Lilia Ziane Khodja

Post-doc à l'INRIA Bordeaux Sud-Ouest

Laboratoire d'Informatique de Paris 6 (LIP6)

Département Calcul Scientifique, équipe PEQUAN

Plan de la présentation

Parcours universitaire et professionnel

Recherche

- Thèse de doctorat
- Stage Post-doctoral

Interactions avec l'équipe PEQUAN

1 Parcours

universitaire et professionnel

Formation

Cursus universitaire :

- 2013 : Doctorat Informatique à l'Université de Franche-Comté
- 2009 : Master Informatique à l'Université de Rennes 1
- 2008 : Ingénieur Informatique à l'Université de Béjaïa, Algérie

Thèse de doctorat :

- Sujet : Résolution de systèmes linéaires et non linéaires creux sur clusters de GPUs
- Laboratoire : FEMTO-ST, département DISC, équipe AND
- Directeurs : Raphaël Couturier et Jacques Bahi

Statut actuel :

- Juillet 2013 : Post-doctorat à l'INRIA Bordeaux Sud-Ouest

Enseignement

Un contrat doctoral pendant 3 années et un ATER durant 6 mois

Environ 276 heures (éq. TD) à l'IUT Belfort-Montbéliard :

Cours	**Niveau**	**Heures**	
		TP	TD
Algorithmique	DUT 1	56	-
Architecture des ordinateurs	DUT 1	39	32
Bases de données	DUT 2	52	-
Réseau/Système	DUT 1	36	-
Réseau/Système	DUT 2	104	55

Responsabilités :

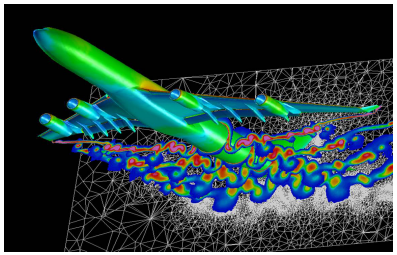
- Contenu TDs/TPs de l'Architectures et Réseau/Système
- Préparation d'examens de l'Architecture et Réseau/Système
- Suivi des stages DUT et licence professionnelle SIL

2 Recherche

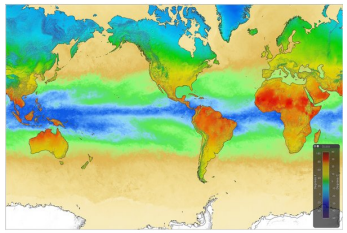
Thèse de doctorat

Contexte et positionnement

Les systèmes linéaires et non linéaires creux de grandes tailles interviennent fréquemment dans différents domaines scientifiques.

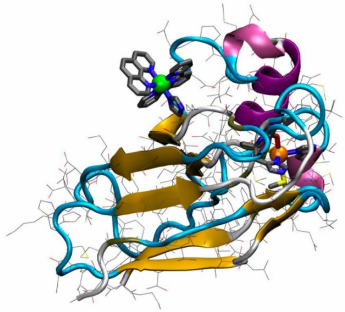


(a) Mécanique des fluides



(b) Simulation météorologique

(c) Calculs financiers



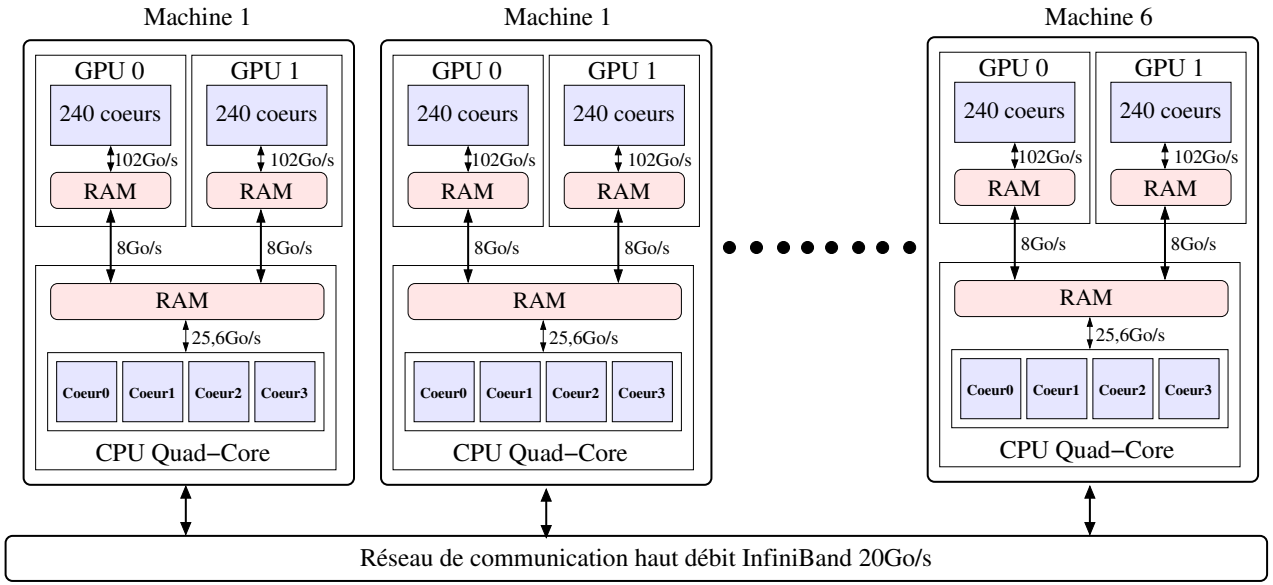
(d) Chimie numérique

Objectifs :

- Résolution de systèmes linéaires et non linéaires creux
- Conception d'algorithmes itératifs pour les clusters GPUs
- Programmation parallèle hétérogène CUDA/MPI

Cluster de GPUs

- Avantages :
  - Exploitation de la capacité de calcul des GPUs
  - Utilisation de moins d'espace de déploiement
  - Faible consommation d'énergie
- Contraintes :
  - Adaptation des algorithmes aux clusters GPUs
  - Programmation hétérogène GPU/CPU
  - Contrôle et gestion des communications



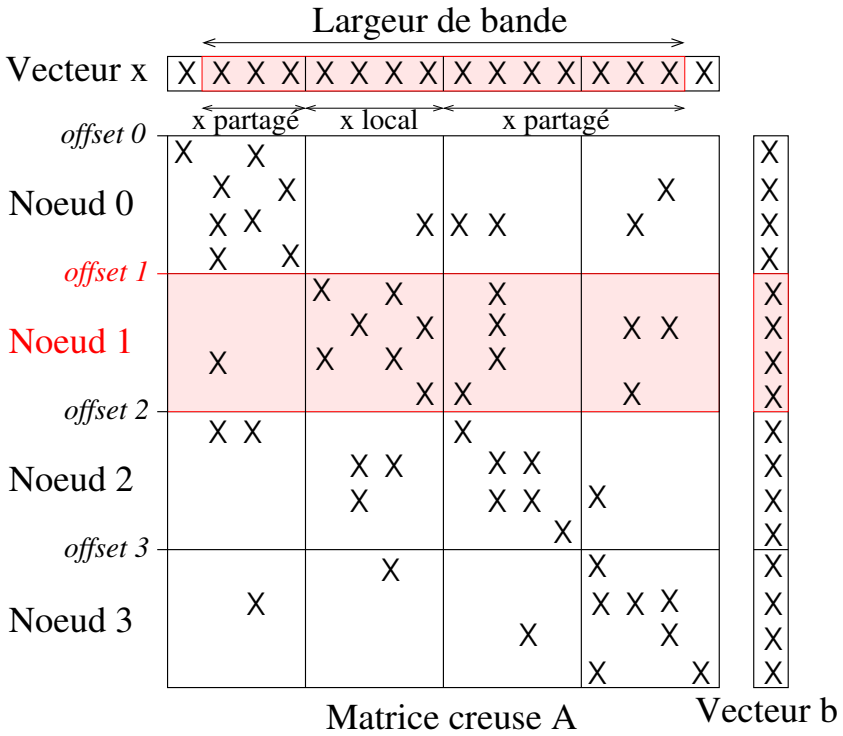
Méthodes de résolution numérique

Résolution d'un système linéaire creux : $Ax = b$

- Méthodes directes :
  - Robustes et prévisibles
  - Résolution de systèmes linéaires
  - Difficiles à paralléliser et supportent mal le passage à l'échelle
- Méthodes itératives :
  - Solutions par approximation
  - Résolution de systèmes linéaires et non linéaires
  - Faciles à paralléliser et supportent mieux le passage à l'échelle
- Méthode itérative la plus connue : GMRES
  - Plus adaptée aux systèmes linéaires asymétriques
  - Préconditionnement du système linéaire : $M^{-1}Ax = M^{-1}b$ où M est la matrice de préconditionnement

Parallélisation sur un cluster de GPUs

- Partitionnement ligne par ligne de A, x et b
- Au niveau de chaque nœud (processus MPI, GPU) :
  - Chargement des données dans la mémoire GPU
  - Exécution du même algorithme itératif
- Synchronisation par passage de messages (routines MPI)



Accélération de calcul sur GPUs

Objectifs :

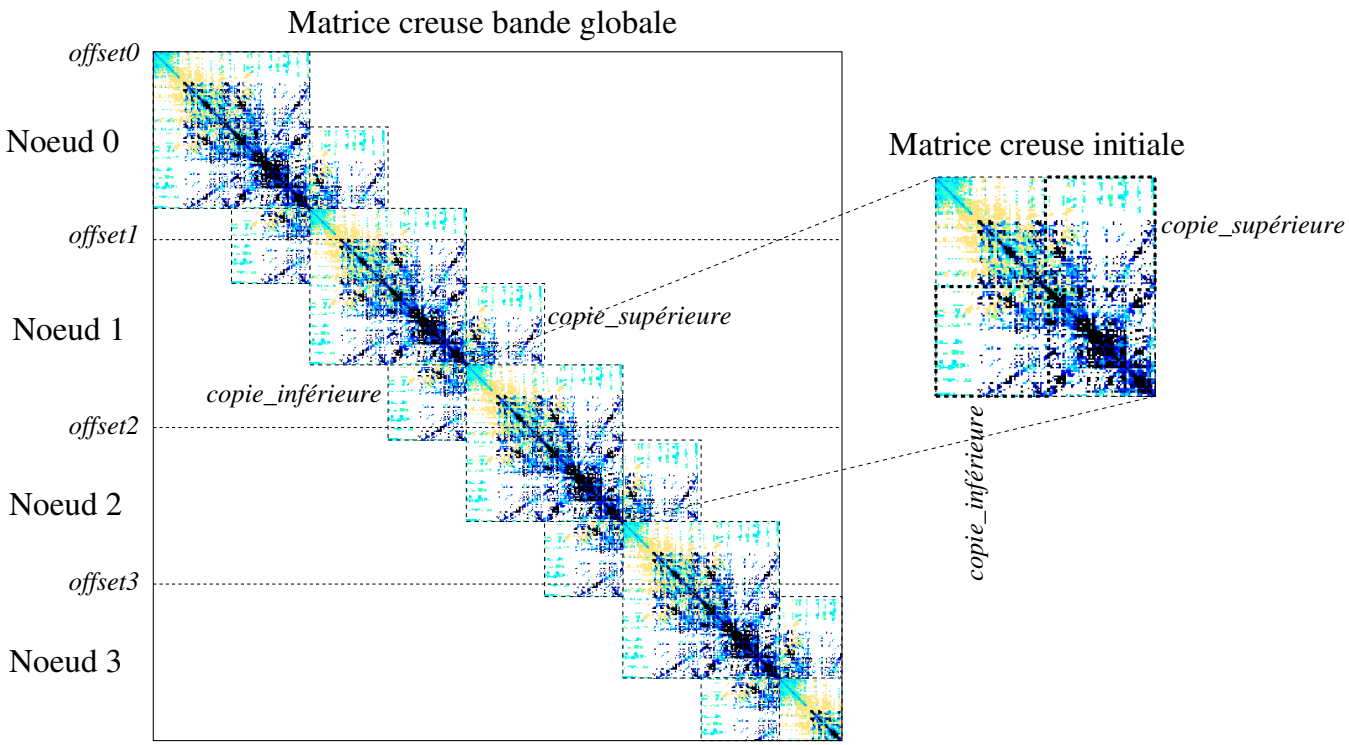
- Effectuer un maximum de calculs sur le GPU
- Lancer autant de threads GPUs que de données

Kernels CUDA :

- Toutes les fonctions opérant sur des vecteurs/matrices
- Préconditionneur M équivalent à la diagonale principale de A :
  - Facile à inverser M^{-1}
  - Produits matrice-vecteur ($M^{-1}Ax$ et $M^{-1}b$) moins coûteux
- Produit matrice creuse-vecteur : Kernel HYB de Nvidia
  - Accès faciles aux éléments non nuls
  - Exécution intelligente de la multiplication
  - Utilisation de la mémoire cache texture

Génération de matrices creuses

- Génération de matrices creuses de très grandes tailles :
  - Maximiser l'utilisation des cœurs GPUs
  - Utiliser les matrices de la bibliothèque de l'université de Floride (applications réelles)
  - Générer des matrices bandes (plus utilisées)



Expérimentations sur un cluster de GPUs

Cluster de GPUs : 6 Xeon Quad-Core E5530 et 12 Tesla C1060

Systèmes linéaires : 25 millions de valeurs inconnues :

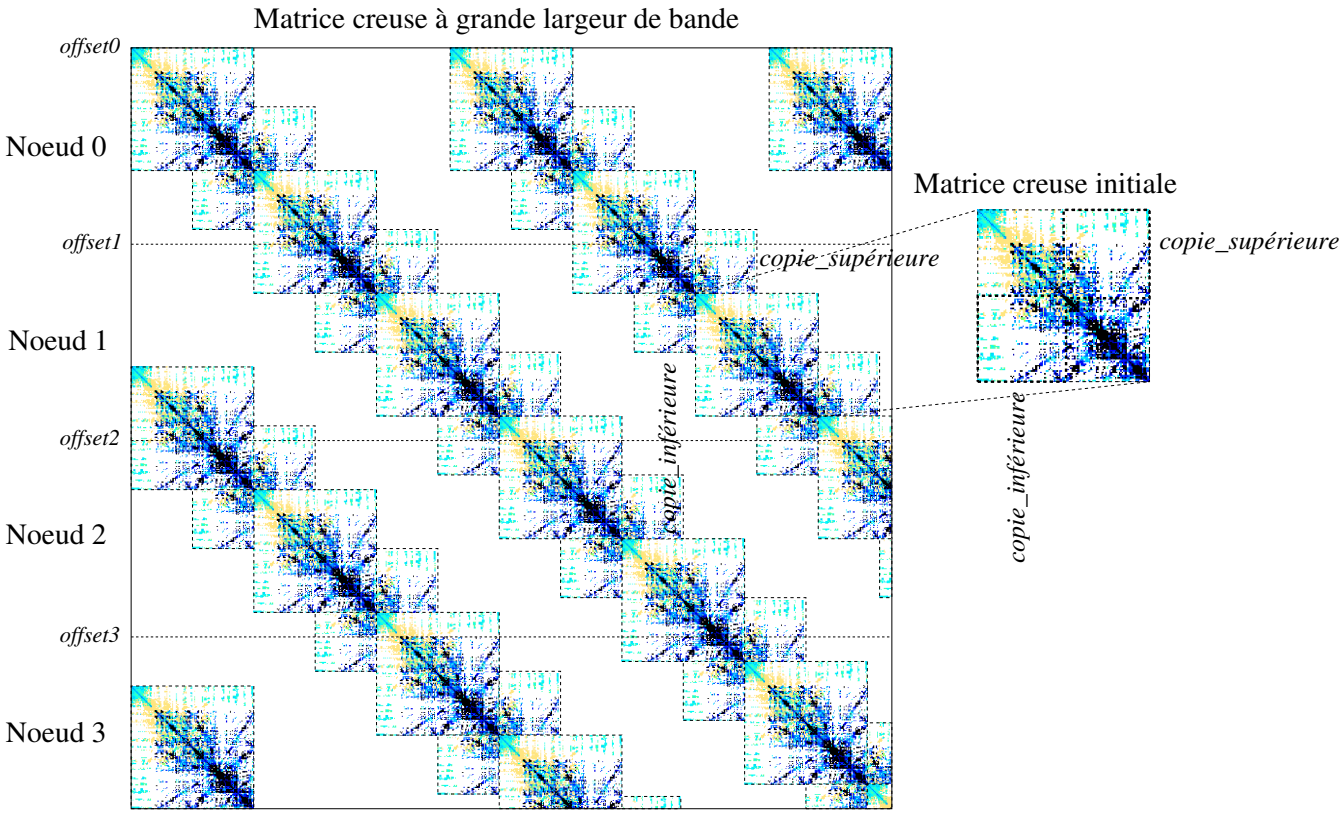
Type	Matrice creuse	# Valeurs non nulles	Largeur de bande
Symétrique	G3_circuit	125 262 292	1 891 887
	shallow_water2	100 235 292	62 806
	thermal2	175 300 284	2 421 285
Asymétrique	cage13	435 770 480	352 566
	language	76 912 824	398 626
	poli_large	53 322 580	15 576

Résultats : Gain $\tau = \frac{Temps_{cpu}}{Temps_{gpu}}$

Matrice	τ	T_{cpu}	T_{gpu}	# *iter*	*prec*	Δ
G3_circuit	4,54	3,14 s	0,48 s	22	1e-12	5e-15
shallow_water2	5,91	2,19 s	0,25 s	17	5e-21	8e-24
thermal2	4,11	3,21 s	0,46 s	21	9e-12	3e-16
cage13	5,75	5,56 s	0,66 s	26	3e-11	1e-14
language	5,16	12,55 s	1,76 s	89	2e-10	2e-10
poli_large	5,85	8,51 s	1,05 s	69	5e-11	6e-12

Matrices creuses à grandes largeurs de bande

- Matrices creuses à grandes largeurs de bande :
  - Un nombre important de dépendances de données
  - Un nœud GPU est lié à plusieurs nœuds voisins



Expérimentations sur un cluster de GPUs

Systèmes linéaires : 25 millions de valeurs inconnues :

Type	Matrice creuse	# Valeurs non nulles	Largeur de bande
Symétrique	G3_circuit	257 982 646	25 000 000
	shallow_water2	200 798 268	25 000 000
	thermal2	359 340 179	24 999 998
Asymétrique	cage13	879 063 379	24 999 998
	language	155 261 826	24 999 492
	poli_large	106 680 819	25 000 000

Résultats : Cluster de 24 cœurs CPU et 12 GPUs

Matrice	τ	T_{cpu}	T_{gpu}	# *iter*	*prec*	Δ
G3_circuit	1,36	61,36 s	45,18 s	22	1e-14	2e-16
shallow_water2	1,23	42,70 s	34,57 s	17	5e-23	3e-26
thermal2	1,31	56,61 s	43,21 s	21	8e-14	4e-18
cage13	1,30	69,01 s	53,17 s	26	3e-13	2e-16
language	1,24	234,76 s	188,82 s	89	2e-12	2e-12
poli_large	1,22	177,05 s	144,66 s	69	5e-13	6e-14

Réduction du volume total de communications

Produit parallèle matrice creuse-vecteur :

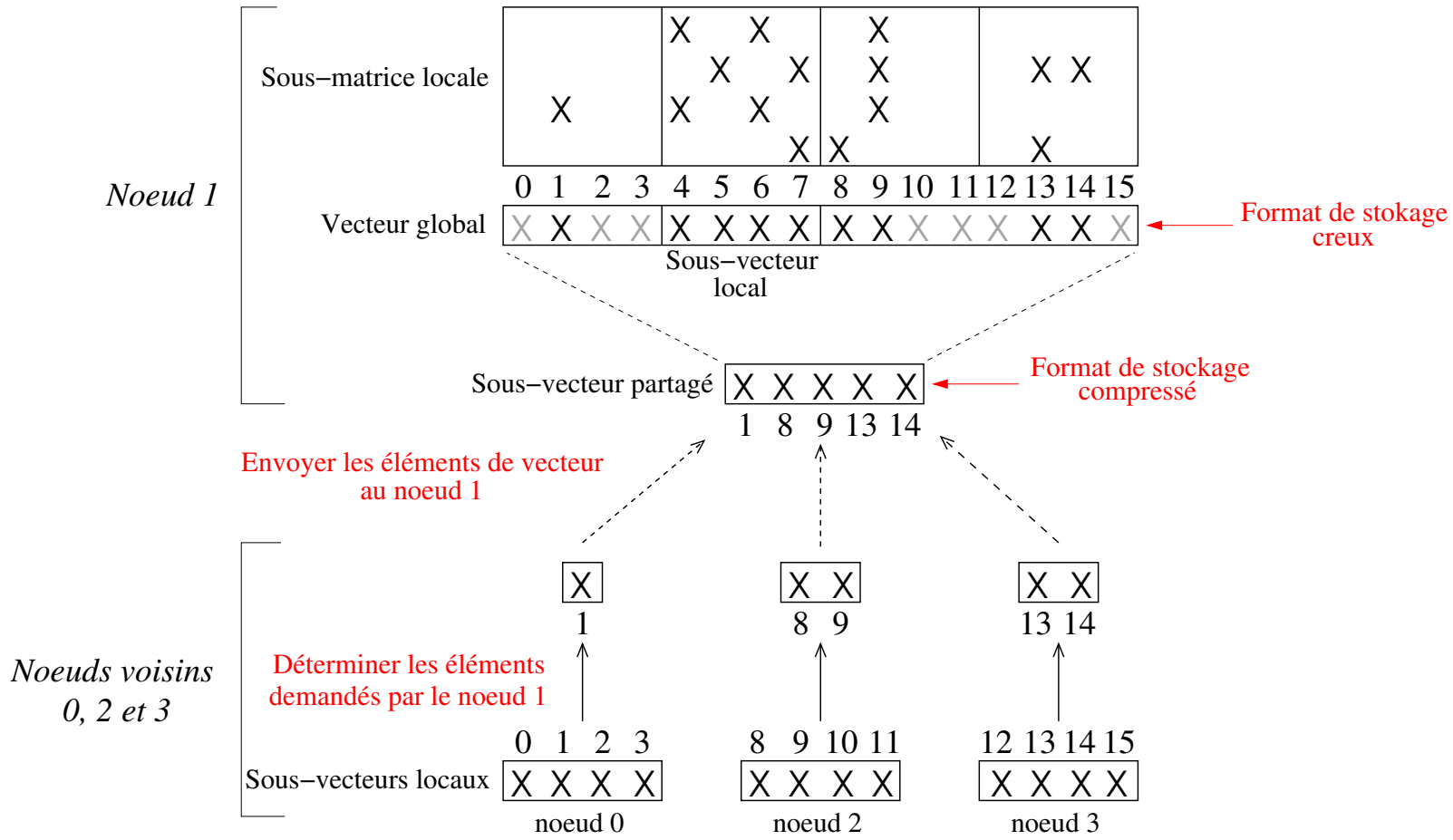
```
1: gpu_vers_cpu() ;   /*transfert de données*/
2: échanges_données() ;   /*par passage de messages MPI*/
3: cpu_vers_gpu() ;   /*transfert de données*/
4: multiplication_matrice_vecteur() ;
```

Partitionnement par hypergraphe :

- Adapté à de nombreuses matrices creuses
- Réduction du volume total des communications inter-GPUs
- Équilibrage de charge entre les GPUs

Minimisation des coûts de communication

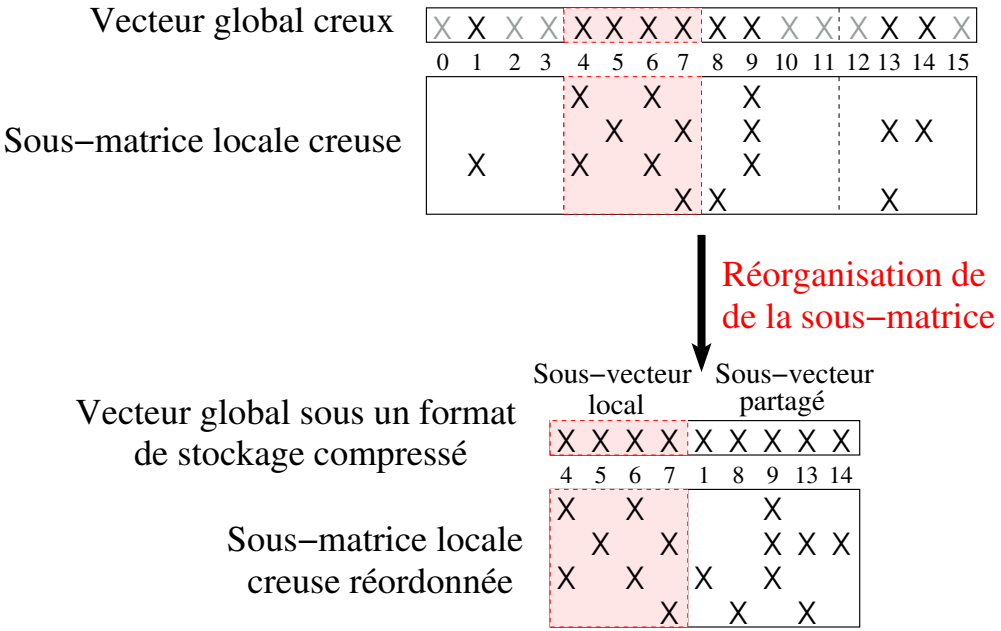
Échanger que les éléments de vecteur nécessaires à la multiplication :



Minimisation des coûts de communication

- Au niveau du nœud source :
  - Un kernel pour déterminer les éléments de vecteur à envoyer aux voisins
- Au niveau du nœud destination :
  - Réception de vecteur partagé sous un format compressé

→ Réorganiser les colonnes de la matrice



Expérimentations sur un cluster de GPUs

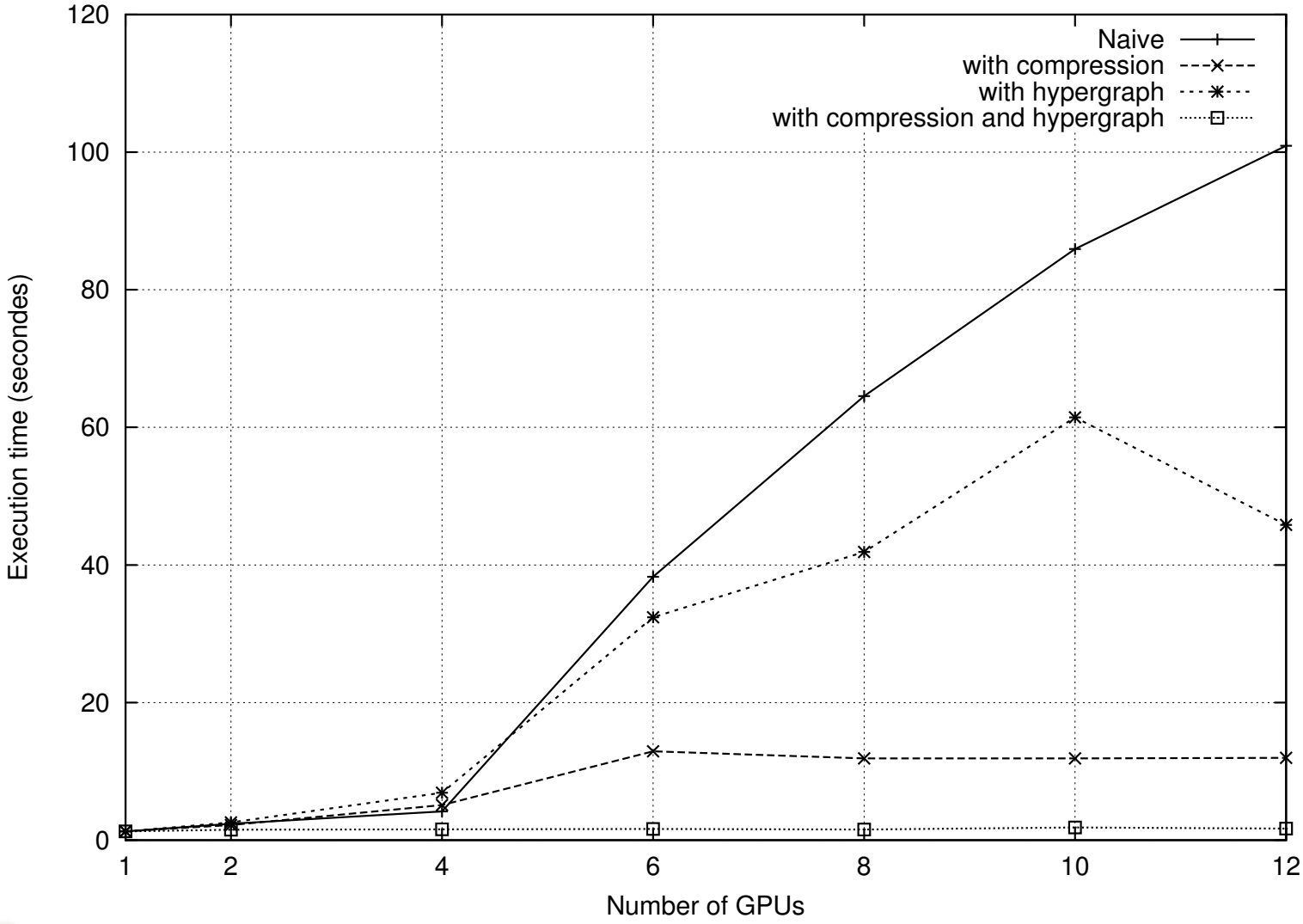
Résultats : Cluster de 24 cœurs CPU et 12 GPUs

Matrice	τ	T_{cpu}	T_{gpu}	# *iter*	*prec*	Δ
G3_circuit	4,60	4,47 s	0,97 s	22	1e-14	2e-16
shallow_water2	8,48	2,65 s	0,31 s	17	5e-23	4e-26
thermal2	6,29	4,19 s	0,67 s	21	8e-14	4e-18
cage13	5,10	8,08 s	2,58 s	26	3e-13	2e-16
language	6,62	16,71 s	2,52 s	89	2e-12	2e-12
poli_large	3,19	12,71 s	3,99 s	69	5e-13	2e-13

Ratios : entre temps d'exécution et temps de communication

Matrice	Parti. ligne par ligne		Format compressé		Parti. hypergraphe & format compressé	
	GPU	CPU	GPU	CPU	GPU	CPU
G3_circuit	0,009	0,047	0,072	2,796	0,558	2,546
shallow_water2	0,009	0,053	0,076	7,214	5,191	4,733
thermal2	0,013	0,060	0,105	3,072	1,497	3,882
cage13	0,021	0,092	0,145	4,961	1,512	3,131
language	0,010	0,054	0,092	5,172	1,875	3,767
poli_large	0,008	0,047	0,077	6,987	0,134	1,302

Passage à l'échelle sur un cluster de GPU



Liste de publications

1 Journal international :

L. Ziane Khodja, R. Couturier, A. Giersch, J.M. Bahi, Parallel Sparse Linear Solver with GMRES Method using Minimization Techniques of Communications for GPU Clusters, J. Supercomputing, 2014. Springer.

2 Conférences internationales :

J.M. Bahi, R. Couturier, L. Ziane Khodja, Parallel sparse linear solver GMRES for GPU clusters with compression of exchanged data, Euro-Par Workshops (1) 2011 : 471-480, Bordeaux, Springer.

J.M. Bahi, R. Couturier, L. Ziane Khodja, Parallel GMRES implementation for solving sparse linear systems on GPU clusters, SpringSim (HPC) 2011 : 12-19, Boston, ACM Press.

2 Chapitres de livres :

R. Couturier, L. Ziane Khodja, Solving sparse linear systems with CG and GMRES methods on a GPU and a GPU cluster, In Patterns for parallel programming on GPUs, Frédéric Magoulès (Ed.), Saxe-Coburg Publications.

L. Ziane Khodja, R. Couturier, J.M. Bahi, Solving sparse linear systems with GMRES and CG methods on GPU clusters, In Designing Scientific Applications on GPUs, Raphaël Couturier (Ed.), 498 pages, November 2013. Chapman and Hall/CRC.

Résolution de problèmes de l'obstacle 3D

- Le problème de l'obstacle est un problème non linéaire :

$$\begin{cases} \frac{\partial u}{\partial t} + b^t \nabla u - \eta \Delta u + cu - f \geq 0, \ u \geq \phi, \ \textit{sur tout } [0,T] \times \Omega, \ \eta > 0, \\ (\frac{\partial u}{\partial t} + b^t \nabla u - \eta \Delta u + cu - f)(u - \phi) = 0, \ \textit{sur tout } [0,T] \times \Omega, \\ u(0,x,y,z) = u_0(x,y,z) \text{ dans un domaine tridimensionnel} \\ C.L. \textit{ sur } u(t,x,y,z) \textit{ défini sur } \partial\Omega, \end{cases}$$

- Résoudre à chaque pas de temps k un système non linéaire :

$$\begin{cases} \text{Trouver } U \in \mathbb{R}^M \text{ tels que} \\ (A + \frac{1}{k} I)U - G \geq 0, U \geq \bar{\Phi}, \\ ((A + \frac{1}{k} I)U - G)^T (U - \bar{\Phi}) = 0. \end{cases}$$

- Exemples d'applications :
  - Mécanique des fluides dans les milieux poreux
  - Bio-mathématique (cicatrisation des plaies)
  - Mathématiques financières (options Américaines)

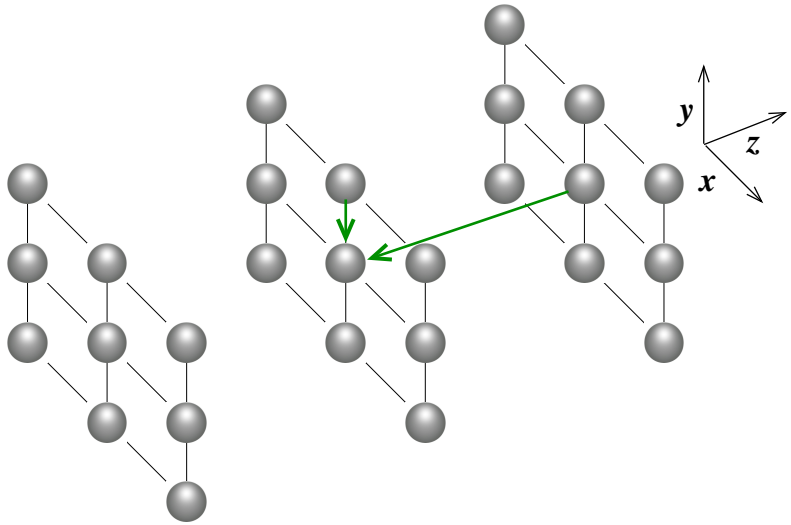
Résolution du problème de l'obstacle 3D

- Méthodes de résolution choisies :
  - Relaxation par blocs :
    - Basée sur la méthode Gauss-Seidel par blocs
    - Convergence rapide et parallélisation difficile
  - Richardson :
    - Basée sur la méthode Jacobi par points
    - Convergence lente et parallélisation facile
- Principaux kernels GPUs des deux méthodes :
  - Multiplication 3D matrice creuse-vecteur
  - Mise à jour du vecteur solution

Méthode de relaxation par blocs

Utiliser les nouvelles valeurs de la solution dès que possible :

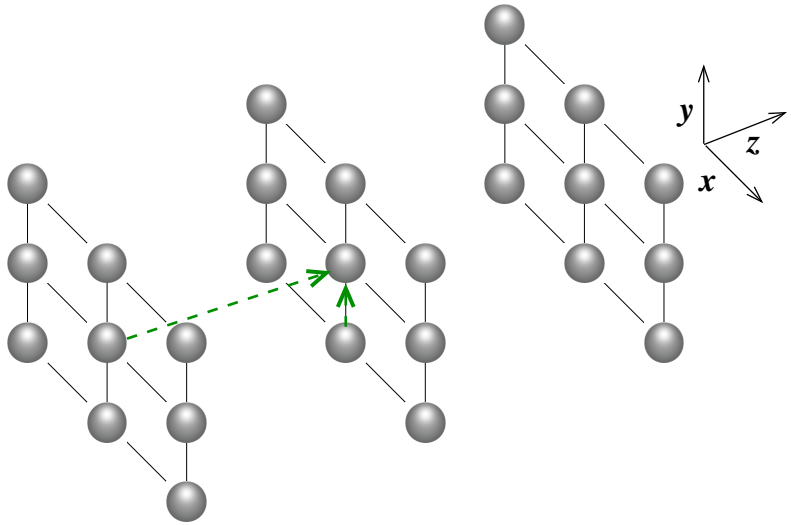
1. Anciennes valeurs des points ayant un rang supérieur
2. Nouvelles valeurs des points ayant un rang inférieur
3. Mise à jour d'un bloc de points par la remontée de Gauss



Méthode de relaxation par blocs

Utiliser les nouvelles valeurs de la solution dès que possible :

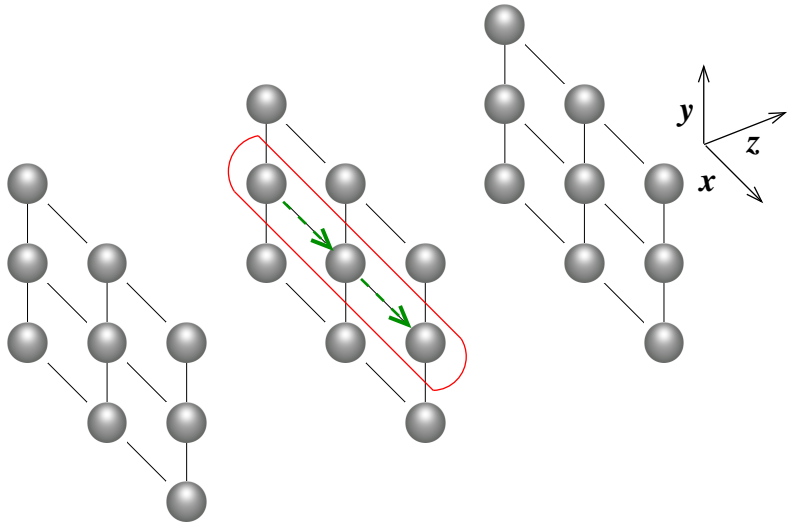
1. Anciennes valeurs des points ayant un rang supérieur
2. Nouvelles valeurs des points ayant un rang inférieur
3. Mise à jour d'un bloc de points par la remontée de Gauss



Méthode de relaxation par blocs

Utiliser les nouvelles valeurs de la solution dès que possible :

1. Anciennes valeurs des points ayant un rang supérieur
2. Nouvelles valeurs des points ayant un rang inférieur
3. Mise à jour d'un bloc de points par la remontée de Gauss



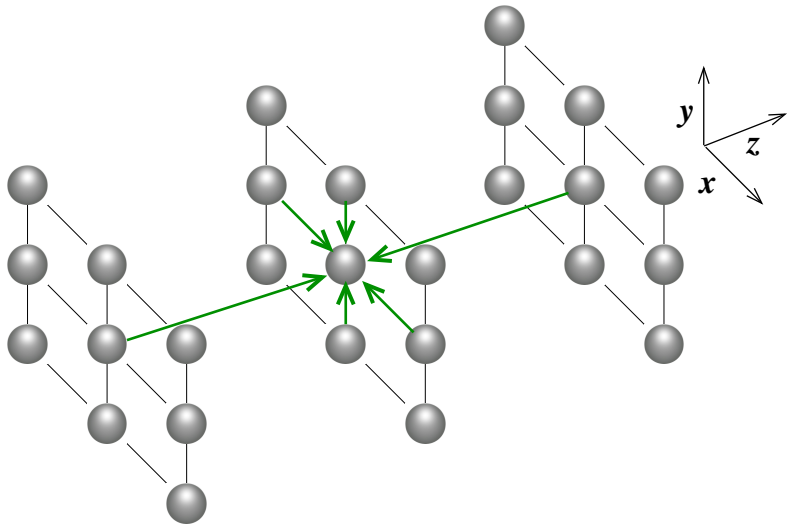
1 seul thread par bloc et mise à jour des points bloc par bloc
$\rightarrow$ Mise en œuvre mixte CPU/GPU

Méthode Richardson

Utiliser uniquement les anciennes valeurs de la solution :

- Multiplication avec les anciennes valeurs des points
- Mise à jour de tous les points à la fois

→ Un thread par point (autant de threads que de points)

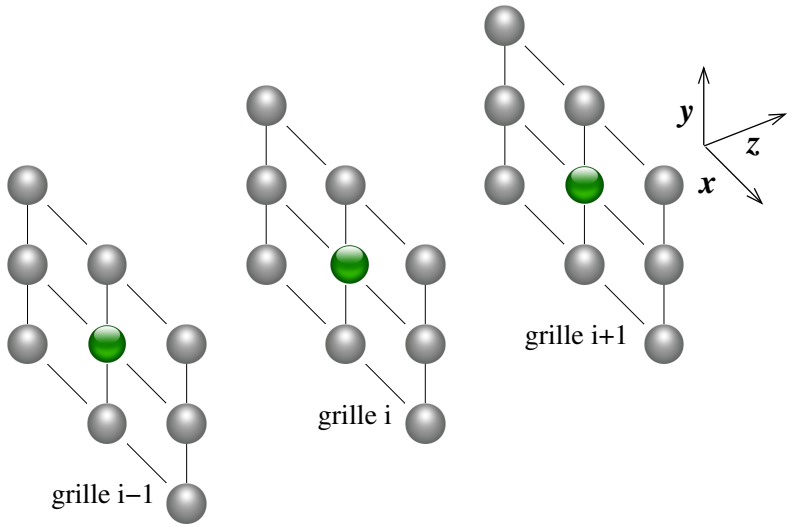


Méthode Richardson

Utiliser uniquement les anciennes valeurs de la solution :

- Multiplication avec les anciennes valeurs des points
- Mise à jour de tous les points à la fois

→ Un thread par point (autant de threads que de points)



Problème de très grande taille : pas assez de threads

→ Un thread est en charge d'un point par grille

Partitionnement de données

Partitionnement 2D du problème de l'obstacle :

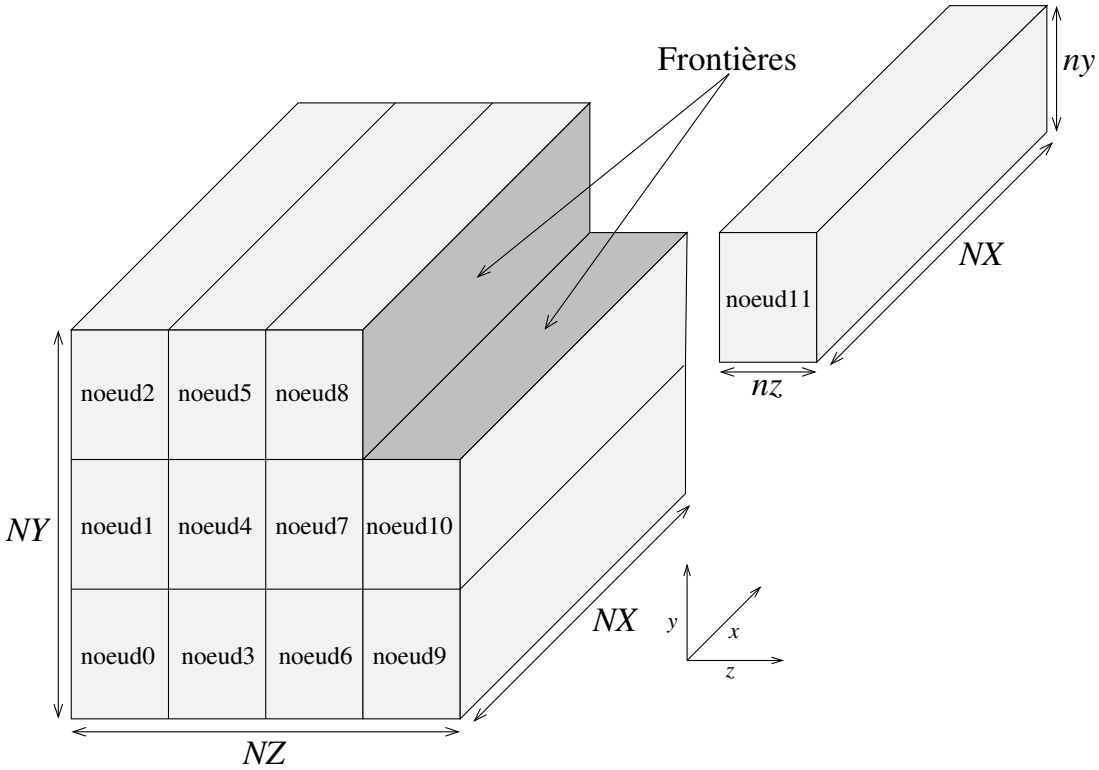


FIGURE: Partitionnement 2D entre 3 × 4 nœuds GPUs

Échange de données entre les nœuds GPUs

A chaque itération :
* Déterminer les points associés aux frontières
* Copier ces points dans la mémoire CPU
* Échanger les points partagés avec les nœuds voisins
* Copier les points reçus des voisins dans la mémoire GPU
* Calculer les nouvelles valeurs du vecteur solution

Échange de données entre les nœuds GPUs

A chaque itération :

* Déterminer les points associés aux frontières (Kernel GPU)
* Copier ces points dans la mémoire CPU
* Échanger les points partagés avec les nœuds voisins
* Copier les points reçus des voisins dans la mémoire GPU
* Calculer les nouvelles valeurs du vecteur solution (Kernel GPU)

Échange de données entre les nœuds GPUs

A chaque itération :
* Déterminer les points associés aux frontières (Kernel GPU)
* Copier ces points dans la mémoire CPU
* Échanger les points partagés avec les nœuds voisins
* Copier les points reçus des voisins dans la mémoire GPU
* Calculer les nouvelles valeurs du vecteur solution (Kernel GPU)

Deux versions :
- Version synchrone : transferts CPU/GPU synchrones et communications MPI bloquantes

- Version asynchrone : transferts CPU/GPU asynchrones et communications MPI non-bloquantes

Échange de données entre les nœuds GPUs

A chaque itération :
* Déterminer les points associés aux frontières (Kernel GPU)
* Copier ces points dans la mémoire CPU
* Calculer les valeurs des points locaux (Kernel GPU)
* Échanger les points partagés avec les nœuds voisins
* Copier les points reçus des voisins dans la mémoire GPU
* Calculer les valeurs des points sur les frontières (Kernel GPU)

$\rightarrow$ Recouvrement entre les kernels et les communications MPI

Stockage de données dans les GPUs

- Vecteur solution : Réutilisation de données
  - Mémoire partagée : pas rentable !
    - Utilisation que de 2 points sur chaque dimension
    - Branchements conditionnels : chargement des points de frontières
  - Mémoires caches : meilleure solution
    - Texture pour GPUs 1.x (ex : Tesla)
    - Caches L1/L2 pour GPUs 2.x et plus (ex : Fermi)
- Matrice creuse : 7 coefficients constants
  - Mémoire cache constante : pas rentable !
    - Chaque thread est en charge d'un point sur chaque grille
    - Accès multiple pour la lecture des coefficients constants
  - Registres : meilleure solution
    - Chargement dans les registres de chaque thread

Performances sur 24 cœurs CPUs

- Méthode de relaxation par blocs :

Taille	Synchrone		Asynchrone	
	$Temps_{cpu}$	# relaxations	$Temps_{cpu}$	# relaxations
256^3	137,50 s	37 080	131,71 s	38 348
512^3	4 814,71 s	132 600	4 371,59 s	141 067
768^3	41 236,56 s	279 768	37 191,07 s	273 515

- Méthode Richardson :

Taille	Synchrone		Asynchrone	
	$Temps_{cpu}$	# relaxations	$Temps_{cpu}$	# relaxations
256^3	575,22 s	198 288	539,25 s	198 613
512^3	19 250,25 s	750 912	18 237,14 s	769 611
768^3	206 159,44 s	1 635 264	183 582,60 s	1 577 004

→ Relaxation par blocs 5 fois plus rapide que Richardson

→ Convergence lente pour la méthode Richardson

Performances sur 12 GPUs

- Méthode de relaxation par blocs :

Taille	Synchrone		Asynchrone	
	$Temps_{gpu}$	# relaxations	$Temps_{gpu}$	# relaxations
256^3	66,22 s	19 548	63,10 s	20 226
512^3	1 822,57 s	69 960	1 763,84 s	71 203
768^3	12 858,47 s	147 672	12 368,78 s	149 734

- Méthode Richardson :

Taille	Synchrone		Asynchrone	
	$Temps_{cpu}$	# relaxations	$Temps_{cpu}$	# relaxations.
256^3	29,67 s	100 692	18,00 s	94 215
512^3	521,83 s	381 300	425,15 s	347 279
768^3	4 112,68 s	831 144	3 313,87 s	750 232

→ Richardson 4 fois plus rapide que relaxation par blocs

→ Parallélisation difficile de la méthode de relaxation par blocs

Comparaison entre les versions 24 cœurs CPU et 12 GPU

τ_{max} : gain de Richardson sur 12 GPUs vs. relaxation par blocs sur 24 CPUs

- Version synchrone :

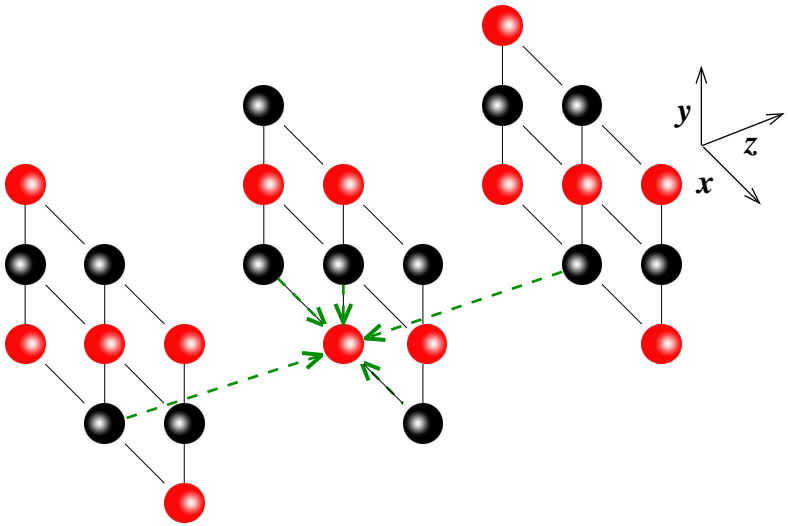
Taille	Relaxation par blocs	Richardson	τ_{max}
256^3	2,08	19,39	4,63
512^3	2,64	36,89	9,23
768^3	3,21	50,13	10,03

- Version asynchrone :

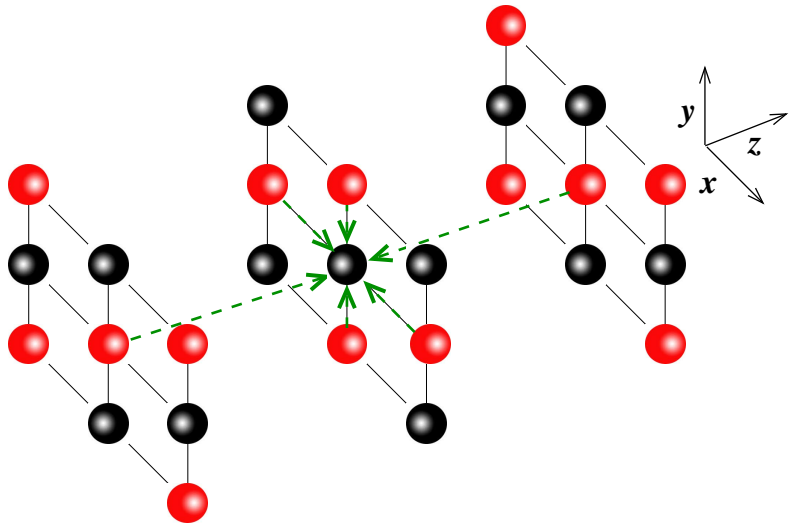
Taille	Relaxation par blocs	Richardson	τ_{max}
256^3	2,09	29,96	7,32
512^3	2,48	42,89	10,28
768^3	3,01	55,40	11,22

Amélioration de la convergence

- Utilisation de la technique de numérotation rouge-noir :
  (a) Calcul des éléments rouges en fonction des éléments noirs
  (b) Calcul des éléments noirs en fonction des éléments rouges



(a) Calcul des éléments rouges

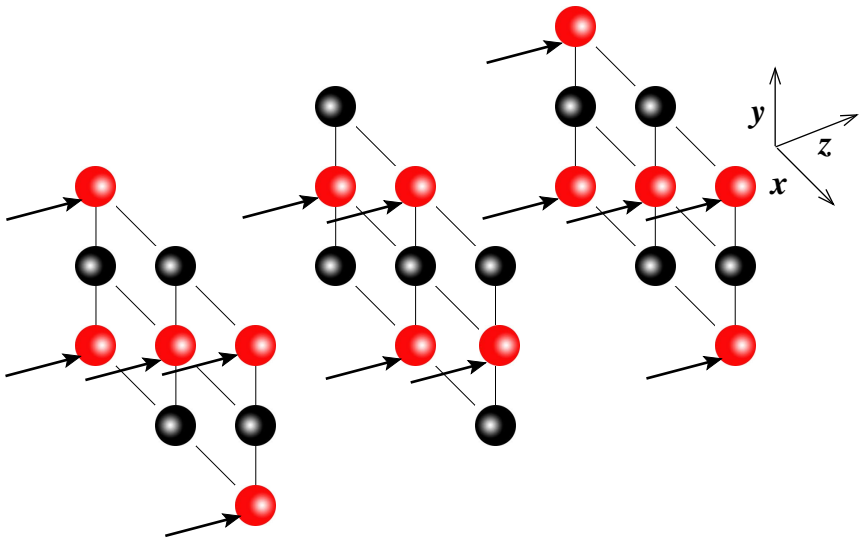


(b) Calcul des éléments noirs

Amélioration de la convergence

- Utilisation de la technique de numérotation rouge-noir :
  (a) Calcul des éléments rouges en fonction des éléments noirs
  (b) Calcul des éléments noirs en fonction des éléments rouges

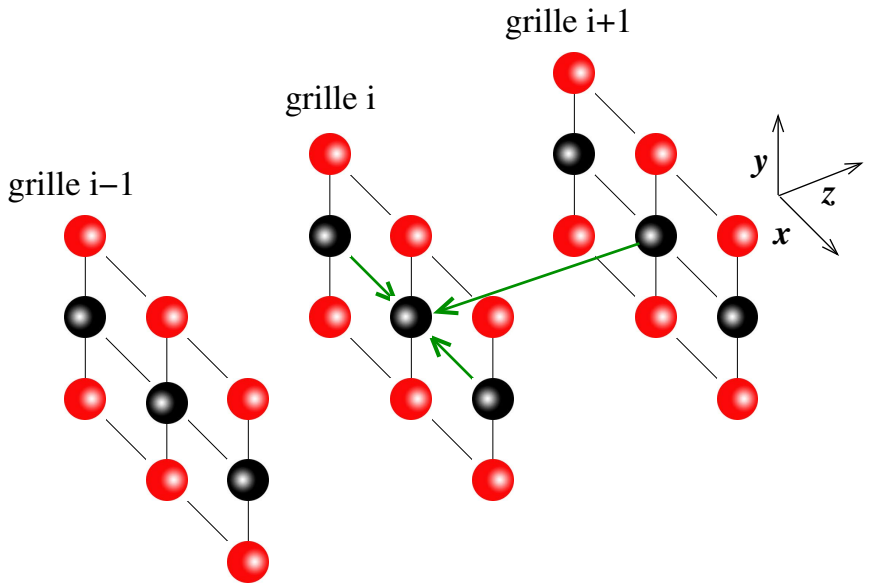
→ Un thread sur deux calcule un élément rouge (ou noir)



→ Accès non contigus à la mémoire !

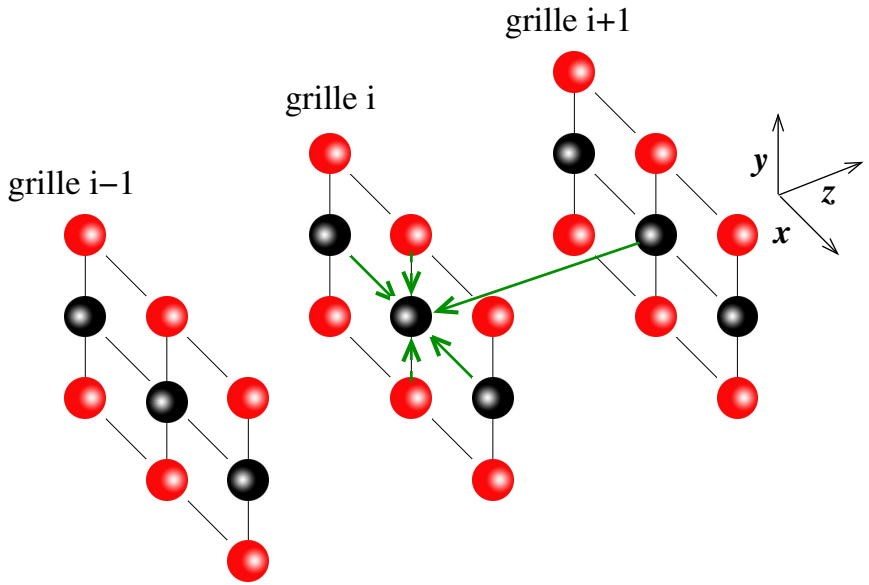
Amélioration de la convergence

- Appliquer la numérotation rouge-noir selon l'axe Y
- Calcul d'un point sur la grille i par un thread GPU :
  - Utiliser les anciennes valeurs des points sur l'axe X
  - Utiliser l'ancienne valeur du point sur la grille $(i+1)$
  -
  -



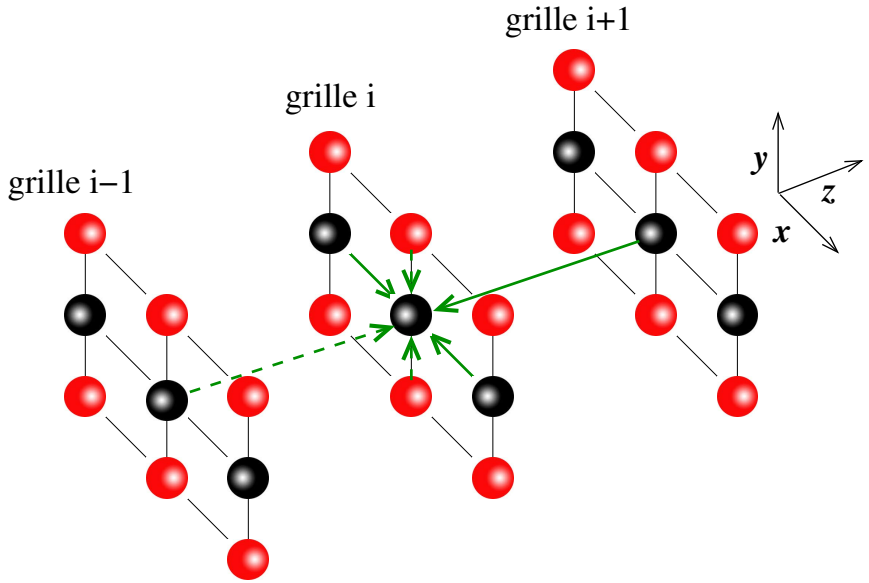
Amélioration de la convergence

- Appliquer la numérotation rouge-noir selon l'axe Y
- Calcul d'un point sur la grille i par un thread GPU :
  - Utiliser les anciennes valeurs des points sur l'axe X
  - Utiliser l'ancienne valeur du point sur la grille $(i+1)$
  - Utiliser les nouvelles valeurs des points sur l'axe Y
  -



Amélioration de la convergence

- Appliquer la numérotation rouge-noir selon l'axe Y
- Calcul d'un point sur la grille i par un thread GPU :
  - Utiliser les anciennes valeurs des points sur l'axe X
  - Utiliser l'ancienne valeur du point sur la grille $(i + 1)$
  - Utiliser les nouvelles valeurs des points sur l'axe Y
  - Utiliser la nouvelle valeur du point sur la grille $(i - 1)$



Résultats expérimentaux sur 12 GPUs

- Performances sur 12 GPUs :

Taille	Synchrone		Asynchrone	
	$Temps_{gpu}$	# relaxations	$Temps_{gpu}$	# relaxations
256^3	18,37 s	71 988	12,58 s	67 638
512^3	349,23 s	271 188	289,41 s	246 036
768^3	2 773,65 s	590 652	2 222,22 s	532 806

- Richardson utilisant la numérotation rouge-noir vs. Richardson simple :

Taille	Synchrone		Asynchrone	
	δ	τ %	δ	τ %
256^3	1,61	38,08%	1,43	30,11%
512^3	1,49	33,07%	1,47	31,93%
768^3	1,48	32,56%	1,49	32,94%

δ : gain en utilisant la numérotation rouge-noir

τ % : gain en % en utilisant la numérotation rouge-noir

Passage à l'échelle sur un cluster de GPUs

La version asynchrone supporte mieux le passage à l'échelle :

- Les GPUs accélèrent les calculs
- Les temps de communications sont importants
- Le rapport calculs/communications est réduit

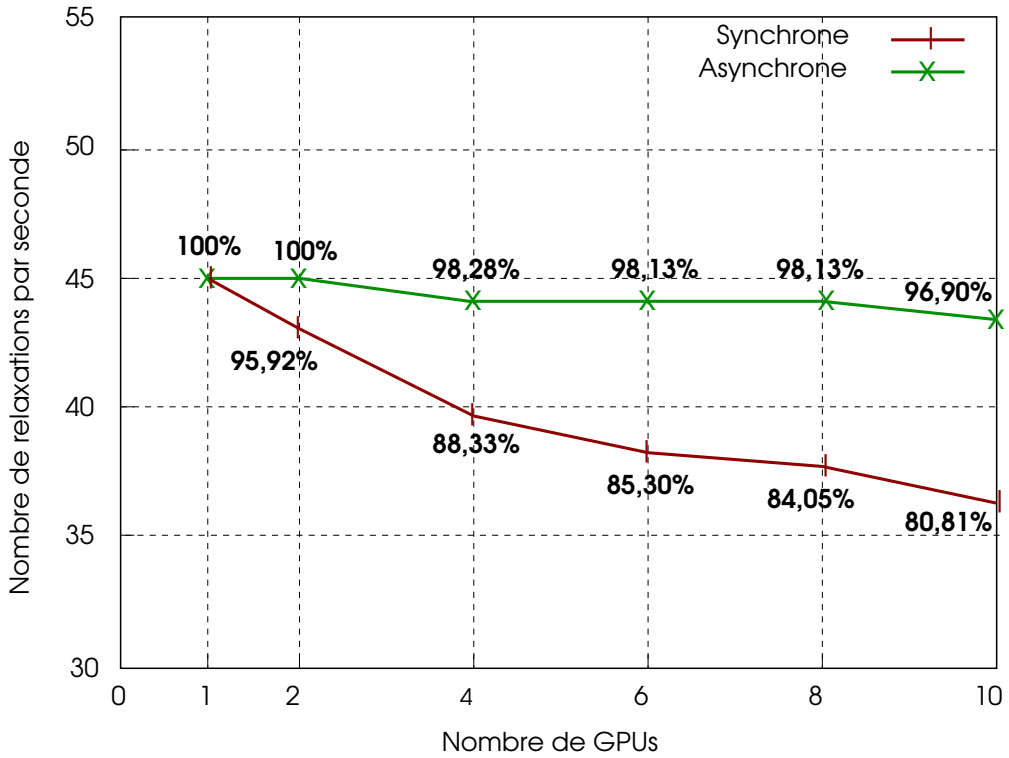


FIGURE: Passage à l'échelle, un sous-problème de 256^3 par GPU

Liste de publications

1 Journal international :

L. Ziane Khodja, M. Chau, R. Couturier, J.M. Bahi, P. Spitéri, Parallel Solution of American Option Derivatives on GPU Clusters, Computers & Mathematics with Applications 65(11) : 1830-1848 (2013), Elsevier

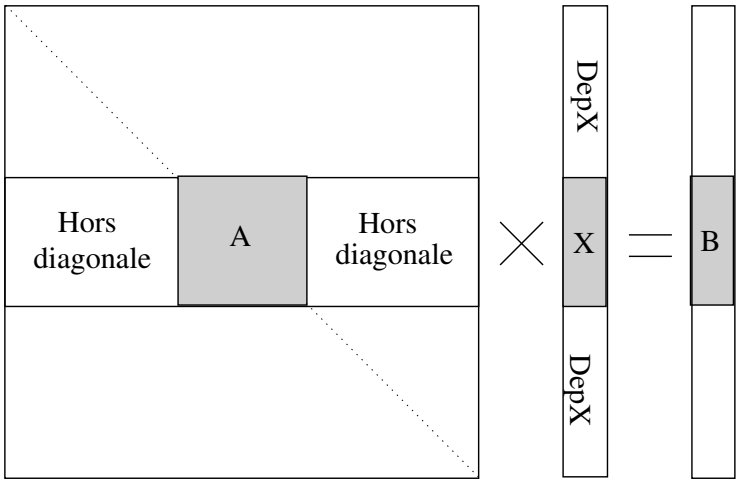
1 Chapitres de livres :

L. Ziane Khodja, M. Chau, R. Couturier, J.M. Bahi, P. Spitéri, Solving sparse nonlinear systems of obstacle problems on GPU Clusters, In Designing Scientific Applications on GPUs, Raphaël Couturier (Ed.), 498 pages, November 2013. Chapman and Hall/CRC.

Méthodes de multisplitting

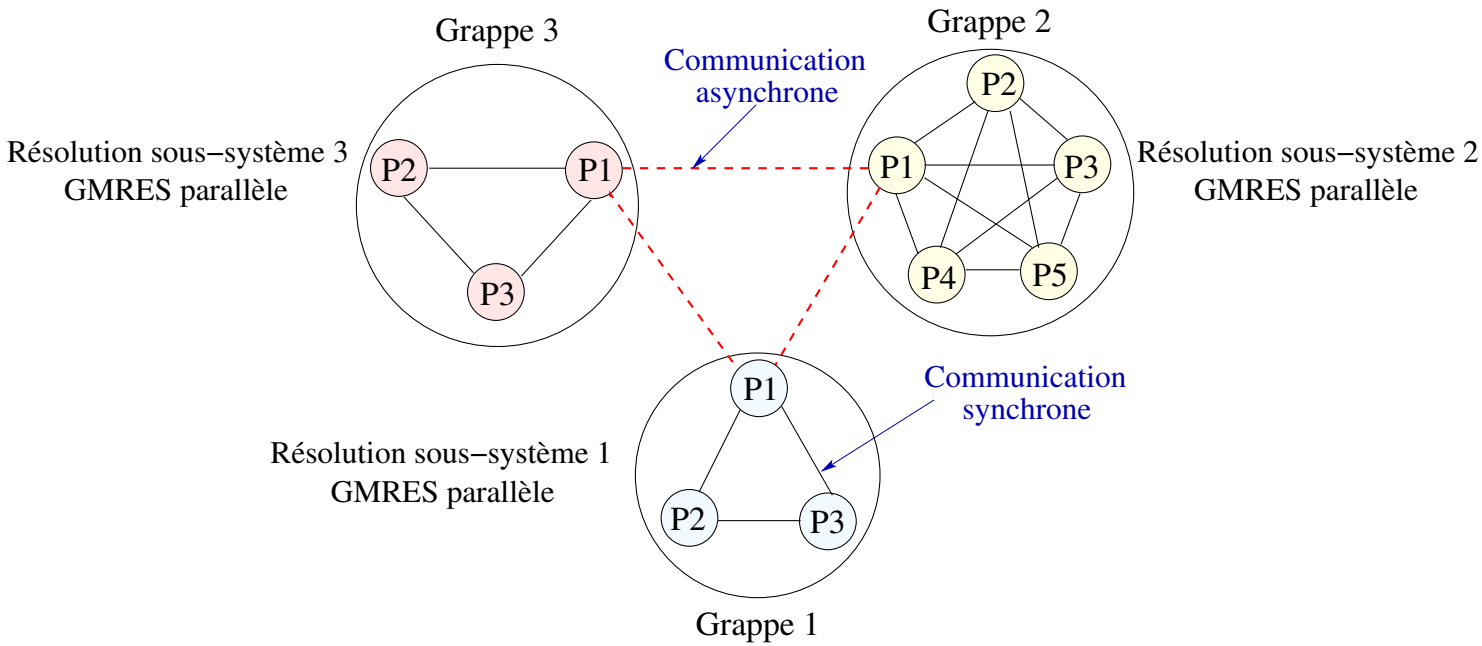
- Résoudre des systèmes linéaires creux sur des plusieurs clusters : communications inter-GPUs très coûteuses
- La méthode de multisplitting permet de combiner entre :
  - Performance des méthodes synchrones dans le contexte local
  - Souplesse des méthodes asynchrones entre les clusters
- Principe de la méthode multisplitting :
  - Décomposition du système en α sous-systèmes linéaires
  - chaque sous-système linéaire est résolu par un processeur

$$\begin{cases} A_{ii}X_i = Y_i, \text{ tel que} \\ Y_i = B_i - \sum_{j=1, j\neq i}^{\alpha} A_{ij}X_j, \end{cases}$$



Algorithme à deux niveaux avec GMRES

- Résolution de chaque sous-système par un cluster GPU
- Méthode à deux niveaux :
  - Itérations internes synchrones : résolution du sous-système local avec GMRES parallèle
  - Itérations externes asynchrones : résolution du système linéaire global



Résultats expérimentaux sur 10 GPUs

δ : gain obtenus avec la méthode à deux niveaux avec GMRES vs. GMRES parallèle

Matrice	GMRES		# grappes × # GPUs	Méthode à deux niveaux avec GMRES					
				Synchrone			Asynchrone		
	Temps	iter.		Temps	iter.	δ	Temps	iter.	δ
cage9	3,17 s	27	2×5	3,08 s	19	1,03	2,03 s	23	**1,56**
			5×2	3,87 s	24	0,82	2,56 s	43	1,24
			10×1	4,13 s	26	0,77	3,78 s	65	0,84
ecology2	2,58 s	21	2×5	2,13 s	13	1,21	1,65 s	24	**1,56**
			5×2	2,37 s	15	1,10	1,74 s	23	1,48
			10×1	2,36 s	15	1,09	2,39 s	43	1,08

$\rightarrow$ Méthode à deux niveaux asynchrone plus performante sur 2 grappes de 5 GPUs

Perspectives

- Clusters à grande échelle ou distants :
  - Valider les performances des algorithmes conçus
  - Concevoir de nouveaux algorithmes adaptés
  - Tester le passage à l'échelle
- Étudier d'autres :
  - systèmes linéaires ou non linéaires creux
  - méthodes de partitionnement
  - méthodes de préconditionnement
  - méthodes de multisplitting sans/avec recouvrement
- Nouvelles architectures matérielles et logicielles :
  - Nouvelles générations de GPUs : Kepler, Maxwell
  - Xeon Phi
  - Clusters hétérogènes

3 Recherche

Stage post-doctoral

Stage post-doctoral

- Projet ANR SONGS :
  - INRIA Bordeaux Sud-Ouest
  - Équipe Runtime
- Objectifs :
  - Modélisation des nouvelles architectures multicœurs
  - Modélisation des nœuds multicœurs interconnectés par des réseaux haute performance
  - Intégration des modèles dans le package SimGrid

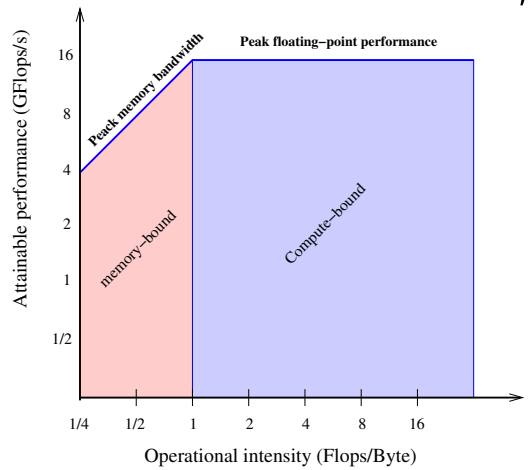
Stage Post-doctoral

Travaux en cours :

- Étudier les performances des applications HPC : bigdft, specfem3d
- Déterminer les parties les plus coûteuses : outils de profiling
- Déterminer les fonctions memory-bound

→ modélisation de la hiérarchie mémoire du multicœur

- Beaucoup de paramètres à prendre en compte ⇒
  - Prédiction des bornes supérieures et bottlenecks
  - Extension du modèle roofline (différents niveaux de cache, cache misses, protocole de cohérence, ...)



4 Recherche

Interactions avec l'équipe PEQUAN

Interactions avec l'équipe PEQUAN

Axes de recherches :

- Arithmétique des ordinateurs et validation numérique
- Calcul numérique distribué
- Optimisation de performance des applications sur différentes architectures
- Systèmes de numération liés à la cryptographie

Mes atouts informatiques :

- Calcul scientifique haute performance
- Algorithmes numériques asynchrones
- Architectures parallèles (multicœurs, GPU, ...)

Interactions avec l'équipe PEQUAN

Mes atouts mathématiques :

Algèbre linéaire :

- Systèmes linéaires/non linéaires creux, méthodes itératives
- Autres pistes : autres problèmes linéaires (par ex. problèmes des moindres carrés), méthodes directes, méthodes multigrilles

Autres connaissances :

- Probabilités, modélisation stochastique (chaînes de Markov)
- 1 journal international :

M. Yazid, L. Bouallouche-Medjkoune, D. Aissani, L. Ziane Khodja, Analytical analysis of applying packet fragmentation mechanism on IEEE 802.11b DCF network in non ideal channel with infinite load conditions, J. Wireless Networks, 2014. Springer.

Piste de recherche : Développement des algorithmes parallèles pour l'exascale

MERCI

Lilia Ziane Khodja

Post-Doc INRIA Bordeaux Sud-Ouest

http ://runtime.bordeaux.inria.fr/lziane/