IS 310 : Méthodes hiérarchiques

O. Coulaud

Olivier.Coulaud@inria.fr

Plan du cours

1. Motivation
2. Méthodes « Particules-Maillage »
3. Méthodes hiérarchiques
   - Les outils : approximations, arbres
   - Méthode Barnes&Hut,
   - Méthode des Multipôles rapide (FMM)
4. Parallélisation
   - Space Filling curves
   - Mémoire partagée : distribution des données, coloriage,
   - Mémoire distribuée : distribution des données, …

1

Introduction

A propos des méthodes hiérarchiques

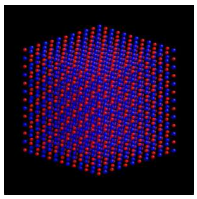
Simulation de particules

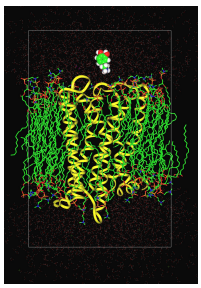
Méthodes très populaires

Historiquement deux domaines moteurs

cosmologie

chimie computationnelle





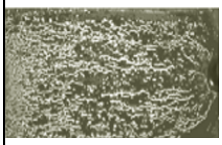
Domaine de simulation : non borné ou périodique

→ des méthodes différentes (mêmes idées)

N particules interagissant entre elles

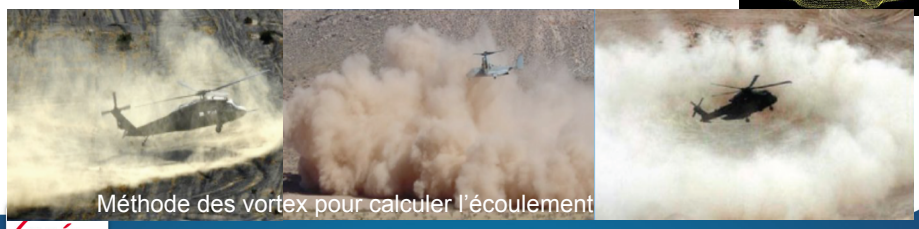
Autres exemples

Beaucoup d'autres applications nécessitent des évaluations rapides d'interactions entre paire de « points » avec le noyau du Laplacien (1/r) et de ses dérivées



Micro and Nano fluidics (complex channel Stokes flows)

Interpolation par des fonctions de base radiales (RBF) interpolation)

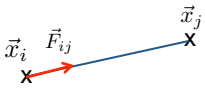


Méthode des vortex pour calculer l'écoulement

Simulation de particules (astrophysique)

N particules interagissant entre elles sous l'action de la gravité $(\vec{x}_i, m_i)$ pour i= 1 .. N

La force entre deux particules est donnée par



$$\vec{F}_{ij} = -G\frac{m_i m_j(\vec{x}_i - \vec{x}_j)}{|\vec{x}_i - \vec{x}_j|^3}$$

La force totale sur la particule i est donnée par

$$\vec{F}_i = -\sum_{j\neq i} G\frac{m_i m_j(\vec{x}_i - \vec{x}_j)}{|\vec{x}_i - \vec{x}_j|^3} - \vec{\nabla}\cdot\phi_{ext}(\vec{x}_i)$$

Le mouvement (Deuxième Loi de Newton)

$$m_i\frac{d^2\vec{x}_i}{dt^2} = \vec{F}_i$$

Simulation de particules (2)

La force entre deux particules est donnée par

$$\vec{F}_{ij} = -G\frac{m_i m_j(\vec{x}_i - \vec{x}_j)}{|\vec{x}_i - \vec{x}_j|^3}$$

Rappel

$$\frac{\vec{x}}{|\vec{x}|^3} = \nabla(\frac{1}{|\vec{x}|})$$

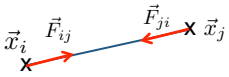
Et donc

$$\vec{F}_i = -m_i \nabla \sum_{j \neq i} G\, m_j \frac{1}{|\vec{x}_i - \vec{x}_j|} \qquad \vec{F}_i = -m_i \nabla V(\vec{x}_i)$$

Simulation de particules (3)

L'énergie potentielle est

$$V_i = V(x_i) = \sum_{j \neq i} \frac{G\ m_j}{|\vec{x}_i - \vec{x}_j|}$$



Et donc la force dérive du potentiel

$$\vec{F}_i = -m_i \nabla V(\vec{x}_i)$$

Troisième loi de Newton : Tout corps A exerçant une force sur un corps B subit une force d'intensité égale, de même direction mais de sens opposé, exercée par le corps B.

$$\vec{F}_{ij} + \vec{F}_{ji} = 0$$

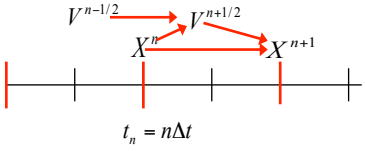
Simulation de particules (astrophysique)

Discrétisation en temps (Schéma de Leapfrog)

On note

$$X = (\vec{x}_1, \ldots, \vec{x}_N)$$

$$\begin{cases} V^{n+1/2} = V^{n-1/2} - M^{-1}\Delta t \nabla V(X^n) \\ X^{n+1} = X^n + \Delta t V^{n+1/2} \end{cases}$$



Possède de bonnes propriétés

- Conserve l'énergie
- ….

Algorithme de base

N : nombre de particules

T_f Temps final de la simulation et il peut être très grand

```
t = 0
while t < t_final

    for i = 1 to N
       calculer f(i) = force sur la particule i

    for i = 1 to n
       Déplacer la particule i        (Utilise  F=ma)

    Calcul de propriétés (énergie, etc.)
    t = t + dt
end while
```

1. $F(x_i,x_j)$
2. Champ externe
3. ….

Comment calculer les forces ?

Nom	FLOPS
yottaFLOPS	10^{24}
zettaFLOPS	10^{21}
exaFLOPS	10^{18}
pétaFLOPS	10^{15}
téraFLOPS	10^{12}
gigaFLOPS	10^{9}
mégaFLOPS	10^{6}
kiloFLOPS	10^{3}

Approche classique

- N particules → N(N-1) interactions
- 3ième loi de Newton → ½ N(N-1)
- ~20 FLOP pour évaluer le potentiel et la force

$$\vec{F}_{ij} = -G\frac{m_i m_j(\vec{x}_i - \vec{x}_j)}{|\vec{x}_i - \vec{x}_j|^3}$$

10^5 particules ~ 10 10^{10} flops par pas de temps (100 Gflops)
10^6 particules ~ 10 10^{12} flops par pas de temps (10 Tflops)

Cas réaliste pour une galaxie 10^{11} particules → 10^8 petaflops !!!

Comment calculer les forces ?

Nom	FLOPS
yottaFLOPS	10^{24}
zettaFLOPS	10^{21}
exaFLOPS	10^{18}
pétaFLOPS	10^{15}
téraFLOPS	10^{12}
gigaFLOPS	10^{9}
mégaFLOPS	10^{6}
kiloFLOPS	10^{3}

$$\text{FLOPS} = \text{cœurs} \times \text{fréquence} \times \frac{\text{FLOPs}}{\text{cycle}}$$

Ivy Bridge (2.5 GHz 10 cœurs, 8 flops/cycle) → 200 GFlops
Haswell (2 Ghz, 14 cœurs, 16 flops/cycle) → 448 GFlops
Skylake (2 Ghz, 22 cœurs, 32 flops/cycle) → 1.4 TFlops

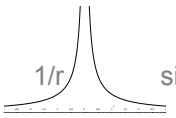
Carte GPU K40 → 4.29 Tflops/s (float)
P100 → 9.3 Tflops/s

~2s pour 10^6 particules

Sunway TaihuLight, (Chine): 93.01 pétaflops (Linpack HPL) 2017
Sunway SW26010 260C 1.45GHz 10,649,600

Comment calculer les forces ?

Contraintes
- Evaluer rapidement
- Avec une bonne précision

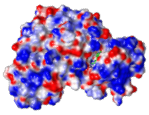


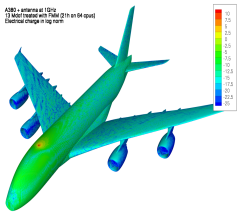
singularité en 0 mais régulier en dehors

Plusieurs Approches
- Résolution du problème de Poisson
- Approches hiérarchiques

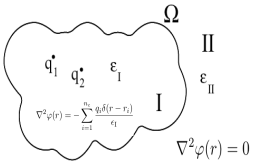
Autre exemple : les équations intégrales

Domaines :
- Électrostatique
- Élasticité
- Acoustique, électromagnétisme





Idée : Ecrire le problème 3D sur une surface 2D



$$\theta(x) + \beta \int_S \frac{\langle x-y, n(x)\rangle}{|x-y|^3}\theta(y)dy = -\beta \frac{\partial}{\partial n}\psi_{vide}(x) \qquad \forall x \in S$$

Système linéaire à résoudre **A x = b** mais A matrice pleine !!

Les équations intégrales

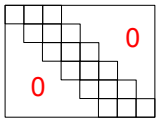
Il faut calculer rapidement $R_i = \sum_{j=1}^{N} A_{i,j} x_j$

Propriété La matrice A vérifie

Si | i - j | >> 1 alors $A_{i,j}$ est petit

Idée :

- Garder les blocs près de la diagonale
- Approcher les blocs lointains

A = 

+ Approximation

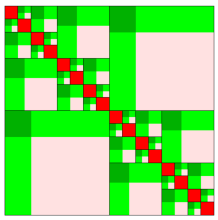
Récapitulatif

Calculer rapidement

- Des interactions entre des entités (=particules)
- Des produits matrices vecteurs

Deux outils

- Séparer le calcul proche
  - méthode de rayon de coupure, boites voisines, Arbres, ….
- Méthode pour approcher le « champ » lointain
  - Développement en série
  - Interpolation
  - Compression d'un bloc par du rang faible



MÉTHODES A BASE D'UN MAILLAGE

Méthode : « Particule-Maillage »

Méthode : « Particule-Maillage » PM

Approche très utilisée pour Hydrodynamique (compressible, incompressible, MHD)

- (~1960) Simulation des plasmas sans collision

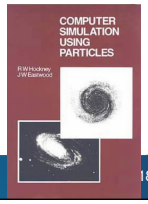
$$-\Delta V = \frac{\rho}{\epsilon_0} \quad \leftarrow \text{densité}$$

$$m\frac{d^2}{dt^2}x = -\nabla V \quad \leftarrow \text{Champ électrique}$$

Code Particule in Cell

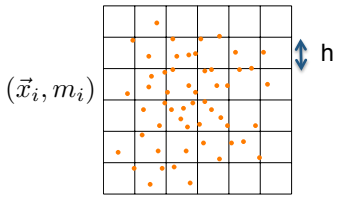
- (1970→1980) utilisation en astrophysique/cosmologie

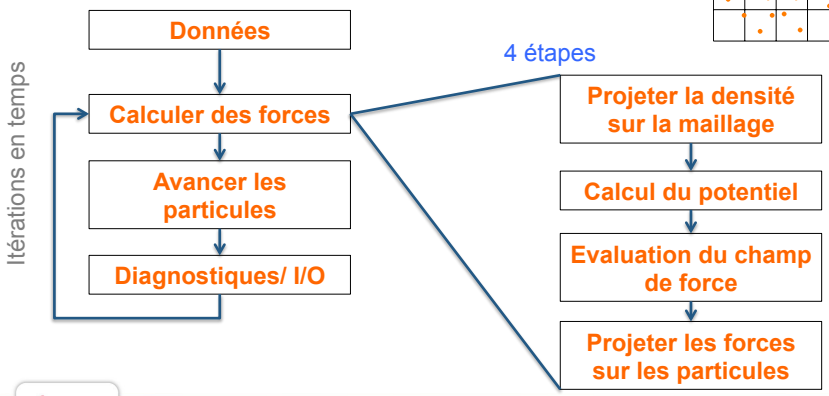
Hockney, R.W., Eastwood, J.W., "*Computer Simulation Using Particles*", CRC Press (1985)



Méthode : « Particule-Maillage » PM

Choisir l'approximation EF, DF, …
Les différentes étapes de la méthode PM



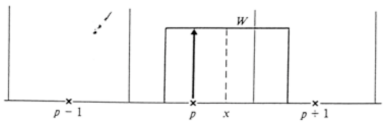


Méthode : « Particule-Maillage » PM

Etape 1 : Projeter la densité sur le maillage

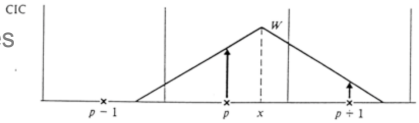
Fonction de forme S pour affecter une fraction de la charge/masse sur le maillage
On centre la fonction S sur les particules

Nearest Grid Point



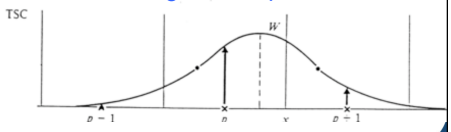
Cloud In Cell

p point
x : particule



Méthode	Nb cellules	Approximation
NGP	1	P0
CIC	2^d	P1
TSC	3^d	P2

Triangular Shape Cloud



Méthode : « Particule-Maillage » PM

Etape 2 : Résoudre le problème de poisson

$$-\Delta V = \frac{\rho}{\epsilon_0} \quad \leftarrow \text{densité}$$

$$m\frac{d^2}{dt^2}x = -\nabla V \quad \leftarrow \text{Champ électrique}$$

Discrétisation classique (DF, VF, EF)

Système linéaire à résoudre (Direct, itératif) - Il faut une méthode rapide et efficace

- Multigrille
- FFT si périodique
- AMR
- ….

Méthode : « Particule-Maillage » PM

Etape 3 : évaluer la force

$$F = -\nabla V$$

- si périodique
  - FFT sur V $\quad \hat{f}_k = -ik\hat{V}_k$ puis FFT^{-1}
- Si non périodique
  - Par exemple différences finies d'ordre 2 ou 4

Méthode : « Particule-Maillage » PM

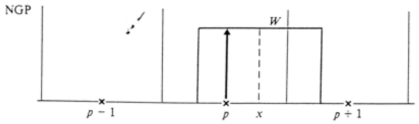
Etape 4 **Projeter les forces sur les particules**

Étape inverse de l'étape 1

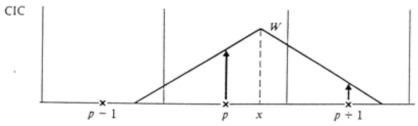
On centre la fonction S sur les points du maillages

Méthode	Nb cellules	Approximation
NGP	1	P0
CIC	2^d	P1
TSC	3^d	P2

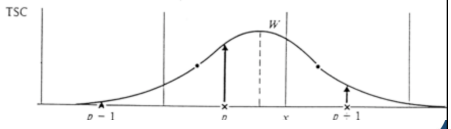
Nearest Grid Point



Cloud In Cell



Triangular Shape Cloud



Méthode : « Particule-Maillage » PM

Avantages

- Facile et simple à mettre en œuvre
- Rapide

Inconvénients

- Dépendance forte entre le pas de la grille et le nombre de particules → système à résoudre
- Que faire si la distributions des particules n'est pas homogènes
  → AMR (pas si facile)
- Équilibrage de charge difficile
  Compromis entre la grille et les particules

2
Les méthodes à base d'un arbre

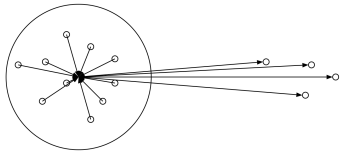
Plan du cours

1. Approximations
2. Méthodes de Barnes & Hut
3. Méthode Fast multipole
4. Méthode Adaptative

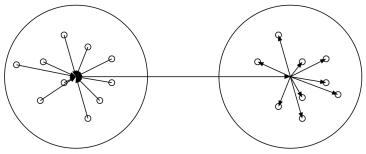
Méthode à base d'arbre

Deux grandes classes

- méthode en arbre :
  - Interactions particules-cellules
  - Complexité N log(N)

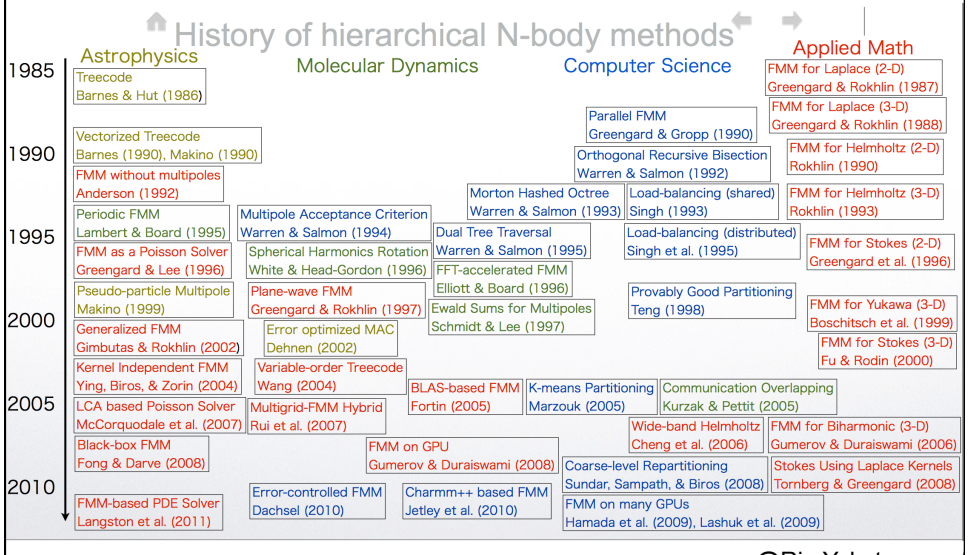


- méthode des multipôles rapide :
  - Interactions cellules-cellules
  - Complexité N



Les idées de base

1. **Séparer le champ proche et le champ lointain**
2. **Approcher le champ lointain**



@Rio Yokota

Exemples (Gordon Bell Prize)

1992 5.2 Gigaflops, Astrophysical N-Body Simulation Intel Delta ; M. Warren and J. Salmon 17 million particles, 600 time steps, 24 hours elapsed time

2009 20 Tflops, Vortex particle simulation of turbulence on Cluster of 256 NVIDIA GeForce 8800 GPUs -- 16.8 million particles ; T. Hamada, R. Yokota, K. Nitadori. T. Narumi, K. Yasoki et al

2010 Petascale direct numerical simulation of blood flow on 200K cores and heterogeneous architectures. A. Rahimian and I. Lashuk, S. Veerapaneni, A Chandramowlishwaran, D Malhotra, L Moon, A Shringarpure, R Sampath, J Vetter, R. Vuduc, G Biros, D Zorin

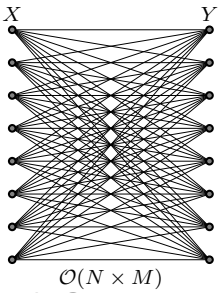
2012 4.45 Pflops Astrophysical N-Body Simulation on K computer T Ishiyama, K Nitadori, J Makino -1 milliard de milliards (Trillon) de particules

Idées de base (1)

Calculer des interactions du type

$$V_i = \sum_{j=1}^{N} K(x_i, y_j)\, w_j \quad \text{for } i = 1, \ldots, M.$$

$$K(x,y) = \frac{1}{\|x-y\|}$$



K est une fonction décroissante et/ou une fonction de Green

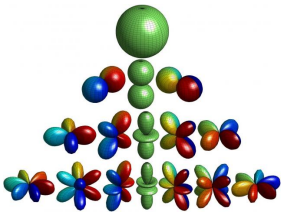
Décomposer en champ proche et champ lointain

Idées de base (2)

Calculs

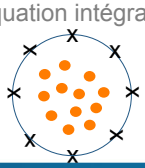
- Champ proche : évaluation directe
- Champ lointain
  - Développements en série :
    - Taylor $x^i y^j z^k$
    - Harmoniques sphériques

$$Y_n^m(\theta,\phi) = \sqrt{\frac{(n+|m|)!}{(n+|m|)!}} P_n^{|m|}(\cos\theta) e^{im\phi}$$

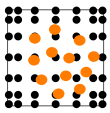


  - Equation intégrale (Biros), Interpolation (Fong & Darve)

2003



2009



LES APPROXIMATIONS

Les arbres

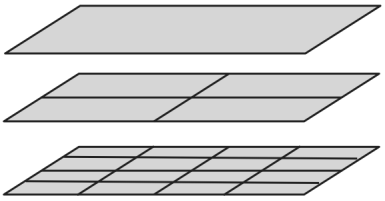
Quad tree

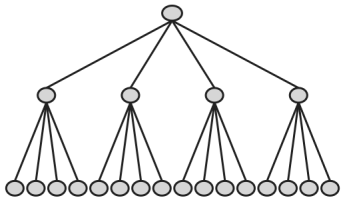
Dimension 2 d'espace

Chaque dimension est coupée en 2 parties

Niveau donné H

Exemple H=2





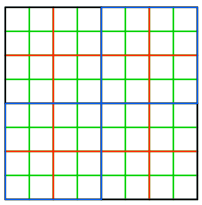
Structure grille

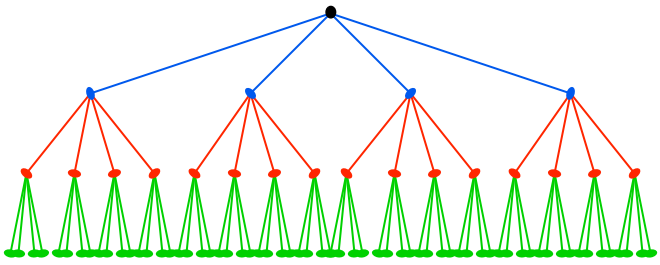
Structure arbre

Quad Tree

Structure de donnée pour diviser un plan

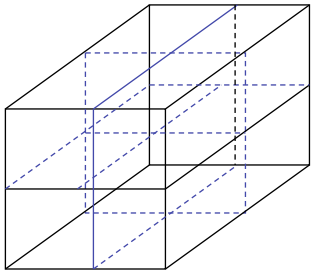
Chaque nœud (non feuille) à 4 enfants



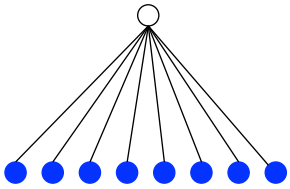


Octree

Dimension 3 d'espace
Chaque cube est divisé en 8



Deux niveaux d'un octree



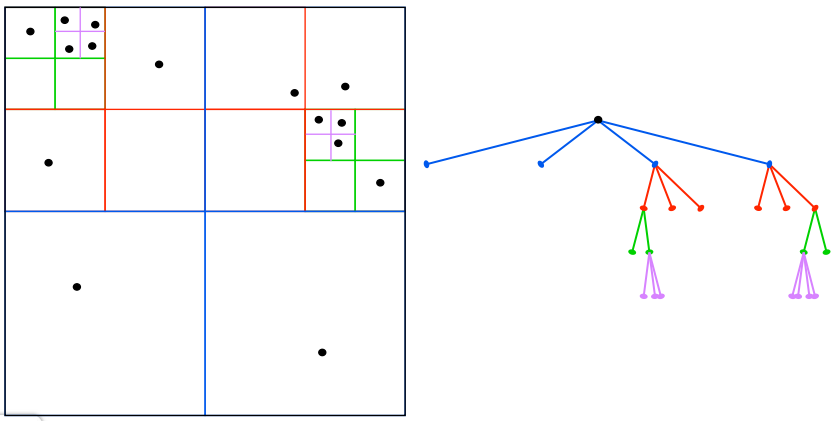
Octree généralisé

Boite initiale = cube dans R^d.

- niveau 0 de l'arbre = racine
- Chaque direction sera découpée en 2 → 2^d cellules filles
- Au niveau *l*
  - $2^{d\,l}$ boites, numérotés de 0 à $2^{d\,l}$ -1
  - Boîte = cube → on parle de nœud ou de cellule
- Au niveau l_{max}
  - Niveau terminal, il n'y a pas de fils
  - Cellule = feuille

Exemple de quadtree adaptatif

Découpage d'une cellule en fonction du nombre de particules



Méthode Adaptative

APPROCHE BARNES & HUT

"A Hierarchical O(n log n) force calculation algorithm", J. Barnes and P. Hut, Nature, v. 324 (1986)

Méthode développée pour une faible précision ~1%

Introduite pour des simulations en cosmologie

Développement de Taylor sur le centre de masse

Système de N particules de masse m_i en position x_i

Quelques définitions des multipôles

- Ordre 1 - monopôle - scalaire

$$M = \sum_{j=1}^{N} m_j$$

- Ordre 2 - dipôle – vecteur

$$\vec{D} = \sum_{j=1}^{N} m_j x_j$$

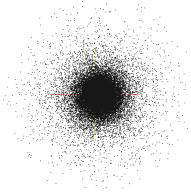
- Ordre 3 - quadripôle matrice 3x3

$$Q_{k,l} = \sum_{j=1}^{N} m_j \, x_{j_k} x_{j_l}$$

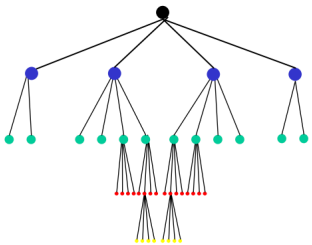
Développement de Taylor sur le centre de masse

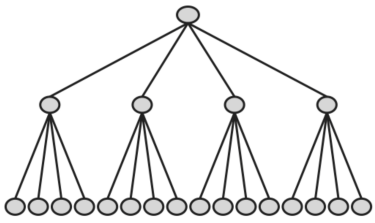
Distribution

Distribution de Plummer



Distribution uniforme

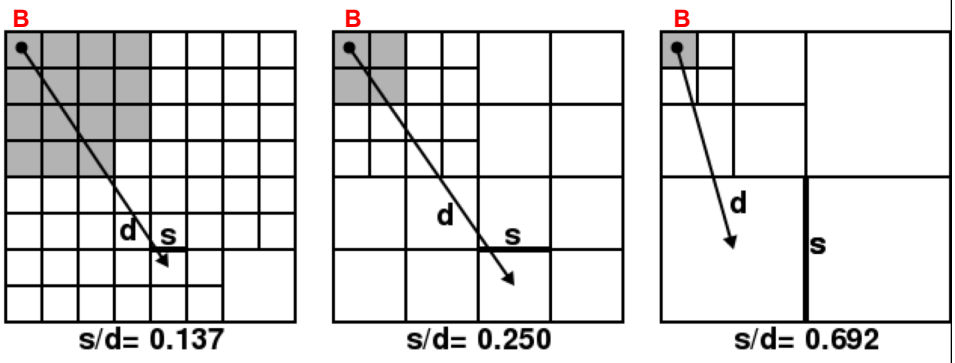




Exemples de listes d'interactions

Boîte de côté 1 ; dx = 1/8 = 0.125

B: [0.0625 0.9375]



Méthode Adaptative

MÉTHODE DES MULTIPÔLES RAPIDE - FMM

Développement de Taylor

Centre des cellules

Quelque soit l'ordre

Evaluation des fonctions par récurrences

Opérateurs

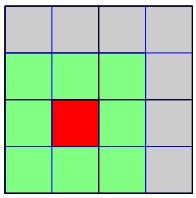
L. Greengard and V. Rokhlin,
A fast algorithm for particle simulation, Comp. Phys. V. 73, 1987

L. Greengard and V. Rokhlin,
A new version of the Fast Multipole Method for the Laplace equation in three dimensions, Acta Numerica, vol. 6, pp. 229–269, 1997.

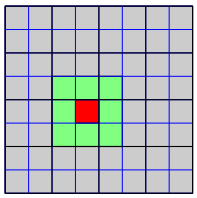
Quelques définitions (1)

Définition 1 On dit que deux boites sont des voisines proches si

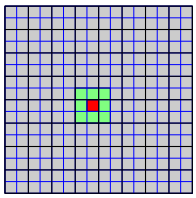
- Même niveau de raffinement
- Partage un point sur sa frontière



Level 2



Level 3



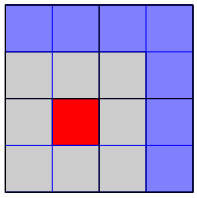
Level 4

Définition 2 On dit que deux boites sont bien séparées si

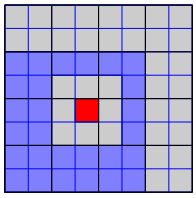
- Même niveau de raffinement
- Ne sont pas des boîtes voisines

Quelques définitions (2)

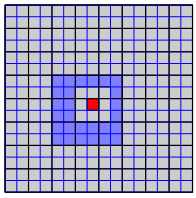
Définition 3 On définit I(B) la liste d'interaction de la boîte B par l'ensemble des boîtes filles des voisins proches de la cellule père de B et qui sont bien séparées.



Level 2



Level 3



Level 4

Quelques définitions (2)

Pour chaque boite B au niveau *l*

- développement multipolaire = approximation de la grandeur physique associée à toutes les particules situées dans les feuilles contenues dans la boite B
- développement local = représente le champ lointain induit par toutes les particules en dehors des boites voisines.

Rôles des opérateurs

P2M Particules → Multipôles feuilles

Construit une approximation d'une grandeur physique : masse, charge

M2M Multipôles → Multipôles Cellules

Propage la grandeur physique équivalente des filles vers le père dans les niveaux de l'arbre

M2L Multipôles Cellule A → Local Cellule B

Construit une approximation du champ dans la cellule B due à la grandeur physique équivalente de la cellule A

L2L Local → Local Cellules

Propage le champ lointain vers les cellules filles

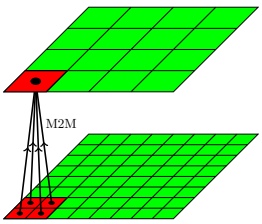
L2P Local → Particules feuilles

• Applique le champ lointain sur les particules

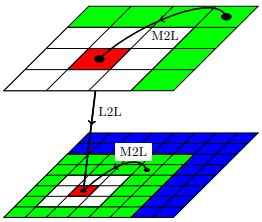
Algorithme en O(N)

Champ lointain : deux phases

Montante :
construit une grandeur physique équivalente



Descendante :
calcul le champ due à cette grandeur équivalente

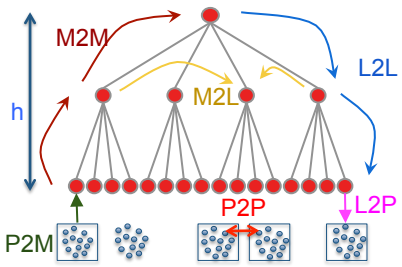


Algorithme en O(N)

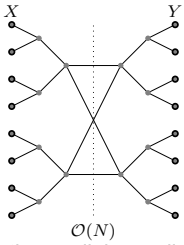
Champ lointain : deux phases

Montante : construit une grandeur physique équivalente

Descendante : calcul le champ due à cette grandeur



Calculs : dépendent de h, P et dominés par M2L (189 #cells)



Interactions cellule – cellule

Algorithme (1)

```
function FMM(tree)
    // Near-field
    P2P(tree.levels[tree.height-1]);
    // Far-field
    P2M(tree.levels[tree.height-1]);
    forall the level l from tree.height-2 to 2 do
        M2M(tree.levels[l]);
    forall the level l from 2 to tree.height-2 do
        M2L(tree.levels[l]);
        L2L(tree.levels[l]);
    M2L(tree.levels[tree.height-1]);
    L2P(tree.levels[tree.height-1]);
```


Algorithme (2)

```
function FMM(tree)
    // Near-field
    P2P(tree.levels[tree.height-1]);
    // Far-field
    P2M(tree.levels[tree.height-1]);
    forall the level l from tree.height-2 to 2 do
        M2M(tree.levels[l]);
    forall the level l from 2 to tree.height-2 do
        M2L(tree.levels[l]);
    forall the level l from 2 to tree.height-2 do
        L2L(tree.levels[l]);
    M2L(tree.levels[tree.height-1]);
    L2P(tree.levels[tree.height-1]);
```

Complexité

N particules
Arbre uniforme de hauteur L

Hypothèse : **distribution uniforme avec s particules par feuilles**

Rappels

- 2^{dL} feuilles → $2^{dL} = N/s$ → $L = \log_8 (N/s)$
- Nombre total de cellules $(2^{dL+1} - 1)(2^d - 1)$
- On doit évaluer M développements multipolaires ou locaux
  $M = p^3$ (Taylor) , p^2 (Harmonique sphérique)

3
Structures de données

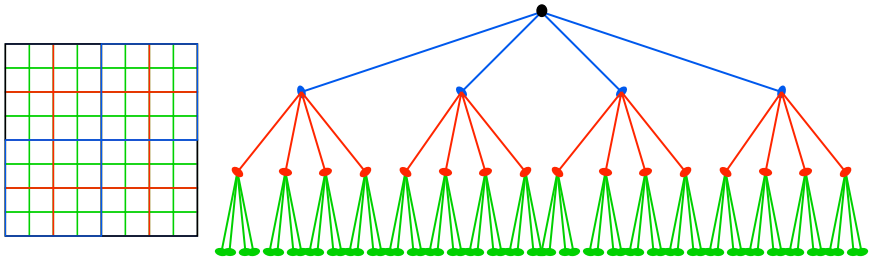
Plan du cours

1. Les arbres
2. Space Filling Curve
3. Numérotation

N.A. Gumerov, R. Duraiswami & E.A. Borovikov
Data Structures, Optimal Choice of Parameters, and Complexity Results for Generalized Multilevel Fast Multipole Methods in d Dimensions
UMIACS TR 2003-28.

Quelques questions

Comment numéroter les cellules ?
Comment trouver rapidement :
- le numéro de la cellule parente, des cellules filles
- le numéro de la cellule à partir d'un point
- le centre de la cellule



Structures de données pour l'octree

Structure d'arbre

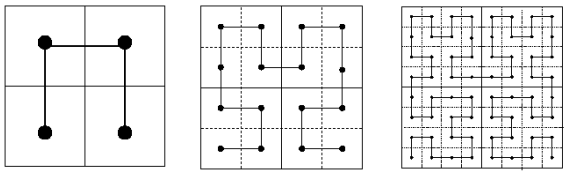
```
Node {
    Node * parent ;
    Node * fils[8] ;
}
```

Permet de se déplacer dans l'arbre mais
- Difficile d'aller à la liste des voisins
- Parcours par niveau difficile

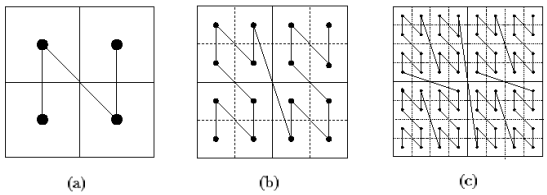
Peu efficace à éviter

Courbes de remplissage de l'espace
Space Filling Curves

Courbe de Hilbert



Z curve

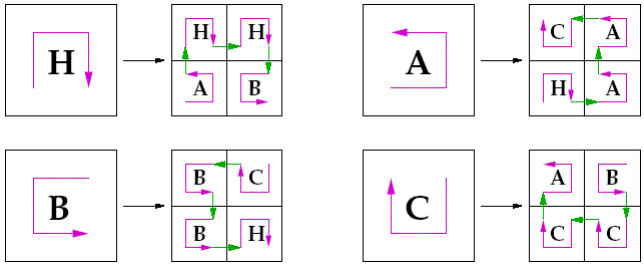


Construction de la courbe de Hilbert

Construction récursive :

Quatre motifs : H,A,B et C.

Ces motifs sont translatés par le schéma suivant

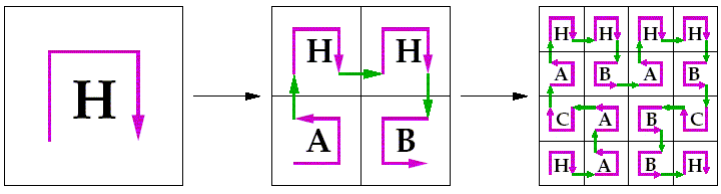


$$H \longrightarrow A\uparrow H \rightarrow H\downarrow B \qquad A \longrightarrow H \rightarrow A\uparrow A \leftarrow C$$
$$B \longrightarrow C \leftarrow B\downarrow B \rightarrow H \qquad C \longrightarrow B\downarrow C \leftarrow C\uparrow A$$

Construction de la courbe de Hilbert

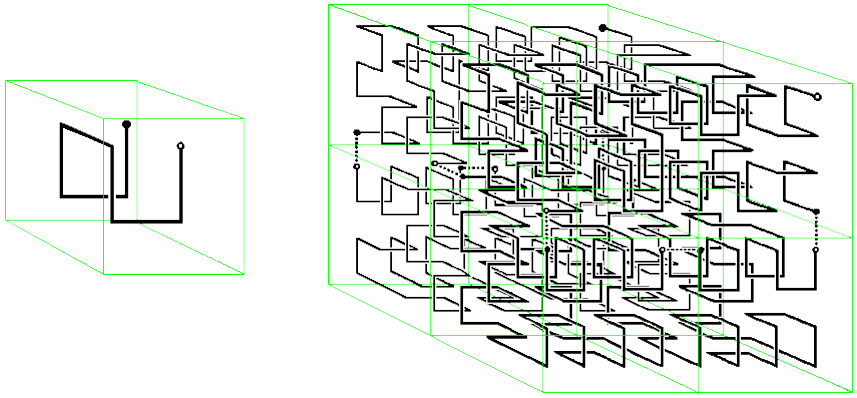
On utilise les règles précédentes pour construire de manière récursive la courbe.

$$H \longrightarrow A\uparrow H \rightarrow H\downarrow B$$



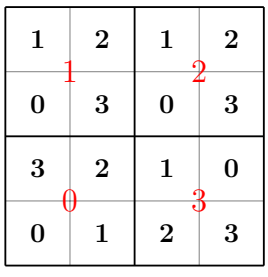
$$(H \rightarrow A\uparrow A \leftarrow C)\uparrow(A\uparrow H \rightarrow H\downarrow B)\rightarrow(A\uparrow H \rightarrow H\downarrow B)\downarrow(C \leftarrow B\downarrow B \rightarrow H)$$

Courbe de Hilbert en dimension 3

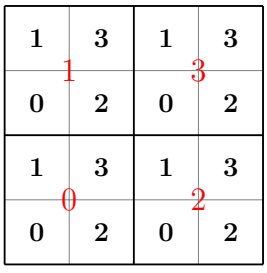


Comparaisons

Numérotations



numérotation de Hilbert



numérotation de Morton

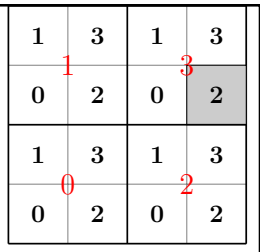
Z, N

Continuité de la numérotation

Facilité de construction

Morton

Numérotation hiérarchique



Boîte(N_1,N_2, …, N_l) = (N_1,N_2, …, N_l) avec $0 \le N_i < 2^d$

Number = $(2^d)^{l-1} N_1 + (2^d)^{l-2} N_2 + \ldots + 2^d N_{l-1} + N_l$

Boîte grise (niveau 2) : indice (3,2) → 32_4 = $4^{2-1}\,3 + 4^0\,2 = 14_{10}$

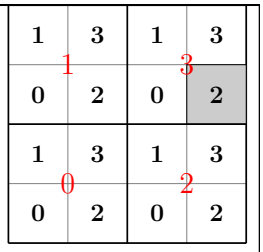
Calcul du parent de la boite B

Number(B) = (N_1,N_2, … , N_{l-1}, N_l)

Parent(B) = (N_1,N_2, … , N_{l-1},) → Parent(B) = [Number(B) / 2^d]

Parent(Boîte grise) = (3) → $3_4 = 3_{10}$ [14 / 4] = 3

Numérotation hiérarchique



Boîte(N_1,N_2, …, N_l) = (N_1,N_2, …, N_l)

Indices des enfants :

Enfants(N_1,N_2, …, N_l) = {(N_1,N_2, …, N_l, N_{l+1}) , N_{l+1} = 0, …, 2^d-1}

$\{(2^d)^l N_1 + (2^d)^{l-1} N_2 + \ldots + (2^d)^2 N_{l-1} + 2^d N_l + N_{l+1},\ N_{l+1} = 0, \ldots, 2^d - 1\}$

Calcul des indices des enfants de la boîte B

Number(B) = (N_1,N_2, … , N_{l-1}, N_l)

Enfants(B) = (2^d * Number(B) + j, j = 0, …, 2^d-1)

Enfants(32_4) = { 320_4, 321_4, 322_4, 323_4}
= {3 4^2 + 2 4^1 + (0,1,2,3) }
= { 56 + (0,1,2,3) }
= {56, 57, 58, 59 }

21	23	29	31	53	55	61	63
20	22	28	30	52	54	60	62
17	19	25	27	49	51	57	59
16	18	24	26	48	50	56	58
5	7	13	15	37	39	45	47
4	6	12	14	36	38	44	46
1	3	9	11	33	35	41	43
0	2	8	10	32	34	40	42

Evaluations rapides

1	3	1	3
0	2	0	2
1	3	1	3
0	2	0	2

1 3 0 2

Multiplication et division par 2^d
sont équivalents à des opérations bit à bit

Opérateur
<< ou ishft (i, d) : décalage à gauche
>> ou ishft (n, -d) : décalage à droite

ex: $3_{10} = 11_2$; $3 << 2 = 1100_2 = 12_{10}$

5	7	13	15
4	6	12	14
1	3	9	11
0	2	8	10

1 3 0 2

Soit n l'indice d'une boîte au niveau l d'un quadtree
Parent(n) = n >> 2
Enfant(n) = { (n << 2) + j , j = 0,…2^d -1 }

Trouver l'indice d'une boîte à partir d'un point

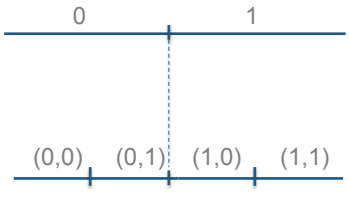
x dans [0,1]

$$x = \sum_{j>0} b_j 2^{-j}$$

$x=0.b_1b_2b_3\ldots$ $b_j = \{0,1\}$

Niveau	Taille boite décimale	Taille boite binaire
0	1	1
1	0.5	0.1
2	0.25	0.01
3	0.125	0.001

Niveau 1
Boîte(0) : $b_1= 0$ et Boîte(1) : $b_1= 1$
$x = 0.N_1b_1b_2b_3\ldots$ appartient à Boîte(N_1)



Niveau 2
Boîte (0) : $b_1= b_2= 0$ et Boîte (1) : $b_1= 0$ et $b_1= 1$
Boîte (2) : $b_1= 1$ et $b_2= 0$ et Boîte (3) : $b_1= b_2= 1$

$x=0.N_1N_2b_1b_2b_3\ldots$ appartient à Boîte(N_1N_2)

Trouver l'indice d'une boîte à partir d'un point

Niveau l

$x = 0.N_1N_2\ldots N_lb_1b_2b_3\ldots$ Appartient à Boîte($(N_1N_{2\ldots}N_l)$)

$b_j = \{0,1\}$

Soit x un point d'une boite(N) comment trouver N ?

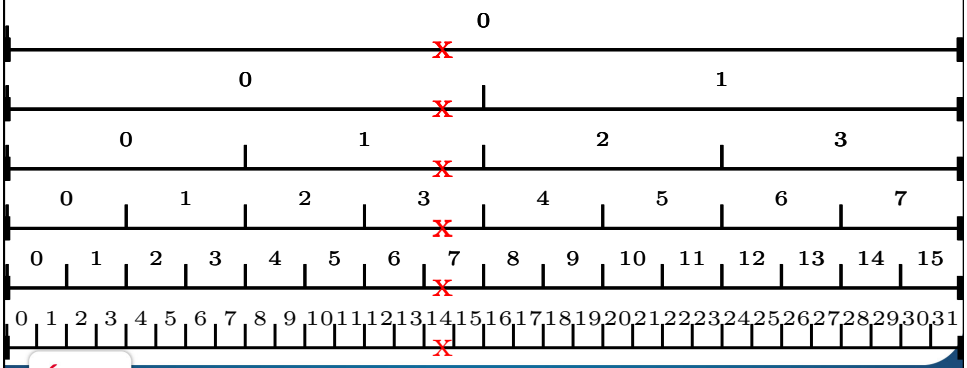
$x = 0.N_1N_2\ldots N_lb_1b_2b_3$ —déplacement de l position→ $N_1N_2\ldots N_l.b_1b_2b_3$

Algorithme

$(N,l) = [\, x * 2^l \,]$

Exemple x = 0.457

Niveau	1	2	3	4	5
$[\, x * 2^l \,]$	0.914	1.828	3.656	7.312	14,624
Numéro	0	1	3	7	14



Numérotation spatiale

Soit **x** = $(x_1,\ldots,x_d)$ un point du cube $[0,1]^d$

$x_i = (0.\ b_{i1}\ b_{i2},b_{i3},\ \ldots)_2$ avec $b_{ij} = 0$ ou 1 $\quad i = 1\ldots d$ et $j > 0$

Générer une clé binaire unique : entrelacer les bits des d coordonnées

x = $(0.\ b_{11}b_{21\ldots}b_{d1}\ b_{12}b_{22\ldots}b_{d2}\ \ldots)_2$

En base 2^d x s'écrit

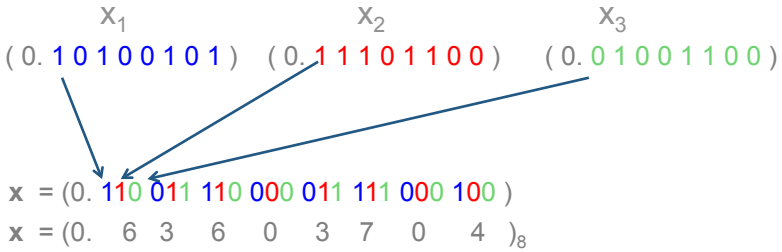
x = $(0.\ N_1N_2\ldots N_j\ldots)_{2^d}$ $\quad N_j = (b_{1j}\ b_{2j},b_{dj})$ $\quad j = 1,2,\ \ldots$ $\quad N_j = 0,\ \ldots,\ 2^d-1$

Marche aussi pour les entiers M = (M1, M2, M3)

Exemple en 3 dimension

Octree

x = (x_1, x_2,x_3) $\qquad (b_1, b_2, b_3)_2 = b_1 2^2 + b_2 2^1 + b_3 2^0$



Une clé unique

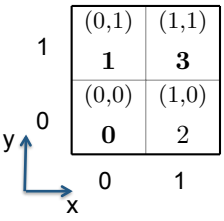
Machine 64 bits → 21 bits par coordonnée → Niveau ≤ 20

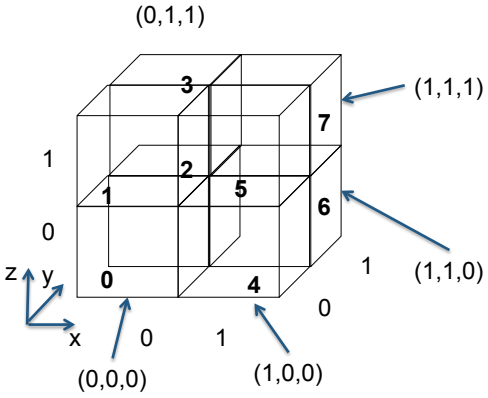
Convention de numérotation

Numérotations des enfants

$b_i = 0, 1$

$(b_1,b_2,\ldots,b_d) = N$





$(b_1, b_2)_2 = b_1 2^1 + b_2 2^0$

$(b_1, b_2, b_3)_2 = b_1 2^2 + b_2 2^1 + b_3 2^0$

Désentrelacer un nombre

Number = numéro d'une boite

Number = $(b_{11},b_{21}, \ldots b_{d1}, b_{12},b_{22}, \ldots b_{d2}, \ldots b_{1n},b_{2n}, \ldots b_{dn})_2$

On décompose ce nombre en d nombre

$Number_1 = (b_{11},b_{12}, \ldots b_{1n})_2$

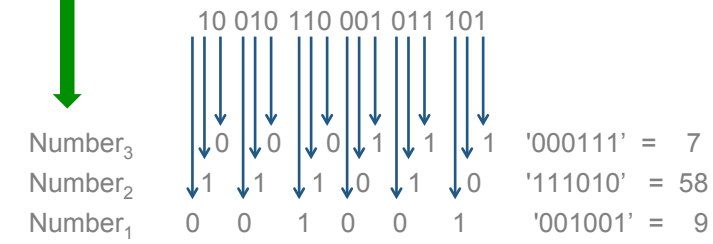
$Number_2 = (b_{21},b_{22}, \ldots b_{2n})_2$

…

$Number_d = (b_{d1},b_{d2}, \ldots b_{dn})_2$

Désentrelacer un nombre

Nombre = 76893 en dimension 3



76893 = (9,58,7)

Calculer le centre d'une boîte

Boîte B au niveau l

Number → (Number$_1$, Number$_2$, Number$_3$)

Centre(B)$_k$ = 2^{-l} * Number$_k$ + 2^{-l-1} = 2^{-l} * (Number$_k$ + 0.5)

Exemple

Centre de la boite

13 niveau 2

13 = 1101_2 →(10_2 , 11_2) = (2 , 3)

Centre = (0.625, 0.875)

49 niveau 3

49 → 110001_2 = (100_2,101_2) = (4, 5)

→ (0.5625, 0.6875)

5	7	13	15
4	6	12	14
1	3	9	11
0	2	8	10

0 1 2 3

21	23	29	31	53	55	61	63
20	22	28	30	52	54	60	62
17	19	25	27	49	51	57	59
16	18	24	26	48	50	56	58
5	7	13	15	37	39	45	47
4	6	12	14	36	38	44	46
1	3	9	11	33	35	41	43
0	2	8	10	32	34	40	42

0 1 2 3 4 5 6 7

Calculer les voisins d'une boîte

Boîte de numéro Number

Nombre de voisins = $3^d - 1$ → en 3d 26 voisins

Etape 1 Désentrelacement

Number → (Number$_1$, Number$_2$, Number$_3$) = (N_1, N_2, N_3)

Etape 2 Construit l'ensemble des voisins avec un déplacement de ± 1

{ (N_1+(-1,0,1), N_2+(-1,0,1), N_3+(-1,1)) ← Face inférieure et supérieure (2x3^2)

\+ (N_1+(-1,0,1), N_2+(-1,1), N_3) ← Coté (2x3)

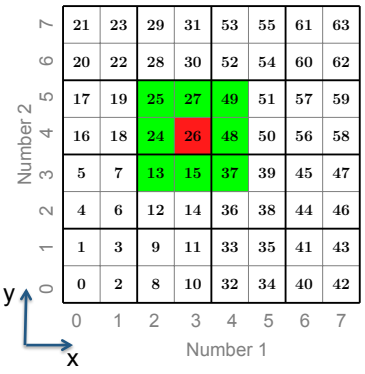
\+ (N_1 , N_2+ (-1,+1), N_3) } ← Devant (2x1)

Etape 3 Entrelacement + passage en décimal

{n_1,n_2,…n_{27}}

26

Calcul des voisins



Cellule 26 = 11010_2= 01 10 10_2

↓ Désentrelace

('011', '100')$_2$ =: $(3,4)_{10}$

Génère les voisins

3±1 et 4 ±1

(2,3), (2,4), (2,5),(3,3),(3,5), (4,3), (4,4), (4,5)

('010', '011')$_2$, ('010', '100')$_2$ …. ('100', '100')$_2$ ('100', '101')$_2$

Entrelace ↓

'001101', '011000', …., '110000', '110001'

13 , 24 , 25, 15, 27, 37, 48, 49

Structures de données pour l'octree

Distribution uniforme

⇒ tableaux unidimensionnels

- Niveau l →Tableau de 8^l pointeurs sur des cellules
- Accès facile via l'indice de Morton
- Total L niveaux → $(8^{L+1} -1)/7$ cellules
- Hauteur 10 → 1.1*sizeof(pointer)*sizeof(Cellule) Go !!

Distributions non uniformes

Hashed octree (HOT) (Warren & Salmon)
Une clé unique pour chaque nœud de l'Octree
→ calcul hash(clé) pour un accès linéaire
→ clé = coordonnées entrelacées de la cellule (centre) préfixé par 1

Table de hachage

Une structure de données qui permet une association clé-élément

Table de hachage = tableau ne comportant pas d'ordre

Le fait de créer une valeur de hachage à partir d'une clé
On accède à un élément par sa clé
→ fonction de hachage : transforme la clé en une valeur de hachage
→ Il peut y avoir des collisions car fonction non injective

Accès en O(1) en moyenne sauf si collision (au pire O(N))

Efficace si N est très grand

FMM 1-D

N particules (x_i, q_i) dans [0,1]
Arbre de hauteur 4 → 2^4 = 16 feuilles

Construire l'indice de Morton des particules → $Im(x_i) = x_i / 2^4$ taille de la cellule
Trier les particules
Construire l'arbre

4

Parallélisme

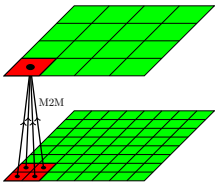
Plan

1. Approche mémoire partagée
   - Distribution de l'arbre
   - Champ direct
   - Modèle fork and join
   - Approche en tâches
2. Mémoire distribuée
   - Distribution des données
   - Algorithme MPI
   - Algorithme hybride

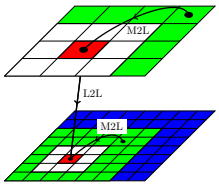
Rappel : algorithme en O(N)

Champ lointain : utilise l'octree

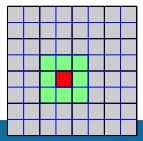
Montante :
On construit une grandeur physique équivalente



Descendante :
On calcul le champ due à cette grandeur équivalente



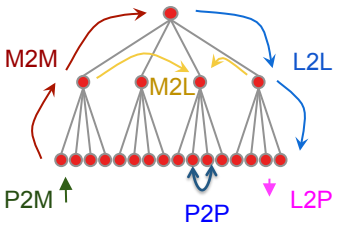
Champ proche : utilise la grille
Calcul des interactions entre particules
opérateur P2P



Parallélisation

Parallélisme important :

- Deux phases indépendantes :
  - P2P et P2M/M2M/M2L/L2L/L2P

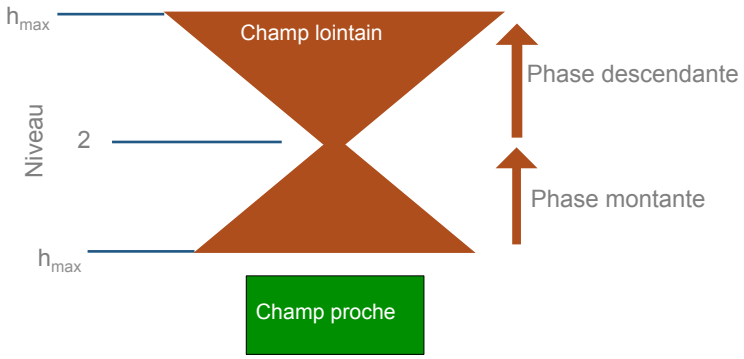


- Beaucoup d'opérateurs
  - # M2L ~ 189 x nbCells
  - # P2P ~ 27 x nbFeuilles

Task	Concurrency	Load-balancing
M2L (lower level)	High	Homogeneous
M2L (higher level)	**Low**	Homogeneous
P2P	High	**Heterogeneous**
M2M/L2L	**Low**	Homogeneous

Parallélisation

Beaucoup de parallélisme en bas de l'arbre / faible en haut
Diminue/augmente d'un facteur 8 à chaque remontée/descente dans l'arbre



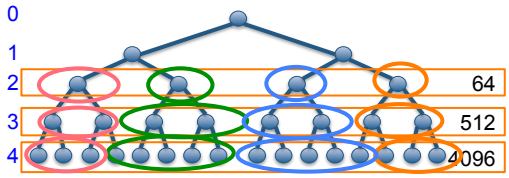
PARALLÉLISATION APPROCHE MÉMOIRE PARTAGÉE

Parallélisation approche mémoire partagée

Distribution Cellules/Feuilles

Indice de Morton → distribution 1D

Fonction de coût = (#particules, # d'interactions)



Une décomposition par niveau

⇒ 1 intervalle par *thread*

thread [StartIndexMorton, EndIndexMorton]

Différentes possibilités

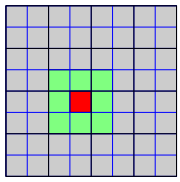
Thread Posix (C)

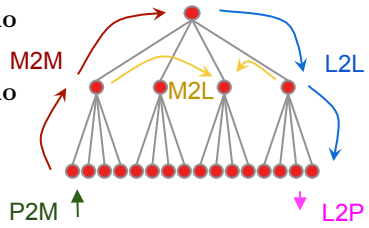
OpenMP (Standard)

Ordonnancement des calculs piloté par le demandeur.

Algorithme

```
function FMM(tree)
    // Near-field
    P2P(tree.levels[tree.height-1]);
    // Far-field
    P2M(tree.levels[tree.height-1]);
    forall the level l from tree.height-2 to 2 do
        M2M(tree.levels[l]);
    forall the level l from 2 to tree.height-2 do
        M2L(tree.levels[l]);
    forall the level l from 2 to tree.height-2 do
        L2L(tree.levels[l]);
    M2L(tree.levels[tree.height-1]);
    L2P(tree.levels[tree.height-1]);
```





Calcul du champ proche (1)

```
function P2P(tree) begin
    foreach leaf lf in setOfLeaves do
        foreach leaf nlf in neighList(lf) do
            foreach particles p1 in lf do
                i = p1.index ;
                foreach particles p2 in nlf do
                    d[x] = pos[p1.index][x] - pos[p2.index][x] ;
                    ...;
                    force = ...;
                    forces[p1.index][x] = d[x]*force ;
                    forces[p2.index][x] = -d[x]*force ;
                    ...
```

Liste des voisins tq mortonIndex (blf)> mortonIndex(lf)

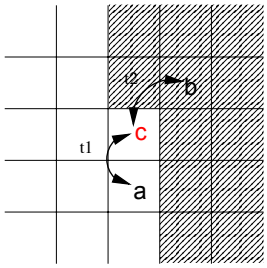
3ème loi de Newton

Calcul du champ proche (2)

Quelques difficultés avec l'approche des interactions réciproques

thread t1 sur l'intervalle blanc

thread t2 sur l'intervalle gris



Si t1 calcule les interactions entre a et c

Si t2 calcule les interactions entre b et c

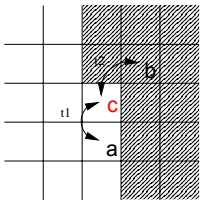
➔ conflit en écriture sur la boite c

Solutions

1. Supprimer les interactions réciproques mais calculs x 2
2. Mettre un verrou sur les cellules impliquées dans l'interaction et traiter plus tard l'interaction avec les deux cellules
3. Utiliser un coloriage

Calcul du champ proche (3) (verrou)

Solution 2 – utilisation d'un verrou



Deux possibilités

- Mettre un verrou (OpenMP/Posix) particule / feuille
- Utiliser un bit dans la structure feuille mais nécessite des opérations atomiques (utiliser un verrou par thread ou *pragma atomic*)

Parallélisation manuelle SPMD en OpenMP

OpenMP: différentes approches

Code séquentiel

```
for (int i=0; i<N; i++) { a[i]=b[i]+c[i]; }
```

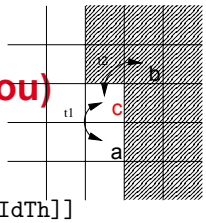
Parallélisation manuelle

```
#pragma omp parallel
{
  int id    = omp_get_thread_num();
  int Nthr = omp_get_num_threads();
  int istart = id*N/Nthr, iend = (id+1)*N/Nthr;
  for (int i=istart; i<iend; i++) { a[i]=b[i]+c[i]; }
}
```

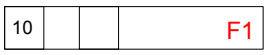
Parallélisation automatique

```
#pragma omp parallel
#pragma omp for schedule(static)
{
  for (int i=0; i<N; i++) { a[i]=b[i]+c[i]; }
}
```


Calcul du champ proche (4) (verrou)




```
function P2P(tree) begin
    foreach leaf lf in [DebutIndMorton[IdTh],FinIndMorton[IdTh]]
    do
        if lf.lock == 1 then
            Mettre lf dans la file F1 ; continue ;
        foreach leaf nlf in neighList(lf) do
            if nlf.lock != 1 then
                nlf.lock =1 ;
                P2P(lf,nlf);
                nlf.lock = 0 ;
            else
                Mettre nlf dans la file F2 ;
        Traiter les cellules de la file F2;
    while F1.notEmpty() do
        Traiter les cellules de la file F1
```

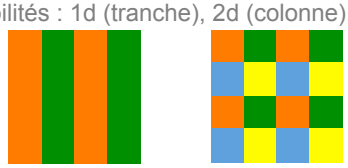


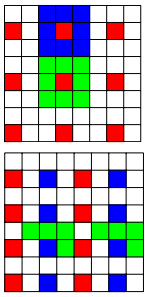
Calcul du champ proche (4) (coloriage SDC)

Spatial Decomposition Coloring

Utiliser un coloriage pour supprimer les dépendances

Différentes possibilités : 1d (tranche), 2d (colonne), 3d





Cas 1 écart de deux entre les cellules

Les cellules rouges peuvent calculer sans conflits

Déplacement de 0,1,2 vers la droite et vers le haut

Coloriage à 9 couleurs en 2d et 27 en 3d

Cas 2 écart de un entre les cellules

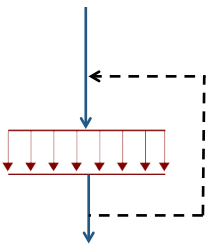
Les cellules rouges peuvent calculer sans conflits

Coloriage à 2x4 couleurs en 2d et 2x8 en 3d

Calcul du champ proche (4) (coloriage SDC)

Algorithme

```
for color=0; color < nbColor; ++color do
    ... ;
    #pragma omp for schedule(dynamic, chunk_Size)
    foreach leaf lf in colors[color] do
        foreach leaf nlf in neighList[lf] do
            P2P(lf,nlf);
```



Ordonnancement

- dynamic → diminuer le temps d'attente (synchronisation car nowait)
- chunk → diminuer le surcoût du démarrage et le temps d'attente

Deux approches

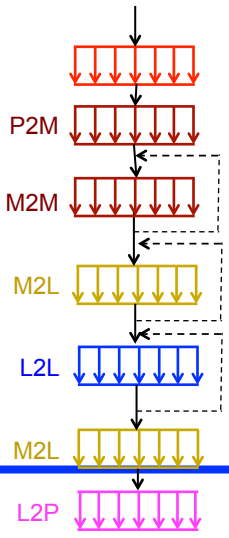
Le cas 2 donne moins de couleurs et plus de cellules par couleur.

→ moins de synchronisation

Champ lointain : fork and Join

```
function FMM(tree)
    // Near-field
    P2P(tree.levels[tree.height-1]);
    // Far-field
    P2M(tree.levels[tree.height-1]);
    forall the level l from tree.height-2 to 2 do
        M2M(tree.levels[l]);
    forall the level l from 2 to tree.height-2 do
        M2L(tree.levels[l]);
    forall the level l from 2 to tree.height-2 do
        L2L(tree.levels[l]);
    M2L(tree.levels[tree.height
    L2P(tree.levels[tree.height
```

```
function M2L(level)
    #pragma omp parallel for
        foreach cell cl in level.cells do
            kernel.m2l(cl.local,
                       cl.far_field.multipole);
```

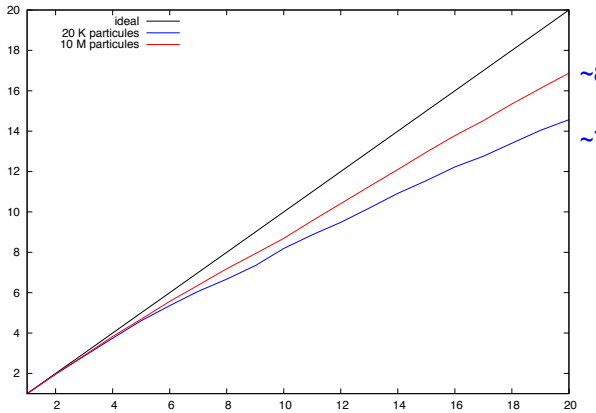


Résultats

Machine : 2 Deca-core Ivy-Bridge à 2,50 GHz

Deux cas tests – précisions 7

- 20 K particules, h = 5
- 10 M particules h = 8



~85%

~70%

Quelques remarques

Parallélisation facile

Mais
- Beaucoup de synchronisations
- Granularité très fine
- Approche séquentielle des calculs
  Calcul du champ proche puis calcul du champ lointain

Peut-on entrelacer les calculs ?

Oui mais il faut utiliser le modèle de tâches

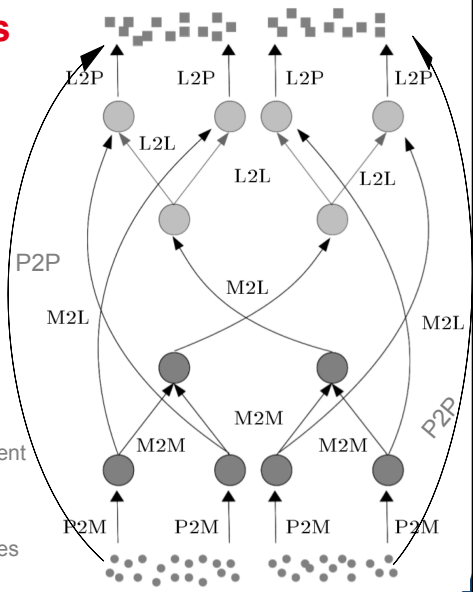
Approche en tâches

Approche par les données
DAG = Directed Acyclic Graph

Nœud = les données = feuille ou cellule
Arête
- Opérateur Op output = Op(input)
- Traduit une dépendance entre les données

Tâche :
Bloc d'instructions + paramètres + environnement
Pas dédié à un thread

Tâches existent dans beaucoup de bibliothèques
OpenMP, moteur d'exécution, …



Rappel des tâches OpenMP (1)

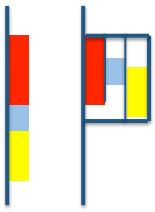
Principale nouveauté avec OpenMP 3.0

Offre un modèle flexible pour les problèmes irréguliers
- Boucles non bornées (boucle while) ;
- Algorithmes récursifs ;
- Schémas producteurs/consommateurs

Rappel des tâches OpenMP (2)

Les tâches OpenMP sont des unités de travail indépendantes
- Les threads sont affectés pour effectuer le travail des tâches ;
- L'exécution des tâches peut être différée ;
- L'exécution des tâches peut être immédiate ;

le système d'exécution décide si l'exécution est différée ou immédiate.



Parallèle

Une tâche est constituée
- D'un code à exécuter ;
- D'un environnement de données initialisées à la création ;
- De variables de contrôle interne (ICV).

Rappel des tâches OpenMP (3)

C/C++:

```
#pragma omp task [clause [[,]clause] ...]
    structured-block
```

Fortran:

```
!$omp task [clause [[,]clause] ...]
    structured-block
!$omp end task
```

Permet de construire une tâche qui sera exécutée par un thread du pool de threads.

On peut imbriquer le constructeur (parallélisme emboité)

Rappel des tâches OpenMP (4)

C/C++:

```
#pragma omp task [clause [[,]clause] ...]
    structured-block
```

Fortran:

```
!$omp task [clause [[,]clause] ...]
    structured-block
!$omp end task
```

Clauses :

- if (expression scalaire)
- final (expression scalaire)
- untied
- default (shared | none)
- mergeable
- firstprivate (list)
- private (list)
- shared (list)

Clauses :

- if (expression scalaire)
- final (expression scalaire)
- untied
- default (shared | private | firstprivate | none)
- mergeable
- firstprivate (list)
- private (list)
- shared (list)

Rappel des tâches OpenMP (liste chainée)

```
    node *p   = listhead ;
#pragma omp parallel
 {
#pragma omp single
  {
    while (p) {
    #pragma omp task  firstprivate (p)
          {
          do_independent_work(p) ;
          }
          p = p->next()
    }
  } // END SINGLE
}  // END PARALLEL
```

Création des threads

Un thread exécute la boucle while

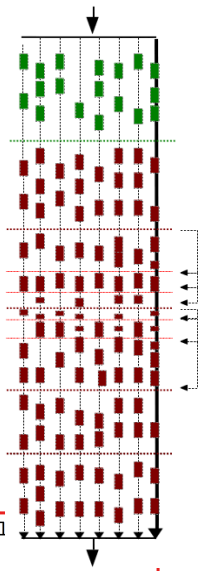
Création des tâches et exécution en parallèle

Chaque tâche s'exécute dans un thread

Quand la tâche se termine, le thread associé attend sur la barrière implicite de la construction ***single***

Parallélisation par directives

```
function FMMtask(kernel, tree)
    #pragma omp parallel
        #pragma omp single
            P2Ptask(kernel, tree.levels[tree.height-1]);
            P2Mtask(kernel, tree.levels[tree.height-1]);
            forall the levels l from tree.height-2 to 2 do
                M2Mtask(kernel, tree.levels[l]);
            forall the levels l from 2 to tree.height-2 do
                M2Ltask(kernel, tree.levels[l]);
                L2Ltask(kernel, tree.levels[l]);
            M2Ltask(kernel, tree.levels[tree.height-1]);
            L2Ptask(kernel, tree.levels[
```

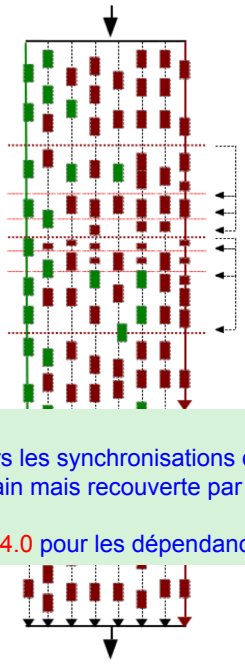


```
function M2M(kernel, level
    foreach Cell cl in cells
        #pragma omp task
            kernel.m2m(cl, cl.children);
    #pragma omp taskwait;
```

Ve ntrelacer FF & NF)

Créer deux sect
- Calcul du
- Calcul du

- Puis une rédu

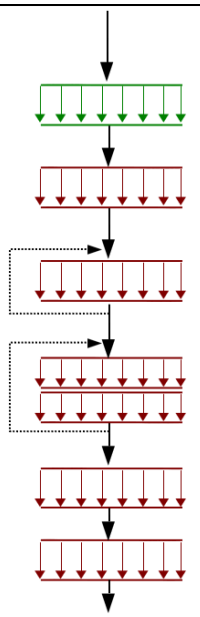


```
#pragma omp sections
  // Near-field
  #pragma omp section
    P2Ptask(tree.levels[tree.height-1]);
  // Far-field without L2P
  #pragma omp section
    P2Mtask(tree.levels[tree.height-1]);
    for l = tree.height-2 → 2 do
      M2Mtask(tree.levels[l]);
    for l = 2 → tree.height-1 do
      M2Ltask_notaskwait(tree.levels[l]);
    Wait for M2L tasks to finish
    pragma omp taskwait;
    r l = 2 → tree.height-2 do
      L2Ltask(tree.levels[l]);
    ction with L2P
    #pragma omp single
      L2Ptask(tree.levels[tree.height-1]);
```

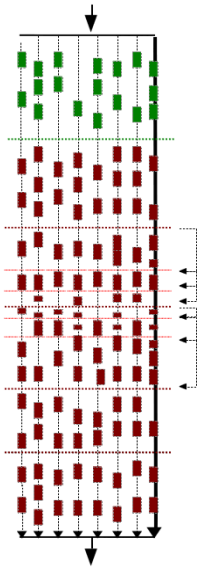
Mais
On a toujours les synchronisations dans le champ lointain mais recouverte par le P2P

→ OpenMP 4.0 pour les dépendances

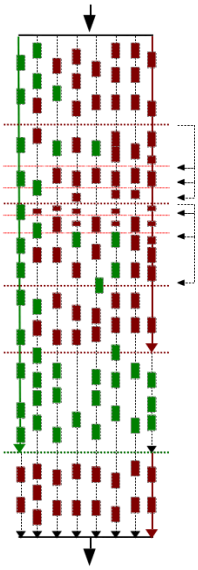
P2P
P2M
M2M
M2L
L2L
M2L
L2P



Boucle OpenMP

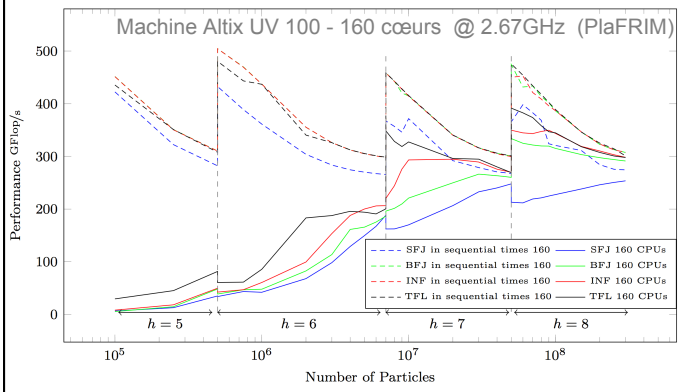


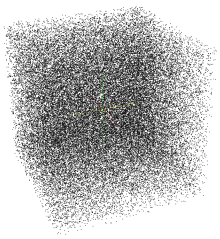
Tâche OpenMP par bloc



Tâche OpenMP + sections

FMM cas Uniforme





P = 5

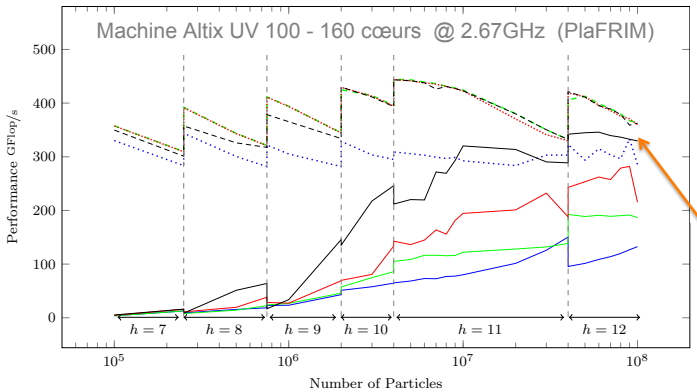
SFJ in sequential times 160 — SFJ 160 CPUs — Boucle OpenMP

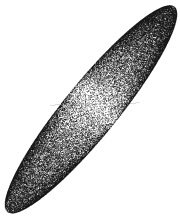
BFJ in sequential times 160 — BFJ 160 CPUs — Boucle OpenMP par bloc

INF in sequential times 160 — INF 160 CPUs — Tâche OpenMP + sections

TFL in sequential times 160 — TFL 160 CPUs — StarPU

FMM cas non uniforme





100 Millions 8.4 s

P = 5

SFJ in sequential times 160 — SFJ 160 CPUs — Boucle OpenMP

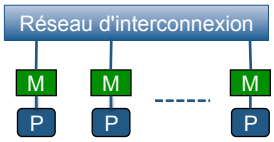
BFJ in sequential times 160 — BFJ 160 CPUs — Boucle OpenMP par bloc

INF in sequential times 160 — INF 160 CPUs — Tâche OpenMP + sections

TFL in sequential times 160 — TFL 160 CPUs — StarPU

APPROCHE MÉMOIRE DISTRIBUÉE

Approche par échange de messages



Parallélisation
- Plusieurs processus
- Un espace d'adressage différent
- Des communications pour échanger les données

→ Standard Message Passing Interface

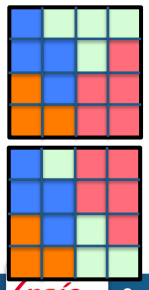
Points à regarder
- Comment distribuer les données
- Structure de données
- Comment gérer les communications

Découpage de l'octree

Objectif :

- Avoir de la localité entre les cellules
  - Les cellules sur un processeur sont spatialement proches.
- Avoir un bon équilibrage de charge

Deux techniques



- Orthogonal Recursive Bissection (ORB)
  - Classique pour découper physiquement l'espace
  - Partitionnement 3D
  - Warren and Salmon, Supercomputing 92

- Costzones : un partitionnement plus simple
  - Utilise la « Space Filling curve »
  - Partitionnement 1D
  - Costzones for Shared Memory PhD thesis, J.P. Singh, Stanford, 1993

Fonctions de coûts

Deux opérateurs sont prédominants P2P et M2L

Fonction de coût : uniquement sur les feuilles

1. champ proche
   - coût uniforme par feuille
   - Nombre de particules par feuille N_{leaf}
   - Prise en compte des voisins :

$$\mathcal{C}_{dir}(i) = \begin{cases} 1 & \text{si la feuille } i \text{ est vide,} \\ \frac{N_i^2}{2} + \sum_{i<j} c_{ij} & \text{sinon,} \end{cases}$$

$C_{ij} = N_i N_j$

2. champ proche + champ lointain
   - Operateur M2L sur les feuilles + liste d'interactions

Découpage (ORB)

Méthode du Bi partitionnement récursif

Idée

Découpage de l'espace de simulation (i.e. la grille)

Barnes-Hut

Uniquement sur les particules, pas sur les cellules

Facile à mettre en place (à droite ou à gauche de la ligne)

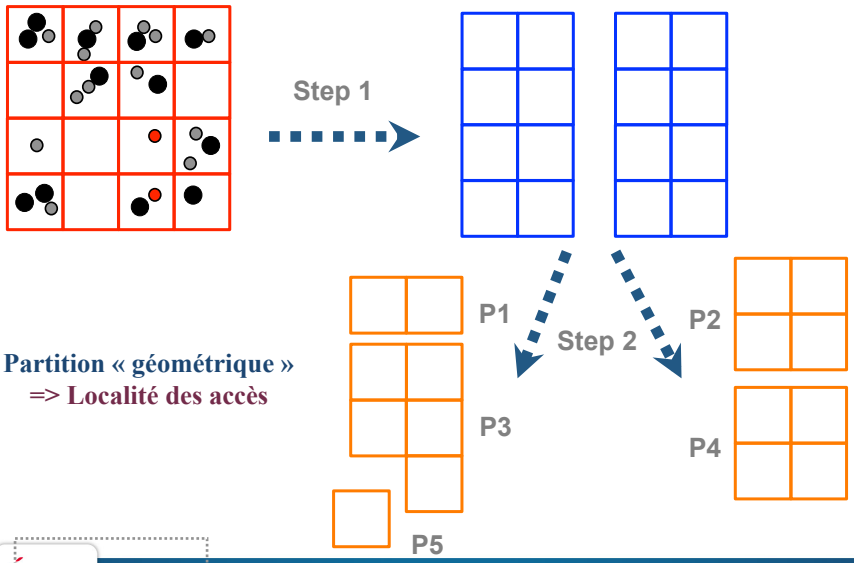
FMM

IL faut prendre en compte toutes les cellules (feuilles+cellules)

On construit un graphe (représente la grille des feuilles)

- Nœud = centre de la cellule
- Arête = si il y a un opérateur entre les deux nœuds
- Coût sur les nœuds et les arêtes
- Partitionneur de graphe scotch ou metis

Idée de l'algorithme



Partition « géométrique »
=> Localité des accès

Costzones

Idée : on va découper l'arbre plutôt que l'espace

- on utilise la numérotation de Morton → 1d (prend en compte l'espace)
- On commence à gauche
- Chaque cellule a une fonction de coût (son coût et le coût total à gauche)
- Variantes : prise en compte des parents ou non

Algorithme : différentes approches

1. **Pré-traitement**
   1. Construit le coût à gauche et le coût total
   2. Evalue son intervalle de cellule meanCost*[rank,rank+1]
      parcours montant puis descendant de l'arbre
2. Construction de la distribution
   Tous les processeurs partent de la racine et élague l'arbre pour arriver dans son intervalle

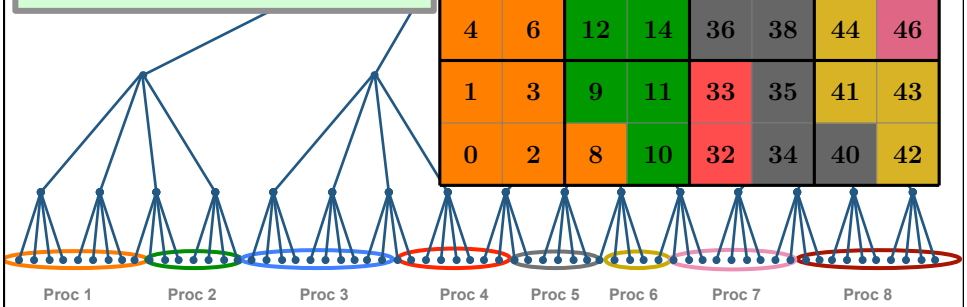
Rapide et efficace

Costzones

Perte de localité rouge et rose

La localité dépend de la numérotation des enfants
→ numérotation uniforme

21	23	29	31	53	55	61	63
20	22	28	30	52	54	60	62
17	19	25	27	49	51	57	59
16	18	24	26	48	50	56	58
5	7	13	15	37	39	45	47
4	6	12	14	36	38	44	46
1	3	9	11	33	35	41	43
0	2	8	10	32	34	40	42

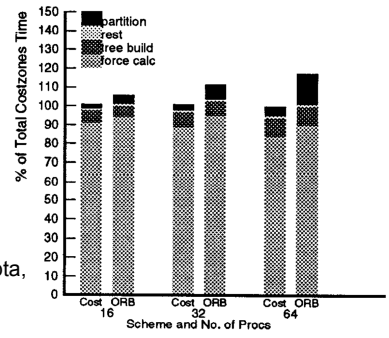


Morton versus Hilbert

21	23	29	31	53	55	61	63
20	22	28	30	52	54	60	62
17	19	25	27	49	51	57	59
16	18	24	26	48	50	56	58
5	7	13	15	37	39	45	47
4	6	12	14	36	38	44	46
1	3	9	11	33	35	41	43
0	2	8	10	32	34	40	42

21	22	25	26	37	38	41	42
20	23	24	27	36	39	40	43
19	18	29	28	35	34	45	44
16	17	30	31	32	33	46	47
15	12	11	10	53	52	51	48
14	13	8	9	54	55	50	49
1	2	7	6	57	56	61	62
0	3	4	5	58	59	60	63

Découpage (bilan)



[1] J. P. Singh, C. Holt, J. L. Hennessy, and A. Gupta, "A parallel adaptive fast multipole method," conference on Supercomputing, 1993, pp. 54–65.

Figure 10: Execution Profiles of *Costzones* and *ORB* on DASH.

On obtient des performances similaires avec un avantage pour l'approche costzone

Raisons

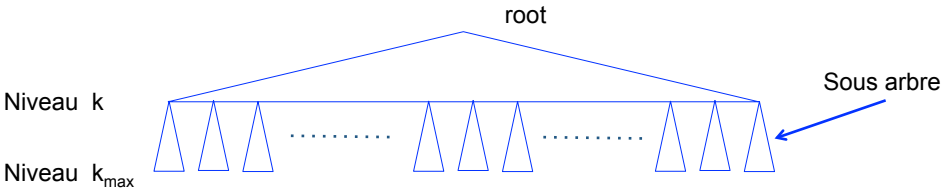
- Meilleur équilibrage → calcul des forces plus rapides
- L'algorithme ORB est plus couteux que celui des costzones

Peu de différences en Hilbert et Morton

Découpage (autres possibilités)

Idée

- On coupe l'arbre à un niveau k < k_{max}.
- 8^k sous arbres ($8=2^d$)
- Un sous arbre = arbre de hauteur L = $k_{max} - k$
- Nombre de sous arbres > nombre de processeurs

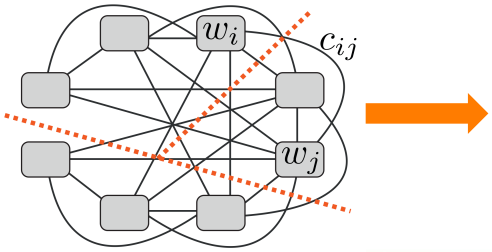


F. A. Cruz, M. G. Knepley, and L. A. Barba, "PetFMM—A dynamically load-balancing fast multipole library," Int. J. Numer. Meth. Engng., vol. 85, no. 4, pp. 403–428, 2011.

Découpage (autres possibilités)

On construit le graphe

- Nœud = sous graphe
- Arête si communication entre deux sous graphes
- Poids
  - ω_i Nœud : quantité de travail par sous arbres
  - C_{ij} Arête : volume de communication par sous arbres



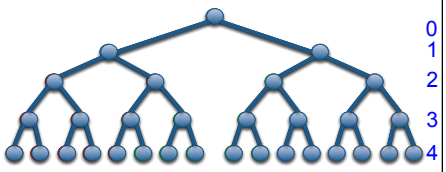
Partitionneur de graphe
(metis, scotch, …)

Octree

Eviter de dupliquer tout l'arbre

Deux approches dans le partitionnement

- l'arbre en entier
- uniquement les feuilles



Distribution uniquement sur les feuilles

- Niveau supérieur

Cellule fille n°1 (là gauche) donne l'appartenance au processeur

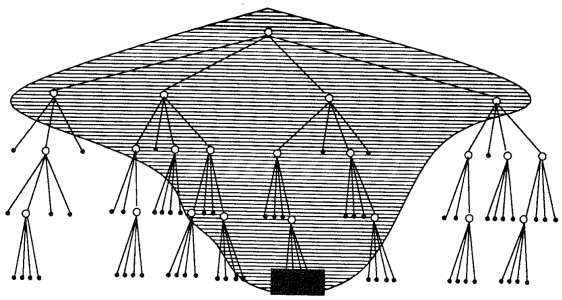
→ déséquilibre sur les niveaux supérieurs

→ minimise les communications pour M2M et M2L

Locally Essential Trees

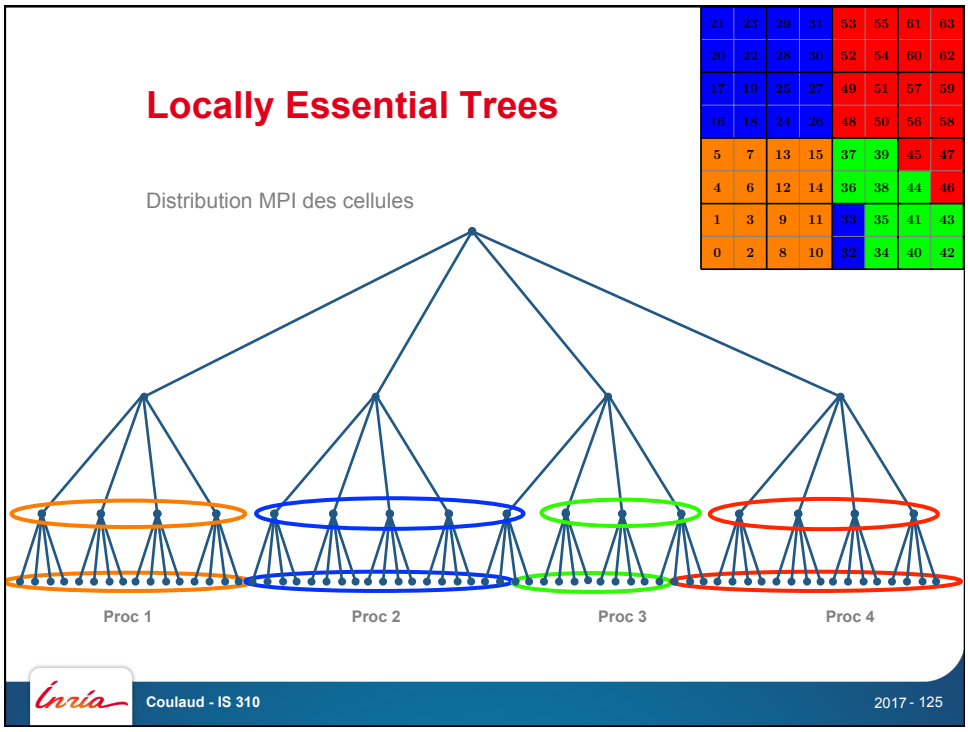
Idée: avoir toute la structure de l'arbre nécessaire aux opérateurs

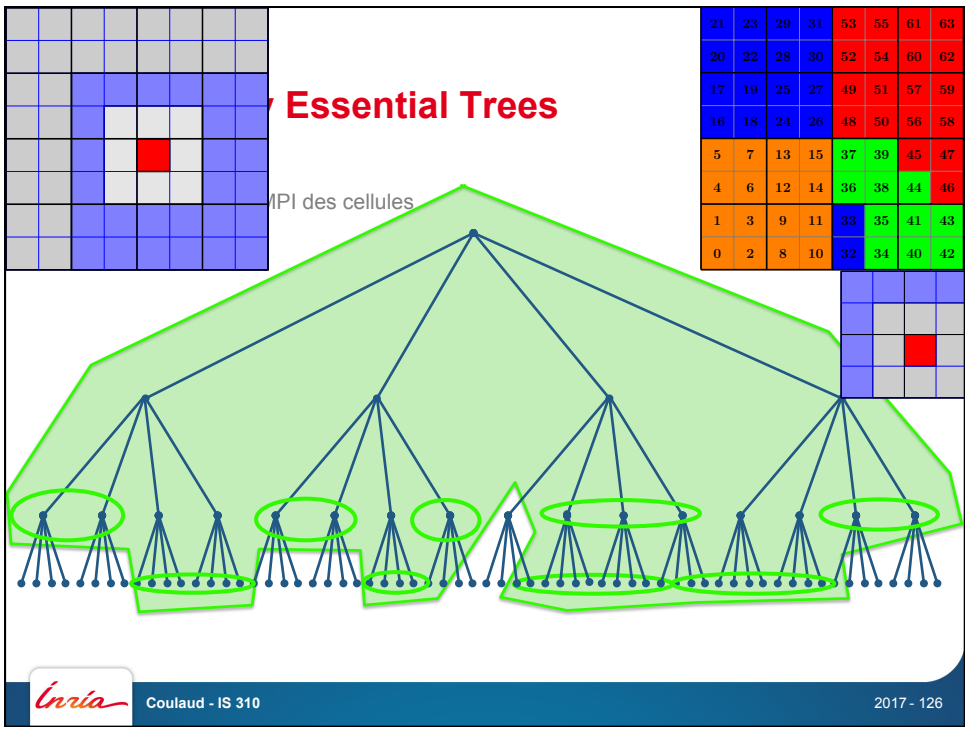
- Ensemble de feuilles sur un processeur
- L'ensemble des cellules/feuilles qui interviennent dans les opérateurs



LET = Arbre local + cellules fantômes

ORB


Locally Essential Trees
Distribution MPI des cellules
21 23 29 31 53 55 61 63
20 22 28 30 52 54 60 62
17 19 25 27 49 51 57 59
16 18 24 26 48 50 56 58
5 7 13 15 37 39 45 47
4 6 12 14 36 38 44 46
1 3 9 11 33 35 41 43
0 2 8 10 32 34 40 42
Proc 1
Proc 2
Proc 3
Proc 4
Coulaud - IS 310
2017 - 125


Essential Trees
MPI des cellules
21 23 29 31 53 55 61 63
20 22 28 30 52 54 60 62
17 19 25 27 49 51 57 59
16 18 24 26 48 50 56 58
5 7 13 15 37 39 45 47
4 6 12 14 36 38 44 46
1 3 9 11 33 35 41 43
0 2 8 10 32 34 40 42
Coulaud - IS 310
2017 - 126

Algorithme

Algorithme

- Supprime la 3ème loi de Newton

  Plus de calculs mais moins de communications à gérer

- Trois catégories de communications
  - calcul du champ proche (communications du calcul direct)
  - opérations M2L (communications de la liste d'interaction)
  - opérations M2M et L2L

  - Recouvrement des calculs par des communications
  - Emetteur pilote les communications

    Costzone : facile de savoir qui doit nous envoyer un message

    Numéro de morton → intervalle (Ok si la cellule existe)

Algorithme MPI

Etape 1 : construction et envoi des messages pour les communications du calcul direct ; (P2P)

Etape 2 : phase de remontée (incluant les communications M2M) ;

Etape 3 : terminaison des communications du calcul direct et traitement des messages reçus ;

Etape 4 : construction et envoi des messages pour les communications de la liste d'interaction ;

Etape 5 : phase du calcul direct ; (P2P)

Etape 6 : terminaison des communications de la liste d'interaction et traitement des messages reçus ;

Etape 7 : phase de descente (incluant les communications L2L) ;

Etape 8 : phase d'évaluation. (L2P)

Parallélisation hybride

Approche MPI/OpenMP

- Distribution uniquement sur les feuilles

- Sur un processus MPI : on découpe comme en mémoire partagée

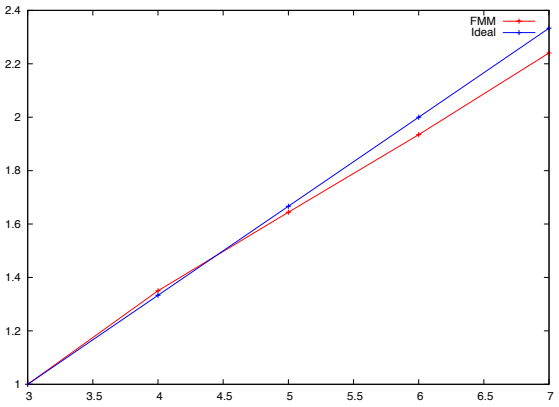
- Un thread dédié aux communications + recouvrement calculs/communications
  Section parallèle + parallélisme de boucle mais 1 thread perdu pour le calcul
  Tâches + section parallèle + délicat

  On travaille opérateur par opérateur :
  Etape 1 lancement des communications
  Etape 2 calculs locaux
  Etape 3 calculs distants

} Fait en parallèle

Scalabilité

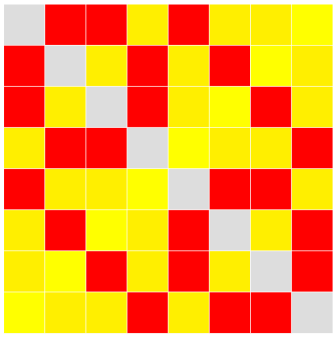
Nœud : 2 Deca-core Ivy-Bridge à 2,50 GHz
200 Millions de particules, Précision de 7 , octree 8
Modèle fork and join

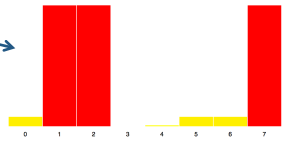


Matrice de communications

Exemple

8 processus MPI/ 8 threads



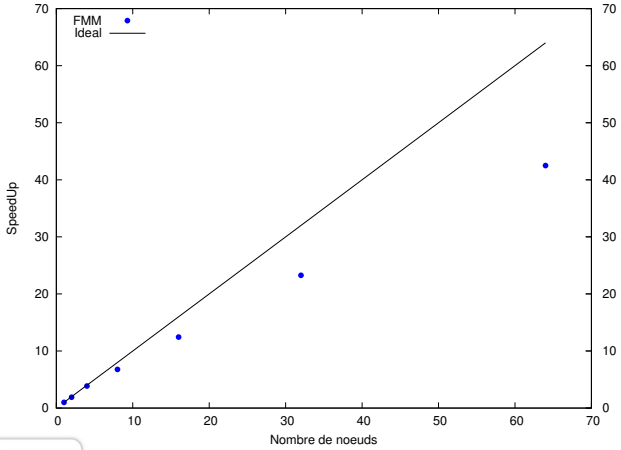


Volume de communications

Scalabilité

Nœud : 2 processeurs Intel Haswell à 2,50 GHz à 12 cœurs

100 Millions de particules, Précision de 7 , octree 8 ; Modèle fork and join



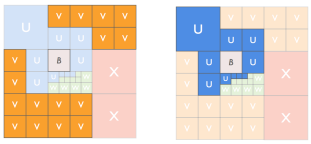
66% sur 1536 cœurs

Beaucoup de variantes

- Découpage différent entre la grille et l'arbre (i.e entre le champ proche et le champ lointain)
- En fonction du coût algorithmique des noyaux
- Sources et cibles différents

Des méthodes adaptatives

- Feuilles à des niveaux différents



- Des évolutions en fonction des nouvelles architectures (Accélérateurs GPU, MIC, …)

- …

Beaucoup de variantes

- Découpage différent entre la grille et l'arbre (i.e entre le champ proche et le champ lointain)
- En fonction du coût algorithmique des noyaux
- Sources et cibles différents

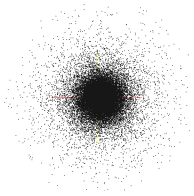
- Des évolutions en fonction des nouvelles architectures (Accélérateurs GPU, MIC, …)

- …

Approche Adaptative

Approche arbre uniforme

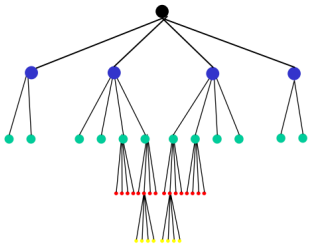
- Ok pour des distribution uniforme
- Ne marche pas si Cost(F1)/Cost(F2) >> 1
- Si arbre est trop profond



Idée

Mettre n0 particules par feuille

Si n > n0 on découpe



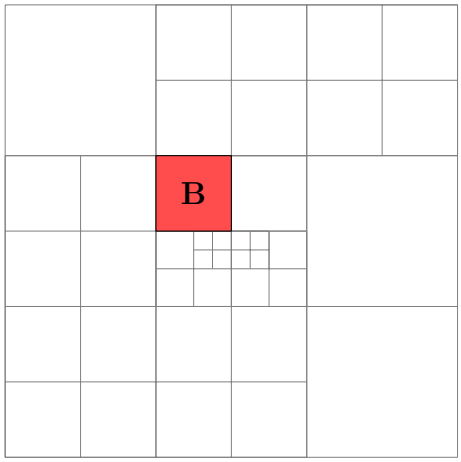
Quelques problèmes

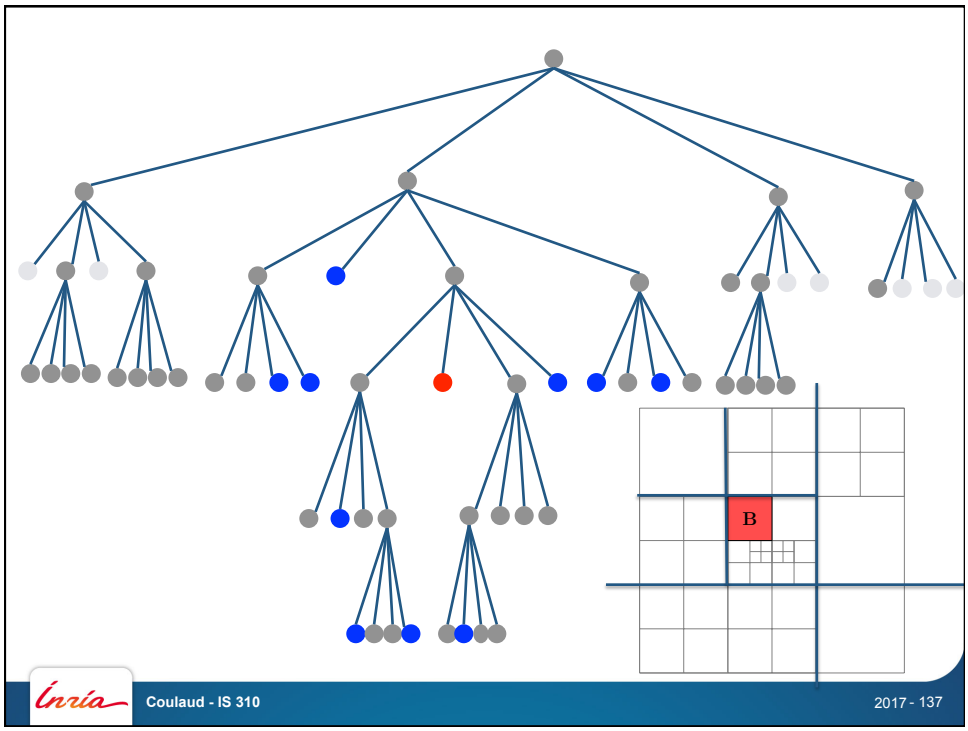
- Voisins sur plusieurs niveaux !
- Construction plus difficile
- Opérateurs entre Niveau

Approche Adaptative

Des listes plus compliquées

B cellule cible





Approche Adaptative

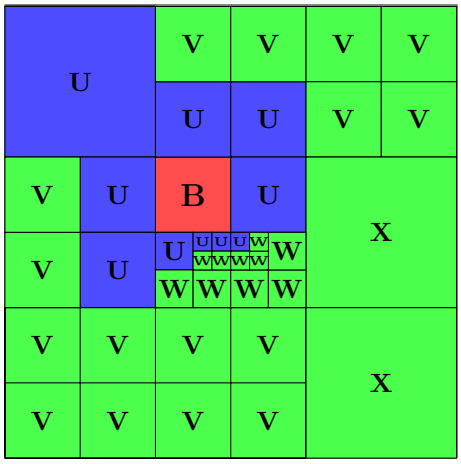
Des listes plus compliquées

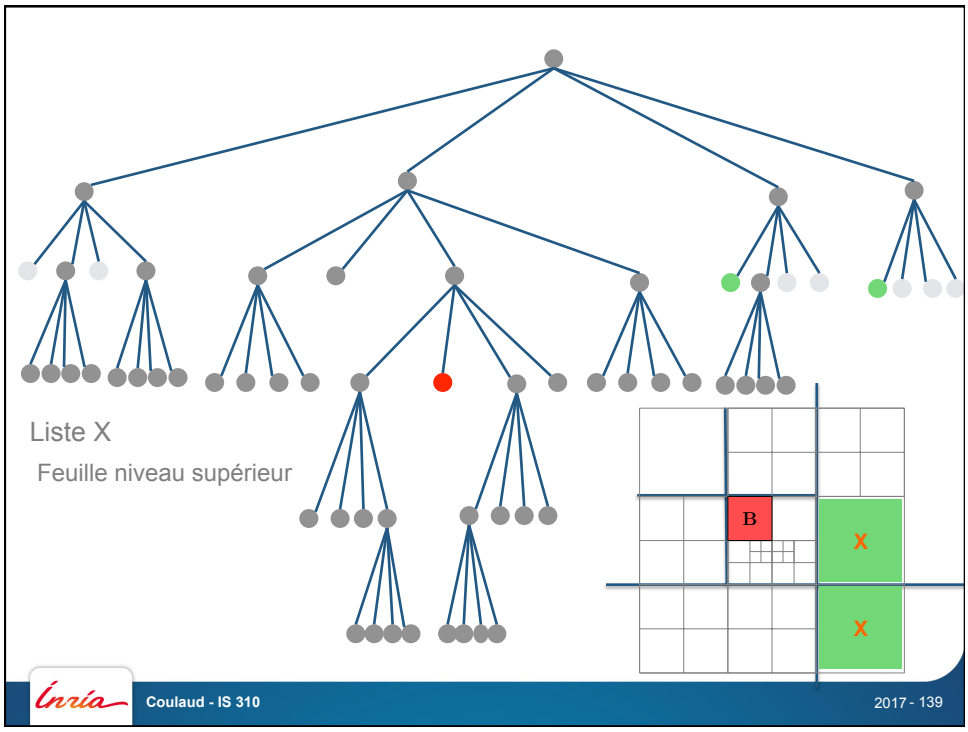
B cellule cible au

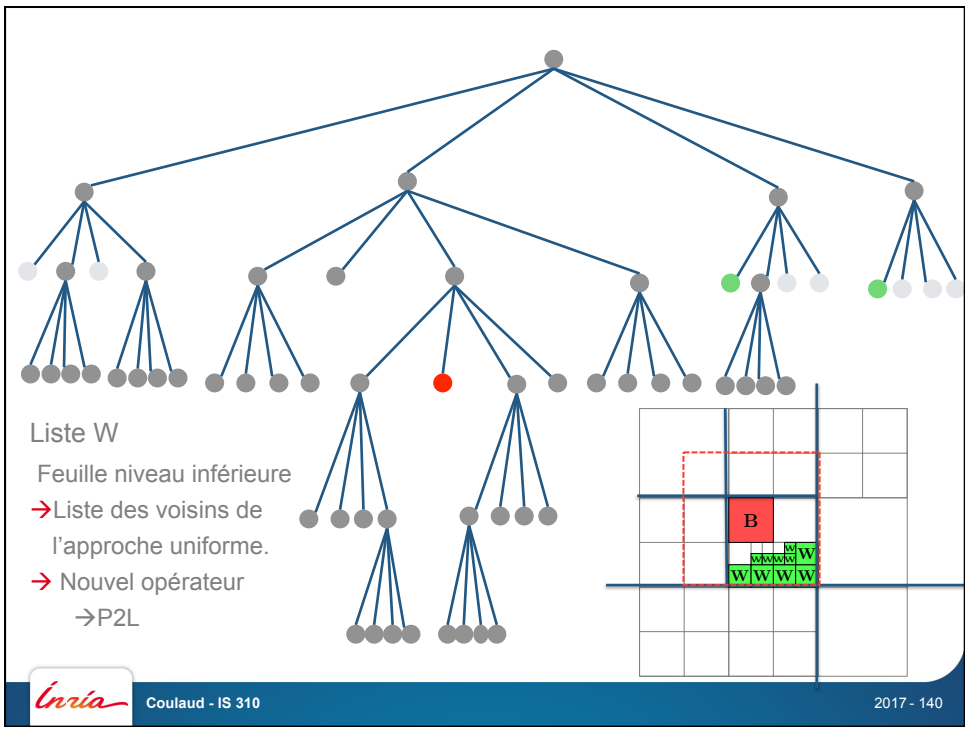
U Liste des voisins

Listes d'interactions

- V même niveau L
- X niveau L+1
- W niveau < L




Liste X
Feuille niveau supérieur
B
X
X
Coulaud - IS 310
2017 - 139


Liste W
Feuille niveau inférieure
→Liste des voisins de
l'approche uniforme.
→ Nouvel opérateur
→P2L
B
W
Coulaud - IS 310
2017 - 140

3 phases

1. Construction de l'octree
2. Calcul des listes U,V,W et X
3. Calcul FMM

Algorithme FMM

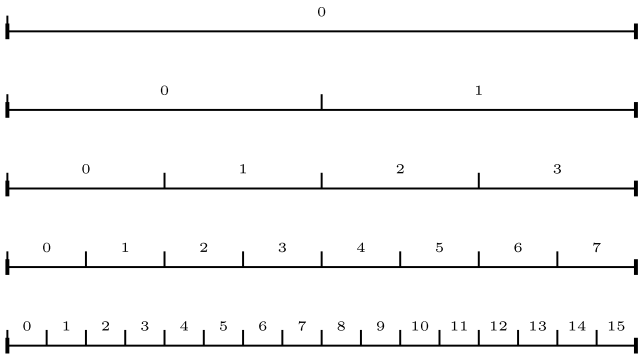
Chaque opérateur on parcourt l'arbre

- Phase de Montante P2M et M2P
- Phase de Descendante
- On applique le champ Lointain
- M2L, M2P où P2L
- Puis L2P
- Calcul Direct

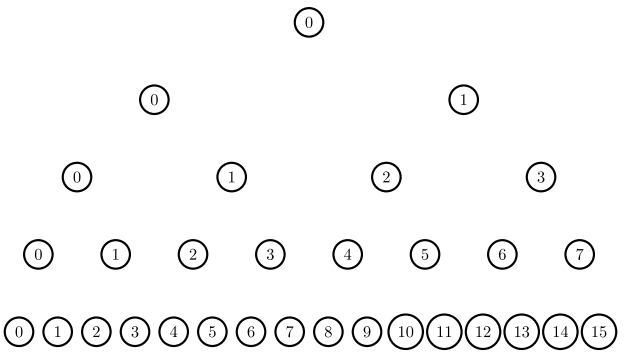
FIN

FMM 1-D

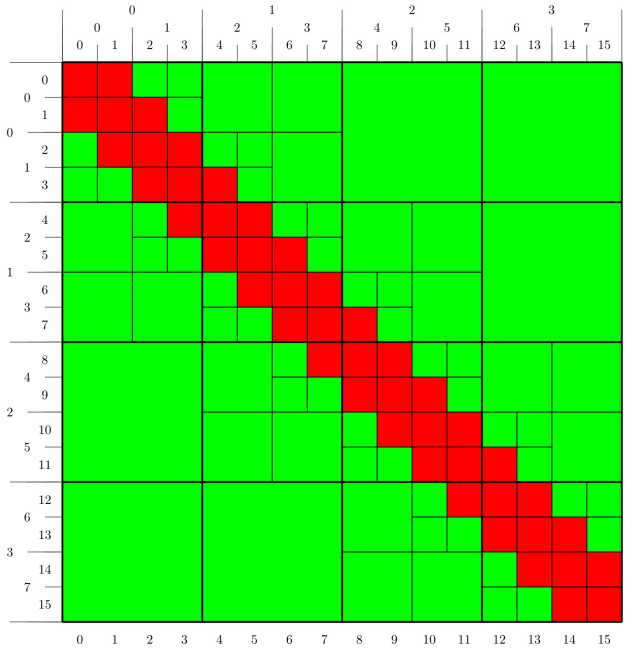
N particules dans [0,1]
Arbre de hauteur 4 → 2^4 = 16 feuilles



Niveau 0
Niveau 1
Niveau 2
Niveau 3
Niveau 4



Niveau 0
Niveau 1
Niveau 2
Niveau 3
Niveau 4



Scéance 1 :

introduction + méthode BH (-> construction de l'arbre)

Scéance 3 :

Fin de la méthode BH, FMM algo + interaction - space filling curve

Scéance 4 :

Sspace filling curve / mémoire partagée

Scéance 5 :

Distribué