

# Multi- $\{\text{échelles, résolution, cœurs}\}$

Thierry Dumont

–Institut Camille Jordan–

LyonCalcul, le 16 Avril 2013.

Quatre parties :

- ① Les problèmes qui nous ont intéressés. *Aspect maths, modélisation* .
- ② Discrétisation en temps. *Méthodes numériques, implantation, parallélisme* .
- ③ La multirésolution. *Implantation, structures de données* .
- ④ Nouveaux paradigmes de programmation pour machines {multi,many}-core. *Vol de tâches* .

Quatre parties :

- ① Les problèmes qui nous ont intéressés. *Aspect maths, modélisation* .
- ② Discrétisation en temps. *Méthodes numériques, implantation, parallélisme* .
- ③ La multirésolution. *Implantation, structures de données* .
- ④ Nouveaux paradigmes de programmation pour machines {multi,many}-core. *Vol de tâches* .

ANR Sechelles :

M. Massot (ECP), S. Descombes (Nice), Max Duarte (ECP, Nice, Lyon),  
C. Renaud (Limsi), Z. Bonaventura (Brno), V.Louvet, T.D. etc....

PEPS : (MCC (AMIES)) collaboration avec Intel.

## Problèmes multi échelles.

- Échelles de temps très différentes.
- Forts gradients spatiaux (mobiles).

# Échelles de temps très différentes

Exemple :

$$\frac{du}{dt} = F(u), \quad u \in \mathbb{R}^n$$

# Échelles de temps très différentes

Exemple :

$$\frac{du}{dt} = F(u), \quad u \in \mathbb{R}^n$$

$$\frac{du_1}{dt} = F_1(u_1, u_2),$$

$$\frac{du_2}{dt} = \frac{1}{\varepsilon} F_2(u_1, u_2).$$

Le système approche *très vite* de la variété d'équilibre partiel  $F_2(u_1, u_2) = 0$ .

# Échelles de temps très différentes

Exemple :

$$\frac{du}{dt} = F(u), \quad u \in \mathbb{R}^n$$

$$\frac{du_1}{dt} = F_1(u_1, u_2),$$

$$\frac{du_2}{dt} = \frac{1}{\varepsilon} F_2(u_1, u_2).$$

Le système approche *très vite* de la variété d'équilibre partiel

$$F_2(u_1, u_2) = 0.$$

Exemple typique : réactions chimiques (mais la variété d'équilibre partiel est inconnue).

## Échelles de temps très différentes

$J$ , matrice Jacobienne :

$$J_{i,j} = \frac{\partial F_i}{\partial u_j}$$

Les valeurs propres de  $J$  sont les inverses des *échelles de temps* du système.



## Échelles de temps très différentes

$J$ , matrice Jacobienne :

$$J_{i,j} = \frac{\partial F_i}{\partial u_j}$$

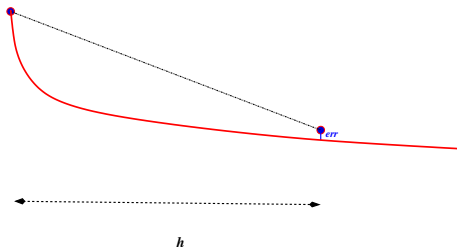
Les valeurs propres de  $J$  sont les inverses des *échelles de temps* du système.

Chimie complexe : 100 équations,  $|\lambda_{\max}|/|\lambda_{\min}| \simeq 10^8$ .

Systèmes raides (stiff).

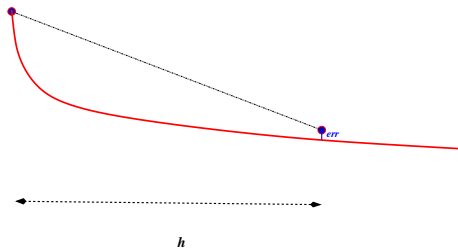
## Que veut on calculer ?

On n'est pas intéressés par la simulation des échelles de temps les plus rapides, mais on veut contrôler l'erreur sur un intervalle de temps  $h$ .



## Que veut on calculer ?

On n'est pas intéressés par la simulation des échelles de temps les plus rapides, mais on veut contrôler l'erreur sur un intervalle de temps  $h$ .



Méthodes numériques adaptées, coûteuses :

- Runge-Kutta implicites,
- Contrôle de l'erreur.

# Un problème raide de tous les jours

$$\frac{\partial u}{\partial t}(x, t) - \varepsilon \Delta u(x, t) = 0.$$

Discrétisation en différences finies de pas  $\delta x$  :  
*raideur* =  $\varepsilon / \delta x^2$ .

Indépendant de la dimension d'espace...

# Échelles spatiales très différentes

Exemple : forts gradients localisés.

# Échelles spatiales très différentes

Exemple : forts gradients localisés.

Problèmes de Réaction–Diffusion :

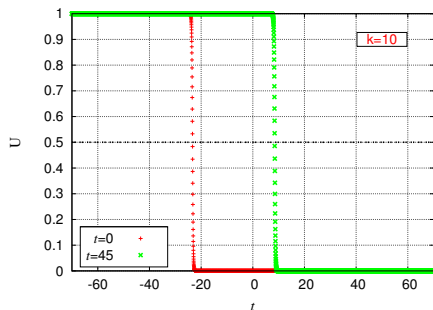
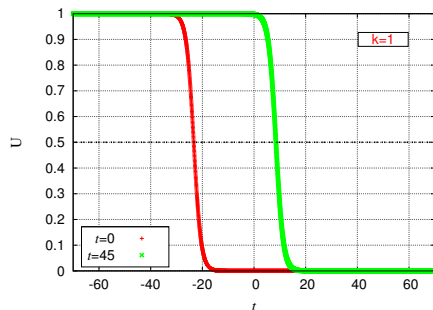
$$\begin{cases} \frac{\partial u_i}{\partial t}(x, t) - \varepsilon \Delta u_i(x, t) = f_i(u_1(x, t), \dots, u_m(x, t)), & 1 \leq i \leq m, x \in \Omega, \\ u_i(x, 0) = u_i^0(x), & 1 \leq i \leq m, x \in \Omega. \end{cases}$$

Chimie complexe.

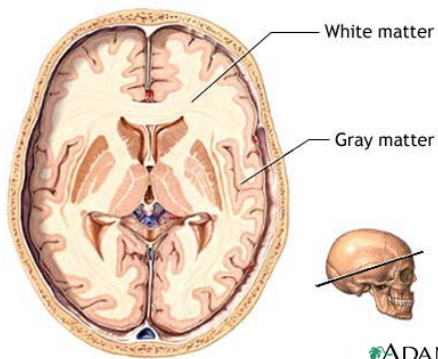
# Réaction-Diffusion

KPP (Kolmogorov-Petrovsky-Piskounov) :

$$\frac{\partial u}{\partial t}(x, t) - \varepsilon \Delta u(x, t) = ku(x, t)^2(1 - u(x, t)).$$



## Autre exemple

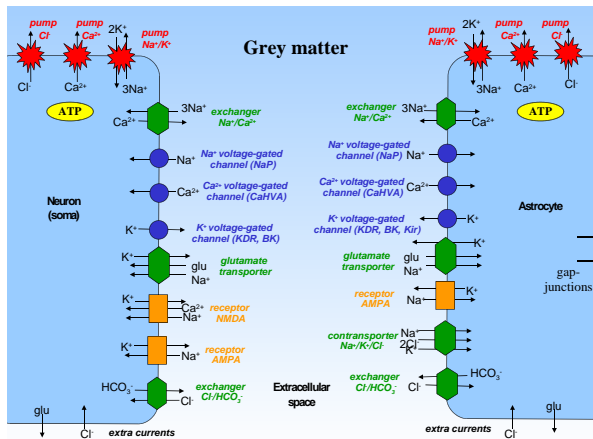


Accidents vasculaires cérébraux ischémiques.

M.A. Dronne, J.P. Boissel & E. Grenier.



# Autre exemple



Accidents vasculaires cérébraux ischémiques.  
M.A. Dronne, J.P. Boissel et E. Grenier.

Système de 20 équations de Réaction–Diffusion, raide ( $10^8$ ).

# Positive Streamer Simulations

$$\partial_t n_e - \partial_x \cdot n_e \mathbf{v}_e - \partial_x \cdot (D_e \partial_x n_e) = n_e \alpha |\mathbf{v}_e| - n_e \eta |\mathbf{v}_e| + n_e n_p \beta_{ep} + n_n \gamma$$

$$\partial_t n_p + \partial_x \cdot n_p \mathbf{v}_p - \partial_x \cdot (D_p \partial_x n_p) = n_e \alpha |\mathbf{v}_e| - n_e n_p \beta_{ep} + n_n n_p \beta_{np}$$

$$\partial_t n_n - \partial_x \cdot n_n \mathbf{v}_n - \partial_x \cdot (D_n \partial_x n_n) = n_e \eta |\mathbf{v}_e| - n_n n_p \beta_{np} - n_n \gamma$$

$$\varepsilon_0 \partial_x^2 V = -q_e (n_p - n_n - n_e) \quad \mathbf{E} = -\partial_x V$$

Décharges pulsées périodiques :

Voltage : 0 → 13 kV

Durée des pulses : 15 ns - Période : 100 μs

Une méthode vieille comme le calcul scientifique : les directions alternées.

$$\begin{cases} \frac{\partial u_i}{\partial t}(x, t) - \varepsilon \Delta u_i(x, t) = f_i(u_1(x, t), \dots, u_m(x, t)), & 1 \leq i \leq m, x \in \Omega, \\ u_i(x, 0) = u_i^0(x), & 1 \leq i \leq m, x \in \Omega. \end{cases}$$







# Questions

- 1 Est-ce adapté au cas raide?
- 2  $h$ ? Rapport avec les échelles de temps du problème?
- 3 Les 2 schémas de Strang?

# Questions

- 1 Est-ce adapté au cas raide?
  - 2  $h$ ? Rapport avec les échelles de temps du problème?
  - 3 Les 2 schémas de Strang?
- 
- 1 Oui, si les échelles rapides sont *filtrées* par les sous pas.
  - 2 Le pas  $h$  n'est pas limité par les échelles rapides.
  - 3 finir avec les problèmes les plus raides.



# Implantation

- 1 Réaction : parallélisme trivial, *mais* :
  - coût très variable selon la distance à l'équilibre,
  - méthodes adaptées, coûteuses (RADAU5).

# Implantation

① Réaction : parallélisme trivial, *mais* :

- coût très variable selon la distance à l'équilibre,
- méthodes adaptées, coûteuses (RADAU5).

② Diffusion :

échelles de temps relativement peu rapides (petit coefficient de diffusion  $\varepsilon$ )  $\Rightarrow$  méthode Rock4.

# Implantation

- 1 Réaction : parallélisme trivial, *mais* :
  - coût très variable selon la distance à l'équilibre,
  - méthodes adaptées, coûteuses (RADAU5).
- 2 Diffusion :

échelles de temps relativement peu rapides (petit coefficient de diffusion  $\varepsilon$ )  $\Rightarrow$  méthode Rock4.

Rock4 : méthodes de Runge-Kutta *explicite*, *mais à grand domaine de stabilité*  $\Rightarrow$  que des produits matrices vecteurs  $\Rightarrow$  parallélisme important.

# Et les échelles spatiales ?

On sait résoudre correctement les échelles temporelles, **qu'en est-il des échelles spatiales ?**

## Et les échelles spatiales ?

On sait résoudre correctement les échelles temporelles, **qu'en est-il des échelles spatiales ?**

Expérience numérique (Duarte, Massot) :

Discrétisation spatiale du problème (AVC) en dimension 1  $\Rightarrow$  système d'équations différentielles ordinaires assez *petit*  $\Rightarrow$  résolution avec un solveur d'EDO d'ordre élevé

## Et les échelles spatiales ?

On sait résoudre correctement les échelles temporelles, **qu'en est-il des échelles spatiales ?**

Expérience numérique (Duarte, Massot) :

Discrétisation spatiale du problème (AVC) en dimension 1  $\Rightarrow$  système d'équations différentielles ordinaires assez *petit*  $\Rightarrow$  résolution avec un solveur d'EDO d'ordre élevé  $\Rightarrow$  solution « exacte » du système discrétisé en espace.

AVC : au moins 1000 points...

En dimension 3,  $10^9$  points !

# Multirésolution.

## Mailler finement où c'est nécessaire

Plusieurs approches (éléments finis adaptatifs, AMR, Multirésolution).



## Mailler finement où c'est nécessaire

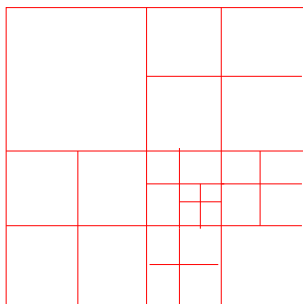
Plusieurs approches (éléments finis adaptatifs, AMR, Multirésolution).

Multirésolution : un solide fondement théorique basé sur les ondelettes.

## Mailler finement où c'est nécessaire

Plusieurs approches (éléments finis adaptatifs, AMR, Multirésolution).

Multirésolution : un solide fondement théorique basé sur les ondelettes.



# L'approche multirésolution

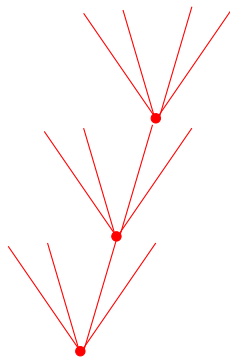
- Quadrees en dimension 2, octrees en dimension 3.

# L'approche multirésolution

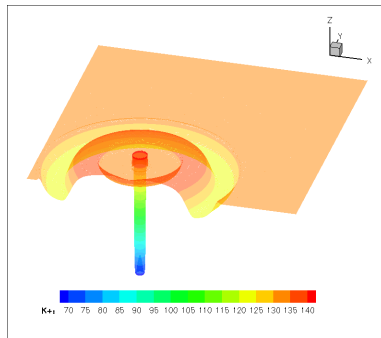
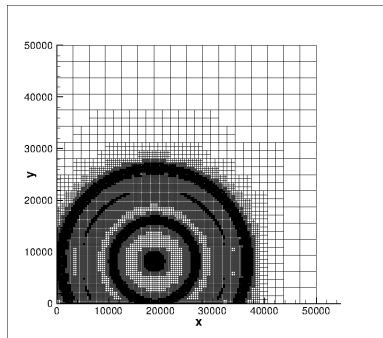
- Quadrees en dimension 2, octrees en dimension 3.
- Un ensemble de **grilles emboîtées**, de la plus grossière à la plus fine.
- Des valeurs aux feuilles *et* aux nœuds de l'arbre (== sur toutes les grilles emboîtées).

# L'approche multirésolution

- Quadrees en dimension 2, octrees en dimension 3.
- Un ensemble de **grilles emboîtées**, de la plus grossière à la plus fine.
- Des valeurs aux feuilles *et* aux nœuds de l'arbre (== sur toutes les grilles emboîtées).



# L'approche multirésolution



# L'approche multirésolution

Ondelettes : valeurs sur une grille = valeurs sur la grille inférieure + détail.

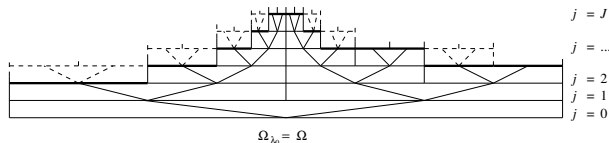
# L'approche multirésolution

Ondelettes : valeurs sur une grille = valeurs sur la grille inférieure + détail.  
Estimer les *détails* pour gérer les raffinements de maillage.



# L'approche multirésolution

Ondelettes : valeurs sur une grille = valeurs sur la grille inférieure + détail.  
Estimer les *détails* pour gérer les raffinements de maillage.



Des transformations entre grilles de différents niveaux :

- **projection** du niveau  $j$  vers le niveau  $j - 1$  :  $R_{j,j-1}$
- **prédiction** du niveau  $j - 1$  vers le niveau  $j$  :  $P_{j-1,j}$

# L'approche multirésolution

- Consistance :  $R_{j,j-1} \circ P_{j-1,j} = Id$
- Détails :  $d = (Id - P_{j-1,j} \circ R_{j,j-1})u$

# L'approche multirésolution

- Consistance :  $R_{j,j-1} \circ P_{j-1,j} = Id$
- Détails :  $d = (Id - P_{j-1,j} \circ R_{j,j-1})u$

## Idée

Raffiner le maillage là où les détails sont trop grands... et le grossir là où ils sont trop petits.

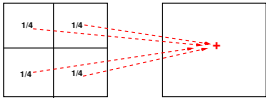
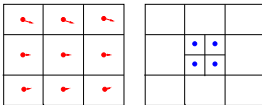
# L'approche multirésolution

- **Consistance** :  $R_{j,j-1} \circ P_{j-1,j} = Id$
- **Détails** :  $d = (Id - P_{j-1,j} \circ R_{j,j-1})u$

## Idée

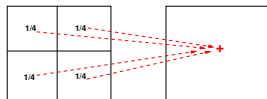
Raffiner le maillage là où les détails sont trop grands... et le grossir là où ils sont trop petits.

Différents choix possibles pour  $R_{j,j-1}$  et  $P_{j-1,j}$  :

- $R_{j,j-1}$  : **moyenne**  

- $P_{j-1,j}$  : **interpolation quadratique**  


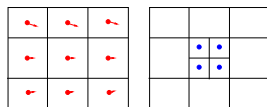
# L'approche multirésolution

- $R_{j,j-1}$  :



Il faut seulement regarder les frères et le père.

- $P_{j-1,j}$  :

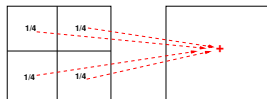


Regarder les frères, cousins et oncles .

C'est la principale difficulté quant à l'implantation !

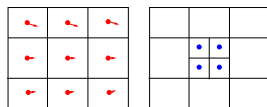
# L'approche multirésolution

- $R_{j,j-1}$  :



Il faut seulement regarder les frères et le père.

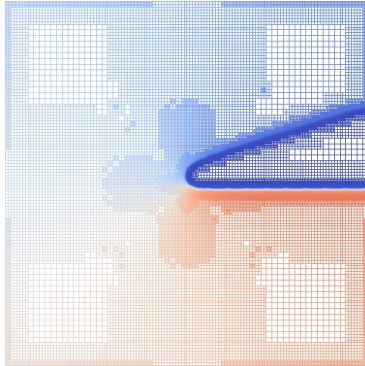
- $P_{j-1,j}$  :



Regarder les frères, cousins et oncles .

C'est la principale difficulté quant à l'implantation !

*Représentation classique des arbres par pointeurs : catastrophe garantie !*



# Multirésolution : implantation

Comment stocker des  $\{2,4,8..\}$ -arbres de manière performante ? Boîte à outils des bases de données géographiques !

- respecter le mieux possible la localité en mémoire !
- recherche facile de la parentele...

Une indexation des nœuds et des feuilles qui respecte le mieux possible la localité des données : [space filling curves](#).



## Multirésolution : implantation

Fonction de Peano : bijection  $P$  de  $[0, 1]^d$  sur le segment  $[0, 1]$ .

## Multirésolution : implantation

Fonction de Peano : bijection  $P$  de  $[0, 1]^d$  sur le segment  $[0, 1]$ .

En dimension  $d = 2$  :

- (en base 2) :

$$x = 0, x_1x_2, \dots, x_nx_{n+1} \dots,$$

$$y = 0, y_1y_2, \dots, y_ny_{n+1} \dots,$$

- $P(x, y) = 0, x_1y_1x_2y_2 \dots x_ny_nx_{n+1}y_{n+1} \dots$

## Multirésolution : implantation

Fonction de Peano : bijection  $P$  de  $[0, 1]^d$  sur le segment  $[0, 1]$ .

En dimension  $d = 2$  :

- (en base 2) :

$$x = 0, x_1x_2, \dots, x_nx_{n+1} \dots,$$

$$y = 0, y_1y_2, \dots, y_ny_{n+1} \dots,$$

- $P(x, y) = 0, x_1y_1x_2y_2 \dots x_ny_nx_{n+1}y_{n+1} \dots$

Généralisation immédiate en dimension  $d$  supérieure.

# Multirésolution : implantation

Fonction de Peano : bijection  $P$  de  $[0, 1]^d$  sur le segment  $[0, 1]$ .

En dimension  $d = 2$  :

- (en base 2) :

$$x = 0, x_1x_2, \dots, x_nx_{n+1} \dots,$$

$$y = 0, y_1y_2, \dots, y_ny_{n+1} \dots,$$

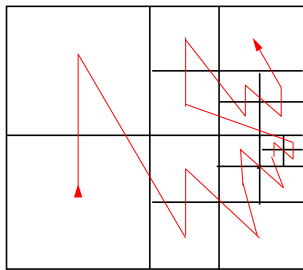
- $P(x, y) = 0, x_1y_1x_2y_2 \dots x_ny_nx_{n+1}y_{n+1} \dots$

Généralisation immédiate en dimension  $d$  supérieure.

Différents noms : Z-curves, Z-fonctions, Hilbert R-tree, k-d tree, etc. voir Knuth.

Ici : le développement en base 2 de  $x$  et  $y$  est de taille bornée par le nombre de grilles emboîtées maximum.

## Implantation (structure de données)

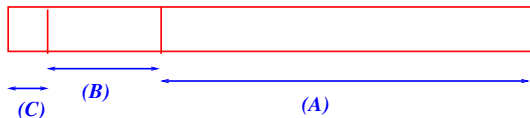


À chaque nœud ou feuille de l'arbre, on associe une **abscisse**  
 $[0, ]p_1p_2 \dots p_{2m}$ .

# Implantation (stockage d'une feuille ou d'un nœud)

Exemple :

- stockage sur un entier 64 bits.
- 15 niveaux.
- Dimension 3.



(A) : abscisse, (B) : flags, (C) : niveau.

## Multirésolution : implantation

Représentation d'une feuille ou d'un nœud : on stocke l'abscisse et le niveau dans l'arbre (hachage parfait).

# Multirésolution : implantation

Représentation d'une feuille ou d'un nœud : on stocke l'abscisse et le niveau dans l'arbre (hachage parfait).

## Structure de données :

- on stocke tous les nœuds pour lesquels  $s_1 \leq P(x, y) < s_2$  dans un vecteur d'entiers, *non triés* (un slot) ;



# Multirésolution : implantation

Représentation d'une feuille ou d'un nœud : on stocke l'**abscisse** et le niveau dans l'arbre (hachage parfait).

## Structure de données :

- on stocke tous les nœuds pour lesquels  $s_1 \leq P(x, y) < s_2$  dans un *vecteur* d'entiers, *non triés* (un slot) ; on a donc une collection de slots, le slot  $i$  contient tous les nœuds dont l'abscisse  $s$  vérifie  $s_1^i \leq s < s_2^i$ .

# Multirésolution : implantation

Représentation d'une feuille ou d'un nœud : on stocke l'abscisse et le niveau dans l'arbre (hachage parfait).

## Structure de données :

- on stocke tous les nœuds pour lesquels  $s_1 \leq P(x, y) < s_2$  dans un vecteur d'entiers, *non triés* (un slot) ; on a donc une collection de slots, le slot  $i$  contient tous les nœuds dont l'abscisse  $s$  vérifie  $s_1^i \leq s < s_2^i$ .
- étant donné l'abscisse  $P(x, y)$  d'un nœud, une *map* C++

$$s_1^i \rightarrow i$$

qui permet de trouver son slot en  $\log_2 N$  opérations ;

# Multirésolution : implantation

Représentation d'une feuille ou d'un nœud : on stocke l'abscisse et le niveau dans l'arbre (hachage parfait).

## Structure de données :

- on stocke tous les nœuds pour lesquels  $s_1 \leq P(x, y) < s_2$  dans un vecteur d'entiers, *non triés* (un slot) ; on a donc une collection de slots, le slot  $i$  contient tous les nœuds dont l'abscisse  $s$  vérifie  $s_1^i \leq s < s_2^i$ .
- étant donné l'abscisse  $P(x, y)$  d'un nœud, une *map* C++

$$s_1^i \rightarrow i$$

qui permet de trouver son slot en  $\log_2 N$  opérations ;

- un système de cache pour accélérer les recherches.

# Multirésolution : implantation

Représentation d'une feuille ou d'un nœud : on stocke l'abscisse et le niveau dans l'arbre (hachage parfait).

## Structure de données :

- on stocke tous les nœuds pour lesquels  $s_1 \leq P(x, y) < s_2$  dans un vecteur d'entiers, *non triés* (un slot) ; on a donc une collection de slots, le slot  $i$  contient tous les nœuds dont l'abscisse  $s$  vérifie  $s_1^i \leq s < s_2^i$ .
- étant donné l'abscisse  $P(x, y)$  d'un nœud, une *map* C++

$$s_1^i \rightarrow i$$

qui permet de trouver son slot en  $\log_2 N$  opérations ;

- un système de cache pour accélérer les recherches.

On vérifie que, étant donné un nœud d'abscisse  $P$ , on calcule facilement les abscisses de ses frères, oncles, cousins, pères etc. (manipulation au niveau bit).

# Multirésolution : implantation

Représentation d'une feuille ou d'un nœud : on stocke l'abscisse et le niveau dans l'arbre (hachage parfait).

## Structure de données :


- on stocke tous les nœuds pour lesquels  $s_1 \leq P(x, y) < s_2$  dans un vecteur d'entiers, *non triés* (un slot) ; on a donc une collection de slots, le slot  $i$  contient tous les nœuds dont l'abscisse  $s$  vérifie  $s_1^i \leq s < s_2^i$ .
- étant donné l'abscisse  $P(x, y)$  d'un nœud, une *map* C++

$$s_1^i \rightarrow i$$

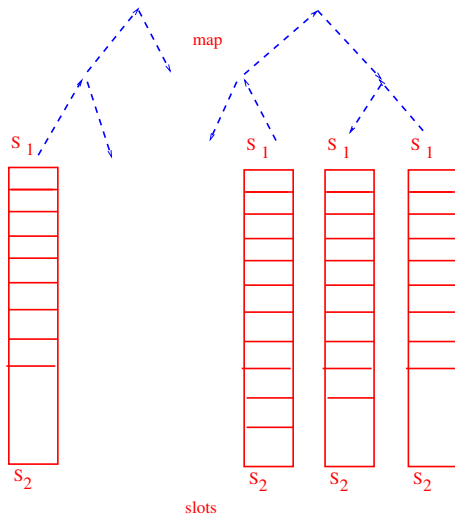
qui permet de trouver son slot en  $\log_2 N$  opérations ;

- un système de cache pour accélérer les recherches.

On vérifie que, étant donné un nœud d'abscisse  $P$ , on calcule facilement les abscisses de ses frères, oncles, cousins, pères etc. (manipulation au niveau bit).

*Idéal pour apprendre l'utilisation de « , », & etc.* 

# Multirésolution : implantation



# Multirésolution : implantation

Opérations sur la collection de *slots* :

- fusion de slots voisins trop petits,
- éclatement de slots trop grands.

# Multirésolution : implantation

Opérations sur la collection de *slots* :

- fusion de slots voisins trop petits,
- éclatement de slots trop grands.

Est-ce que tous les calculs sont parallélisables ?



Calculs effectués par Max Duarte et Zdenek Bonaventura.

vidéos :

Calculs effectués par TD.

{Multi-Many}-core.

# Architectures

- Augmentation du nombre de cœurs,
- Many core : Xeon Phi.

# Architectures

- Augmentation du nombre de cœurs,
- Many core : Xeon Phi.

## Xeon Phi

Architecture compatible Intel.

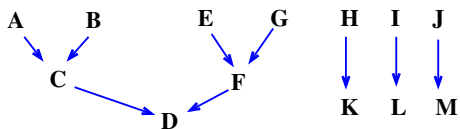
- $n \gg 1$  core ( $n = 60?$ )
- 4  $n$  threads,
- Unités vectorielles (4 doubles, 8 float).

# Programmation par vol de tâches

(X)kaapi (Grenoble Inria), [Threading Building Blocks \(Intel TBB\)](#).

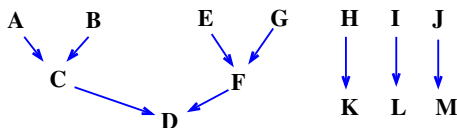
# Programmation par vol de tâches

(X)kaapi (Grenoble Inria), [Threading Building Blocks \(Intel TBB\)](#).



# Programmation par vol de tâches

(X)kaapi (Grenoble Inria), [Threading Building Blocks \(Intel TBB\)](#).



À chaque instant :

- stock de tâches disponibles,
- les unités de calcul (threads) *volent* les tâches disponibles quand elles sont desœuvrées.

# Programmation par vol de tâches

## Avantage

Pas de problèmes d'équilibrage des charges !

Exemple : pas réactif.



# Programmation par vol de tâches

## Avantage

Pas de problèmes d'équilibrage des charges !

Exemple : pas réactif.

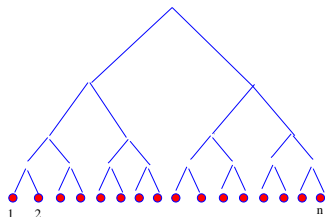
## TBB :

Mémoire partagée.

- bibliothèque C++ (g++, icc...),
- libre,
- simple !
- gestion des tâches,
- `parallel_for`, `parallel_reduce` etc.
- allocation multithread (remplace `malloc`).
- conteneurs type `stl`, concurrents.
- bibliothèque de choix pour le Xeon Phi.

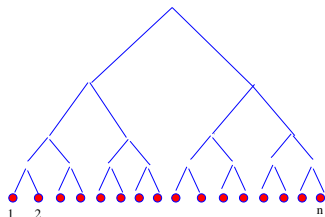
## Exemple : parallel\_for

$n$  tâches indépendantes.



## Exemple : parallel\_for

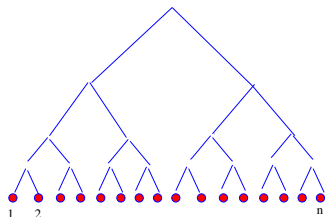
$n$  tâches indépendantes.



Décomposition récursive de l'intervalle  $[1, n]$  (jusqu'à un *grain* minimal).

## Exemple : parallel\_for

$n$  tâches indépendantes.



Décomposition récursive de l'intervalle  $[1, n]$  (jusqu'à un *grain* minimal).

Exemple de programme minimal :

Données :

```
double x[1000];  
for (int i=0; i<n; i++)  
    x[i]=0.0;
```

Ajouter 1 à chaque composant de  $x$ .

## Exemple : parallel\_for

```
class addOne{  
  
    double *X;  
public:  
    // Constructor.  
    addOne(double *x): X(x){}  
  
    // Copy Constructor.  
    addOne(const addOne& C): X(C.X){}  
  
    // operator  
    void operator()(blocked_range<size_t> range)  
    {  
        for (size_t i=range.begin(); i!=range.end(); i++)  
            x[i]+=1.0;  
    }  
}
```

## Exemple : parallel\_for

```
double x[1000];  
for (int i=0; i<n; i++)  
    x[i]=0.0;  
  
parallel_for(blocked_range<size_t>(1,1000), addOne(x));
```

## Exemple : parallel\_for

```
double x[1000];  
for (int i=0; i<n; i++)  
    x[i]=0.0;
```

```
parallel_for (blocked_range<size_t>(1,1000), addOne(x));
```

- découpe récursive de l'intervalle (1,1000),
- pour chaque sous intervalle créé, instantiation (copy constructor) d'un objet de type addOne,
- arrivé au niveau minimum, appel de l'opérateur sur le dernier objet instancié.

## Xeon Phi et “Vintage programming”

Unités vectorielles sur le Xeon Phi ! retour de la vectorisation comme sur les Cray des années 80.

Exemple de problème : résolution numérique des EDOs par méthode implicite : on doit calculer la matrice Jacobienne du second membre :

$$du/dt = F(u)$$

$$J_{i,j} = \frac{\partial F_i}{\partial u_j}$$

en général on procède par approximation numérique :

$$(F(u + he_j) - F(u))/h,$$

où  $e_j = (0, 0, \dots, 1, 0, \dots, 0)^t$  1 en  $j$ ème position.

On peut, *faut* vectoriser ce calcul.



# Micro expérience de portage sur un prototype du Xeon Phi.

Code multi résolution : 20 000 ligne de C++.

- ① on vérifie qu'il peut être compilé avec icc.
- ② ajout d'une option de compilation !
- ③ compilation et recopie du binaire exécutable sur le Xeon Phi.

# Micro expérience de portage sur un prototype du Xeon Phi.

Code multi résolution : 20 000 ligne de C++.

- ① on vérifie qu'il peut être compilé avec icc.
- ② ajout d'une option de compilation !
- ③ compilation et copie du binaire exécutable sur le Xeon Phi.
- ④ ça tourne !

# Micro expérience de portage sur un prototype du Xeon Phi.

Code multi résolution : 20 000 ligne de C++.

- ① on vérifie qu'il peut être compilé avec icc.
- ② ajout d'une option de compilation !
- ③ compilation et recopie du binaire exécutable sur le Xeon Phi.
- ④ ça tourne ! plus lentement.

Travail d'optimisation à faire : nécessité d'outils de mesure (VTune).

