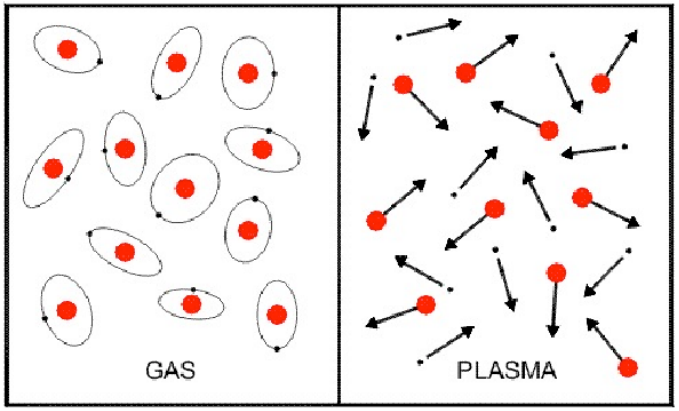
Accélérer des simulations d'accélérateurs

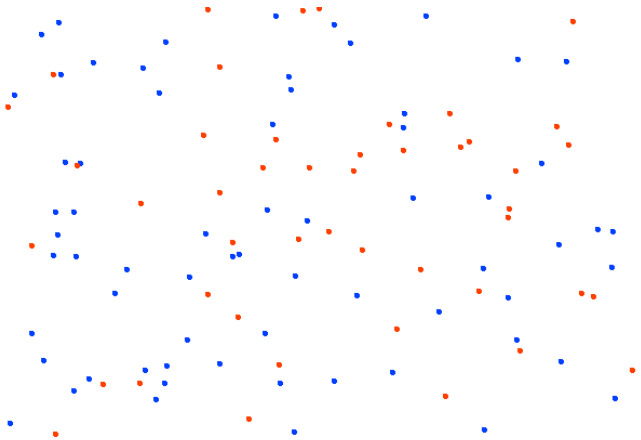
De la physique à l'optimisation numérique

Structure

Cours de l'exposé

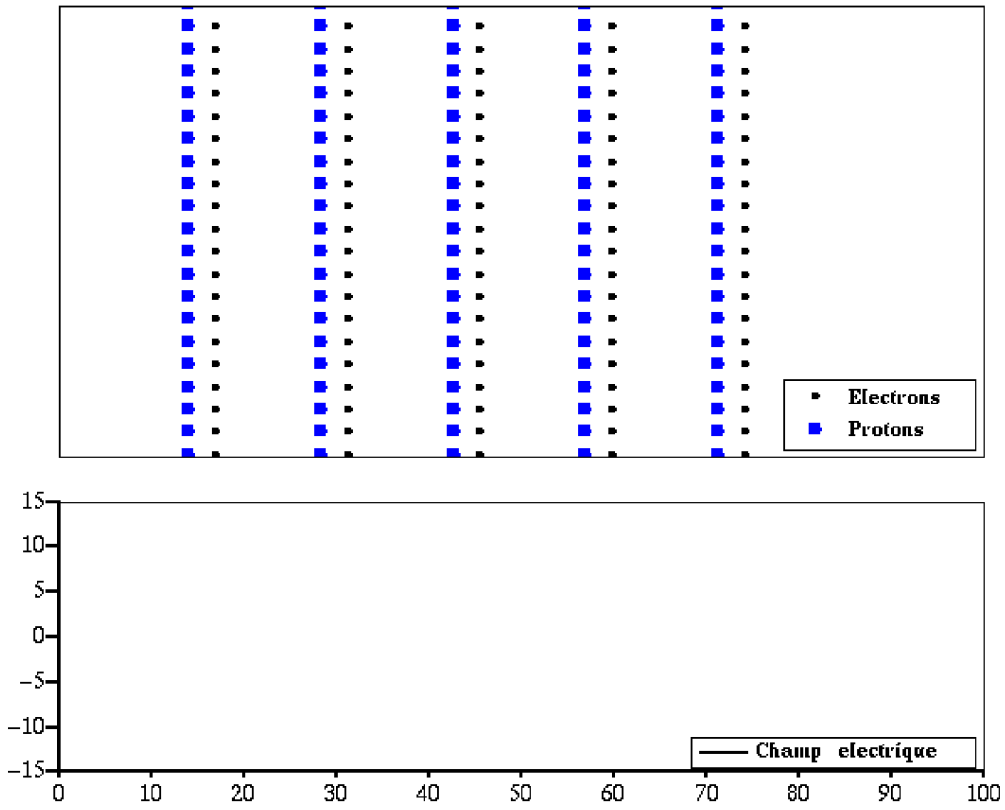
Gaz ionisé



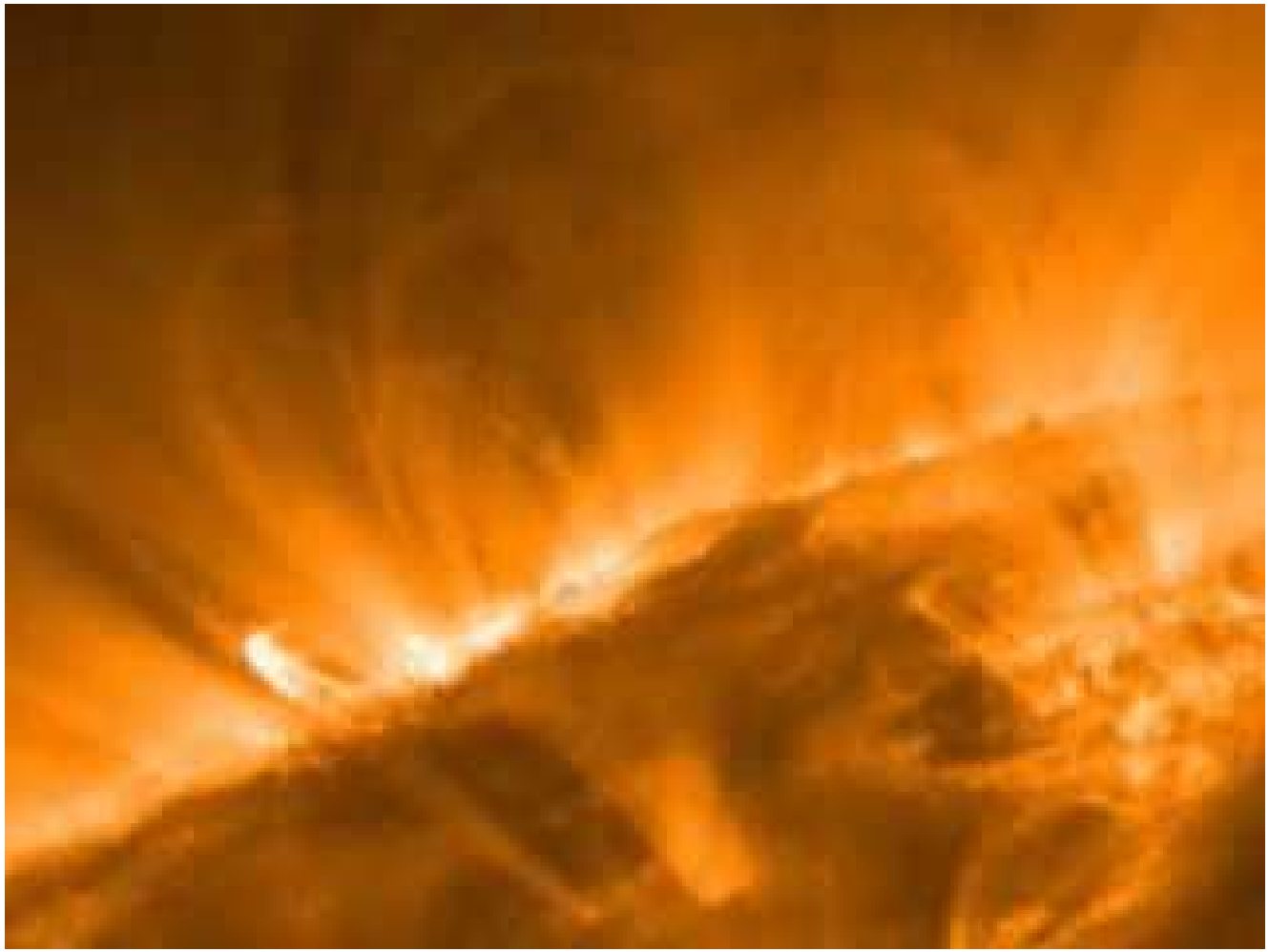


Plasma $\equiv$ Gaz ionisé dominé par les effets collectifs

Un exemple à petite échelle



Un exemple à grande échelle



A grande échelle le champ magnétique est structurant.

Alors comment décrire un plasma ?

Aux grandes échelles

Un nombre infini de particules dans un champ magnétique

- Approche fluide : Magnéto-hydrodynamique (MHD)
- Approche statistique : description de l'évolution de la fonction de distribution des particules, équation de Vlasov.

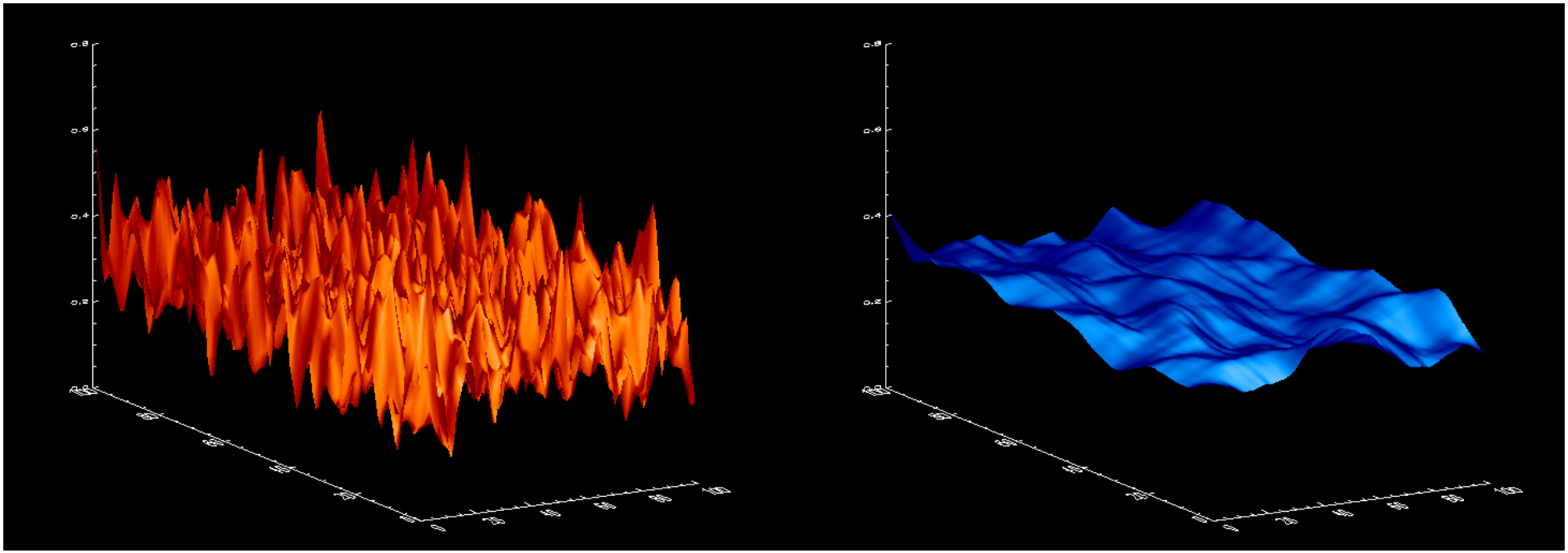
Aux petites échelles

Un nombre fini, mais grand, de particules dans un champ électromagnétique :

- Approche statistique valable car les effets collectifs dominent.
- N-Corps pour les cas pathologiques (ex : zone de transition solaire).

La représentation Vlasovienne

L'approche 'Vlasov' : les champs électriques sont des champs moyens et lisses produits par des effets collectifs et auto-consistants avec la dynamique des particules.



Champ électrique 'réel'

Champ électrique 'Vlasov'

Cours de l'exposé

Un code PIC résout des équations

$f_s(\mathbf{x},\mathbf{v})d\mathbf{x}d\mathbf{v}$ is the probability to find a particle of species s in the phase space point ($\mathbf{x}$,$\mathbf{v}$) around d$\mathbf{x}$d$\mathbf{v}$.

Vlasov equation

No collisions

$$\frac{\partial f_s}{\partial t} + \mathbf{v}\cdot\frac{\partial f_s}{\partial \mathbf{x}} + q_s/m_s(\mathbf{E} + \frac{\mathbf{v}\times\mathbf{B}}{c})\cdot\frac{\partial f_s}{\partial \mathbf{v}} = 0$$

Maxwell's equations

$$\begin{cases} \nabla\cdot\mathbf{E} = 4\pi\rho \\ \nabla\cdot\mathbf{B} = 0 \\ \nabla\times\mathbf{E} = -\frac{1}{c}\frac{\partial\mathbf{B}}{\partial t} \\ \nabla\times\mathbf{B} = \frac{1}{c}\frac{\partial\mathbf{E}}{\partial t} + \frac{4\pi}{c}\mathbf{J} \end{cases}$$

Moments equations

$$\rho = \sum_s^{n_s} q_s \int f_s d\mathbf{v}$$

$$\mathbf{J} = \sum_s^{n_s} q_s \int \mathbf{v} f_s d\mathbf{v}$$

Difficulté ==> les équations sont couplées !

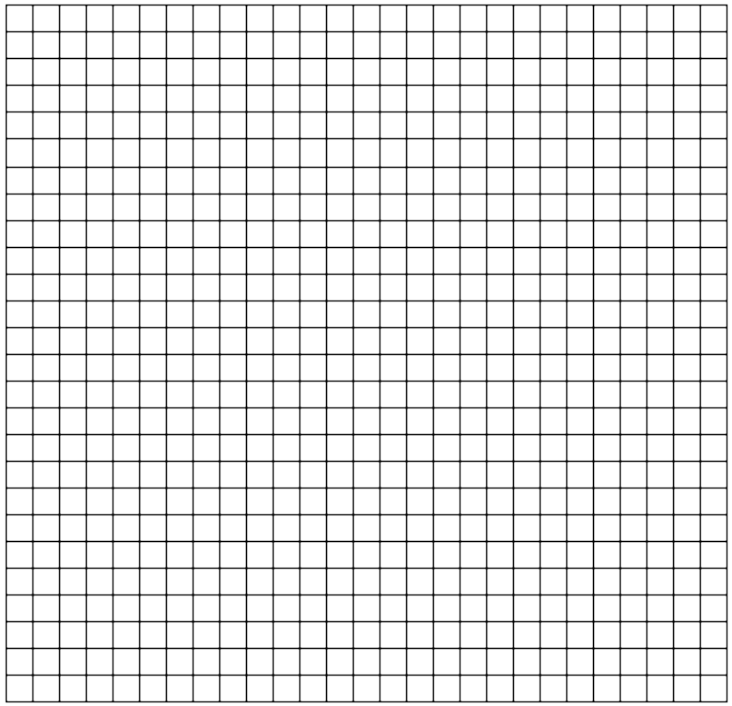
Comment fait on pour les champs ?

Les champs sont 3D (x, y, z).

Pour les discrétiser, il faut discrétiser l'espace.

$\downarrow$

Grille ou Maillage



Les équations de Maxwell sont discrétisées par différences finies et résolues sur cette grille.

Comment fait on pour la fonction de distribution ?

f est 6D (x, y, z, v_x, v_y, v_z).

Pour la discrétiser il nous faut discrétiser l'espace des phases.

$\downarrow$

Un maillage 6D est très coûteux !

L'astuce

On échantillonne l'espace des phases avec des macro-particules qui sont des objets qui ressemblent à des particules (position & vitesse) mais qui ont une forme et représentent un petit morceau d'espace des phases.

$$f = \sum_{p=0}^{N-1} f_p$$

PIC = Particle In Cell

Pourquoi ça marche ?

L'équation de Vlasov est linéaire en f donc si chacun des f_p est solution, leur somme est solution.

On peut démontrer que résoudre l'équation de Vlasov pour tous les f_p est équivalent à déplacer les macro-particules comme si elles étaient des particules normales. Pour le cas non relativiste :

$$\vec{v_p} = \frac{d\vec{x_p}}{dt} \tag{1}$$

$$\frac{d\vec{v_p}}{dt} = q_p(\vec{E_p} + \vec{v_p} \times \vec{B_p}) \tag{2}$$

Mais le couplage est encore là !

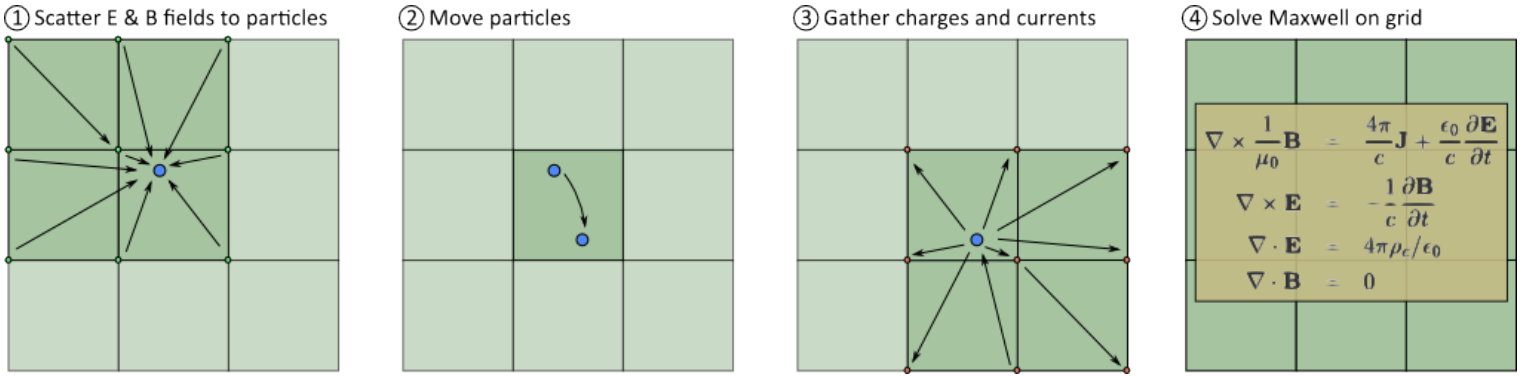
Une manière de découpler : les codes PIC explicites

Les deux jeux d'équations sont artificiellement résolues séparément.

- Gèle des particules - Avancement des champs
- Gèle des champs - Avancement des particules

On garantie la stabilité du schéma numérique en choisissant correctement les résolutions en temps et en espace (analyse numérique).

Les étapes d'un code PIC explicite



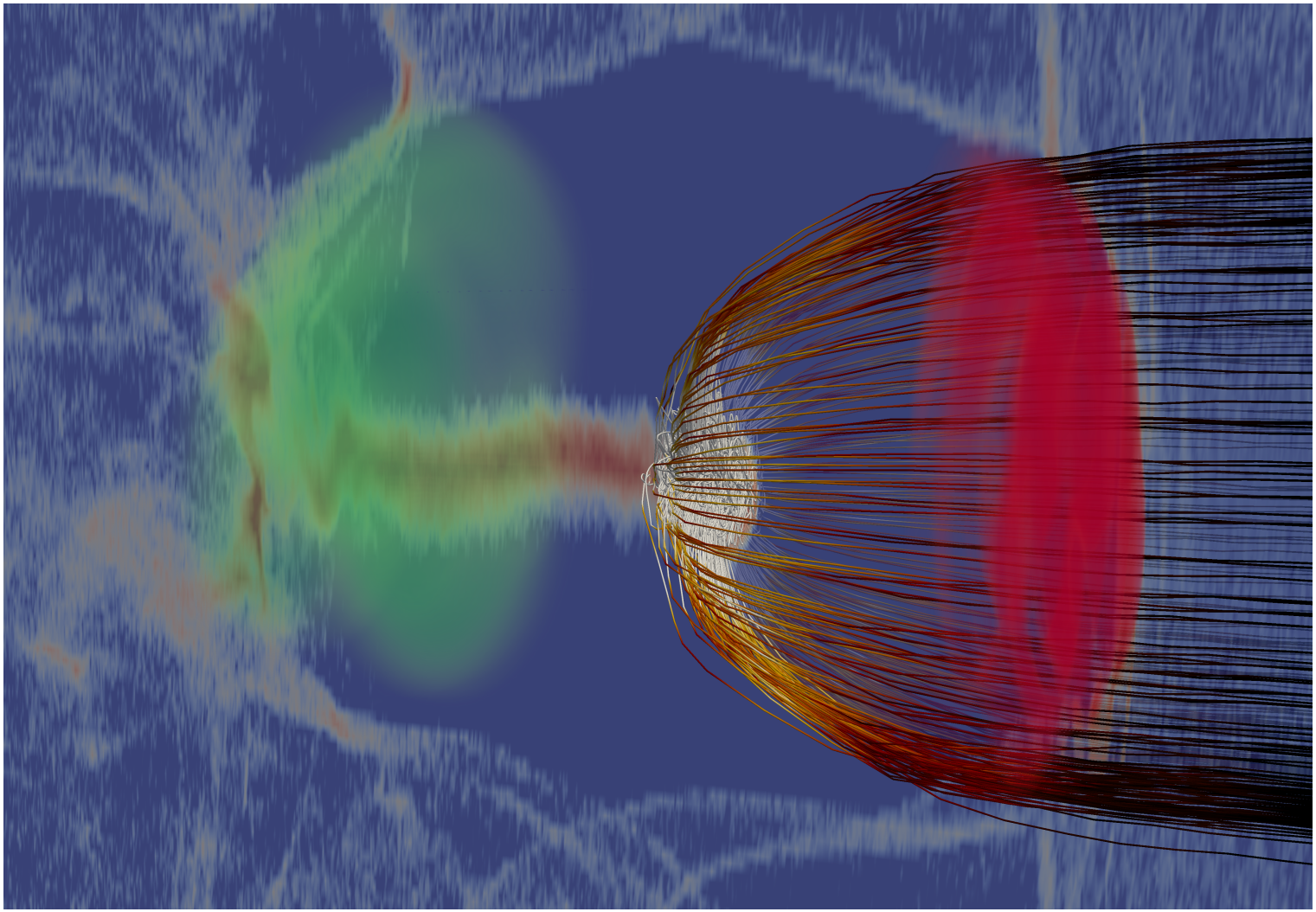
Étapes

1. Interpolation
2. Pousseur
3. Projection
4. Maxwell

Problèmes

- Accès aléatoires en mémoire.
- Équilibrage de charge.

Une vraie simulation



Les challenges numériques pour la simulation laser-plasma

Challenge 1 : Efficacité

- Les simulations 3D sont LOURDES ! $\simeq 10^9$ cellules, 8 particules/cellule, $\simeq 10^7$ itérations ==> 10^{19} opérations <==> 85 heures sur 3M+ coeurs
- Soit 255 Mheures CPU.
- Les plus grandes allocations PRACE < 100 Mheures CPU.
- ... en considérant que le code passe à l'échelle et utilise la performance crête de la machine (haha).
- Problème critique de l'équilibrage de charge : les performances chutent jusqu'à 40X.

R. Fonseca *et al*, *Exploiting multi-scale parallelism for large scale numerical modeling of laser wakefield accelerators*, Plasma Phys. Control. Fusion (2013)
A. Beck *et al*, *Load management strategy for Particle-In-Cell simulations in high energy particle acceleration*, Nucl. Instrum. Methods in Phys. Res. A (2016)

Les challenges numériques pour la simulation laser-plasma

Précision

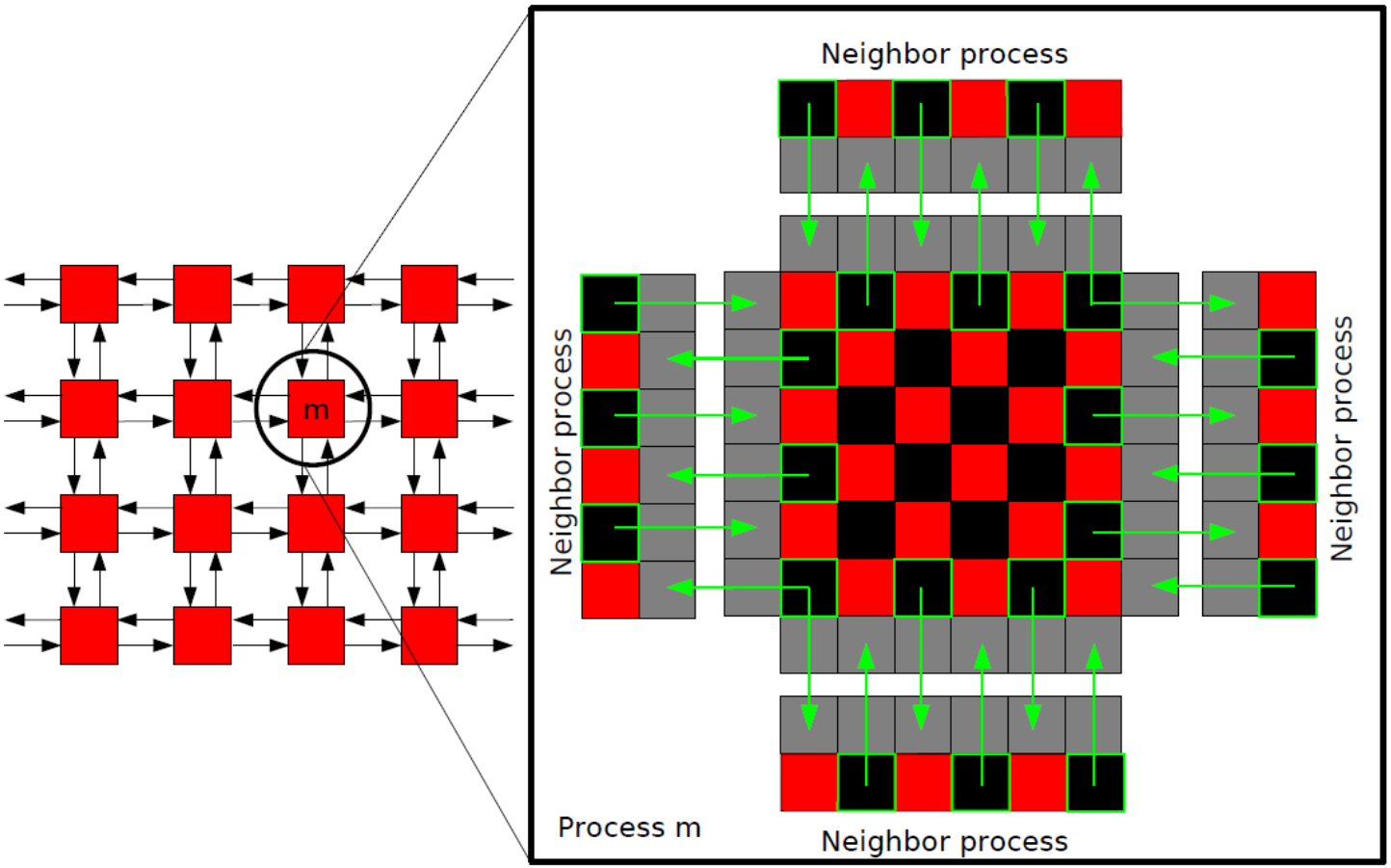
- Dispersion numérique : le laser doit se propager à la vitesse de groupe théorique.
- Pas d'accord quantitatif sur plusieurs grandeurs importantes et notamment la charge injectée.
- Physique manquante ? Collisions, QED, pertes radiatives ?

Cours de l'exposé

Un code PIC sur une architecture massivement parallèle

Bonne pratique #1 : Exposer un maximum de parallélisme de l'algorithme.

Premier niveau, la décomposition de domaine pour les mémoires distribuées.



Interpolation de particule à grille

Bonne pratique #2 : Adapter l'algorithme et les structures de données pour exposer encore plus de parallélisme (Cf B.P.#1).

Contents lists available at ScienceDirect

J. Parallel Distrib. Comput.

journal homepage: www.elsevier.com/locate/jpdc

Fast parallel particle-to-grid interpolation for plasma PIC simulations on the GPU

George Stantchev [a,*], William Dorland [a], Nail Gumerov [b]

[a] Center for Scientific Computing and Mathematical Modeling, University of Maryland, United States

[b] Institute for Advanced Computer Studies, University of Maryland, United States

ARTICLE INFO

Article history:
Received 12 March 2008
Received in revised form
10 May 2008
Accepted 10 May 2008
Available online xxxx

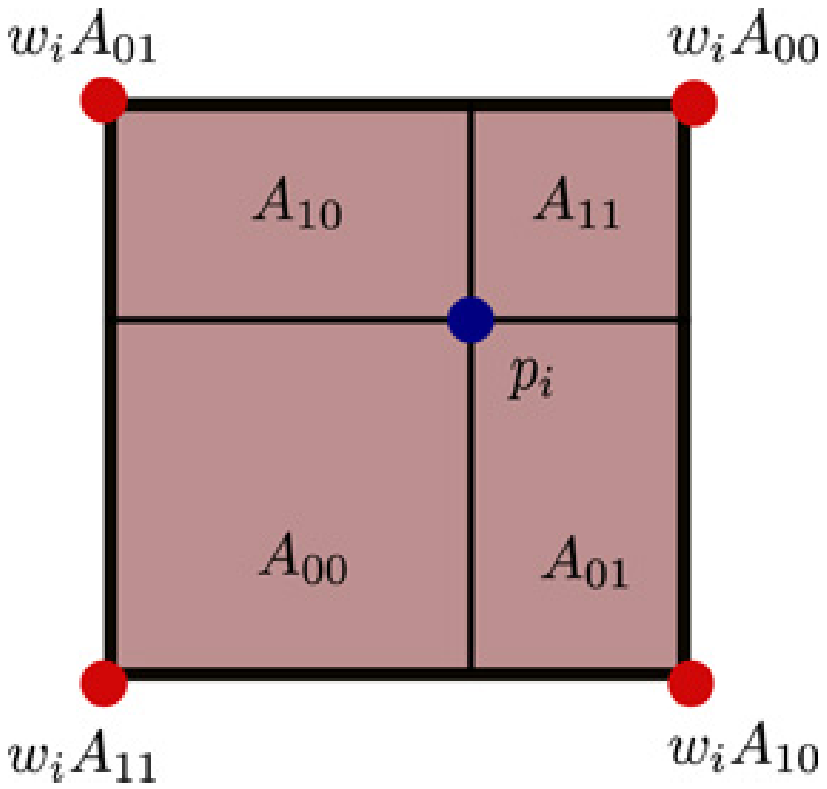
Keywords:
GPU computing
Scientific computing
Parallel algorithms
Numerical simulations
Particle-in-cell methods
Plasma physics

ABSTRACT

Particle-in-Cell (PIC) methods have been widely used for plasma physics simulations in the past three decades. To ensure an acceptable level of statistical accuracy relatively large numbers of particles are needed. State-of-the-art Graphics Processing Units (GPUs), with their high memory bandwidth, hundreds of SPMD processors, and half-a-teraflop performance potential, offer a viable alternative to distributed memory parallel computers for running medium-scale PIC plasma simulations on inexpensive commodity hardware. In this paper, we present an overview of a typical plasma PIC code and discuss its GPU implementation. In particular we focus on fast algorithms for the performance bottleneck operation of particle-to-grid interpolation.

Interpolation de particule à grille

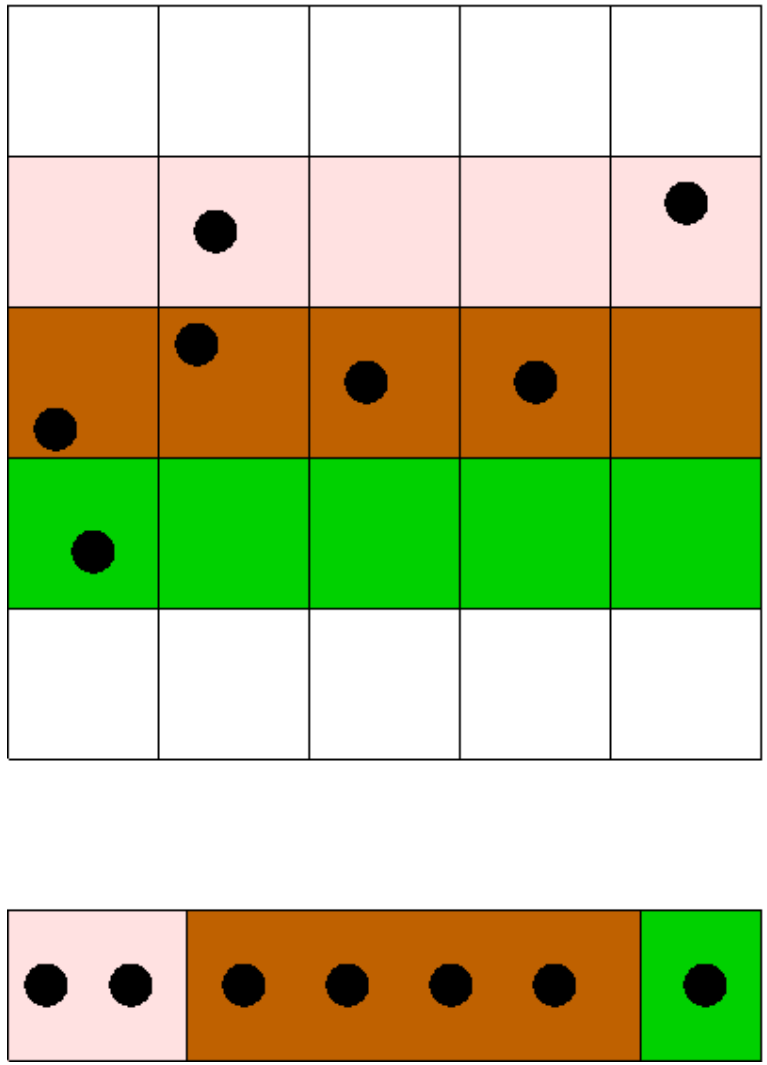
```
// Initialize f(v_s)
foreach vertex v_s ∈ G do
    f(v_s) ← 0 ;
end
// Loop over particles first
foreach particle p_i ∈ D do
    find V(p_i);
    foreach v_s ∈ V(p_i) do
        f(v_s) ← f(v_s) + w_i K(v_s, p_i)
    end
end
```



Problème d'accès mémoire aléatoire. Il faut trier, mais comment ?

Découpage du domaine en cluster

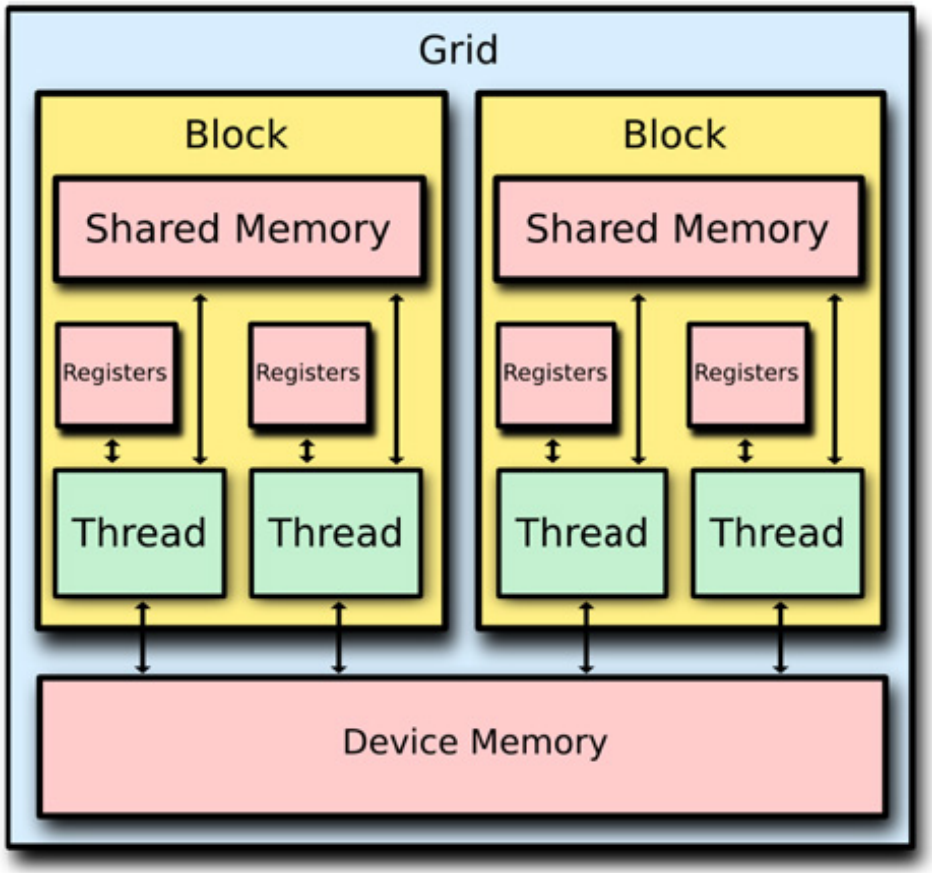
- Il faut des particules ordonnées en mémoire.
- On introduit une “granularité” dans le tri.
- Les clusters peuvent être traités indépendamment, on a bien extrait du parallélisme en plus en structurant nos données.
- 1 cluster = 1 block ou 1 thread openMP ou ...



Mémoire

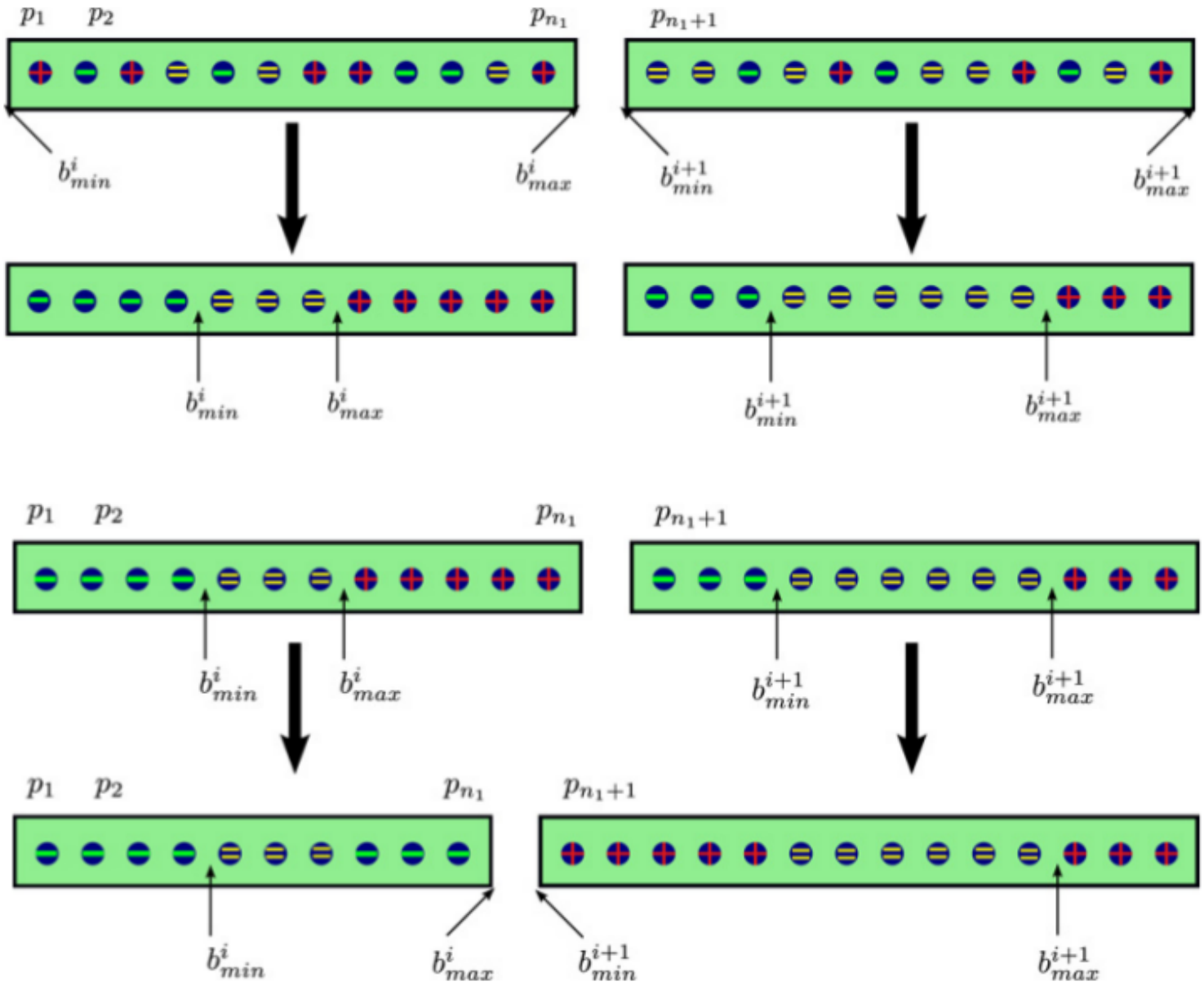
Architecture d'un GPU

- Threads structurés en block/grille
- registre = cache "très" local
- shared memory = cache "moins" local
- Gain en latence 100x
- Peu de mémoire shared
- Les blocks s'exécutent en parallèle (si possible)
- Bien choisir le nombre et la taille des blocks est primordial



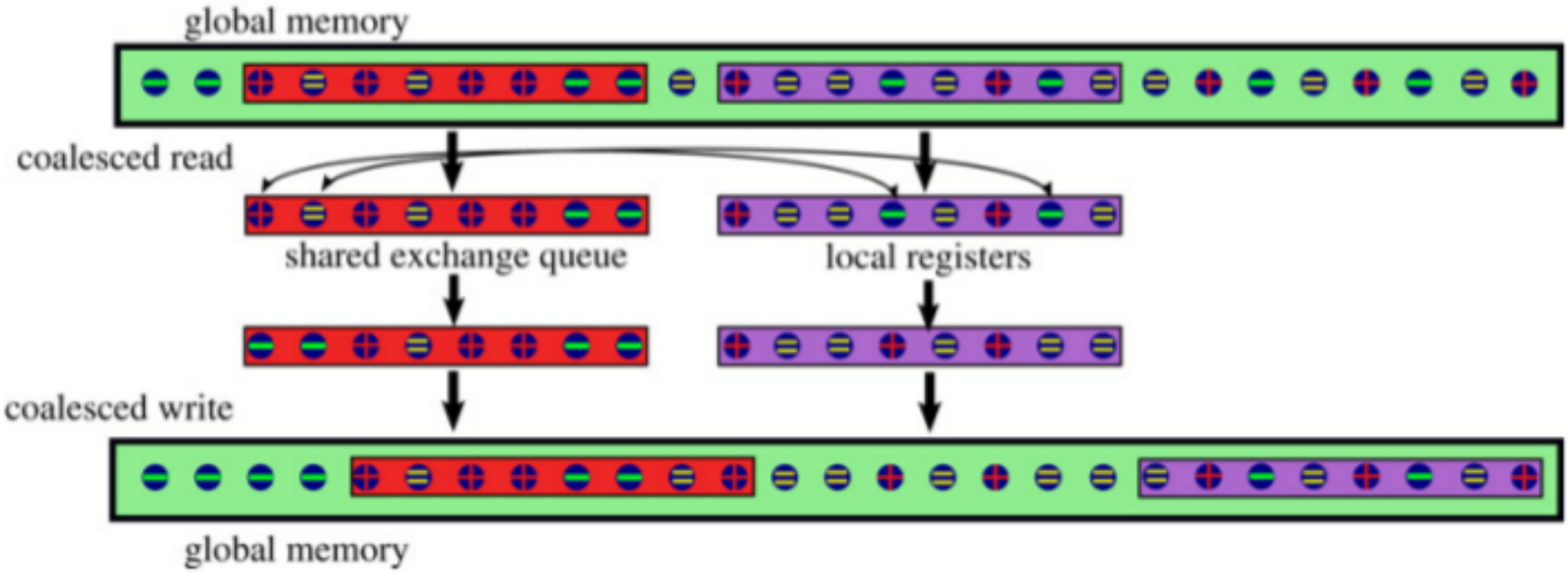
Bonne pratique #3 : avoir une structure de donnée adaptée à sa hiérarchie mémoire pour optimiser l'utilisation des caches.

Trie des particules (algorithme)



Implémentation du tri des particules pour un GPU

exemple de la “forward pass”



Optimise aussi sur CPU via SIMD

Bonne pratique #4 : Essayer de traiter des données contiguës en mémoire pour vectoriser.

Découle un peu naturellement des bonnes pratiques précédentes. Tout a été fait pour avoir des choses contiguës en mémoire ==> gain directe également sur le code CPU car les compilateurs peuvent vectoriser certaines opérations.

On peut aider le compilateur en utilisant des instructions SIMD. Ex : “memcpy” pour déplacer directement des blocs de mémoire de manière optimale.

Optimise aussi pour d'autres approches

Les efforts d'exposition de parallélisme ne sont jamais perdus.

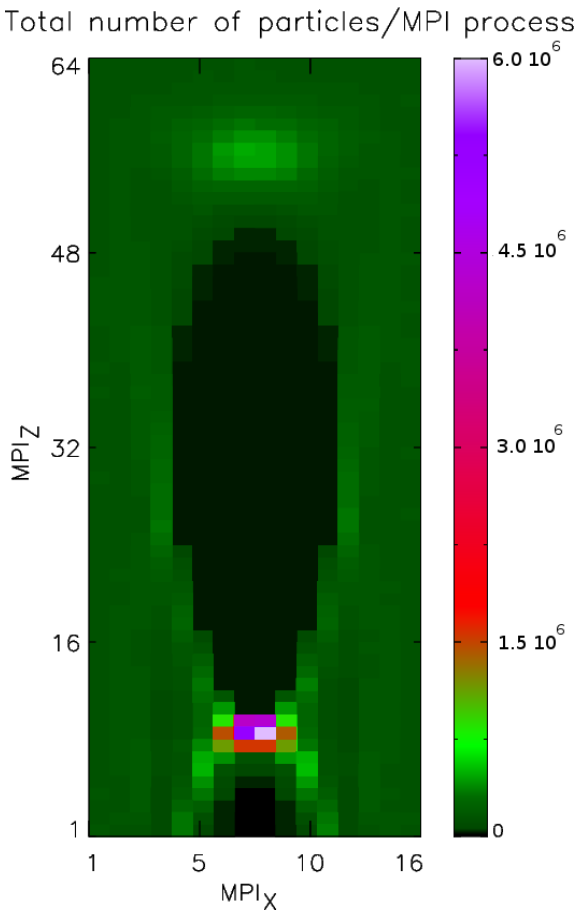
Les jobs répartis entre les différents "blocks" du GPU peuvent se paralléliser de manière équivalente entre les différents threads openMP.

$$\textit{CudaBlock} \simeq \textit{openMP thread}$$

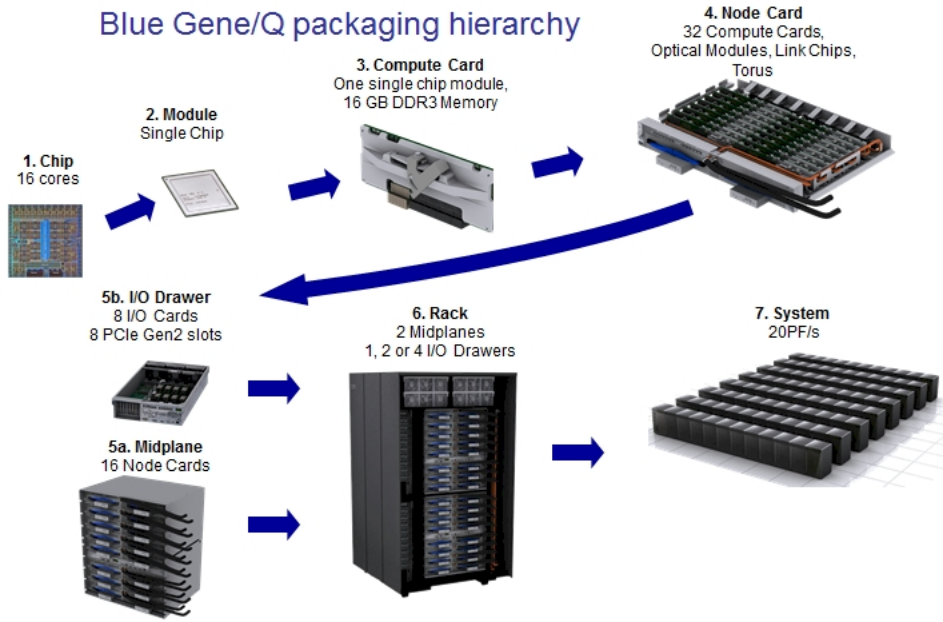
Cours de l'exposé

Le problème

Exemple d'accélération par sillage laser en vraie 3D,
code Photon-Plasma, Niels Bohr Institute.



2048 noeuds BG/Q.
16X16X64 processus MPI.
8 openMP threads.
Déséquilibre de 40X.



Beaucoup d'autres cas sont sensibles à ce problème (reconnection magnétique, interaction laser-solide ...)

La stratégie

Inspirée par des travaux préliminaires de K. Germaschewski (University of New Hampshire, arXiv :1310.7866).

- Éclater les domaines MPI en de nombreuses petites briques élémentaires appelées “Patch”.
- Les patchs peuvent être vus comme des sous domaines MPI, avec leurs propres particules et de champs.
- Chaque processus MPI possède un grand nombre de patchs.
- Les patchs définissent une granularité de tri et peuvent être traités indépendamment.
- Les patchs sont échangés entre processus MPI pour équilibrer la charge.

Space filling curve

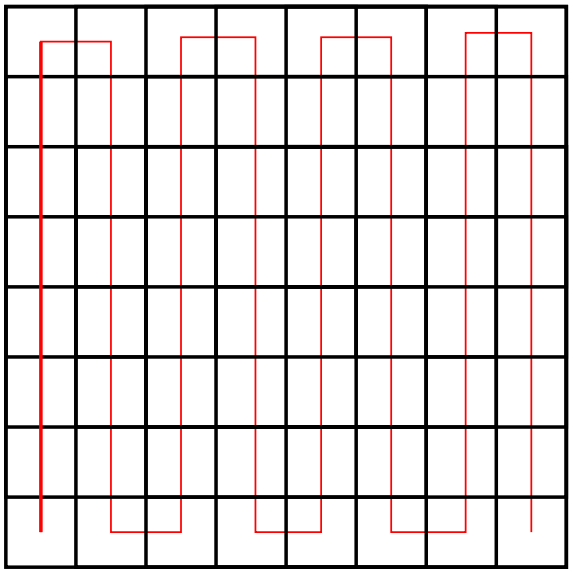
Nous avons besoin d'une politique pour assigner les patchs aux différents processus MPI (reconstruction du domaine). Pour cela, on organise les patchs le long d'une space-filling curve.

1. Courbe continue qui traverse chaque patch.
2. Chaque patch n'est visité qu'une seul fois.
3. Deux patchs consécutif sont voisins.
4. On veut en plus que les sous groupes de patchs ainsi définis soient le plus "compacts" possible.

Cette courbe est ensuite divisée en "*MPI_Comm_Size*" morceaux.

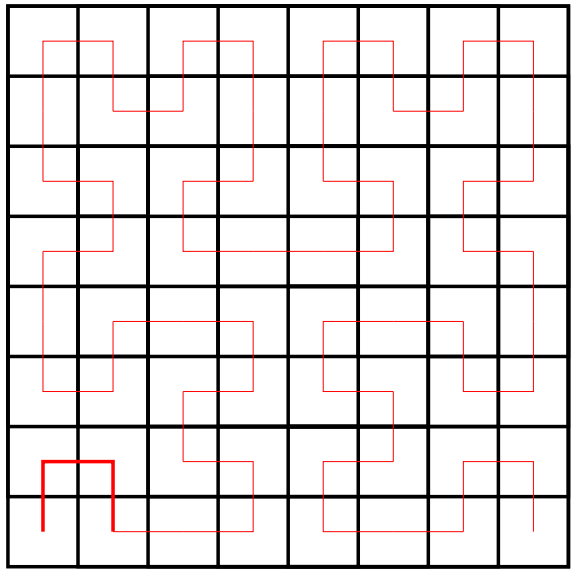
Exemples

Implémentation naïve



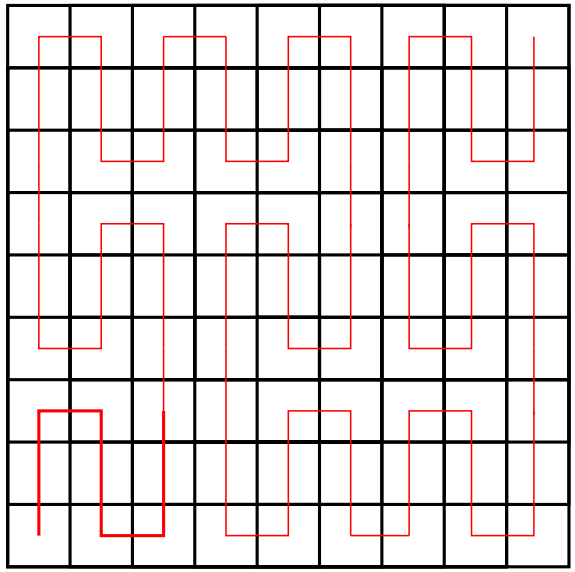
Base n.

Hilbert curve



Base 2.

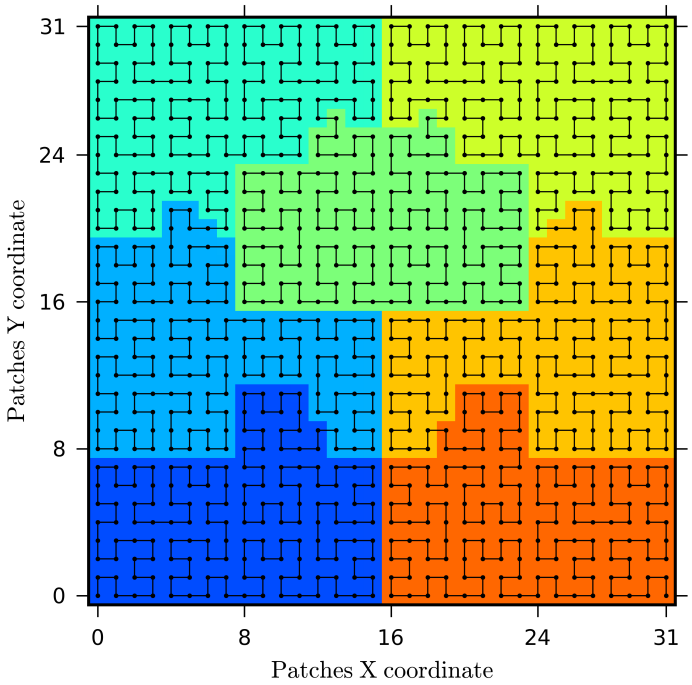
Peano curve



Base 3.

Exemple de partition initiale

32 X 32 patchs partagés entre 7 processus MPI.



Les synchronisations entre patchs sont majoritairement intra-noeud.
On utilise une généralisation des courbes de Hilbert pour supporter les grilles qui ne sont pas carrées.

L'algorithme d'équilibrage

1. Évaluation de la charge totale.
2. Évaluation de la charge optimale pour chaque processus.
3. Chaque processus réclame des patchs jusqu'à ce que la charge optimale soit approchée.
4. Échange de patchs pour satisfaire les réclamations si nécessaire.

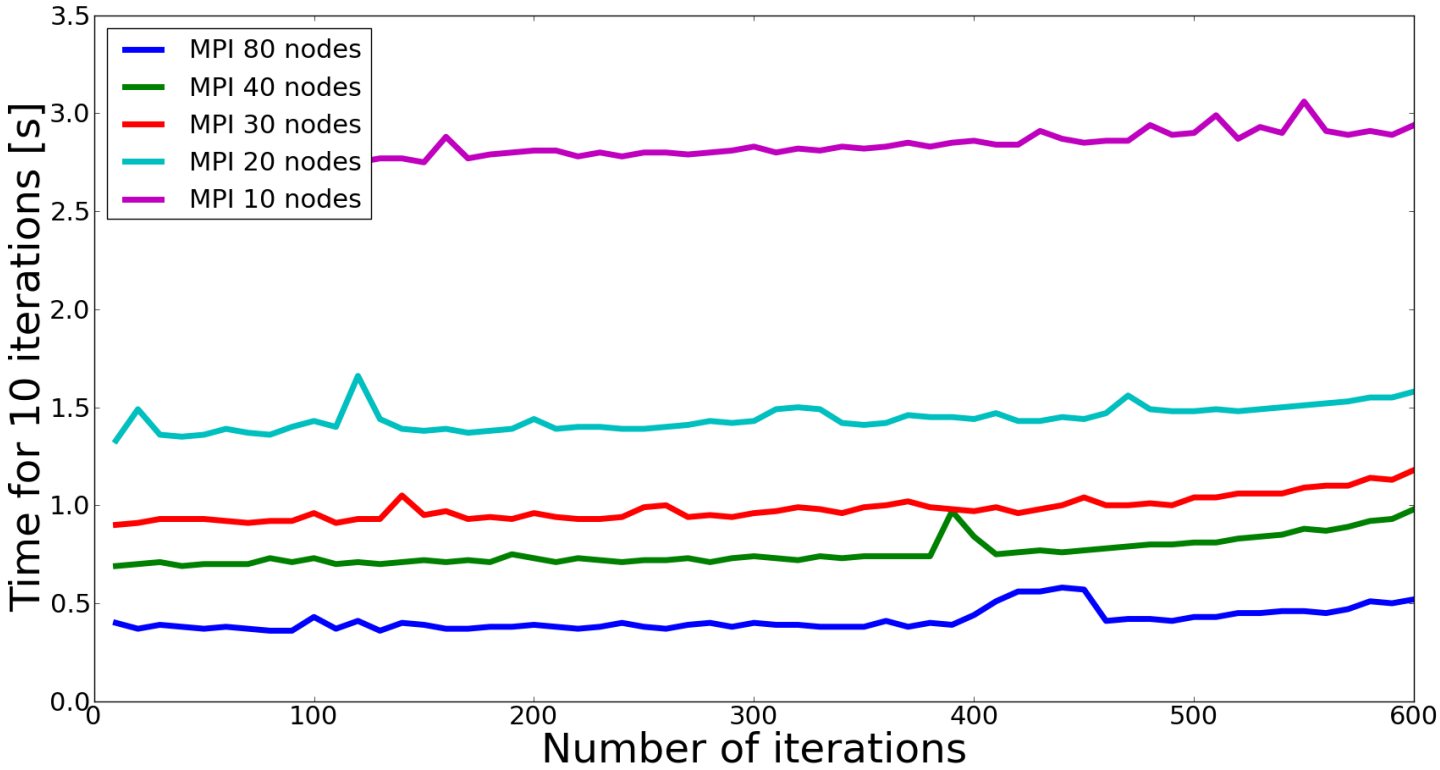
Paramètres

- Load = $N_{part} + \mathbf{L_{cell}} \times Ncell + \mathbf{L_{frozen}} \times N_{partfrozen}$
- **Capacité** de chaque processus MPI.
- **Fréquence d'équilibrage.**

Traditionnellement on a $L_{cell} = 2 - 10$, $L_{frozen} = 0.1$, équilibrage toutes les 20 itérations et la même capacité pour chaque noeud (cluster homogène).

So you think you can scale ?

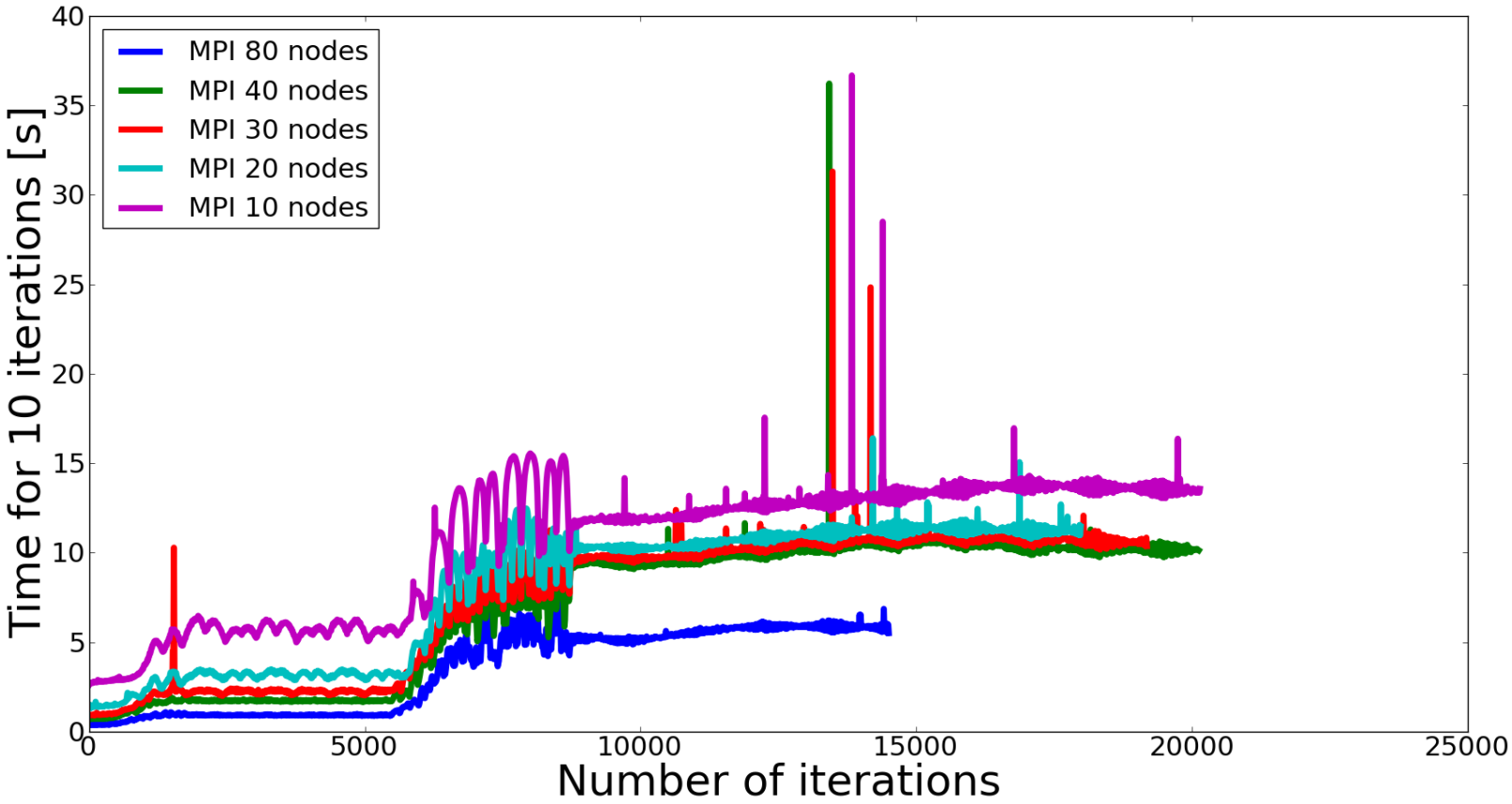
"Strong scaling" sur un domaine de 10240 $\times$ 1280 cell. 24 processus MPI processus par noeud (pure MPI)



En conditions triviales, un bon scaling MPI a été démontré jusqu'à plus de 50k+ coeur (système OCCIGEN).

La triste réalité

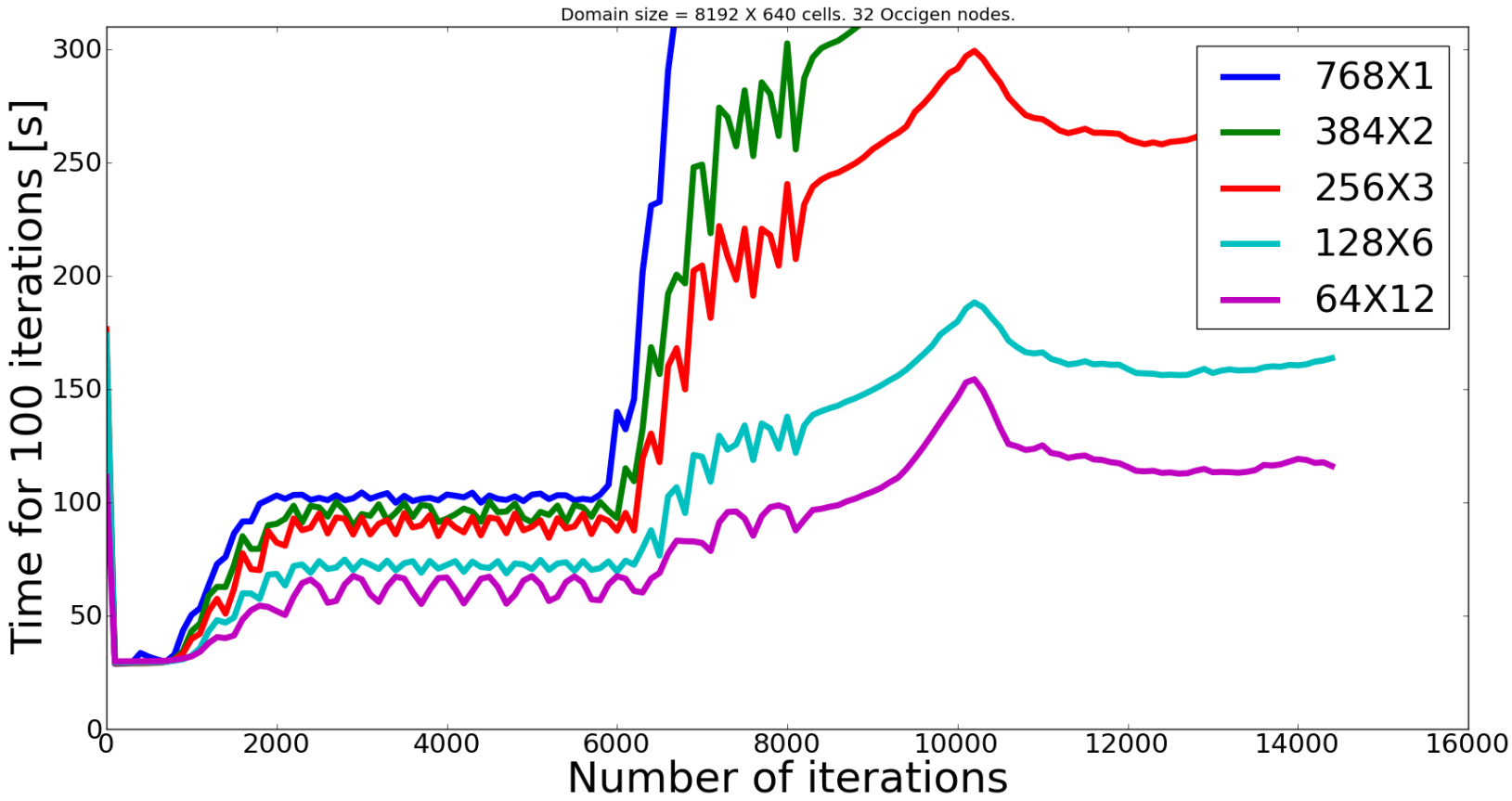
"Strong scaling" sur un domaine de 10240 $\times$ 1280 cell. 24 processus MPI processus par noeud (pure MPI)



Le déséquilibre fait perdre beaucoup de perf et du scaling.

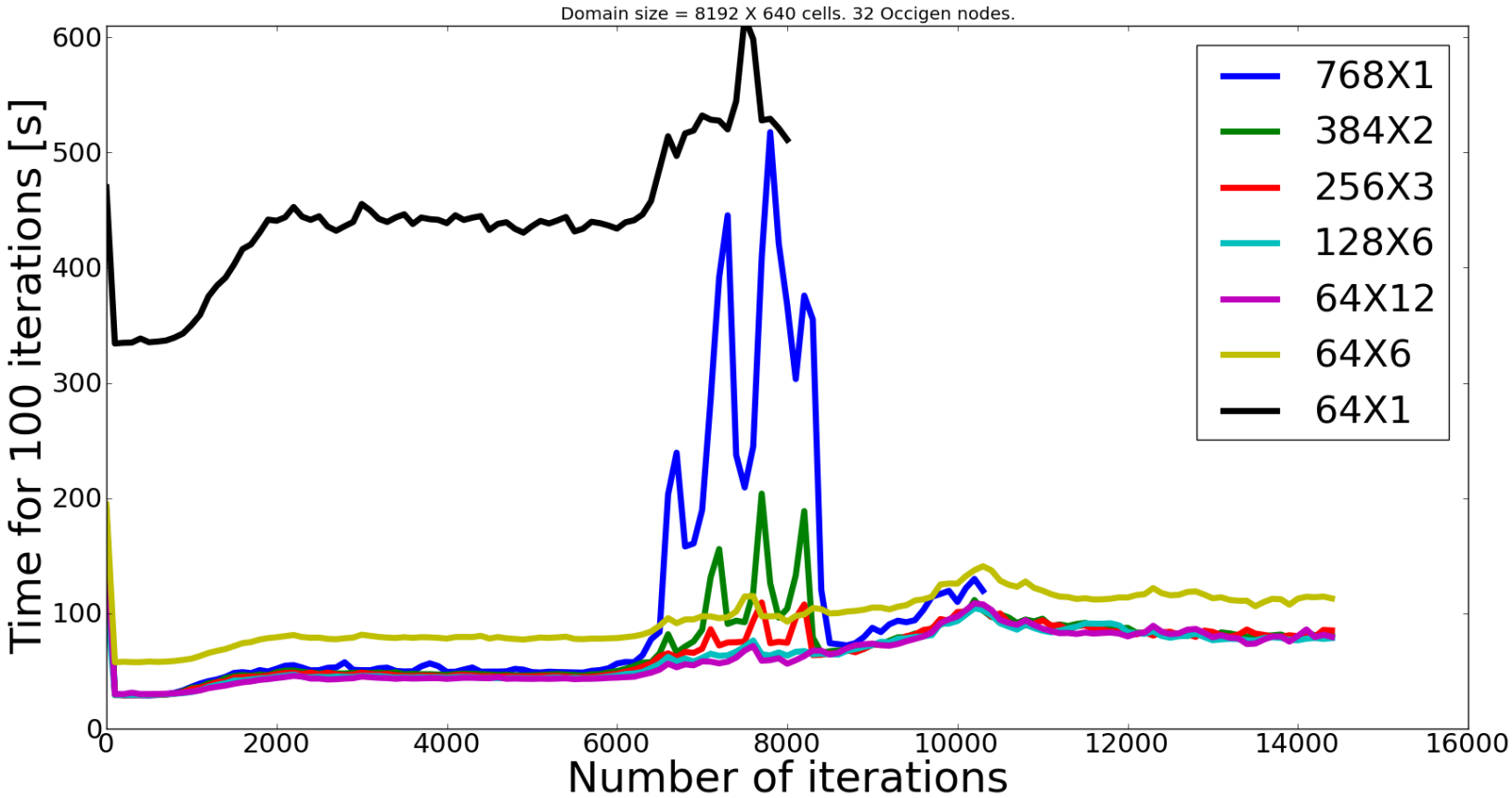
Une première solution, openMP

Scheduler dynamique



Boucle openMP sur les patchs. Le scheduler dynamique d'openMP fait relativement bien son boulot.

Avec l'équilibrage de charge



Les performances redeviennent correctes.

Les problèmes dont je n'ai pas parlé

- Conditions initiales (smooth start)
- Conditions limites (open boundaries, incoming/outgoing laser)
- Conservation de la quantité de mouvement
- Conservation de l'énergie
- Équation de Poisson et Charge Conserving Schemes
- Facteur de forme des particules
- Solveurs implicites
- Pousseur invariant par transformation de Lorentz
- Boosted frame
- I/O
- Énergie de rayonnement
- Ionisation
- Méthodes spectrales
- ...

Smilei

a collaborative, open-source, multi-purpose PIC code for the next generation of super-computers

Thank you for your attention!

SMILEI is still a young project but both the code and the community around it are developing quickly

http://www.maisondelasimulation.fr/projects/Smilei/html/index.html