Calcul scientifique sur grilles

F. Jézéquel
Fabienne.Jezequel @ lip6.fr
www-pequan.lip6.fr/~jezequel

Master 2 SFPN
HPC Avancé

Plan

- grilles de calcul / cloud computing
- algorithmes itératifs parallèles
  - version synchrone/asynchrone
  - détection de convergence
  - algorithmes itératifs et consommation d'énergie
- résolution sur grilles de systèmes linéaires
  - les bibliothèques parallèles
  - résolution sur grilles : le multisplitting
  - implantation et performances sur Grid5000

Grilles de calcul / cloud computing

Motivations du parallélisme

Faire travailler simultanément plusieurs processeurs pour résoudre un même problème afin :

- de diminuer le temps de calcul,
- d'augmenter la taille du problème traité.

Dû aux besoins en calcul toujours croissants de la recherche et de l'industrie.

C'est quoi une grille de calcul ?

- cluster : ensemble de machines homogènes sur un même site
- grille : ensemble de clusters distants (hétérogènes en général)

Les grilles sont une alternative bon marché aux super-calculateurs.

particularités techniques des grilles : latences non négligeables, débits variables, hétérogénéité des machines et des réseaux

  - grilles de production
  - grilles de recherche

Le cloud

“ensemble de matériel, de raccordements réseau et de logiciels qui fournit des services qui peuvent être exploités à volonté”

paiement à l’usage :
la quantité de service consommée dans le cloud est mesurée
$\Rightarrow$ contrôle, adaptation des moyens, facturation

Cloud computing : modes d'utilisation

- Infrastructure as a Service (IaaS)
  Le matériel est fourni sous forme de machines virtuelles sur lesquelles on installe son image disque
  ex : Amazon EC2, Rackspace, GoGRID, Orange, ...

- Platform as a Service (PaaS)
  On peut développer ses propres applications en utilisant les services fournis
  ex : Google Apps, Windows Azure, Amazon S3, IBM CloudBurst et Websphere, ...

- Software as a Service (SaaS)
  Des applications entières sont disponibles à distance
  ex : Googledocs, Facebook, Orange, IBM LotusLive, ...

Ref : G. Mathieu, "Les grilles et le could computing"

Notion de conteneur (1/3)

virtualisation à base de conteneurs : méthode de virtualisation dans laquelle la couche de virtualisation s'exécute au sein même du système d'exploitation (OS).

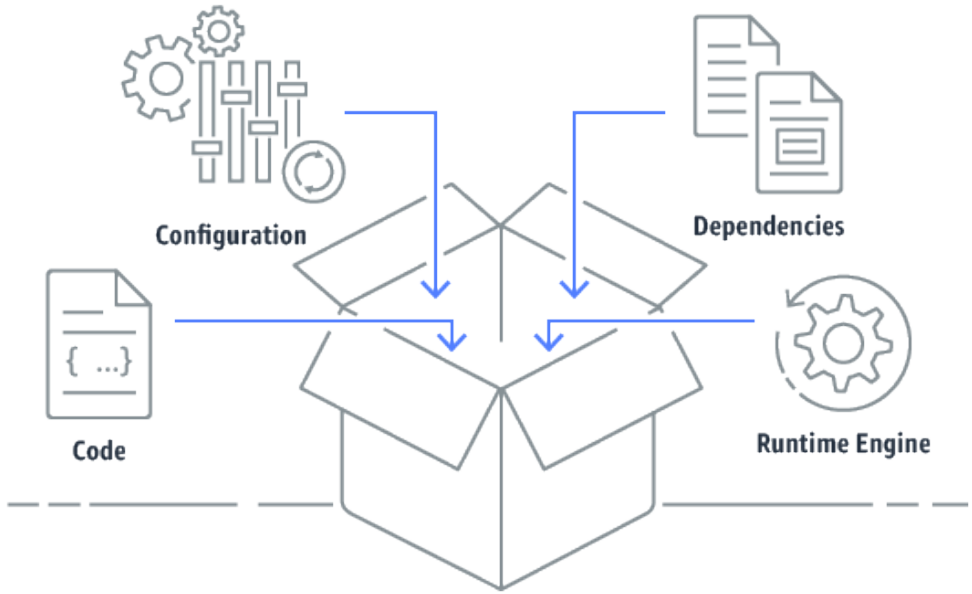
Le noyau de l'OS hôte exécute plusieurs machines virtuelles invitées et autonomes sans passer par une couche logicielle supplémentaire. Ces machines invitées s'appellent des conteneurs.

avantage : on peut héberger sur un serveur beaucoup plus d'instances virtualisées que la virtualisation traditionnelle.

inconvénient : chaque invité doit utiliser le même OS que celui de la machine hôte (on ne peut pas faire tourner une application Windows sur un conteneur Linux).

Notion de conteneur (2/3)

Les conteneurs fournissent un moyen standard de joindre le code, les configurations et les dépendances d'une application en un seul objet.



ex : Docker

Notion de conteneur (3/3)

CaaS (Containers as a Service) : services de prise en charge de conteneurs d'un fournisseur de cloud.

à l'utilisateur d'administrer ses conteneurs

se situe entre l'infrastructure en tant que service (IaaS, Infrastructure as a Service) et la plateforme en tant que service (PaaS, Platform as a Service).

Refs : `https://www.lemagit.fr`,
`https://aws.amazon.com/fr/containers`

Initiatives européennes

- PRACE `www.prace-ri.eu`
  Mise en commun de ressources des centres de calcul nationaux
- EUDAT `www.eudat.eu`
  Objectif : répondre aux besoins des chercheurs en matière d'accès, de préservation, d'analyse de données scientifiques.
- EGI : European Grid Initiative `www.egi.eu`

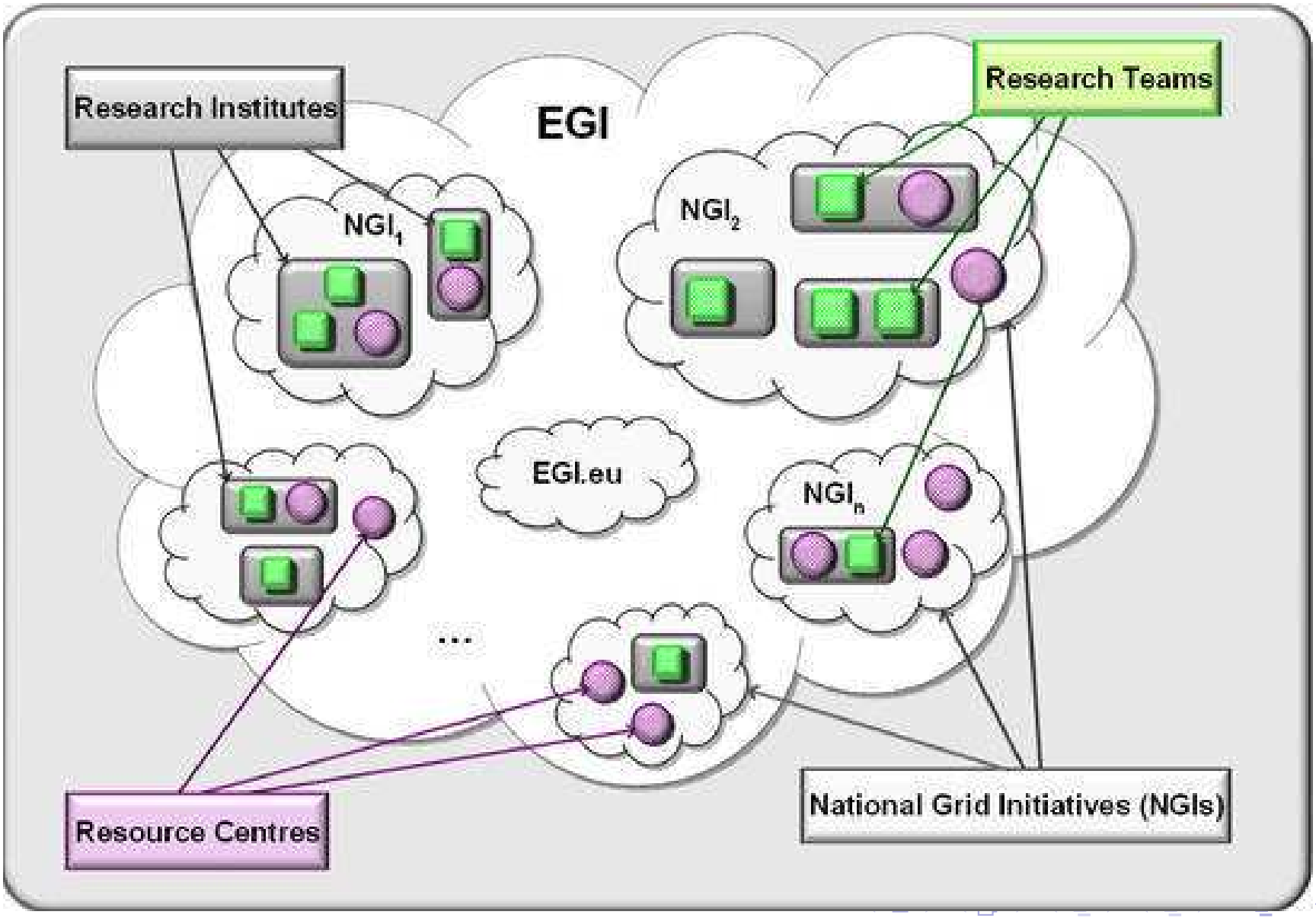
Le projet EGI (1/4)

EGI : European Grid Initiative `http://www.egi.eu`

EGI vise à doter l'ensemble des pays européens d'une grille commune. EGI se compose :

- dans chaque pays : d'une entité responsable, appelée NGI (National Grid Initiative)
- d'une structure de coordination et d'opération centrale siégeant à Amsterdam.

Le projet EGI (2/4)



Le projet EGI (3/4)

la plus grand grille du monde !

e-infrastructure publique (soutien de l'UE)

- 350 centres
- 56 pays en Europe, dans la région Asie-Pacifique, au Canada et en Amérique Latine.
- $>$370.000 CPU
- $>$170 Po (1 Po$=10^{15}$ octets)

s'appuie sur les réseaux GEANT en Europe et RENATER en France

Le projet EGI (4/4)

regroupe différentes communautés de la recherche ou de l'industrie

Domaines applicatifs :

- recherche biomédicale (recherche de nouvelles molécules)
- sciences de la Terre (climat, hydrologie, géophysique,...)
- astronomie
- économie
- physique des particules

En France

Le Groupement d'Intérêt Scientifique France Grilles joue le rôle de NGI-France `http://www.france-grilles.fr`

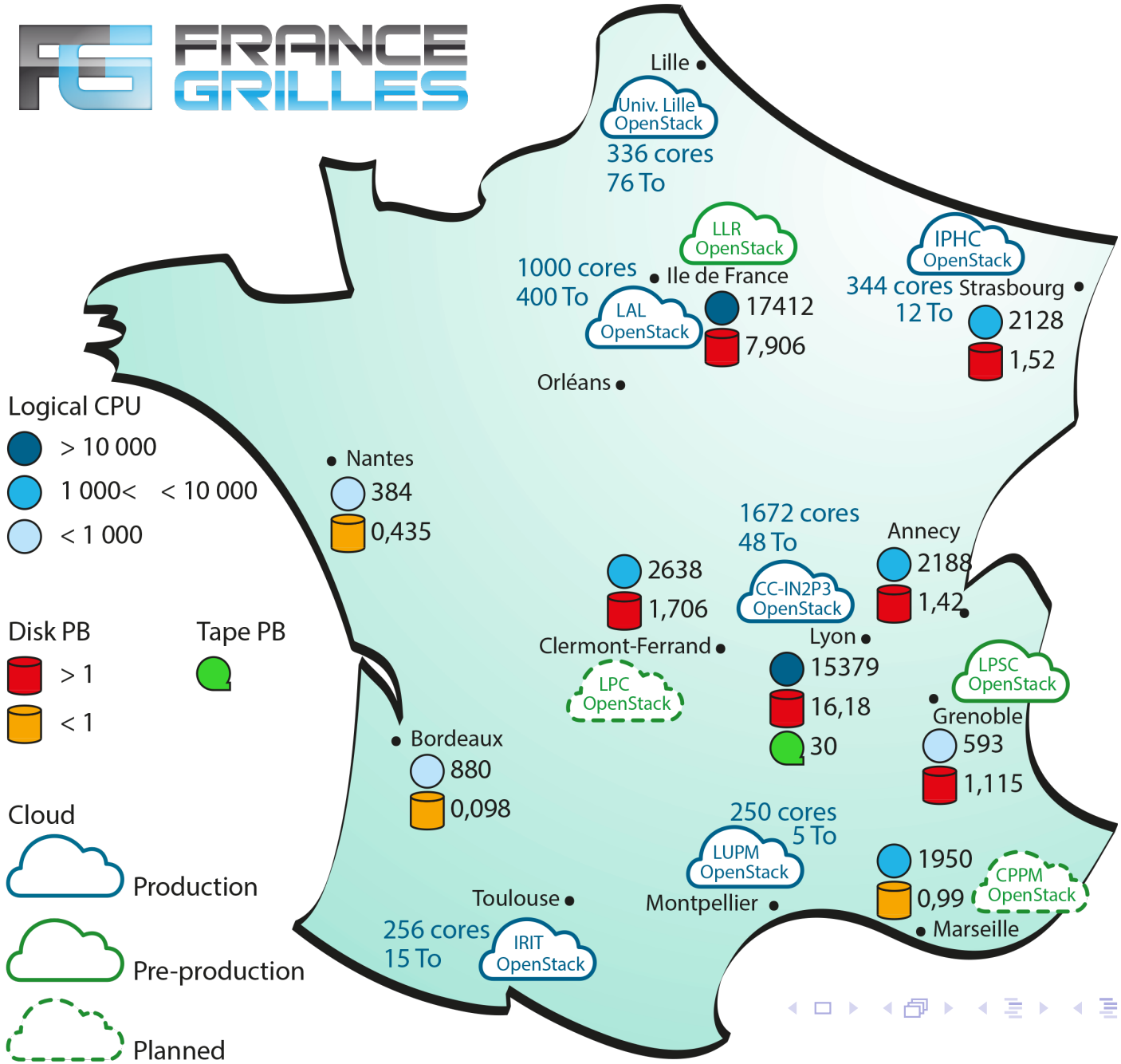
>1000 utilisateurs

Infrastructure de France Grilles (chiffres-clefs de 2015 : `http://www.france-grilles.fr/category/statistiques`)

- grille
  - 12 sites
  - 41231 CPU logiques
  - 24 Po de stockage.
- cloud
  - 9 sites
  - 2000 CPU logiques
  - 100 To de stockage
  - OpenStack : ensemble de logiciels open source permettant de déployer des infrastructures de cloud computing (infrastructure en tant que service).

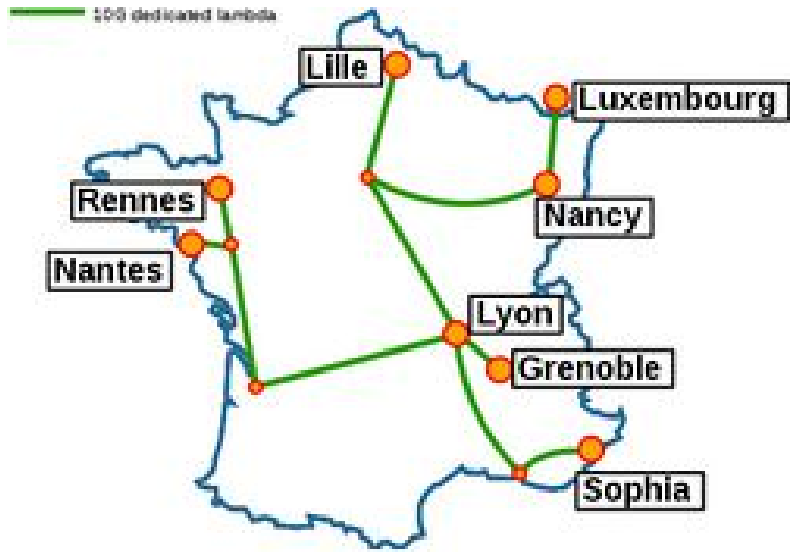
N.B. : 1 To = 10^{12} octets, 1 Po = 10^{15} octets.

Infrastructure de France Grilles



GRID5000 : grille de recherche www.grid5000.fr

grille expérimentale répartie sur 7 sites en France et le site du Luxembourg (plus de 500 utilisateurs).
Sur chaque site : au moins un cluster (76 à 486 processeurs)
sites connectés par le réseau RENATER : connexions 10 Gb/s.



GRID5000

Au 25 avril 2019 :

- 31 clusters
- 828 nœuds
- 12328 cœurs
- accélérateurs : 4 Intel Xeon Phi, 88 GPU Nvidia

grille reconfigurable

réservation de noeuds pour une certaine durée (en prévoir toujours plus !)

possibilité de déployer ses propres images linux
L'image déployée contient :

- l'environnement nécessaire (compilateurs, bibliothèques)
- aucun protocole inutile (FTP, LDAP, NFS,...) qui pourrait ralentir l'exécution des calculs.

Gestion des tâches sur Grid5000

OAR (*batch scheduler*) `http://oar.imag.fr`

Fonctionnalités : jobs en batch et en interactif, règles d'admission, gestion des priorités, réservation à l'avance des noeuds, ajout/suppression dynamique de noeuds, ...

commandes de base décrites dans
`www.grid5000.fr/mediawiki/index.php/Getting_Started`

Réservation de noeuds

Exemples

```
oarsub -I -l nodes=2,walltime=0:30
```

2 noeuds pourront être utilisés pendant 30 min dès qu'ils seront disponibles

`-I` : mode interactif

`-l` : permet de spécifier les ressources voulues

```
oarsub -l nodes=3,walltime=3 -r '2019-12-23 16:30:00'
```

réservation à une date future

```
oarsub -p "cluster='granduc'" -l nodes=5,walltime=2 -I
```

choix du cluster

```
oarsub -p "wattmeter='YES' and gpu='YES'" -l
nodes=2,walltime=2 -I
```

noeuds avec GPU et mesure de consommation électrique

Connexion à un noeud / gestion des jobs

`oarsh` (par rapport à `ssh`) améliore le partage des ressources par plusieurs jobs sur un même nœud.

`oarsh` : englobe `ssh` en tant qu'utilisateur `oar`, puis `su` de l'utilisateur

Par ex, on suppose que les noeuds utilisés sont griffon-49 et griffon-54.
sur griffon-49 :

```
uniq $OAR_NODEFILE # list of resources of your reservation
oarsh griffon-1 # use a node not in the file (will fail)
oarsh griffon-54 # use the other node of your reservation
ssh griffon-54 # will fail
```

liste des jobs en mode exécution : `oarstat`

destruction d'un job : `oardel ...`

Réservation de noeuds puis déploiement d'une image

Exemple

Réservation :

```
oarsub -I -l nodes=1,walltime=1:45 -t deploy
```

-t deploy donne le droit de déployer ultérieurement une image sur les noeuds réservés

Déploiement d'une image (5 à 10 minutes) :

```
kadeploy3 -f $OAR_NODE_FILE -e wheezy-x64-base -k
```

-k permet de copier sa clef ssh sur le compte root afin de se connecter sans mot de passe après le déploiement

Utilisation de l'image :

```
ssh root@griffon-42
```

L'image peut ensuite être modifiée (apt-get install ...) puis sauvegardée.

Algorithmes itératifs parallèles

Méthodes numériques directes ou itératives

- Méthodes directes :
  fournissent la solution exacte, modulo les erreurs d'arrondi ...
  qui peuvent devenir problématiques !
- Méthodes itératives :
  - convergent vers la solution par approximations successives
  - des critères de convergence doivent être respectés

Exemple de critères de convergence : résolution de système linéaire

On souhaite résoudre $Ax = b$.
On décompose la matrice A : $A = M - N$, avec M inversible.

$$Ax = b \quad \Leftrightarrow \quad Mx = Nx + b$$

On calcule la suite de vecteurs x^i à partir d'un vecteur x^0 choisi arbitrairement :

$$M\,x^{k+1} = N\,x^k + b \quad \Leftrightarrow \quad x^{k+1} = M^{-1}N\,x^k + M^{-1}b$$

$B = M^{-1}N$ est la matrice d'itération.

Le processus sera convergent si $\rho(B) < 1$.

Parallélisation d'une méthode itérative

- on distribue le travail effectué lors d'une itération aux différents processeurs
- à chaque nouvelle itération,
  les processeurs s'échangent les résultats dont ils ont besoin et déterminent si la convergence est atteinte

Ces algorithmes parallèles sont synchrones et ont le même comportement que les algo. séquentiels dont ils sont dérivés.

Méthodes synchrones ou asynchrones

Les algorithmes itératifs parallèles synchrones peuvent être très coûteux dans un contexte hétérogène et distant (on attend toujours le process le plus lent).
Recouvrir une partie des communications par du calcul permet de réduire les temps de synchronisation.

Les algorithmes asynchrones sont adaptés aux grilles :
ils supportent l'hétérogénéité des processeurs et des réseaux.
Les itérations sont différentes du mode synchrone.
$\Rightarrow$ Il faut étudier minutieusement la convergence des algo. asynchrones.

Modèle d'algorithme itératif séquentiel

x^0 donné
for k=0,1,...
    $x^{k+1} = f(x^k)$
end for

x^k : vecteur de dimension n
f : fonction de $\mathbb{R}^n$ dans $\mathbb{R}^n$

Test d'arrêt :

- cas général : $\max_{i=1..n} |x_i^{k+1} - x_i^k| < \varepsilon$
- système linéaire $Ax = b$: calcul du résidu $R = Ax - b$
  $\max_{i=1..n} |R_i| < \varepsilon$

Modèle d'algorithme itératif parallèle

On découpe le vecteur x^k en m blocs ($m < n$).
$x^k = (x_1^k, x_2^k, ..., x_n^k) = (X_1^k, X_2^k, ..., X_m^k)$

Modèle synchrone :
$(X_1^0, ..., X_m^0)$ donné
for k=0,1,...
    for i=1,...,m
        $X_i^{k+1} = F_i(X_1^k, ..., X_m^k)$
    end for
end for

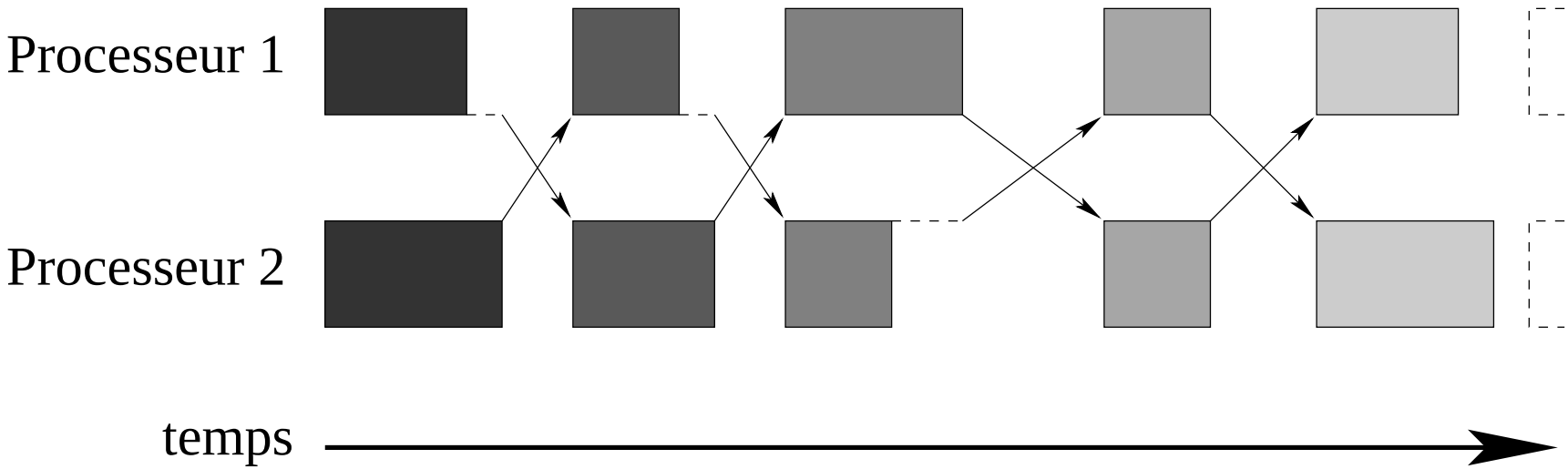
A l'itération k, chaque process met à jour son bloc de composantes.

Dans le modèle asynchrone, toutes les composantes ne sont pas mises à jour à l'itération k.
Il existe des blocs i pour lesquels $X_i^{k+1} = X_i^k$.

Classification des algorithmes itératifs parallèles

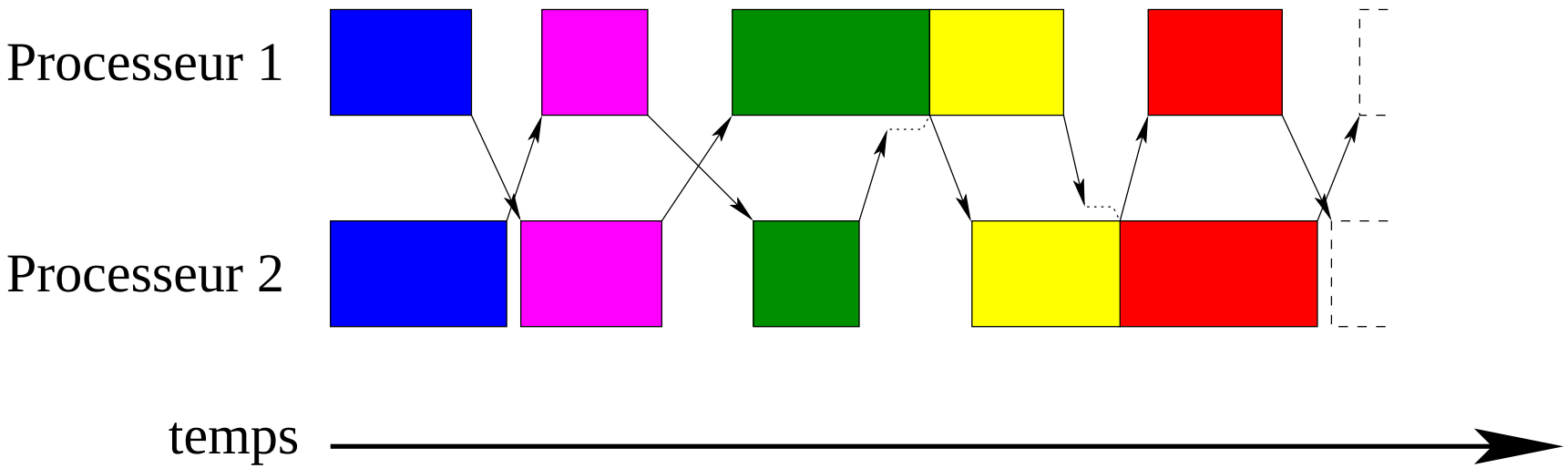
- ISCS : Itération Synchrone, Communication Synchrone
- ISCA : Itération Synchrone, Communication Asynchrone
- IACA : Itération Asynchrone, Communication Asynchrone

Algorithme ISCS



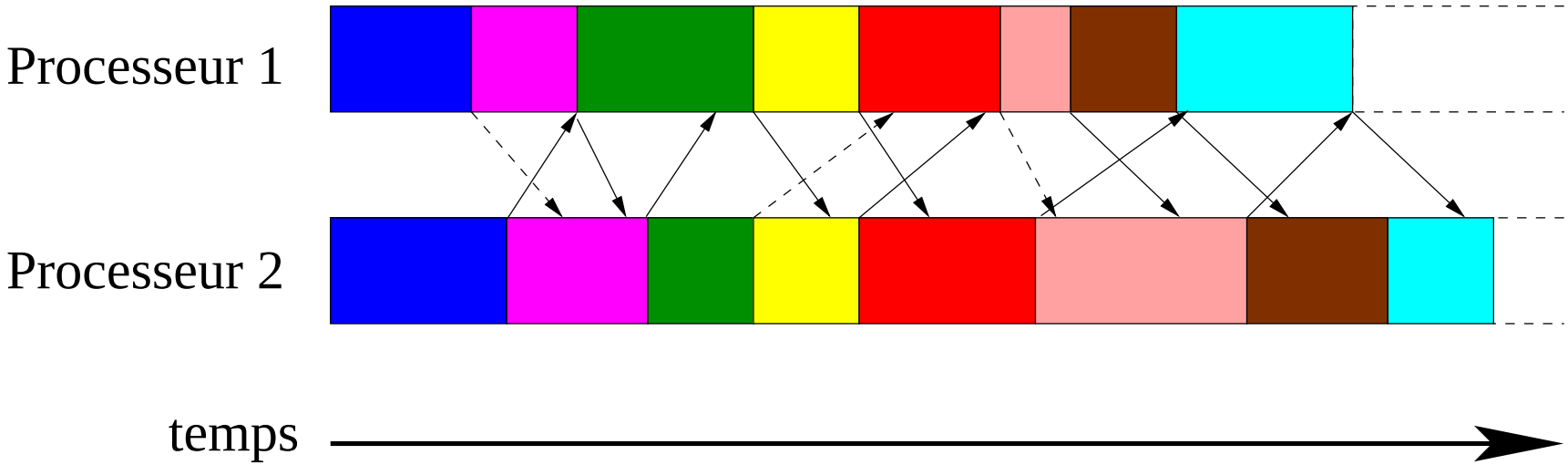
- Les échanges de données sont effectués à la fin d'une itération.
- Les processeurs doivent s'attendre.

Algorithme ISCA



- Dès qu'un résultat est calculé, il est envoyé. L'envoi peut donc être recouvert par du calcul (cf 1e itération).
- Si un message arrive alors qu'un processeur n'a pas fini son itération, la réception n'a lieu qu'à la fin de l'itération (cf pointillés).

Algorithme IACA (1)



Un noeud n'attend pas ses voisins.
Donc aucun temps mort entre les itérations.

Un processeur peut recevoir plusieurs messages du même voisin correspondant à des itérations différentes. Seul le dernier message est intéressant.
En pointillés : envois de résultats qui ne seront pas utilisés.

Algorithme IACA (2)

Particularités des algo. IACA :

- ils tolèrent les pertes de messages
- ils permettent de recouvrir les communications par du calcul
- adaptés aux environnements de clusters distants (latences non négligeables, débits variables, hétérogénéité des machines et des réseaux)
- en général, plus d'itérations qu'avec des itérations synchrones
- le nombre d'itérations peut varier d'une exécution à une autre et d'un processeur à un autre
- détection de convergence plus complexe.

Détection de convergence

- Algorithmes synchrones
  On détermine le max des résidus locaux.
  Cette opération synchronise l'ensemble des processeurs.

- Algorithmes asynchrones
  - approche centralisée : un processeur centralise le résidu local de chaque nœud
  - approche décentralisée : fondée sur des communications entre voisins. Lorsqu'un processeur a convergé, il en informe un voisin.

Détection de convergence locale dans les algorithmes asynchrones

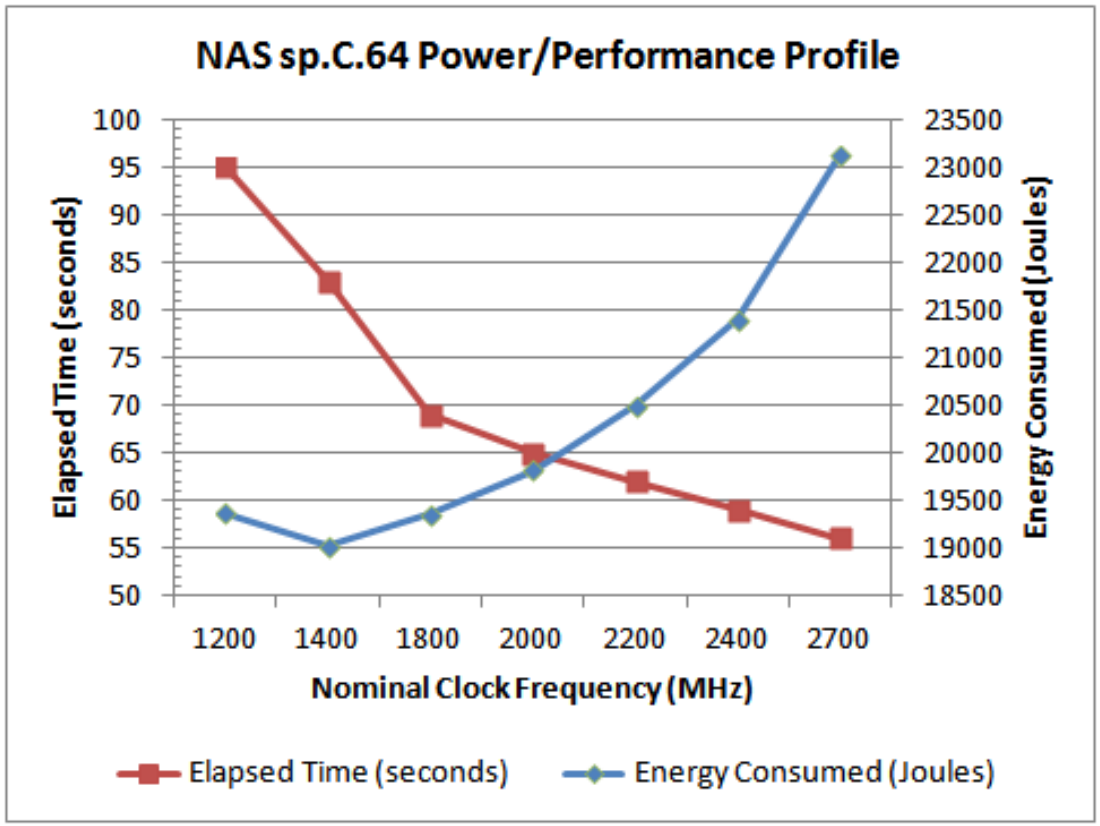
Chaque résidu local ne décroît pas nécessairement de manière monotone.
Le résidu peut augmenter suite à la réception d'un message et donc osciller autour du seuil d'arrêt.

Pour ne pas détecter une fausse convergence globale, un processeur compte le nb de fois consécutives où il se trouve sous le seuil de convergence.
Quand ce nombre atteint une valeur choisie par l'utilisateur, on a une convergence locale.

Algorithmes itératifs et consommation d'énergie

Importance des économies d'énergie dans le HPC (green computing)

Lien entre fréquence des CPU, temps d'exécution et énergie consommée



source : `http://docs.adaptivecomputing.com`

Compromis entre économie d'énergie et dégradation des performances

Puissance consommée

- puissance statique : consommée tant que la machine est allumée
- puissance dynamique : consommée pendant les calculs (pas pendant les communications)

DVFS (dynamic voltage and frequency scaling) : technique qui réduit la puissance dynamique en baissant la fréquence du CPU

Stratégies de réduction de l'énergie consommée particulièrement intéressantes pour les méthodes asynchrones.

- Applications asynchrones : en général plus performantes que les applications synchrones sur grilles
- Cependant comme elles effectuent plus d'itérations, elles sont plus consommatrices d'énergie.

La réduction de la fréquence peut varier d'un processeur à un autre dans un contexte hétérogène.

Dans un calcul itératif, des économies d'énergie peuvent être réalisées dès la 2e itération à partir d'informations collectées à la 1e itération.

Résolution sur grilles de systèmes linéaires

Résolution de systèmes linéaires

- La résolution de systèmes linéaires $Ax = b$ intervient dans de nombreux problèmes
- Utilisation de plusieurs processeurs afin d'accélérer les temps de calcul et traiter des problèmes de grande taille
- Il n'est pas question de calculer la matrice inverse A^{-1}, ce qui serait inutilement coûteux.
- 2 grandes classes de solveurs :
  - méthodes directes
  - méthodes itératives

Méthodes directes de résolution de systèmes linéaires (1/2)

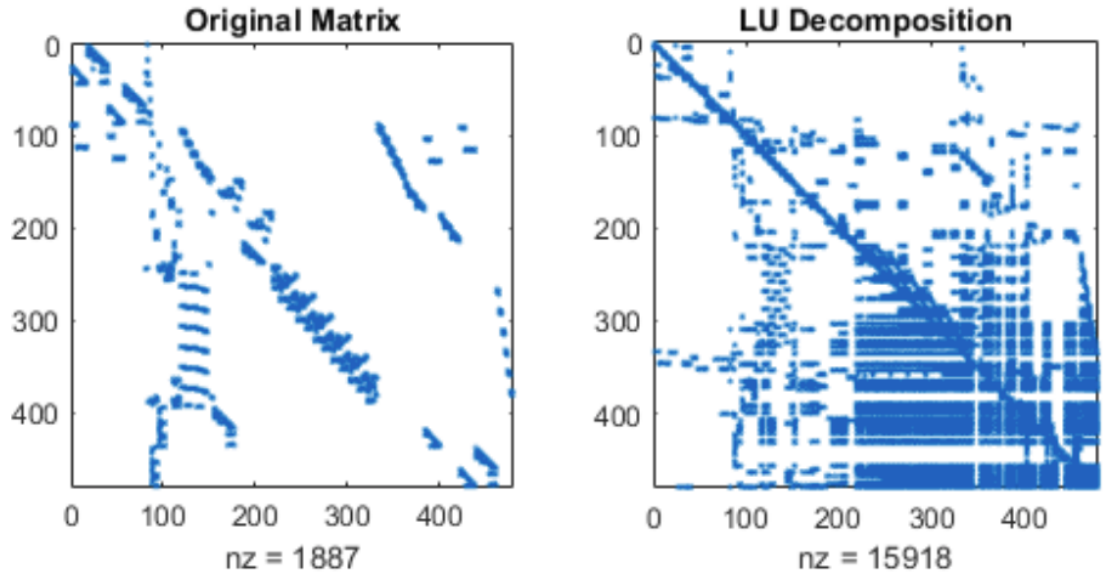
- Elles fournissent (en l'absence d'erreur d'arrondi) la solution exacte en un nombre fini d'opérations.
- 2 étapes :
  - factorisation de la matrice
  - résolution par substitutions successives

- Exemple : méthode fondée sur une décomposition $A = LU$
  L : matrice triangulaire inférieure formée de 1 sur la diagonale
  U : matrice triangulaire supérieure.
  $Ax = b \quad \Leftrightarrow \quad LUx = b$
  Soit $y = Ux$.
  On résout sucessivement 2 systèmes triangulaires : $Ly = b$,
  puis $Ux = y$.

Méthodes directes de résolution de systèmes linéaires (2/2)

- bien adaptées aux matrices pleines

  Une méthode directe appliquée à une matrice creuse peut engendrer des pb de *fill-in* (apparition de nouveaux éléments non nuls)



source : `https://fr.mathworks.com/help/matlab/ref/dissect.html`

  $\Rightarrow$ logiciels de réordonnancement (METIS, ParMETIS : `http://glaros.dtc.umn.edu/gkhome/views/metis`)

  Par ex, pour grouper les éléments autour de la diag. principale

- factorisation coûteuse, parallélisation difficile

Méthodes itératives de résolution de systèmes linéaires (1/2)

- on détermine à chaque itération $x^{k+1} = f(x^k)$
- exemples :
  - méthode de Jacobi, de Gauss-Seidel
    On décompose $A = M - N$

    $Ax = B \Leftrightarrow Mx = Nx + B$

    A partir d'un vecteur x^0, on calcule $x^{k+1} = M^{-1}Nx^k + M^{-1}b$.
  - méthodes de Krylov (CG, GMRES, BiCGStab)
    basées sur des projections sur un espace vectoriel de taille inférieure à celle de la matrice

Méthodes itératives de résolution de systèmes linéaires (2/2)

- bien adaptées aux matrices creuses
- parallélisation plus facile
- utilisation fréquente de préconditionneurs

  Au lieu de résoudre $Ax = b$, on résout $P^{-1}Ax = P^{-1}b$ avec P facilement inversible, par ex : $P = diag(A)$
  $\Rightarrow$ accélération de la convergence

Remarque : approche hybride (directe/itérative) pour les matrices structurées en blocs

Format de stockage des matrices creuses (1)

Considérons une matrice carrée de taille size et avec nnz éléments non nuls.

COO (COOrdinate list)
formé de trois tableaux de taille nnz : *rows*, *columns* et *values*.

Exemple :

$$\begin{bmatrix} 1 & 1 & -3 & 0 & 0 \\ -2 & 5 & 0 & 0 & 0 \\ 0 & 0 & 4 & 6 & 4 \\ -4 & 0 & 2 & 7 & 0 \\ 0 & 8 & 0 & 0 & -5 \end{bmatrix}$$

$$\begin{aligned} \textit{rows} &= \begin{bmatrix} 0 & 0 & 0 & 1 & 1 & 2 & 2 & 2 & 3 & 3 & 3 & 4 & 4 \end{bmatrix} \\ \textit{columns} &= \begin{bmatrix} 0 & 1 & 2 & 0 & 1 & 2 & 3 & 4 & 0 & 2 & 3 & 1 & 4 \end{bmatrix} \\ \textit{values} &= \begin{bmatrix} 1 & 1 & -3 & -2 & 5 & 4 & 6 & 4 & -4 & 2 & 7 & 8 & -5 \end{bmatrix} \end{aligned}$$

Format de stockage des matrices creuses (2)

CSR (Compressed Sparse Row) : le plus répandu

formé d'un tableau *pointer* de taille size+1 et de deux tableaux *columns* et *values* de taille nnz.

columns et *values* sont identiques à COO.
pointer donne la position dans le tableau *values* du 1er élément non nul d'une ligne
Par convention, $pointer[size] = nnz$.

Exemple :

$$\begin{bmatrix} 1 & 1 & -3 & 0 & 0 \\ -2 & 5 & 0 & 0 & 0 \\ 0 & 0 & 4 & 6 & 4 \\ -4 & 0 & 2 & 7 & 0 \\ 0 & 8 & 0 & 0 & -5 \end{bmatrix}$$

$$columns = \begin{bmatrix} 0 & 1 & 2 & 0 & 1 & 2 & 3 & 4 & 0 & 2 & 3 & 1 & 4 \end{bmatrix}$$
$$values = \begin{bmatrix} 1 & 1 & -3 & -2 & 5 & 4 & 6 & 4 & -4 & 2 & 7 & 8 & -5 \end{bmatrix}$$
$$pointer = \begin{bmatrix} 0 & 3 & 5 & 8 & 11 & 13 \end{bmatrix}$$

Format de stockage des matrices creuses (2)

CSR (Compressed Sparse Row) : le plus répandu

formé d'un tableau *pointer* de taille size+1 et de deux tableaux *columns* et *values* de taille nnz.

columns et *values* sont identiques à COO.
pointer donne la position dans le tableau *values* du 1er élément non nul d'une ligne
Par convention, $pointer[size] = nnz$.

Exemple :

$$\begin{bmatrix} 1 & 1 & -3 & 0 & 0 \\ -2 & 5 & 0 & 0 & 0 \\ 0 & 0 & 4 & 6 & 4 \\ -4 & 0 & 2 & 7 & 0 \\ 0 & 8 & 0 & 0 & -5 \end{bmatrix}$$

$$columns = \begin{bmatrix} 0 & 1 & 2 & 0 & 1 & 2 & 3 & 4 & 0 & 2 & 3 & 1 & 4 \end{bmatrix}$$

$$values = \begin{bmatrix} \mathbf{1} & 1 & -3 & -2 & 5 & 4 & 6 & 4 & -4 & 2 & 7 & 8 & -5 \end{bmatrix}$$

$$pointer = \begin{bmatrix} \mathbf{0} & 3 & 5 & 8 & 11 & 13 \end{bmatrix}$$

Format de stockage des matrices creuses (2)

CSR (Compressed Sparse Row) : le plus répandu

formé d'un tableau *pointer* de taille size+1 et de deux tableaux *columns* et *values* de taille nnz.

columns et *values* sont identiques à COO.
pointer donne la position dans le tableau *values* du 1er élément non nul d'une ligne
Par convention, $pointer[size] = nnz$.

Exemple :

$$\begin{bmatrix} 1 & 1 & -3 & 0 & 0 \\ -2 & 5 & 0 & 0 & 0 \\ 0 & 0 & 4 & 6 & 4 \\ -4 & 0 & 2 & 7 & 0 \\ 0 & 8 & 0 & 0 & -5 \end{bmatrix}$$

$$columns = \begin{bmatrix} 0 & 1 & 2 & 0 & 1 & 2 & 3 & 4 & 0 & 2 & 3 & 1 & 4 \end{bmatrix}$$

$$values = \begin{bmatrix} 1 & 1 & -3 & \mathbf{-2} & 5 & 4 & 6 & 4 & -4 & 2 & 7 & 8 & -5 \end{bmatrix}$$

$$pointer = \begin{bmatrix} 0 & \mathbf{3} & 5 & 8 & 11 & 13 \end{bmatrix}$$

Format de stockage des matrices creuses (2)

CSR (Compressed Sparse Row) : le plus répandu

formé d'un tableau *pointer* de taille size+1 et de deux tableaux *columns* et *values* de taille nnz.

columns et *values* sont identiques à COO.
pointer donne la position dans le tableau *values* du 1er élément non nul d'une ligne
Par convention, $pointer[size] = nnz$.

Exemple :

$$\begin{bmatrix} 1 & 1 & -3 & 0 & 0 \\ -2 & 5 & 0 & 0 & 0 \\ 0 & 0 & 4 & 6 & 4 \\ -4 & 0 & 2 & 7 & 0 \\ 0 & 8 & 0 & 0 & -5 \end{bmatrix}$$

$$columns = \begin{bmatrix} 0 & 1 & 2 & 0 & 1 & 2 & 3 & 4 & 0 & 2 & 3 & 1 & 4 \end{bmatrix}$$
$$values = \begin{bmatrix} 1 & 1 & -3 & -2 & 5 & \mathbf{4} & 6 & 4 & -4 & 2 & 7 & 8 & -5 \end{bmatrix}$$
$$pointer = \begin{bmatrix} 0 & 3 & \mathbf{5} & 8 & 11 & 13 \end{bmatrix}$$

Format de stockage des matrices creuses (3)

CSC (Compressed Sparse Column) aussi appelé format *Harwell-Boeing*
Idem que pour CSR, mais la matrice se lit en colonne.
pointer donne la position dans le tableau *values* du 1er élément non nul d'une colonne.

Exemple :

$$\begin{bmatrix} 1 & 1 & -3 & 0 & 0 \\ -2 & 5 & 0 & 0 & 0 \\ 0 & 0 & 4 & 6 & 4 \\ -4 & 0 & 2 & 7 & 0 \\ 0 & 8 & 0 & 0 & -5 \end{bmatrix}$$

$$rows = \begin{bmatrix} 0 & 1 & 3 & 0 & 1 & 4 & 0 & 2 & 3 & 2 & 3 & 2 & 4 \end{bmatrix}$$
$$values = \begin{bmatrix} 1 & -2 & -4 & 1 & 5 & 8 & -3 & 4 & 2 & 6 & 7 & 4 & -5 \end{bmatrix}$$
$$pointer = \begin{bmatrix} 0 & 3 & 6 & 9 & 11 & 13 \end{bmatrix}$$

Format de stockage des matrices creuses (3)

CSC (Compressed Sparse Column) aussi appelé format *Harwell-Boeing*
Idem que pour CSR, mais la matrice se lit en colonne.
pointer donne la position dans le tableau *values* du 1er élément non nul d'une colonne.

Exemple :

$$\begin{bmatrix} 1 & 1 & -3 & 0 & 0 \\ -2 & 5 & 0 & 0 & 0 \\ 0 & 0 & 4 & 6 & 4 \\ -4 & 0 & 2 & 7 & 0 \\ 0 & 8 & 0 & 0 & -5 \end{bmatrix}$$

$$rows = \begin{bmatrix} 0 & 1 & 3 & 0 & 1 & 4 & 0 & 2 & 3 & 2 & 3 & 2 & 4 \end{bmatrix}$$
$$values = \begin{bmatrix} \mathbf{1} & -2 & -4 & 1 & 5 & 8 & -3 & 4 & 2 & 6 & 7 & 4 & -5 \end{bmatrix}$$
$$pointer = \begin{bmatrix} \mathbf{0} & 3 & 6 & 9 & 11 & 13 \end{bmatrix}$$

Format de stockage des matrices creuses (3)

CSC (Compressed Sparse Column) aussi appelé format *Harwell-Boeing*
Idem que pour CSR, mais la matrice se lit en colonne.
pointer donne la position dans le tableau *values* du 1er élément non nul d'une colonne.

Exemple :

$$\begin{bmatrix} 1 & 1 & -3 & 0 & 0 \\ -2 & 5 & 0 & 0 & 0 \\ 0 & 0 & 4 & 6 & 4 \\ -4 & 0 & 2 & 7 & 0 \\ 0 & 8 & 0 & 0 & -5 \end{bmatrix}$$

$$rows = \begin{bmatrix} 0 & 1 & 3 & 0 & 1 & 4 & 0 & 2 & 3 & 2 & 3 & 2 & 4 \end{bmatrix}$$
$$values = \begin{bmatrix} 1 & -2 & -4 & \mathbf{1} & 5 & 8 & -3 & 4 & 2 & 6 & 7 & 4 & -5 \end{bmatrix}$$
$$pointer = \begin{bmatrix} 0 & \mathbf{3} & 6 & 9 & 11 & 13 \end{bmatrix}$$

Format de stockage des matrices creuses (3)

CSC (Compressed Sparse Column) aussi appelé format *Harwell-Boeing*
Idem que pour CSR, mais la matrice se lit en colonne.
pointer donne la position dans le tableau *values* du 1er élément non nul d'une colonne.

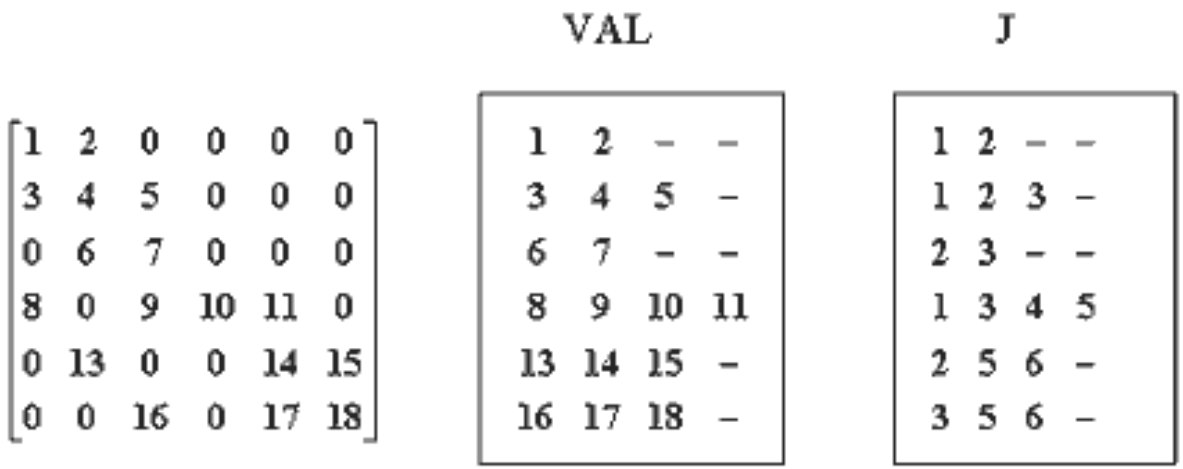
Exemple :

$$\begin{bmatrix} 1 & 1 & -3 & 0 & 0 \\ -2 & 5 & 0 & 0 & 0 \\ 0 & 0 & 4 & 6 & 4 \\ -4 & 0 & 2 & 7 & 0 \\ 0 & 8 & 0 & 0 & -5 \end{bmatrix}$$

$$rows = \begin{bmatrix} 0 & 1 & 3 & 0 & 1 & 4 & 0 & 2 & 3 & 2 & 3 & 2 & 4 \end{bmatrix}$$
$$values = \begin{bmatrix} 1 & -2 & -4 & 1 & 5 & 8 & \mathbf{-3} & 4 & 2 & 6 & 7 & 4 & -5 \end{bmatrix}$$
$$pointer = \begin{bmatrix} 0 & 3 & \mathbf{6} & 9 & 11 & 13 \end{bmatrix}$$

Format de stockage des matrices creuses (4)

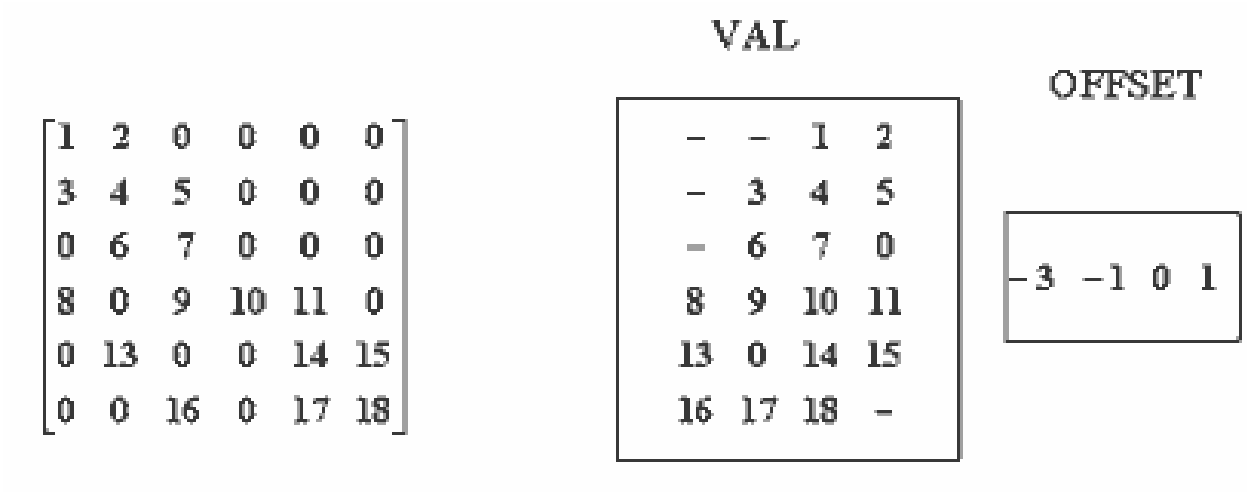
ELL (ELLPACK/ITPACK) : pour stocker une matrice creuse avec compression de lignes ou de colonnes.
On stocke 2 tableaux de petite taille : une matrice de valeurs (par ex les lignes successives compressées) et une matrice d'indices (par ex de colonnes).



source : `http://icis.pcz.pl`

Format de stockage des matrices creuses (5)

DIA : pour compresser des matrices creuses dont les éléments non nuls sont situés sur un petit nombre de diagonales.
On stocke dans un tableau 2D les diagonales de la matrice et dans un tableau 1D les décalages par rapport à la diagonale principale.



source : `http://icis.pcz.pl`

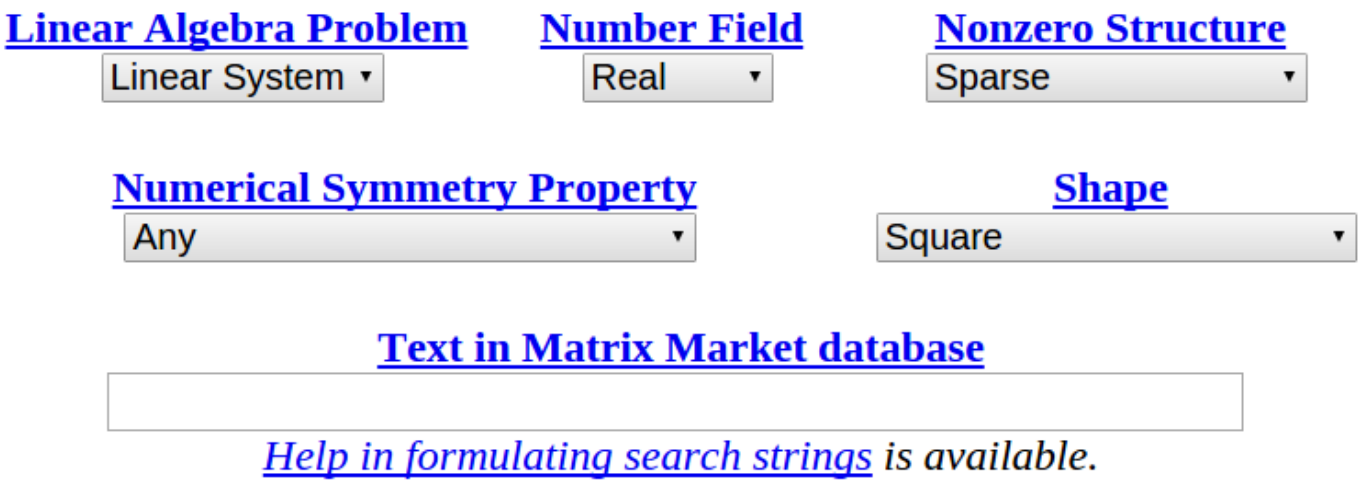
Tester les performances d'un solveur

exemples de collections de matrices

- University of Florida sparse matrix collection :
  `http://www.cise.ufl.edu/research/sparse/matrices/`
- MatrixMarket :
  `http://math.nist.gov/MatrixMarket/`

A visual repository of test data for use in comparative studies of algorithms for numerical linear algebra, featuring nearly 500 sparse matrices from a variety of applications, as well as matrix generation tools and services.

Critères de recherche dans “Matrix Market”

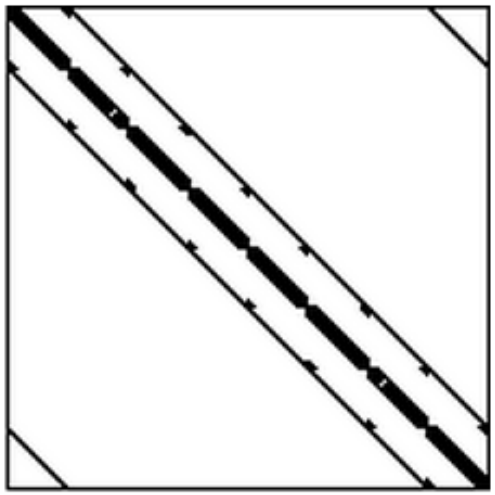


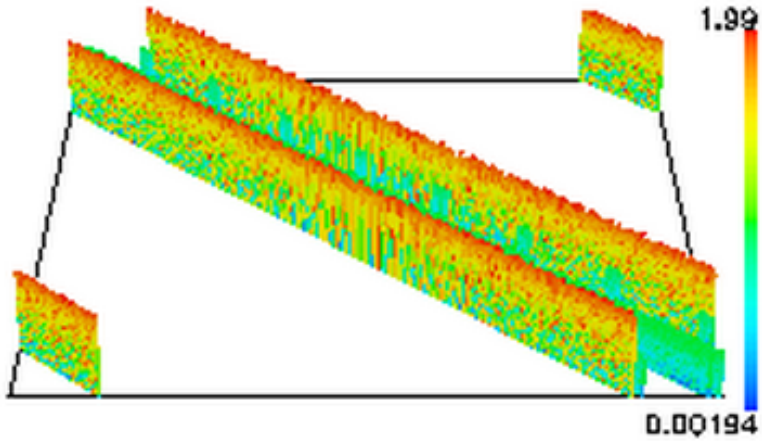
+ critères sur la taille, le format de stockage,...

Exemple de matrice proposée par “Matrix Market”

CONF6.0-00L8X8-8000 : Quantum Chromodynamics

Structure Plot	City Plot





“Matrix Statistics” (extrait) :
Dimension M = 49152
Dimension N = 49152
Number of matrix entries = 1916928
Average number of nonzero elements/Row = 39.00

Quelques bibliothèques pour la résolution de systèmes linéaires (1)

- PETSc (Portable, Extensible Toolkit for Scientific Computation)
  - solveurs itératifs (Krylov) ou solveurs directs, mais via d'autres bibliothèques (SuperLU, MUMPS)
  - MPI, GPUs (via CUDA ou OpenCL), parallélisme hybride MPI-GPU
  - même code quel que soit l'environnement (séquentiel ou parallèle)
    ex : fonction KSPSolve
  - `http://www.mcs.anl.gov/petsc`
- Sparselib++ (pour matrices creuses)
  - itératif
  - séquentiel, thread-safe : possibilité d'exécution simultanée par plusieurs threads (processus légers).
  - `http://math.nist.gov/sparselib`

Quelques bibliothèques pour la résolution de systèmes linéaires (2)

- MUMPS (MUltifrontal Massively Parallel sparse direct Solver)
  - direct, interfacé avec différents logiciels de réordonnancement
  - MPI et OpenMP
  - `http://graal.ens-lyon.fr/MUMPS`
- SuperLU
  - direct
  - mémoire partagée (threads POSIX et OpenMP)
    mémoire distribuée (MPI)
  - `http://crd.lbl.gov/~xiaoye/SuperLU`

Pas de bibliothèque adaptée aux grilles...

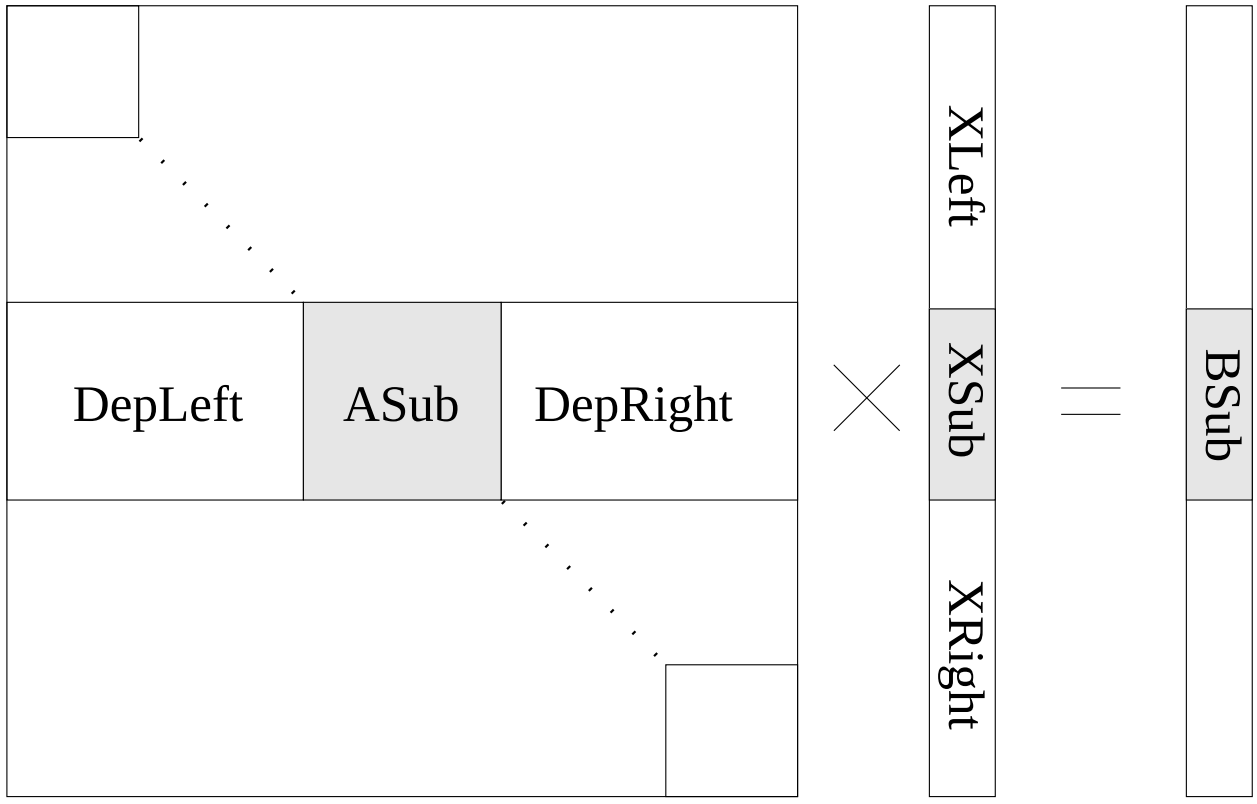
Résolution de systèmes linéaires sur grilles

Un algorithme efficace sur grille doit être à gros grain et asynchrone.

Multisplitting (multidécomposition) :

- Découper le système linéaire en matrices rectangles et itérer :
  - résoudre chaque système de manière indépendante
  - échanger des dépendances

Décomposition du système AX=B



$DepLeft * XLeft + ASub * XSub + DepRight * XRight = BSub$

$\Rightarrow$ À chaque itération un processeur résout : $ASub * XSub = BLoc$
avec $BLoc = BSub - DepLeft * XLeft - DepRight * XRight$

Principe de l'algorithme

1. **Initialisation**
   La matrice A provient d'un fichier ou est générée à l'exécution.
   Chaque processeur charge la matrice rectangulaire
   $[DepLeft, ASub, DepRight]$
   puis effectue de manière itérative les étapes suivantes :
2. **Calcul**
   Chaque processeur calcule
   $BLoc = BSub - DepLeft * XLeft - DepRight * XRight$
   puis résout $ASub * XSub = BLoc$.
3. **Echange des résultats**
   Chaque processeur :
   - envoie $XSub$ aux autres processeurs.
   - reçoit des parties du vecteur solution et met donc à jour ses vecteurs $XLeft$ et $XRight$.
4. **Détection de la convergence**

Caractéristiques de la méthode

- Chaque sous-système est résolu en utilisant un algorithme séquentiel
- Si on utilise une méthode directe la factorisation est faite une seule fois
- Différents solveurs peuvent utilisés (même simultanément)

Conditions de convergence

- version synchrone : rayon spectral de la matrice d'itération strictement inférieur à 1
- version asynchrone : rayon spectral de la valeur absolue de la matrice d'itération strictement inférieur à 1

La convergence est assurée si la matrice A est à diagonale dominante : $\forall i, |a_{i,i}| \geq \sum_{j=1, j\neq i}^{n} |a_{i,j}|$

Elle l'est aussi pour les Z matrices, matrices à éléments diagonaux > 0 et éléments hors diagonaux ≤ 0.
Ces matrices interviennent dans les domaines de la physique, la biologie ou les sciences sociales.

Recouvrement

La décomposition de la matrice en bandes n'implique pas que les bandes soient disjointes.
Lorsqu'elles ont des parties communes, une partie des données est calculée par au moins deux processeurs.

L'intérêt est de diminuer le nombre d'itérations en contrepartie d'un temps de calcul moyen d'itération plus important.

Mise en œuvre du multisplitting sur grilles

Le code GREMLINS (GRid Efficient Methods for LINear Systems)

`http://info.iut-bm.univ-fcomte.fr/gremlins`

Solveurs internes (résolution séquentielle des sous-systèmes) : utilisation de bibliothèques

- PETSc, Sparselib (solveurs itératifs)
- MUMPS, SuperLU (solveurs directs)

Solveur externe : méthode itérative (multisplitting)
même code quelle que soit la bibliothèque choisie

$\Rightarrow$ conversion des sous-matrices pour les adapter au format imposé par la bibliothèque choisie

Mise en œuvre du multisplitting sur grilles

Le code GREMLINS (GRid Efficient Methods for LINear Systems)

`http://info.iut-bm.univ-fcomte.fr/gremlins`

Solveurs internes (résolution séquentielle des sous-systèmes) : utilisation de bibliothèques

- PETSc, Sparselib (solveurs itératifs)
- MUMPS, SuperLU (solveurs directs)

Solveur externe : méthode itérative (multisplitting)
même code quelle que soit la bibliothèque choisie

$\Rightarrow$ conversion des sous-matrices pour les adapter au format imposé par la bibliothèque choisie

Grâce à un paramètre, exécution synchrone ou asynchrone :

- En mode synchrone, tous les messages reçus sont sauvegardés.
- En mode asynchrone, si plusieurs messages sont reçus de même provenance et même identifiant, seul le dernier message est sauvegardé.

Implantation du multisplitting sur GRID5000

Selon le site : différentes versions de MPI (MPI-LAM, MPICH, MPICH2, OpenMPI), différents compilateurs

- réservation de noeuds
- déploiement sur tous les noeuds de la même image linux qui contient le code GREMLINS, les 4 bibliothèques scientifiques, la bibliothèque de communication, les compilateurs,...

Génération de la matrice

- **A partir d'un fichier**
  sites spécialisés :
  - MatrixMarket :
    `http://math.nist.gov/MatrixMarket/`
  - University of Florida sparse matrix collection :
    `http://www.cise.ufl.edu/research/sparse/matrices/`

  Le fichier est stocké sur un processeur qui distribue les données aux autres processeurs.
  La mémoire vive du processeur limite la taille du fichier qui peut être traité.
  Largeur maximale de matrice testée : 130 228 sur au plus 20 processeurs.

Génération de la matrice

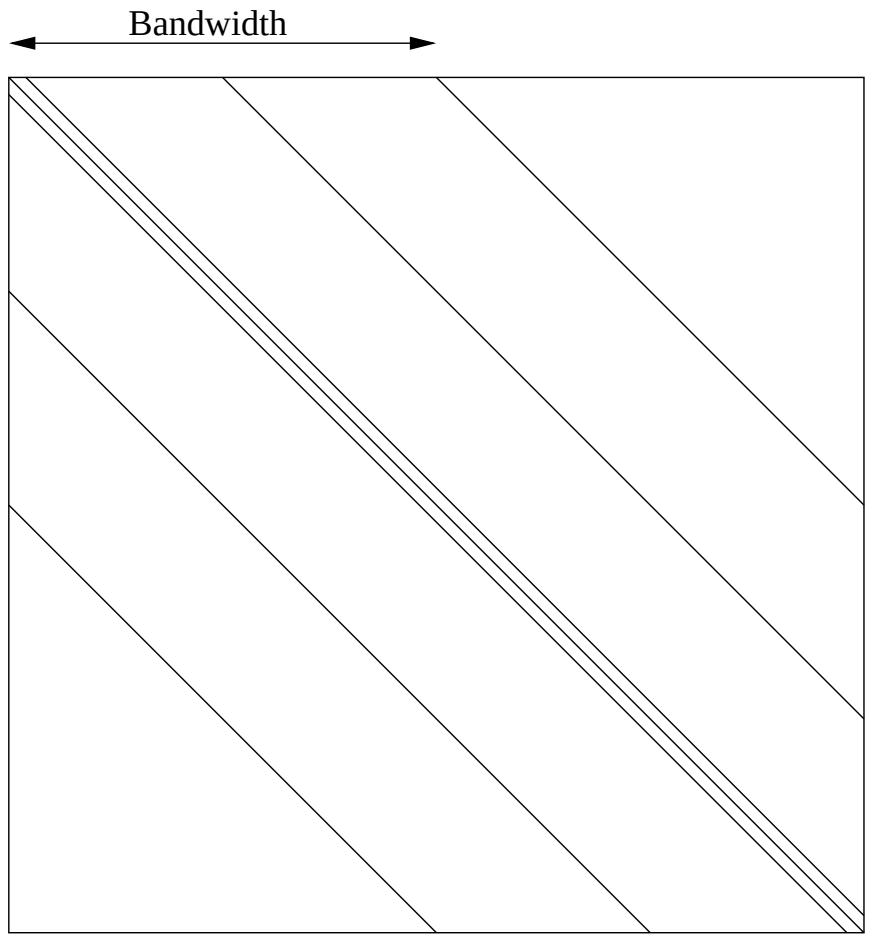
- **A partir d'un fichier**
  sites spécialisés :
  - MatrixMarket :
    `http://math.nist.gov/MatrixMarket/`
  - University of Florida sparse matrix collection :
    `http://www.cise.ufl.edu/research/sparse/matrices/`

  Le fichier est stocké sur un processeur qui distribue les données aux autres processeurs.
  La mémoire vive du processeur limite la taille du fichier qui peut être traité.
  Largeur maximale de matrice testée : 130 228 sur au plus 20 processeurs.
- **A l'exécution**
  Pour résoudre de plus gros problèmes en évitant d'importants transferts de fichiers, on génère le système linéaire à l'exécution.
  Chaque processeur génère la sous-matrice rectangulaire qu'il va traiter.

Génération des matrices à l'exécution

Z matrices générées aléatoirement (convergence en mode synchrone et asynchrone)

Exemple de matrice avec 7 diagonales et une largeur de bande égale à la moitié de la taille de la matrice :



Impact de la matrice sur les performances

Plus le rapport *tps communication / tps calcul* **augmente, plus la version asynchrone est favorisée.**

Plus la taille de la matrice ou le nb de diagonales diminue,

- plus le tps de calcul diminue
- plus la version asynchrone est favorisée.

Plus la largeur de bande augmente,

- plus le tps de communication augmente,
- plus la version asynchrone est favorisée.

Remarques :

- Sur un cluster local, la version synchrone peut être plus rapide.
- Sur Grid5000, le réseau est meilleur que sur une grille traditionnelle.

Expérimentations (1)

120 machines : 40 Rennes, 40 Paris, 25 Nancy et 15 Lille (solveur MUMPS)

Taille des matrices	Nombre de diag.	Largeur de bande	Synchrone		Asynchrone	
			temps d'exec (s)	nb. iter.	temps d'exec (s)	nb. iter.
1 000 000	23	1 000	10.05	72	4.55	[303-475]
2 000 000	23	2 000	14.98	69	5.39	[195-229]
4 000 000	23	4 000	19.33	68	12.31	[204-268]
6 000 000	23	6 000	24.19	69	13.34	[146-176]
8 000 000	23	8 000	27.87	68	18.18	[142-143]
10 000 000	23	5 000	28.10	67	22.22	[136-144]

Plus la taille de la matrice augmente,

- plus le tps de calcul augmente,
- plus la version synchrone est favorisée.

Expérimentations (2)

matrice de taille 20 000 000 avec 200 bi-processeurs : 120 à Paris et 80 à Nice (solveur SparseLib)

Nombre de diagonales	Largeur de bande	Synchrone		Asynchrone	
		temps d'exec (s)	nb. iter.	temps d'exec (s)	nb. iter.
13	300 000	132.51	134	87.14	[634-859]
23	300 000	163.88	141	104.37	[576-809]
13	3 000 000	353.80	142	245.68	[980-1279]

Plus le nb de diagonales augmente,

- plus le tps de calcul augmente,
- plus la version synchrone devrait être favorisée.

Plus le tps de comm. augmente, plus la version asynchrone devrait être favorisée...

Expérimentations (3) : comparaison avec PETSc contexte local

matrice de taille 20 000 000, largeur de bande 2 000 000, sur 100 processeurs à Orsay

Nb. diagonals	Multisplitting (MUMPS) Synchronous time (s)	Multisplitting (MUMPS) Synchronous nb. iter.	Multisplitting (MUMPS) Asynchronous time (s)	Multisplitting (MUMPS) Asynchronous nb. iter.	PETSc time (s)	PETSc nb. iter.
13	15.50	12	20.59	[54-56]	17.56	12
23	32.04	17	39.56	[66-72]	24.28	14
33	42.11	21	48.90	[81-82]	27.69	15
43	54.95	25	58.65	[78-81]	30.58	16
53	62.49	28	66.48	[97-100]	34.13	17
63	73.40	32	76.33	[104-110]	37.78	18

Expérimentations (4) : comparaison avec PETSc

contexte distant

matrice de taille 20 000 000, largeur de bande 20 000,
avec 198 machines : 60 à Nice 68 à Paris et 70 à Rennes

Nombre de diagonales	Multidécomposition (MUMPS) Synchrone		Multidécomposition (MUMPS) Asynchrone		PETSc	
	temps d'exec (s)	nb. iter.	temps d'exec (s)	nb. iter.	temps d'exec (s)	nb. iter.
13	20.76	37	23.14	[172-198]	42.10	47
23	27.56	46	33.02	[215-245]	49.09	54
33	38.71	57	33.58	[169-186]	55.41	60
43	51.48	68	43.50	[173-189]	57.75	68
53	65.03	78	53.58	[187-204]	69.20	71
63	75.04	91	72.44	[243-286]	76.92	80

- Performances du multiplitting généralement meilleures que celles de PETSc dans un contexte distant
- Peu d'influence du solveur interne (direct ou itératif) sur les performances

Multisplitting et GPU

Les algorithmes asynchrones permettent le recouvrement implicite des communications et des calculs... et les communications CPU-GPU sont pénalisantes !

Décomposition du système linéaire en sous-systèmes

- solveur interne : méthode itérative (GMRES) sur plusieurs CPU et/ou GPU
- solveur externe : multisplitting

nombre optimal de sous-systèmes ?

Multisplitting vs GMRES sur GPU

Architecture (nb grappes $\times$ nb GPU/grappe)	Multisplitting (+GMRES)	
	T_{GMRES}/T_{sync}	T_{GMRES}/T_{async}
2×5	1.21	1.56
5×2	1.10	1.48
10×1	1.09	1.08

Multisplitting plus intéressant avec des sous-systèmes de grande taille.

[thèse L. Ziane Khodja, univ. de Franche-Comté, 2013]

Travaux connexes

- augmentation de la charge du réseau en faveur de l'asynchronisme

Travaux connexes

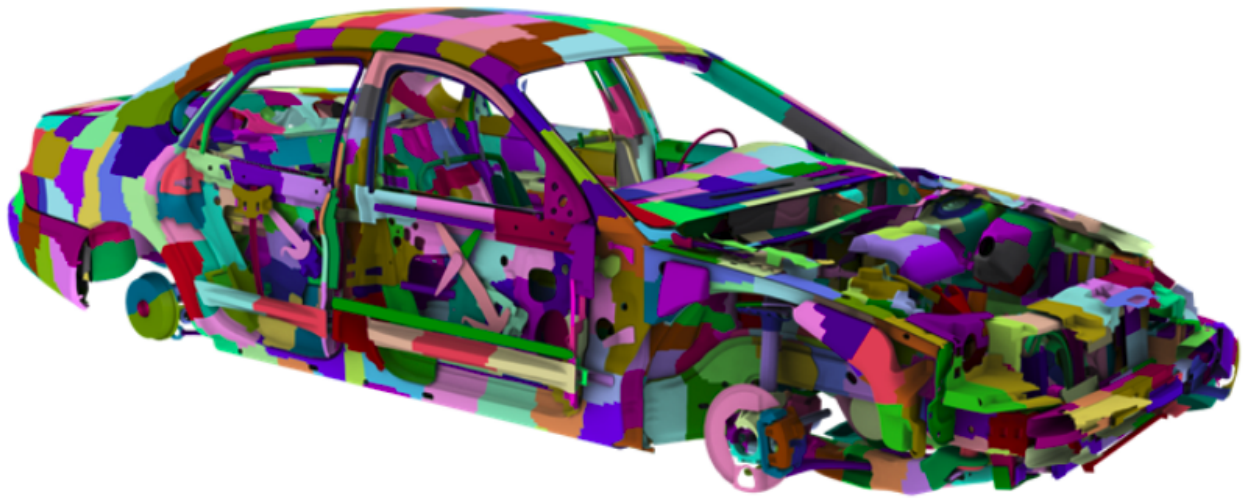
- augmentation de la charge du réseau en faveur de l'asynchronisme
- implantation en Java
  - sur grille : le déploiement d'une image linux n'est plus nécessaire

Travaux connexes

- augmentation de la charge du réseau en faveur de l'asynchronisme
- implantation en Java
  - sur grille : le déploiement d'une image linux n'est plus nécessaire
- déploiement plus convivial (par exemple obtention du nb de processeurs désiré)

Travaux connexes

- augmentation de la charge du réseau en faveur de l'asynchronisme
- implantation en Java
  - sur grille : le déploiement d'une image linux n'est plus nécessaire
- déploiement plus convivial (par exemple obtention du nb de processeurs désiré)
- méthodes de décomposition de domaines avec itérations asynchrones



source : http://industry.it4i.cz

Perspectives

Exécution optimale ?

- choix :
  - synchrone/asynchrone
  - CPU/GPU
  - temps d'exécution/consommation d'énergie
  - formats de stockage
- guidé par :
  - heuristique
  - modèle des performances (énergie et temps)
  - benchmarks élémentaires
- effectué par :
  - utilisateur
  - scheduler
  - algos d'IA

L'asynchronisme : une nécessité pour les simulations à large échelle

Critical Issues at Peta & Exascale for Algorithm and Software Design

- Synchronization-reducing algorithms
  - Break Fork-Join model
- Communication-reducing algorithms
  - Use methods which have lower bound on communication
- Mixed precision methods
  - 2x speed of ops and 2x speed for data movement
- Autotuning
  - Today's machines are too complicated, build "smarts" into software to adapt to the hardware
- Fault resilient algorithms
  - Implement algorithms that can recover from failures/bit flips
- Reproducibility of results
  - Today we can't guarantee this. We understand the issues, but some of our "colleagues" have a hard time with this.

Ref : Jack Dongarra, "Algorithmic and Software Challenges when Moving Towards Exascale"