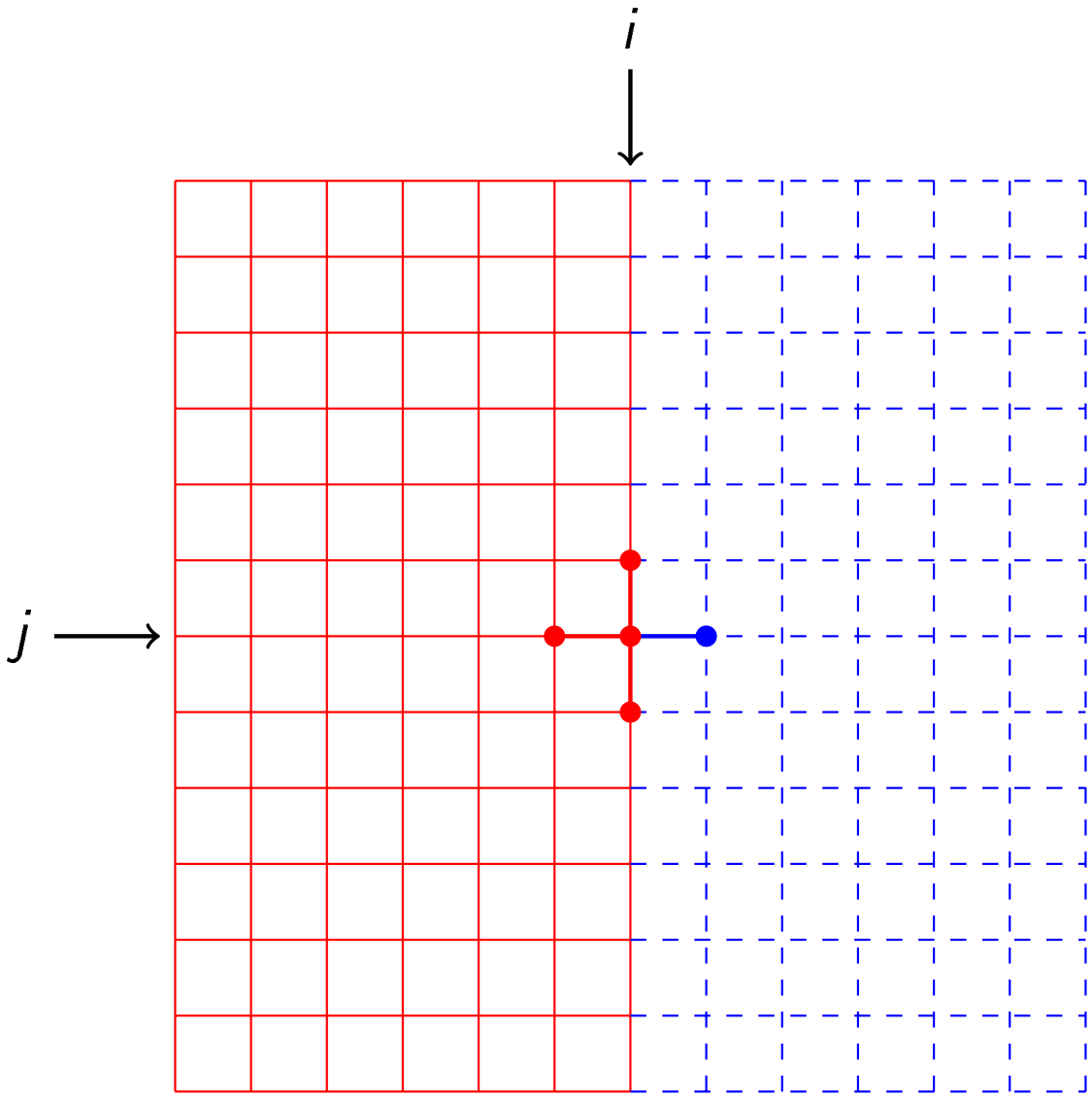
Un exemple simple(suite)

```
id = mpi.COMM_WORLD.rank

if id==0:
    sendValues = [point(1,2.,4.5), array([3,4,8]), \
                          {1:'un',2:'deux',3:'trois'}]
    mpi.COMM_WORLD.Send(sendValues, dest=1, tag=0)
else:
    recvValues = mpi.COMM_WORLD.Recv(source=0, tag=0)
    for v in recvValues:
        print v
```

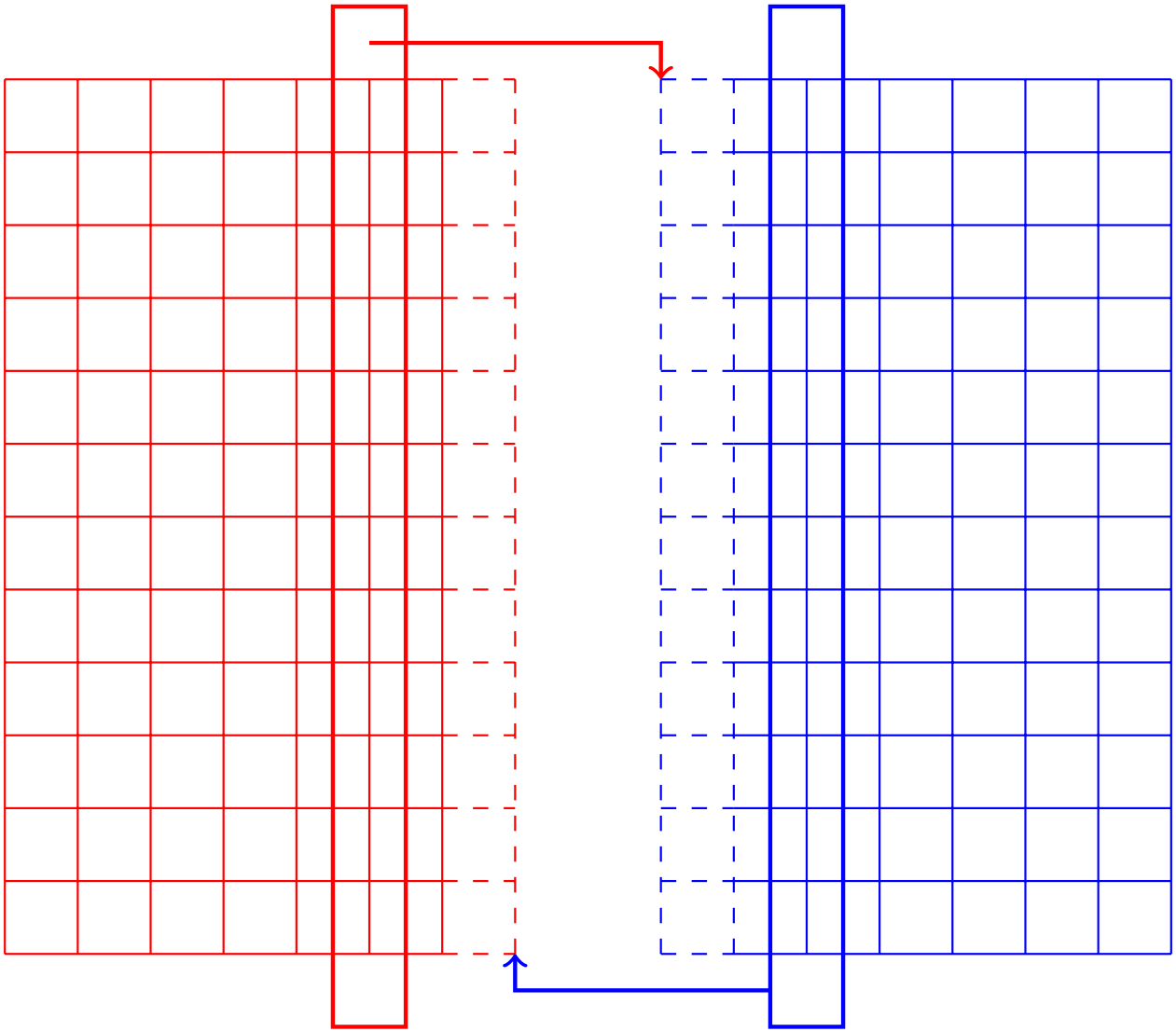
Passer au parallèle



Passer au parallèle



Passer au parallèle



En résumé

- on suppose qu'on a la solution à l'instant n
- calcul de la solution à l'instant $n+1$
- mise à jour des points fictifs par échange entre les sous-domaines

Initialisation de l'instant initial

```
id = mpi.COMM_WORLD.rank

if id == 0:
    x = linspace(0.,.5+dx,nx+1)
    y = linspace(0.,1.,ny)
    tabBC = [1,0,1,1]
else:
    x = linspace(.5-dx,1.,nx+1)
    y = linspace(0.,1.,ny)
    tabBC = [1,1,1,0]

X,Y = meshgrid(x, y)

u = 100.*exp(-100.*((X-.5)**2+(Y-.5)**2))
up = u.copy()
```

Calcul de la solution

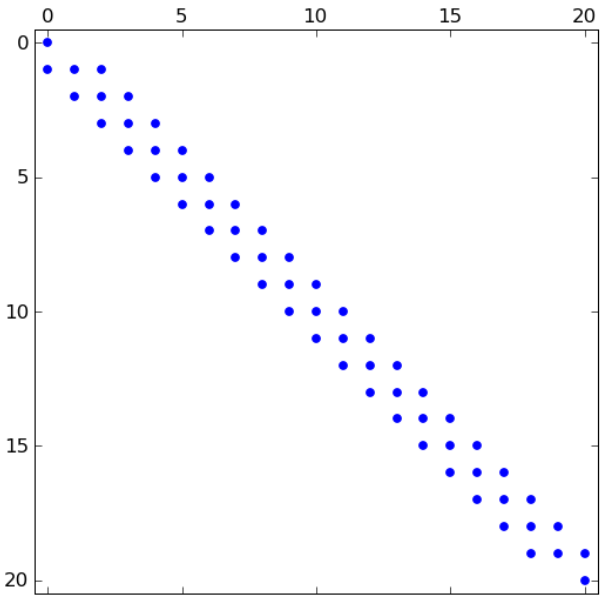
```
for t in xrange(nbStep):
    up = chaleurC(up,u,dx,dy,dt,nu)
    bc(tabBC,up)

    # actualisation de l'interface
    if id==0:
        mpi.COMM_WORLD.Send(up[:,-3],1,0)
        up[:,-1] = mpi.COMM_WORLD.Recv(None,1,1)
    else:
        up[:,0] = mpi.COMM_WORLD.Recv(None,0,0)
        mpi.COMM_WORLD.Send(up[:,2],0,1)

    u[:] = up[:]
```

Schéma à l'intérieur

Euler implicite + différences finies

$$u_i^{n+1} = u_i^n + \frac{\Delta t}{\Delta x^2}\left(u_{i-1}^{n+1} - 2u_i^{n+1} + u_{i+1}^{n+1}\right)$$

On doit résoudre $Au^{n+1} = b$ où A est de la forme



Il nous faut

1. des outils pour matrice creuse

scipy.sparse

- csc_matrix : Compressed Sparse Column
- csr_matrix : Compressed Sparse Row
- lil_matrix : List of Lists
- ...

2. des outils pour résoudre des systèmes linéaires

scipy.linsolve

basé sur le solveur SuperLU

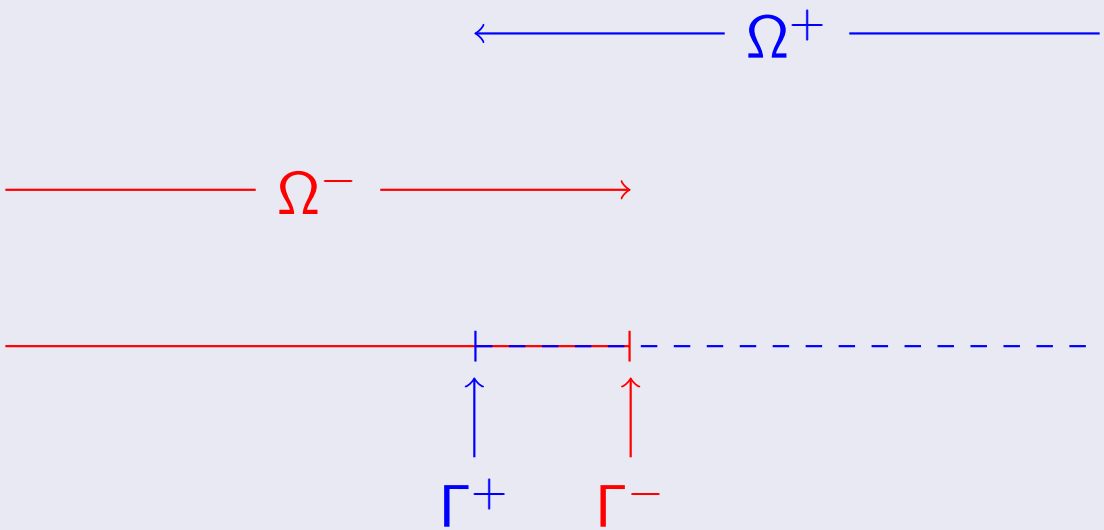
Assemblage de la matrice A et résolution

```
def heat(nx,dx,dt,nu):
    A = lil_matrix((nx,nx))
    A.setdiag((1. + 2.*dt*nu/dx**2)*ones(nx))
    T = -dt*nu*ones(nx)/dx**2
    A.setdiag(T,1)
    A.setdiag(T,-1)

# conditions de Dirichlet
...
# condition initiale
u0 = ...

A = heat(nx,dx,dt,nu)
LU = linsolve.factorized(A.tocsr())

for t in xrange(nbStep):
    u = LU(u0)
    u0[:] = u[:]
```

Parallélisation

On ne connait pas la solution du voisin à l'instant $n+1$.

Algorithme de Schwarz classique avec recouvrement



Parallélisation

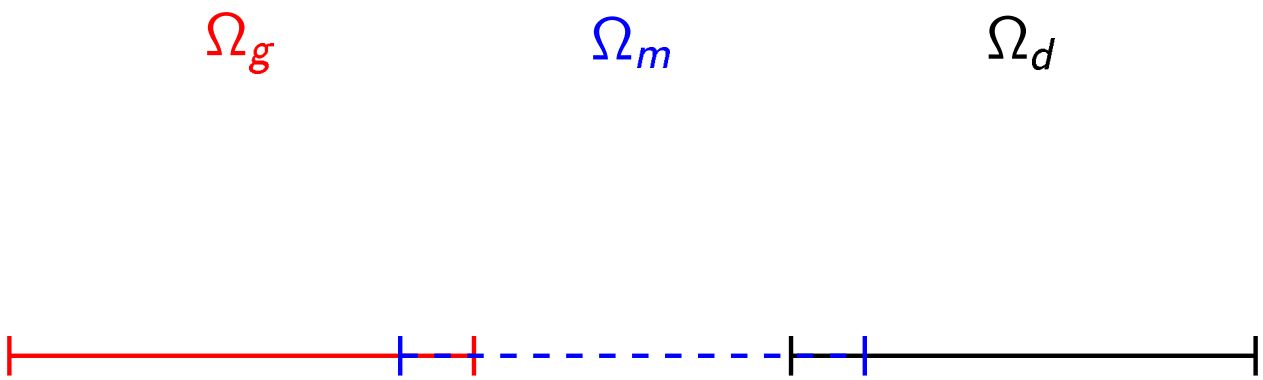
Il faut donc itérer.

Algorithme de Schwarz classique avec recouvrement

$$\begin{cases} \mathcal{L}u^{k+1} = 0 \quad \text{dans} \quad \Omega^- \quad \text{et} \quad t \geq 0, \\ u^{k+1}(.,.,0) = w_0 \quad \text{dans} \quad \Omega^-, \\ u^{k+1} = v^k \quad \text{sur} \quad \Gamma^- \quad \text{et} \quad t \geq 0. \end{cases}$$

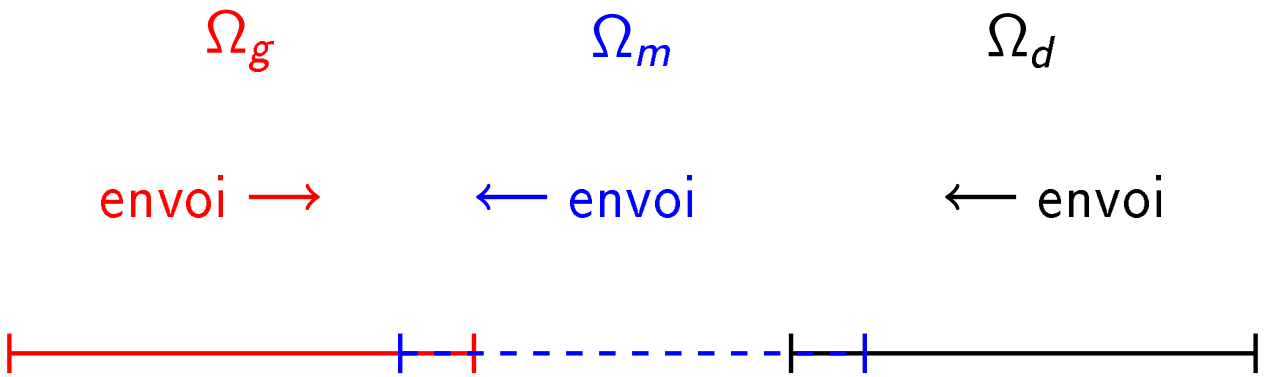
$$\begin{cases} \mathcal{L}v^{k+1} = 0 \quad \text{dans} \quad \Omega^+ \quad \text{et} \quad t \geq 0, \\ v^{k+1}(.,.,0) = w_0 \quad \text{dans} \quad \Omega^+, \\ v^{k+1} = u^k \quad \text{sur} \quad \Gamma^+ \quad \text{et} \quad t \geq 0. \end{cases}$$

Cas général : 3 cas possibles



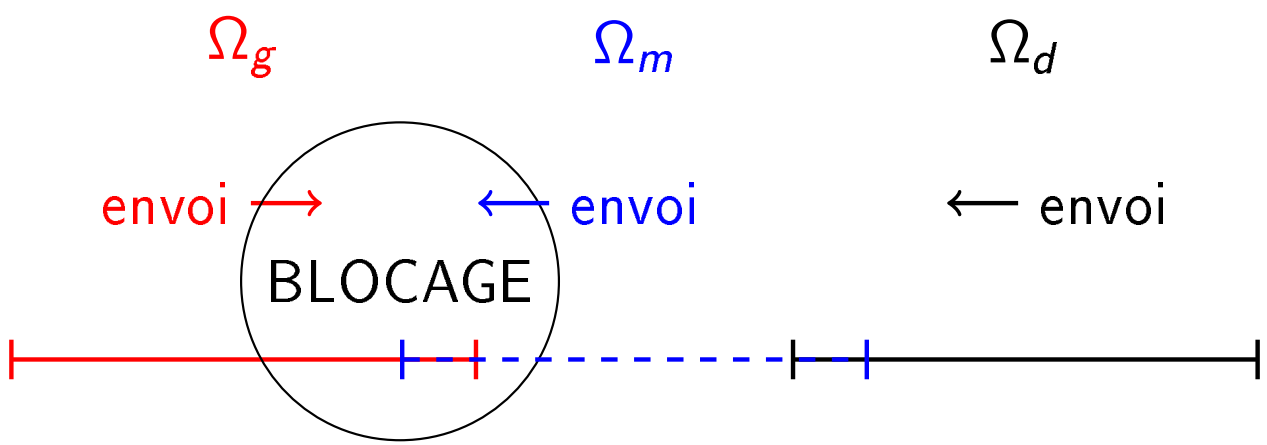
Cas général : échange des données

On ne veut utiliser que des *send* et des *recv*.



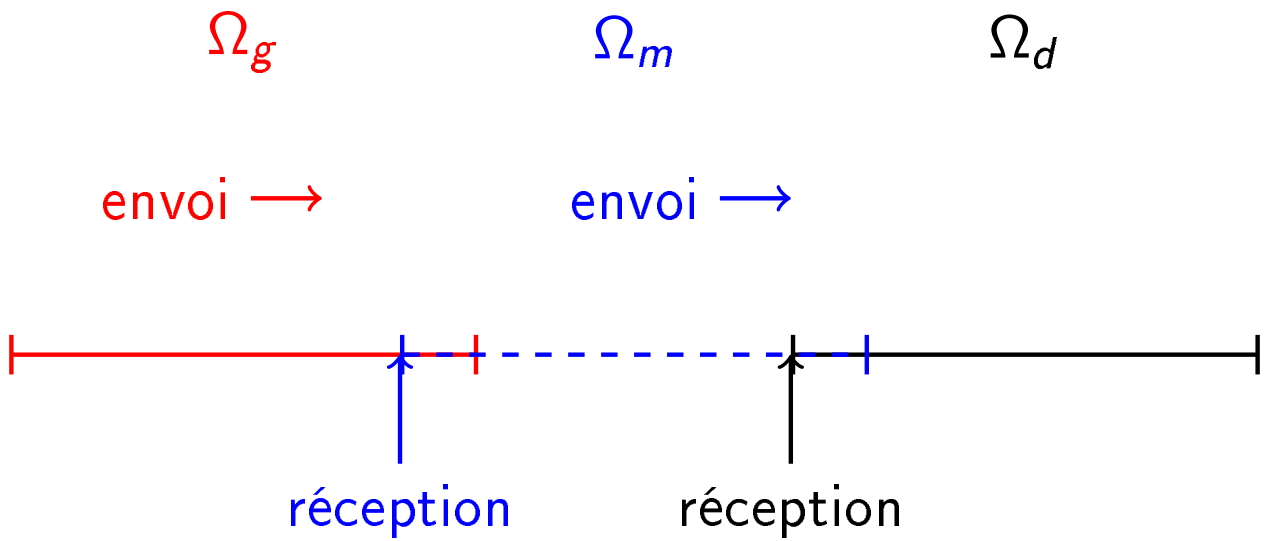
Cas général : échange des données

On ne veut utiliser que des *send* et des *recv*.



Cas général : échange des données

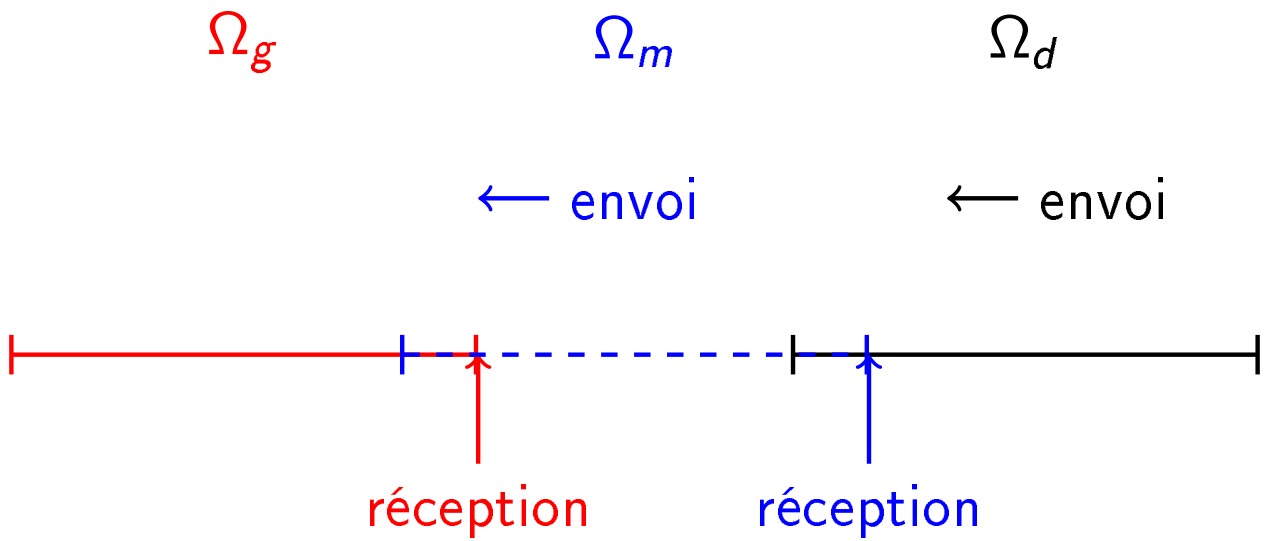
On ne veut utiliser que des *send* et des *recv*.



première étape

Cas général

On ne veut utiliser que des *send* et des *recv*.



deuxième étape

Calcul de la solution

```
for t in xrange(nbStep):
    # debut de l'algorithme de Schwarz
    for k in xrange(15):
        u = LU(u0)

        # recuperation du u interface du voisin
        if id==0: # domaine à gauche
            mpi.COMM_WORLD.Send(u[-1-2*recouv],1,0)
            uv=mpi.COMM_WORLD.Recv(None,1,1)
            u0[-1]=uv
        elif id==mpi.COMM_WORLD.size-1: # domaine à droite
            uv=mpi.COMM_WORLD.Recv(None,id-1,id-1)
            mpi.COMM_WORLD.Send(u[2*recouv],id-1,id)
            u0[0]=uv
        else: # domaine milieu
            mpi.COMM_WORLD.Send(u[-1-2*recouv],id+1,id)
            uvg=mpi.COMM_WORLD.Recv(None,id-1,id-1)

            mpi.COMM_WORLD.Send(u[2*recouv],id-1,id)
            uvd=mpi.COMM_WORLD.Recv(None,id+1,id+1)

            u0[0]=uvg
            u0[-1]=uvd

    u0[:] = u[:]
```

Code Optimism

But

Résoudre

$$\frac{\partial}{\partial t}u(\mathbf{x},t) + \nabla(M.\nabla u(\mathbf{x},t)) + p.\nabla u(\mathbf{x},t) + cu(\mathbf{x},t) = f(\mathbf{x},t)$$

$$\text{pour} \quad \mathbf{x} \in \Omega \subset \mathrm{R}^2, \quad t > 0$$

avec des conditions interfaces :

$$\alpha_1^i \frac{\partial}{\partial t}u(\mathbf{x},t) + M\nabla_{n^i}u(\mathbf{x},t) + \alpha_2^i u(\mathbf{x},t) + \alpha_3^i \nabla_{\tau^i}u(\mathbf{x},t)$$

$$+\nabla_{\tau^i}(a_{\tau^i}\nabla_{\tau^i}u)(\mathbf{x},t) = g^i(\mathbf{x},t)$$

$$\text{pour} \quad \mathbf{x} \in \Gamma_i, \quad t > 0$$

Outils utilisés

- petsc4py
- mpi4py
- swig
- VTK
- WxWidgets

Exemple d'application : équation de convection-diffusion-réaction

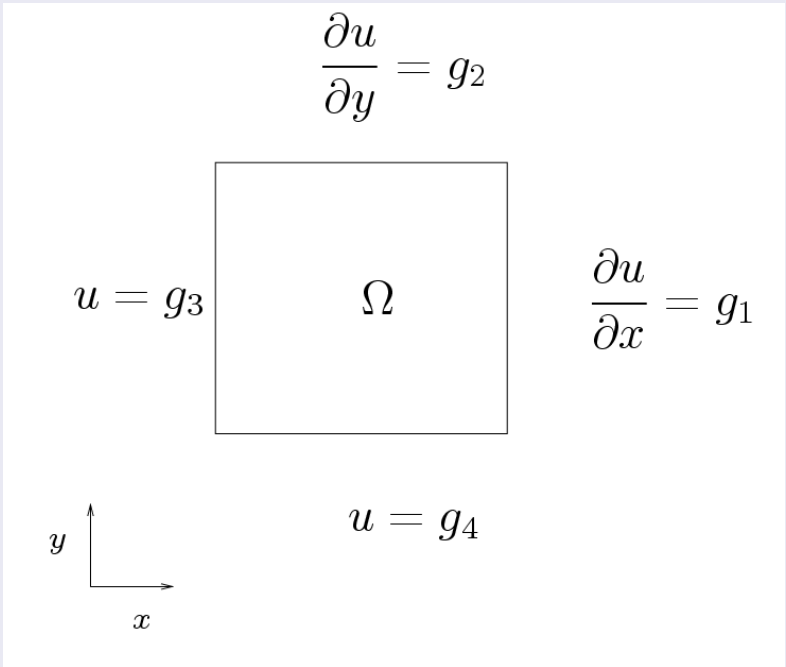
$$\begin{cases} \partial_t u - \nu \Delta u + \boldsymbol{a}.\nabla u + u = 0 \quad \text{sur} \Omega = [0,1] \times [0,1] \quad \text{et} \quad t > 0 \\ \\ u_0 = 10 exp(-100((x-0.5)^2 + (y-0.5)^2)) \end{cases}$$

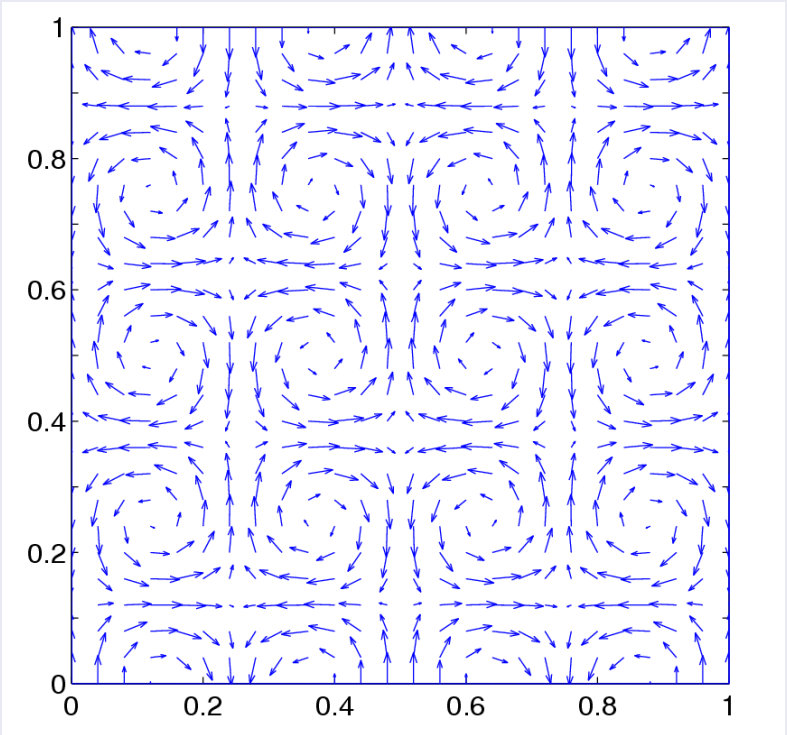
avec $\nu = 10^{-3}$ et

$$\boldsymbol{a} = \begin{pmatrix} 0.32\pi \sin(4\pi x) \sin(4\pi y) \\ 0.32\pi \cos(4\pi x) \cos(4\pi y) \end{pmatrix}$$

Exemple d'application : équation de convection-diffusion-réaction

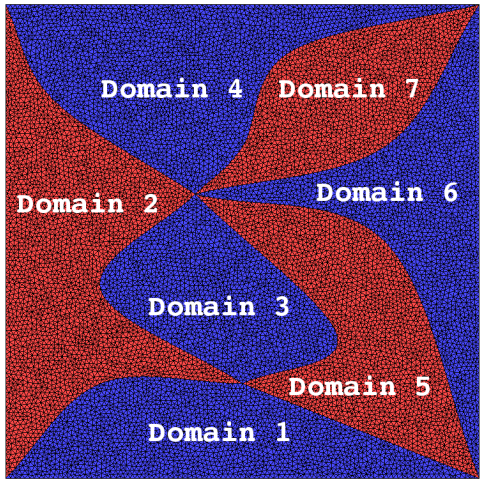
Conditions aux limites et champs de vitesse





Exemple d'application : équation de convection-diffusion-réaction

Maillage



Condition aux interfaces

$$\partial_{n_i} u_i + p_i u_i = \partial_{n_i} u_j + p_i u_j \quad \text{sur} \quad \Gamma_{ij}$$

Exemple d'application : équation de St-Venant

On souhaite résoudre sur $\Omega = [0, L_x] \times [0, Ly], t > 0$:

$$\begin{cases} \frac{\partial}{\partial t} u = fv - \frac{c^2}{H} \frac{\partial h}{\partial x} + \nu \Delta u + \frac{\tau^x}{\rho_0 H}, \\ \frac{\partial}{\partial t} v = -fu - \frac{c^2}{H} \frac{\partial h}{\partial y} + \nu \Delta v + \frac{\tau^y}{\rho_0 H}, \\ \frac{\partial}{\partial t} h = -H \left(\frac{\partial u}{\partial y} + \frac{\partial v}{\partial y} \right) \end{cases}$$

avec $L_x = 15000$ km, $L_y = 3000$ km, $\tau_0 = 5.10^{-2} N/m^2$,$x_0 = 3000$ km, $L = 300$ km et

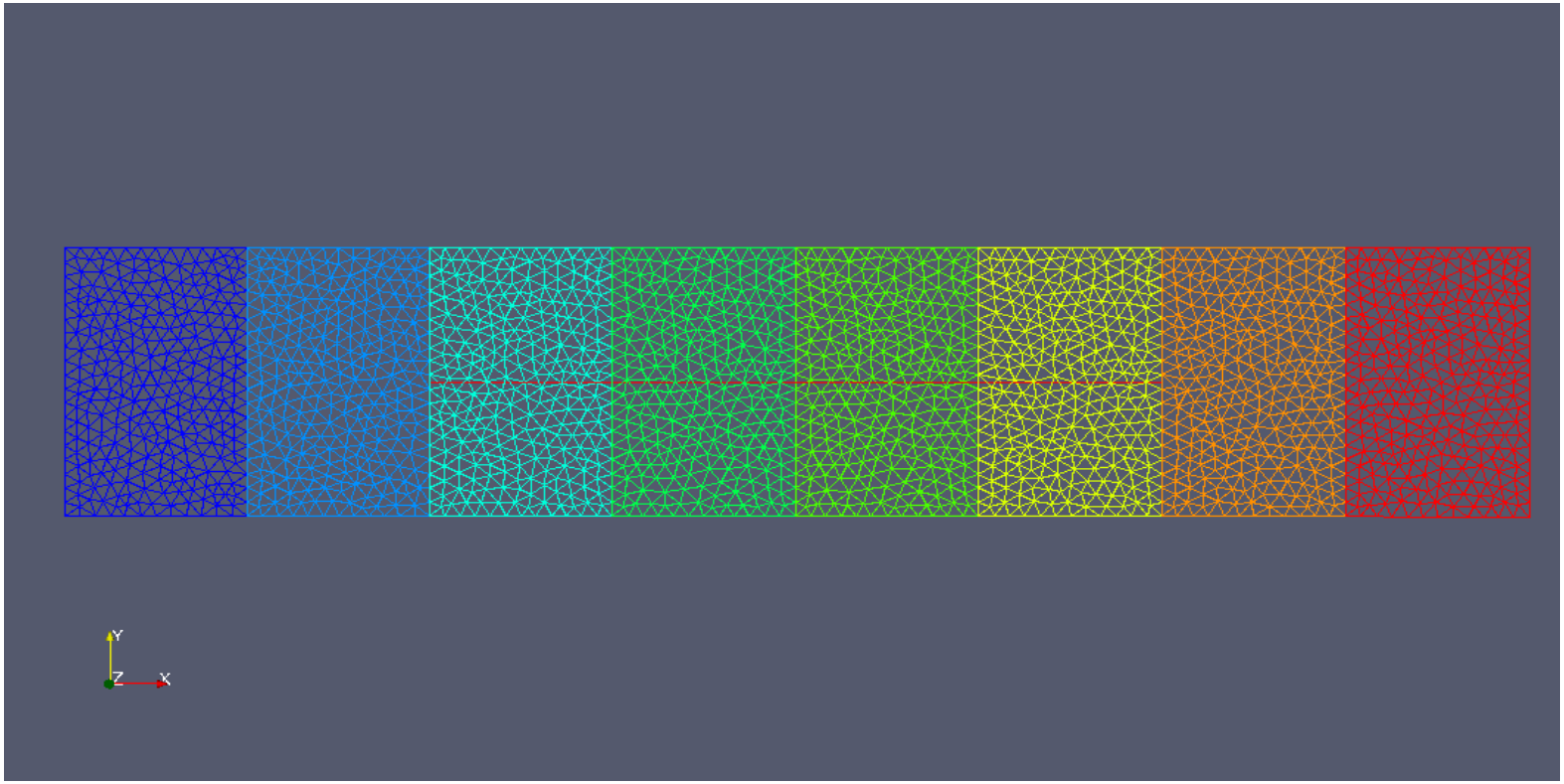
$$\begin{aligned} &\nu = 500 m^2/s, \\ &\rho_0 = 1000 kg/m^3, \\ &H = 1m, \\ &c = 3m/s, \\ &f = 2.10^{-11} y, \tau_x = \frac{\tau_0}{2} \left(1 + tanh(\frac{x_0 - x}{L})\right), \tau_y = 0, \\ &\Delta t = 30 min \end{aligned}$$

Exemple d'application : équation de St-Venant

Conditions interfaces :

$$\begin{cases} -\nu \frac{\partial u_i^{k+1}}{\partial \mathbf{n_i}} + c^2(n_{ix} + n_{iy}) h_i^{k+1} & -cu_i^{k+1} = -\nu \frac{\partial u_j^{k+1}}{\partial \mathbf{n_i}} \\ & \quad + c^2(n_{ix} + n_{iy}) h_j^{k+1} - cu_j^{k+1} \\ -\nu \frac{\partial v_i^{k+1}}{\partial \mathbf{n_i}} - cv_i^{k+1} = & -\nu \frac{\partial v_j^{k+1}}{\partial \mathbf{n_i}} - cv_j^{k+1} \end{cases}$$

Exemple d'application : équation de St-Venant



Conclusion

1. Python est facile à utiliser
2. attention aux performances
3. langage de plus en plus utilisé
4. ne s'applique pas qu'au calcul scientifique

Références

- www.python.org
- www.scipy.org
- http ://wiki.python.org/moin/ParallelProcessing
- diveintopython.org
- G. Swinnen. *Apprendre à programmer avec python*.
- H. P. Langtangen. *Python Scripting for Computational Science*. Springer, 3rd edition, 2008.

Références

- H. P. Langtangen, and X. Cai. *On the efficiency of Python for high-performance computing : a case study involving stencil updates for partial differential equations*, In : Proceedings of the HPSC2006 Conference, ed. by H. G. Bock et al, Springer Verlag. LNCSE, 2008.
- X. Cai, H. P. Langtangen, *Parallelizing PDE solvers using the Python programming language*. In A. M. Bruaset and A. Tveito, editors, *Numerical Solution of Partial Differential Equations on Parallel Computers*. Volume 51 of Springer Lecture Notes in Computational Science and Engineering. pp. 295-325, Springer, 2006