Travaux pratiques en décomposition de domaine

Loïc Gouarin[1] et Laurent Series[2]

[1]Laboratoire de mathématiques d'Orsay

[2]Ecole Centrale Paris

17 novembre 2011

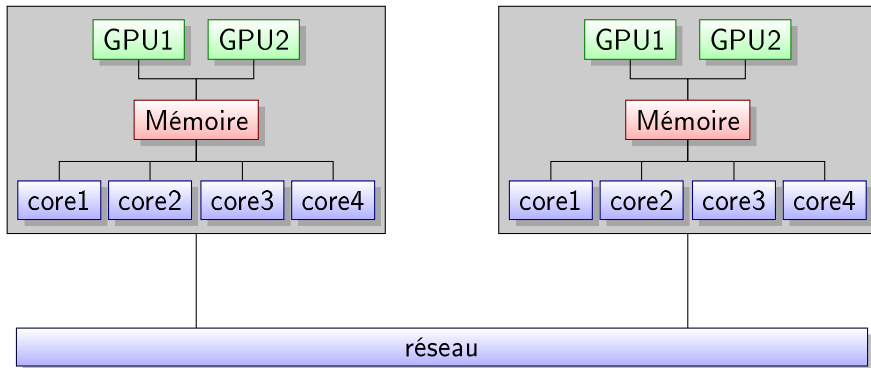
Plan

1. Introduction
2. MPI
3. Introduction aux méthodes de Schwarz
4. Parallélisation du gradient conjugué

Plan

1. Introduction
2. MPI
3. Introduction aux méthodes de Schwarz
4. Parallélisation du gradient conjugué

Comment paralléliser son code ?



Quels sont les outils pour paralléliser son code ?

1. Threads
2. PVM
   - pypvm
   - pynpvm
3. MPI
   - pyMPI
   - mpi4py
4. GPU
   - PyCUDA
   - PyOpenCL

Plan

1. Introduction
2. MPI
3. Introduction aux méthodes de Schwarz
4. Parallélisation du gradient conjugué

MPI : Message Passing Interface

- conçue en 1993,
- norme définissant une bibliothèque de fonctions, utilisable avec les langages C et Fortran,
- permet d'exploiter des ordinateurs distants ou multiprocesseurs par passage de messages.

MPI : Message Passing Interface

Fonctionnalités de MPI-1

- nombre de processus, numéro du processus,...
- communications point à point,
- communications collectives,
- communicateurs,
- types dérivées,
- topologies.

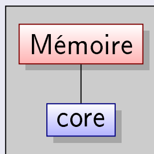
MPI : Message Passing Interface

Fonctionnalités de MPI-2

- gestion dynamique des processus,
- I/O parallèle,
- interfaçage avec Fortran95 et C++,
- extension des communications collectives aux intercommunicateurs,
- communications de mémoire à mémoire,
- ...

MPI : Message Passing Interface

Programme séquentiel

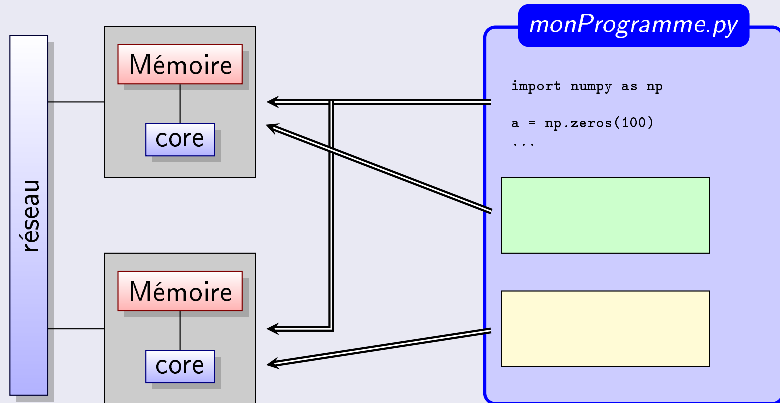


exécution

monProgramme.py

```
import numpy as np

a = np.zeros(100)
...
```

SPMD : Single Program Multiple Data



PyMPI

- projet initié en 2002 par Patrick Miller
- interfaçage de la bibliothèque MPI-1 directement en API C
- ce module gère
  - nombre de processus, numéro du processus,...
  - communications point à point,
  - communications collectives,
  - création de sous-communicateurs.

mpi4py

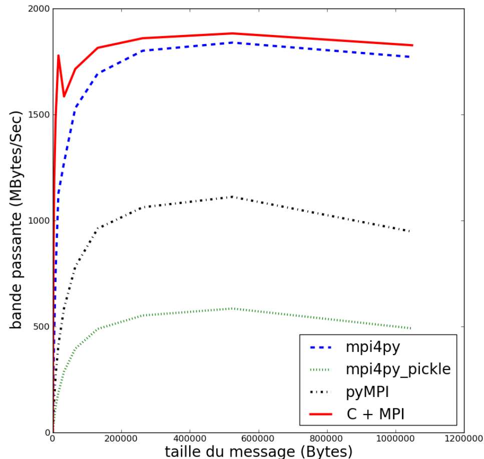
- projet inititié en 2006 par Lissandro Dalcin
- interfaçage de la bibliothèque MPI-1/2 avec Swig ($version < 1$)
- interfaçage de la bibliothèque MPI-1/2 avec Cython ($version >= 1$)
- ce module gère toutes les fonctionnalités que l'on peut trouver dans MPI-1/2

Tests de performance

Ping pong

- Chaque processus dispose d'un tableau de double.
- Envois alternés de paquets de données entre 2 processus.
- Les paquets sont de plus en plus grands.
- On regarde combien de temps il faut pour recevoir ces paquets.

Tests de performance



Un petit mot sur pickle

Le module **pickle** permet de convertir n'importe quel objet complexe Python en suites d'octets (*sérialisation*).

Ce flux peut

- être sauvegardé,
- être transmis via le réseau.

On peut ensuite le reconstruire (*désérialisation*).

Le module **cPickle** est une implémentation en C plus rapide que **pickle**.

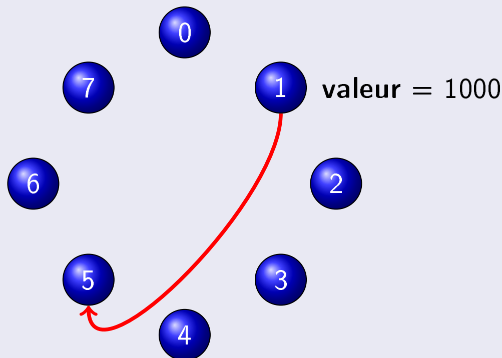
Un petit mot sur pickle

Exemple

```
import cPickle
import numpy as np

a = np.linspace(0., 1., 100)
b = "une chaine"

fichier = open("pickleData", "w")
cPickle.dump([a, b], fichier, 0)
```

Communication point à point bloquante

Exemple



Exemple Fortran

```
program point_a_point
  implicit none
  include 'mpif.h'
  integer, dimension(MPI_STATUS_SIZE) :: statut
  integer, parameter                  :: etiquette=100
  integer                             :: rang,valeur,code

  call MPI_INIT(code)

  call MPI_COMM_RANK(MPI_COMM_WORLD,rang,code)

  if (rang == 1) then
     valeur=1000
     call MPI_SEND(valeur,1,MPI_INTEGER,5,etiquette,MPI_COMM_WORLD,code)
  elseif (rang == 5) then
     call MPI_RECV(valeur,1,MPI_INTEGER,1,etiquette,MPI_COMM_WORLD,statut,code)
     print *,'Moi, processus 5, j''ai reçu ',valeur,' du processus 1.'
  end if

  call MPI_FINALIZE(code)
```

Même exemple avec Python

```
import mpi4py.MPI as mpi

rang = mpi.COMM_WORLD.rank

if rang == 1:
    valeur = 1000
    mpi.COMM_WORLD.send(valeur, dest = 5)
elif rang == 5:
    valeur = mpi.COMM_WORLD.recv(source = 1)
    print "Moi, Processus 5, j'ai recu", valeur, "du processus 1."
```

Un exemple un peu plus complexe

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

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

Un exemple un peu plus complexe (suite)

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

rank = mpi.COMM_WORLD.rank

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

Envoi d'un tableau Numpy

```
import mpi4py.MPI as mpi
import numpy as np

rank = mpi.COMM_WORLD.rank
n = 10
if rank == 0:
    sendarray = np.linspace(0., 1., n)
    mpi.COMM_WORLD.Send([sendarray, mpi.DOUBLE], dest = 1)
else:
    recvarray = np.empty(n)
    mpi.COMM_WORLD.Recv([recvarray, mpi.DOUBLE], source = 0)
    print recvarray
```

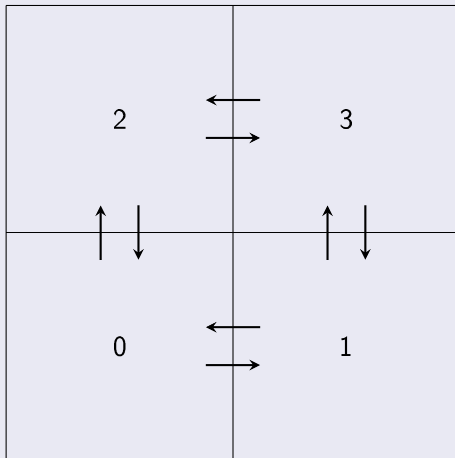
Résumé

Dans le module mpi4py.MPI

- *COMM_WORLD* : communicateur par défaut (englobe tous les processus lancés)
- *COMM_WORLD.size* : nombre de processus
- *COMM_WORLD.rank* : rang du processus
- *COMM_WORLD.send, COMM_WORLD.recv* : envoi et réception de messages via cPickle
- *COMM_WORLD.Send, COMM_WORLD.Recv* : envoi et réception de tableaux Numpy
- *INTEGER, FLOAT, DOUBLE, COMPLEX, ...* : types des données des tableaux Numpy envoyés et reçus

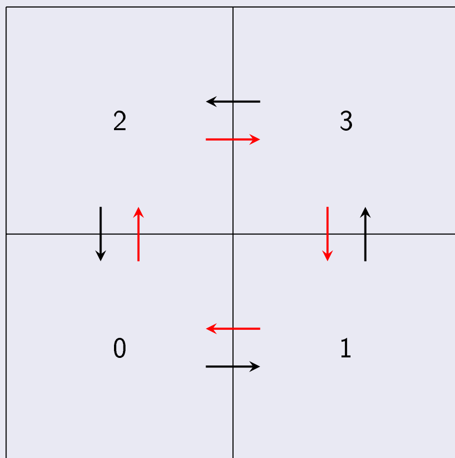
Communication point à point non bloquante

Exemple



Communication point à point non bloquante

Exemple

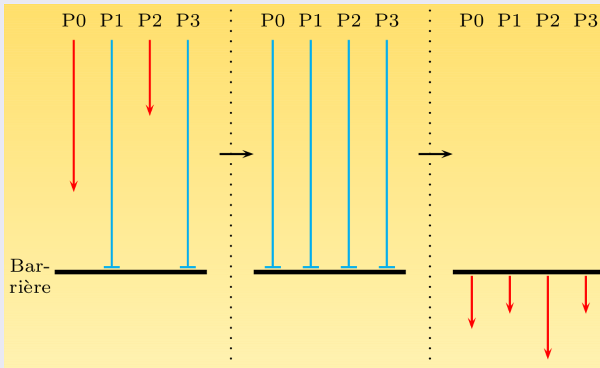


Communication point à point non bloquante

```
import mpi4py.MPI as mpi

rang, size = mpi.COMM_WORLD.rank, mpi.COMM_WORLD.size

if rang == 0:
    voisins = [2, 1]
if rang == 1:
    voisins = [0, 3]
if rang == 2:
    voisins = [3, 0]
if rang == 3:
    voisins = [1, 2]

recvalue = []
for v in voisins:
    mpi.COMM_WORLD.Isend([np.arange(1000), mpi.INT], v, tag=10*v + rang)
for v in voisins:
    recvalue.append(np.empty(1000, dtype=np.int))
    mpi.COMM_WORLD.Recv([recvalue[-1], mpi.INT], v, tag=10*rang + v)
print rang, voisins, recvalue
```

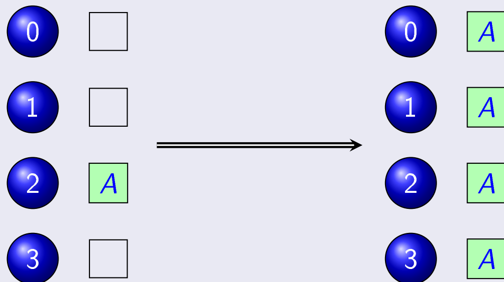
Synchronisation globale

mpi4py.MPI.COMM_WORLD.Barrier()



Diffusion générale

- bcast(obj = None, root = 0)
- Bcast(buf, root = 0)



Diffusion générale

Exemple avec cPickle

```
import mpi4py.MPI as mpi

if mpi.COMM_WORLD.rank == 0:
    a = [(1, 2), {2: 'toto', 3: 'titi'}]
    a = mpi.COMM_WORLD.bcast(a, 0)
else:
    a = mpi.COMM_WORLD.bcast(None, 0)
print a
```

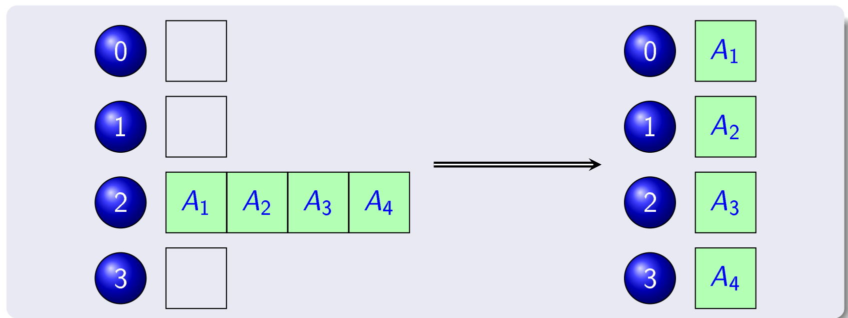
Diffusion générale

Exemple avec numpy

```
import mpi4py.MPI as mpi
import numpy as np

n = 10
a = np.empty(n)

if mpi.COMM_WORLD.rank == 0:
    a = np.linspace(0, 1, n)

mpi.COMM_WORLD.Bcast([a, mpi.DOUBLE], 0)
print a
```

Communication dispersive

- scatter(sendobj=None, recvobj=None, root = 0)
- Scatter(sendbuf, recvbuf, root = 0)



Communication dispersive

Exemple avec cPickle

```
import mpi4py.MPI as mpi

if mpi.COMM_WORLD.rank == 0:
    a = [(1, 2), {2: 'toto', 3: 'titi'}]
    a = mpi.COMM_WORLD.scatter(a, 0)
else:
    a = mpi.COMM_WORLD.scatter(None, 0)
print a
```

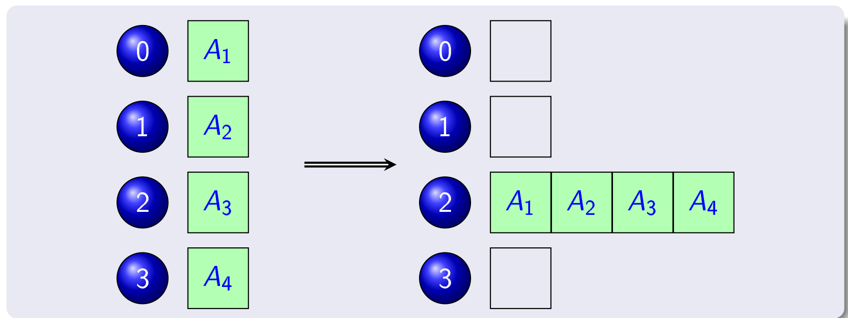
Communication dispersive

Exemple avec numpy

```
import mpi4py.MPI as mpi
import numpy as np

n = 16
b = np.empty(n/4)

if mpi.COMM_WORLD.rank == 0:
    a = np.linspace(0, 1, n)
    mpi.COMM_WORLD.Scatter([a, mpi.DOUBLE], [b, mpi.DOUBLE], 0)
else:
    mpi.COMM_WORLD.Scatter(None, [b, mpi.DOUBLE], 0)
print b
```

Rassembler

- gather(sendobj=None, recvobj=None, root = 0)
- Gather(sendbuf, recvbuf, root = 0)



Rassembler

Exemple avec cPickle

```
import mpi4py.MPI as mpi

if mpi.COMM_WORLD.rank == 0:
    a = (1, 2)
    a = mpi.COMM_WORLD.gather(a, 0)
    print a
else:
    a = {2: 'toto', 3: 'titi'}
    mpi.COMM_WORLD.gather(a, 0)
```

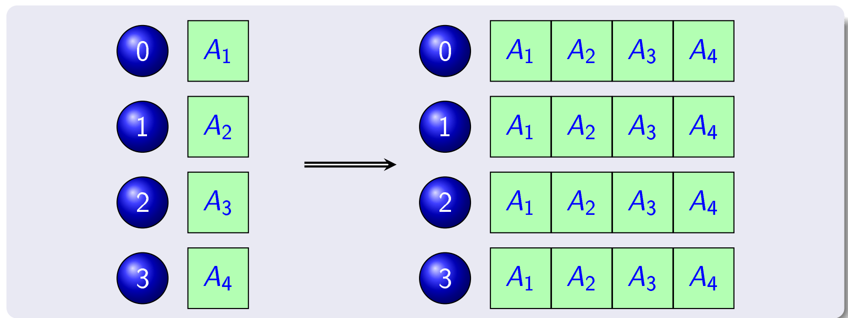
Rassembler

Exemple avec numpy

```
import mpi4py.MPI as mpi
import numpy as np

n = 4
rank = mpi.COMM_WORLD.rank
size = mpi.COMM_WORLD.size
interval = mpi.COMM_WORLD.size*n - 1

deb = float(n*rank)/interval
fin = float(n*(rank + 1) - 1)/interval
a = np.linspace(deb, fin, n)

if mpi.COMM_WORLD.rank == 0:
    b = np.empty(n*size)
    mpi.COMM_WORLD.Gather([a, mpi.DOUBLE], [b, mpi.DOUBLE], 0)
else:
    mpi.COMM_WORLD.Gather([a, mpi.DOUBLE], None, 0)

if rank == 0:
    print b
```

Tout rassembler

- allgather(sendobj=None, recvobj=None)
- Allgather(sendbuf, recvbuf)



Rassembler

Exemple avec cPickle

```
import mpi4py.MPI as mpi

if mpi.COMM_WORLD.rank == 0:
    a = (1, 2)
    a = mpi.COMM_WORLD.gather(a, 0)
    print a
else:
    a = {2: 'toto', 3: 'titi'}
    mpi.COMM_WORLD.gather(a, 0)
```

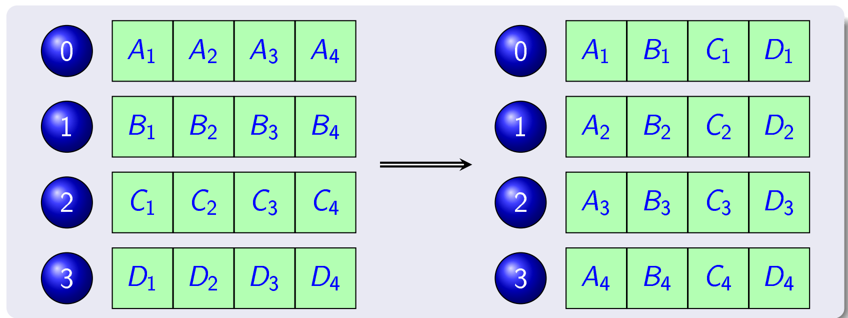
Rassembler

Exemple avec numpy

```
import mpi4py.MPI as mpi
import numpy as np

n = 4
rank = mpi.COMM_WORLD.rank
size = mpi.COMM_WORLD.size
interval = mpi.COMM_WORLD.size*n - 1

deb = float(n*rank)/interval
fin = float(n*(rank + 1) - 1)/interval
a = np.linspace(deb, fin, n)

if mpi.COMM_WORLD.rank == 0:
    b = np.empty(n*size)
    mpi.COMM_WORLD.Gather([a, mpi.DOUBLE], [b, mpi.DOUBLE], 0)
else:
    mpi.COMM_WORLD.Gather([a, mpi.DOUBLE], None, 0)

if rank == 0:
    print b
```

Echanges croisés

- alltoall(sendobj=None, recvobj=None)
- Alltoall(sendbuf, recvbuf)



Réduction

- reduce(sendobj=None, recvobj=None, op=SUM, root=0)
- Reduce(sendbuf, recvbuf, op=SUM, root=0)
- allreduce(sendobj=None, recvobj=None, op=SUM)
- Alleduce(sendbuf, recvbuf, op=SUM)

- *mpi4py.MPI.SUM* : somme des éléments
- *mpi4py.MPI.PROD* : produit des éléments
- *mpi4py.MPI.MAX* : recherche du maximum
- *mpi4py.MPI.MIN* : recherche du minimum
- *mpi4py.MPI.MAXLOC* : recherche de l'indice du maximum
- *mpi4py.MPI.MINLOC* : recherche de l'indice du minimum
- ...

Réduction

On reprend notre classe point en y ajoutant

```
    def __add__(self, p2):
        return point(0, self.x + p2.x, self.y + p2.y)
```

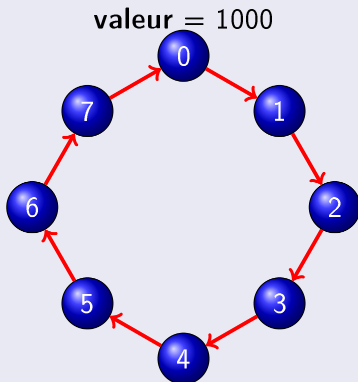
Exemple avec cPickle

```
import mpi4py.MPI as mpi
from point import point

rank = mpi.COMM_WORLD.rank
p1 = point(0, rank, rank + 1)

p2 = mpi.COMM_WORLD.allreduce(p1, mpi.SUM)
print p2
```

- cours MPI de l'IDRIS
- site de mpi4py
- site de pyMPI

Communication en anneau

Ecrire un script Python réalisant une communication en anneau. Le processus 0 a une valeur initialisée à 1000 et envoie à son voisin, le voisin rajoute 1 à cette valeur et envoie au suivant...



Plan

1 Introduction

2 MPI

3 Introduction aux méthodes de Schwarz

4 Parallélisation du gradient conjugué

Problème modèle 1D

On souhaite résoudre le problème

$$
\begin{aligned}
&-\Delta u(x) = f(x) \quad \text{pour} \quad x \in [0,1] \\
&u(0) = u(1) = 0
\end{aligned}
$$

discrétisé par un shéma aux différences finies à trois points sur la grille $x_j = jh, 0 \leq j \leq n-1$ où $h = 1/(n-1)$.

On a besoin

1. de matrices creuses

scipy.sparse

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

2. d'un solveur pour notre

scipy.sparse.linalg

basé sur le solveur SuperLU.

Assemblage de la matrice

```
def laplacian(nx, dx, nu, alphag=None, alphad=None):
    """
    Assemblage de la matrice
    """
    A = sp.lil_matrix((nx, nx))

    d = (2.*nu/dx**2 + 1.)*np.ones(nx)
    T = -nu*np.ones(nx)/dx**2
    A.setdiag(2.*nu/dx**2*np.ones(nx))
    A.setdiag(T, 1)
    A.setdiag(T, -1)

    if alphag is not None:
        A[0, 0] = 1./dx + alphag
        A[0, 1] = -1./dx

    if alphad is not None:
        A[-1, -1] = 1./dx + alphad
        A[-1, -2] = -1./dx

    return A
```

Mise en place des conditions de Dirichlet

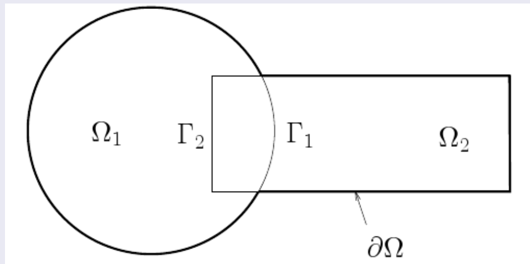
```
def dirichletCondition(A, b, alphag=None, alphad=None):
    """
    Condition de Dirichlet homogene a gauche et a droite
    """

    b[0] = 0.
    b[-1] = 0.
    if alphag is None:
        A[0, :] = 0.
        A[0, 0] = 1.
        b[0] = 0.

    if alphad is None:
        A[-1, :] = 0.
        A[-1, -1] = 1.
        b[-1] = 0.
```

Résolution du problème

```
# set second member
b = ...

A = laplacian(nx, dx, nu, eta)
# set Dirichlet condition
...
LU = linalg.factorized(A.tocsc())

u = LU(b)
```

Parallélisation

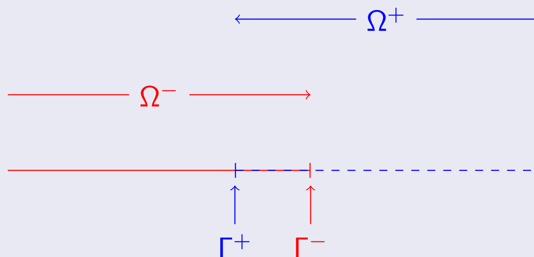
Algorithme de Schwarz (1870)



$$\begin{cases} \mathcal{L}(u_1^{k+1}) = f(x), & x \in \Omega_1 \\ u_1^{k+1}(x) = g(x), & x \in \partial\Omega_1 \bigcap \overline{\Omega_1} \\ u_1^{k+1}(x) = u_2^k(x), & x \in \Gamma_1 \end{cases} \qquad \begin{cases} \mathcal{L}(u_2^{k+1}) = f(x), & x \in \Omega_2 \\ u_2^{k+1}(x) = g(x), & x \in \partial\Omega_2 \bigcap \overline{\Omega_2} \\ u_2^{k+1}(x) = u_1^{k+1}(x), & x \in \Gamma_2 \end{cases}$$

Parallélisation

Algorithme de Schwarz classique avec recouvrement (Lions 1988)



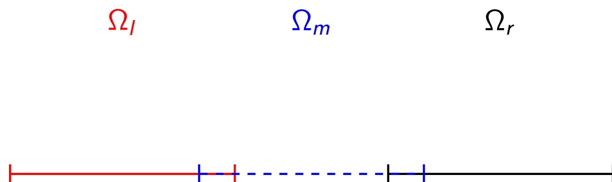
Parallélisation

Algorithme de Schwarz classique avec recouvrement (Lions 1988)

$$\left\{\begin{array}{llll} \mathcal{L}u^{k+1} = f(x) & \text{for} \quad x \in \Omega^- & \text{and} & t \geq 0, \\ u^{k+1} = v^k & \text{on} \quad x \in \Gamma^- & \text{and} & t \geq 0. \end{array}\right.$$

$$\left\{\begin{array}{llll} \mathcal{L}v^{k+1} = f(x) & \text{for} \quad x \in \Omega^+ & \text{and} & t \geq 0, \\ v^{k+1} = u^k & \text{on} \quad x \in \Gamma^+ & \text{and} & t \geq 0. \end{array}\right.$$

3 cas possibles



Exercices

- découper le domaine $[0, 1]$ en fonction du sous-domaine i et du recouverment (modifier le *linspace*),
- assembler le système pour chaque sous-domaine,
- appliquer les conditions de Dirichlet,
- résoudre le système multi domaines dans le cas du Schwarz classique.
- Refaire la même chose en mettant des conditions de Robin aux interfaces,

Plan

1. Introduction
2. MPI
3. Introduction aux méthodes de Schwarz
4. Parallélisation du gradient conjugué

Parallélisation du gradient conjugué

On souhaite résoudre le problème

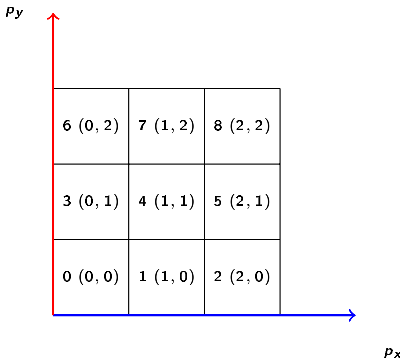
$$\begin{aligned} -\Delta u(x) = f(x) \quad \text{pour} \quad x \in [0,1] \times [0,1] \\ u = u_d \quad \text{sur} \quad \Gamma \quad \text{(bord du domaine)} \end{aligned}$$

discrétisé par la méthode des éléments finis (Q1).

Découpage du domaine

- nombre de découpages suivant x : $np_x = 3$
- nombre de découpages suivant y : $np_y = 3$
- coordonnées cartésiennes du processus i : (p_x, p_y)

avec $p_x = i\%np_x$ et $p_y = i/np_x$



Correspondance locale vers globale

numérotation globale

$\Omega = \Omega_1 \cup \Omega_2$

20	21	22	23	24
15	16	17	18	19
10	11	12	13	14
5	6	7	8	9
0	1	2	3	4

Ω_1

20	21	22
15	16	17
10	11	12
5	6	7
0	1	2

Ω_2

22	23	24
17	18	19
12	13	14
7	8	9
2	3	4

Correspondance locale vers globale

numérotation locale

$\Omega = \Omega_1 \cup \Omega_2$

20	21	22	23	24
15	16	17	18	19
10	11	12	13	14
5	6	7	8	9
0	1	2	3	4

Ω_1

12	13	14
9	10	11
6	7	8
3	4	5
0	1	2

Ω_2

12	13	14
9	10	11
6	7	8
3	4	5
0	1	2

Exercices

Compléter la méthode `fromCoord` de la classe `Mesh` :

- calcul des coordonnées cartésiennes des sous domaines
- calcul du nombre de noeuds par sous domaine en fonction de la taille des vecteurs x et y
- construction du tableau des correspondances locales vers globales
- extraction des coordonnées locales

Gradient conjugué

Résoudre Ax = b

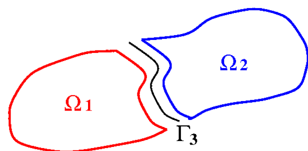
Initialisation

- $x^0 = 0$, $g^0 = Ax^0 - b$, $w^0 = g^0$

Itération p+1 (connaissant x^p, g^p et w^p) :

- Calcul du produit de A par la direction w^p : Aw^p
- Calcul de $\rho = -\frac{(g^p, w^p)}{(Aw^p, w^p)}$
- Mise à jour de la solution et du gradient :
  $x^{p+1} = x^p + \rho w^p$
  $g^{p+1} = x^p + \rho A w^p$
- Calcul de $\gamma = -\frac{(g^{p+1}, Aw^p)}{(Aw^p, w^p)}$
- Mise à jour de la direction de descente :
  $w^{p+1} = g^{p+1} + \gamma w^p$

Produit matrice vecteur



Matrice globale : $\begin{pmatrix} A_{11} & 0 & A_{13} \\ 0 & A_{22} & A_{23} \\ A_{31} & A_{32} & A_{33} \end{pmatrix}$

Matrice Ω_1 : $\begin{pmatrix} A_{11} & A_{13} \\ A_{31} & A_{33}^{(1)} \end{pmatrix}$ Matrice Ω_2 : $\begin{pmatrix} A_{22} & A_{23} \\ A_{32} & A_{33}^{(2)} \end{pmatrix}$

avec $A_{33}^{(1)} + A_{33}^{(2)} = A_{33}$

Produit matrice vecteur (2)

Produit global :

$$\begin{pmatrix} A_{11} & 0 & A_{13} \\ 0 & A_{22} & A_{23} \\ A_{31} & A_{32} & A_{33} \end{pmatrix} \begin{pmatrix} x_1 \\ x_2 \\ x_3 \end{pmatrix} = \begin{pmatrix} A_{11}x_1 + A_{13}x_3 \\ A_{22}x_2 + A_{23}x_3 \\ A_{31}x_1 + A_{32}x_2 + A_{33}x_3 \end{pmatrix}$$

Produits locaux :

$$\begin{pmatrix} A_{11} & A_{13} \\ A_{31} & A_{33}^{(1)} \end{pmatrix} \begin{pmatrix} x_1 \\ x_3 \end{pmatrix} = \begin{pmatrix} A_{11}x_1 + A_{13}x_3 \\ A_{31}x_1 + A_{33}^{(1)}x_3 \end{pmatrix}$$

$$\begin{pmatrix} A_{22} & A_{23} \\ A_{31} & A_{33}^{(2)} \end{pmatrix} \begin{pmatrix} x_2 \\ x_3 \end{pmatrix} = \begin{pmatrix} A_{22}x_2 + A_{23}x_3 \\ A_{32}x_2 + A_{33}^{(2)}x_3 \end{pmatrix}$$

Produit matrice vecteur (3)

Afin d'obtenir dans chaque sous-domaine la trace sur l'interface du produit Ax, il est nécessaire d'échanger des données entre les sous-domaines.

$$\text{Produit global : } \begin{pmatrix} A_{11}x_1 + A_{13}x_3 \\ A_{22}x_2 + A_{23}x_3 \\ A_{31}x_1 + A_{32}x_2 + A_{33}x_3 \end{pmatrix}$$

$$\Omega_1 \quad \begin{pmatrix} A_{11}x_1 + A_{23}x_3 \\ A_{31}x_1 + A_{33}^{(1)}x_3 \end{pmatrix} \quad + \quad \begin{pmatrix} 0 \\ A_{32}x_2 + A_{33}^{(1)}x_3 \end{pmatrix}$$

$$\Omega_2 \quad \begin{pmatrix} A_{22}x_2 + A_{23}x_3 \\ A_{32}x_2 + A_{33}^{(2)}x_3 \end{pmatrix} \quad + \quad \begin{pmatrix} 0 \\ A_{31}x_1 + A_{33}^{(2)}x_3 \end{pmatrix}$$

Exercices

Compléter la classe `Interface` :

- compléter la méthode `build_interface`
- compléter la méthode `interf_assemble`

Produit scalaire

Produits locaux :

$$\Omega_1 \quad \begin{pmatrix} x_1 \\ x_3 \end{pmatrix} \begin{pmatrix} y_1 \\ y_3 \end{pmatrix} = (x_1 y_1) + (x_3 y_3)$$

$$\Omega_2 \quad \begin{pmatrix} x_2 \\ x_3 \end{pmatrix} \begin{pmatrix} y_2 \\ y_3 \end{pmatrix} = (x_2 y_2) + (x_3 y_3)$$

Somme globale des produits locaux :

$$(x_2 y_2) + (x_2 y_2) + 2(x_3 y_3)$$

Produit scalaire (2)

Deux solutions :

- Somme les produits locaux pondérés pour chaque composante par un facteur égale à l'inverse du nombre de sous-domaines auxquels elle appartient :

$$(x_1 y_1) + \frac{1}{2}(x_3 y_3) \text{ dans } \Omega_1 \quad \text{et} \quad (x_2 y_2) + \frac{1}{2}(x_3 y_3) \text{ dans } \Omega_2$$

- Utilisation des vecteurs non assemblés :

$$(x_1 y_1) + (x_3 y_3^{(1)}) \text{ dans } \Omega_1 \quad \text{et} \quad (x_2 y_2) + (x_3 y_3^{(2)}) \text{ dans } \Omega_2$$

$$\text{avec} \quad y_3^{(1)} + y_3^{(2)} = y_3$$

Gradient conjugué parallèle

Résoudre Ax = b

Initialisation

- $x^0 = 0$, $g^0 = Ax^0 - b$, assemblage de g^0 : g^0_{ass}, $w^0 = g^0_{ass}$

Itération p+1 (connaissant x^p, g^p et w^p) :

- Calcul du produit de A par la direction w^p : Aw^p
- Calcul de $\rho = -\frac{(g^p, w^p)}{(Aw^p, w^p)}$
- Mise à jour de la solution et du gradient :
  $x^{p+1} = x^p + \rho w^p$
  $g^{p+1} = g^p + \rho A w^p$
- Assemblage du gradient g^{p+1} : g^{p+1}_{ass}
- Calcul de $\gamma = -\frac{(g_{ass}{p+1}, Aw^p)}{(Aw^p, w^p)}$
- Mise à jour de la direction de descente :
  $w^{p+1} = g^{p+1}_{ass} + \gamma w^p$

Exercices

Modifier la fonction `cgPara` du fichier `cg.py` :

- assemblage du gradient
- traitement des coefficients ρ et γ obtenus par produit scalaire

Gradient conjugué préconditionné

Résoudre M Ax = M b

Initialisation

- $x^0 = 0$, $g^0 = Ax^0 - b$, $w^0 = Mg^0$

Itération p+1 (connaissant x^p, g^p et w^p) :

- Calcul du produit de A par la direction w^p : Aw^p
- Calcul de $\rho = -\frac{(g^p, w^p)}{(Aw^p, w^p)}$
- Mise à jour de la solution et du gradient :
  $x^{p+1} = x^p + \rho w^p$
  $g^{p+1} = g^p + \rho A w^p$
- Calcul du produit de A par le gradient g^{p+1} : Mg^{p+1}
- Calcul de $\gamma = -\frac{(Mg^{p+1}, Aw^p)}{(Aw^p, w^p)}$
- Mise à jour de la direction de descente :
  $w^{p+1} = Mg^{p+1} + \gamma w^p$

Gradient conjugué préconditionné

On choisi un préconditionneur M tel que Mg :

$$Mg = \begin{pmatrix} -A_{11}^{-1} A_{13} g_3 \\ -A_{22}^{-1} A_{23} g_3 \\ g_3 \end{pmatrix}$$

Si on pose $w = Mg$ alors les composantes de w sont telles que :

$$A_{11} w_1 + A_{13} w_3 = 0$$

$$A_{22} w_2 + A_{23} w_3 = 0$$

Gradient conjugué préconditionné

Ce préconditionneur est qualifié de localement optimal dans ce sens que le gradient est nul à l'intérieur des sous-domaines.
En effet, le produit de A par w lorsque $w = Mg$ est égal :

$$\begin{pmatrix} A_{11} & 0 & A_{13} \\ 0 & A_{22} & A_{23} \\ A_{31} & A_{32} & A_{33} \end{pmatrix} \begin{pmatrix} w_1 \\ w_2 \\ w_3 \end{pmatrix} = \begin{pmatrix} -A_{11}A_{11}^{-1}A_{13}g_3 + A_{13}w_3 = 0 \\ -A_{22}A_{22}^{-1}A_{23}g_3 + A_{23}w_3 = 0 \\ (A_{33} - A_{31}A_{11}^{-1}A_{13} - A_{32}A_{22}^{-1}A_{23})w_3 \end{pmatrix}$$

Puisque la mise à jour du gradient s'écrit $g^{p+1} = g^p + \rho A w^p$, on conserve bien un gradient nul à l'intérieur des sous-domaines si l'on part d'un gradient initial nul.

Gradient conjugué préconditionné

Pour obtenir un gradient initial nul à l'intérieur des sous-domaines, il suffit de chosir la solution initiale telle que :

$$A_{11}x_1^0 + A_{13}x_3^0 = b_1 \quad \text{soit} \quad x_1^0 = A_{11}^{-1}b_1 - A_{11}^{-1}A_{13}x_3^0$$

$$A_{22}x_2^0 + A_{23}x_3^0 = b_2 \quad \text{soit} \quad x_2^0 = A_{22}^{-1}b_1 - A_{22}^{-1}A_{23}x_3^0$$

En effet, en substituant x_1^0 et x_1^0 dans $g = Ax - b$, on obtient :

$$\begin{pmatrix} A_{11} & 0 & A_{13} \\ 0 & A_{22} & A_{23} \\ A_{31} & A_{32} & A_{33} \end{pmatrix} \begin{pmatrix} A_{11}^{-1}b_1 - A_{11}^{-1}A_{13}x_3^0 \\ A_{22}^{-1}b_1 - A_{22}^{-1}A_{23}x_3^0 \\ x_3^0 \end{pmatrix} - \begin{pmatrix} b_1 \\ b_2 \\ b_3 \end{pmatrix} =$$

$$\begin{pmatrix} 0 \\ 0 \\ (A_{33} - A_{31}A_{11}^{-1}A_{13} - A_{32}A_{22}^{-1}A_{23})x_3^0 - (b_3 - A_{31}A_{11}^{-1}b_1 - A_{32}A_{22}^{-1}b_2) \end{pmatrix}$$

Exercices

Créer la fonction *cgPrecPar* implémentant le gradient conjugué parallèle préconditionné.

Si on pose

$$S_{33} = A_{33} - A_{31}A_{11}^{-1}A_{13} - A_{32}A_{22}^{-1}A_{23}$$

et

$$c_3 = b_3 - A_{31}A_{11}^{-1}b_1 - A_{32}A_{22}^{-1}b_2$$

créer la fonction *schur* résolvant le système de la méthode de schur $S_{33}x_3 = c_3$ par un gradient conjugué.