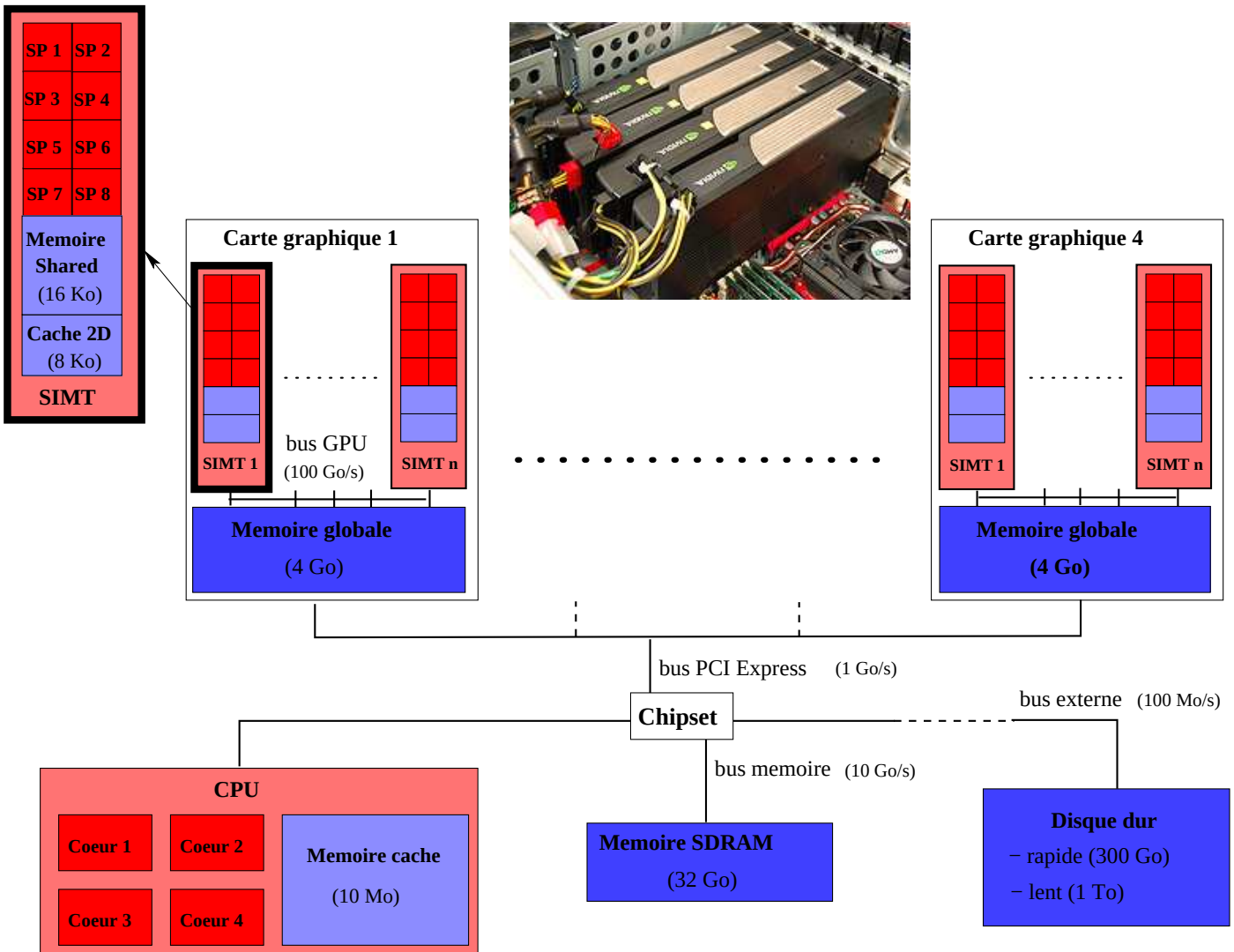
Parallélisation des calculs sur serveur multi-GPUs pour la résolution de problèmes inverses

Nicolas GAC, MCF Université Paris Sud
Groupe Problèmes Inverses (GPI)
L2S (Supélec/CNRS/Univ Paris Sud)

Cours ESIEE, Paris, 14 septembre 2012

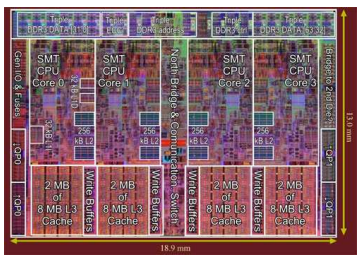
Serveur de calcul multi GPU



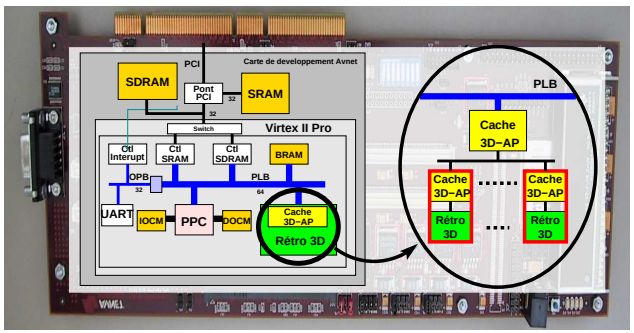
Calcul haute performance

High Performance Computing (HPC)

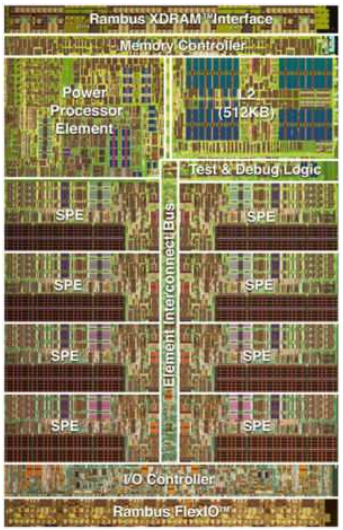
- Parallélisation sur machines multi-processeurs
  ➲ Efficace sur machine à mémoire distribuée
- Noeuds de calculs performants
  ➲ processeurs multi-core, many-core ou FPGA/ASIC



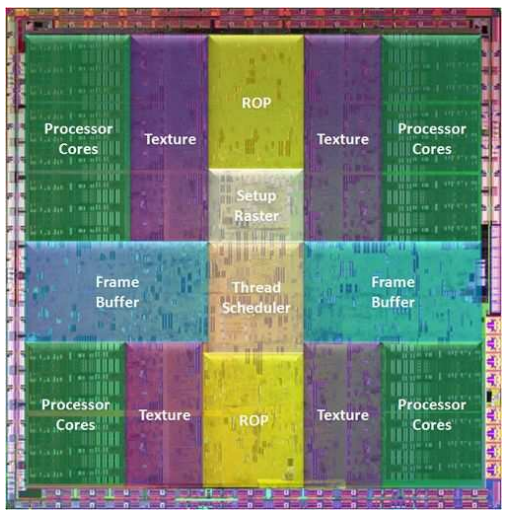
Intel Nehalem (4 coeurs)



SoPC (prototypage)

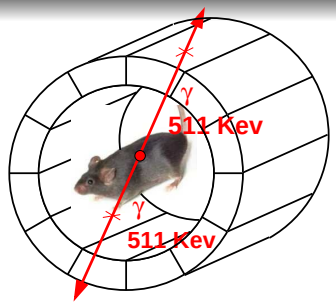


IBM Cell (8+1 coeurs)



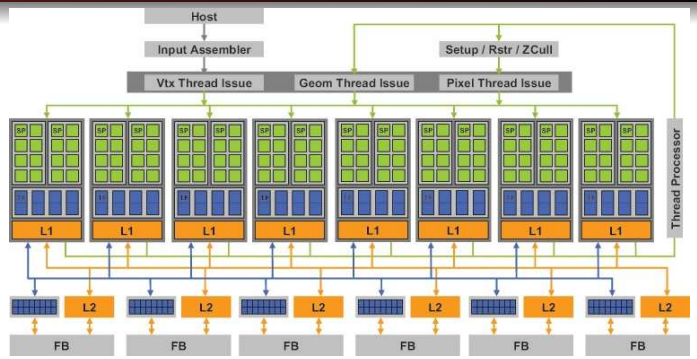
Nvidia GTX 200 (240 coeurs)

Thèse soutenue en 2008 : "Adéquation Algorithme Architecture pour la reconstruction 3D en imagerie médicale TEP" (Gipsa-lab, Grenoble-INP sous la direction de M. Desvignes et S. Mancini)

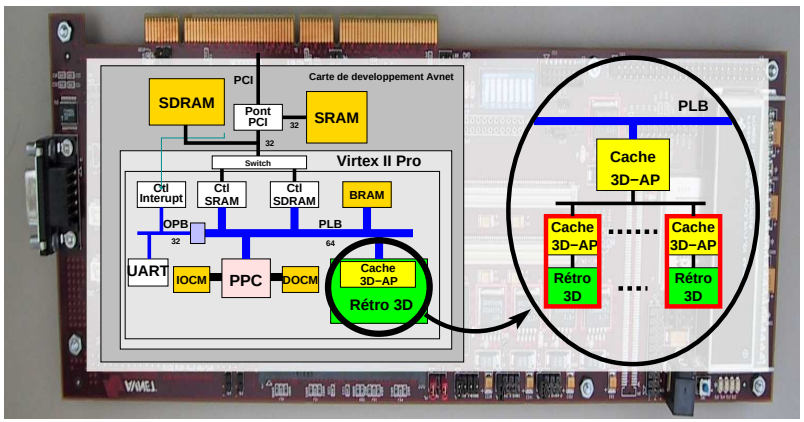


① Injection du traceur ② Acquisition ③ Enregistrement des données (Sinogrammes) ④ Reconstruction ⑤ Visualisation

Tomographie TEP



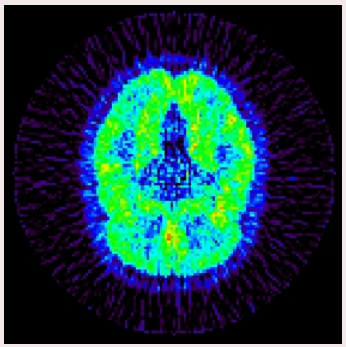
GPU (carte graphique)



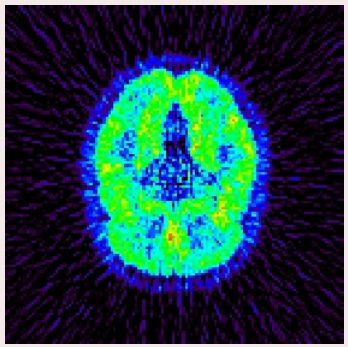
SoPC (prototypage)

Exploration architecturale

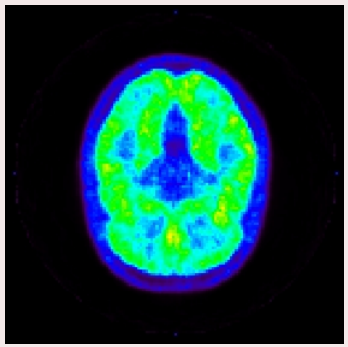
Conclusion de la thèse



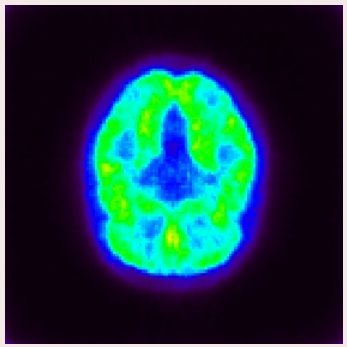
CPU
(non itératif)



FPGA
(non itératif)



CPU
(itératif)



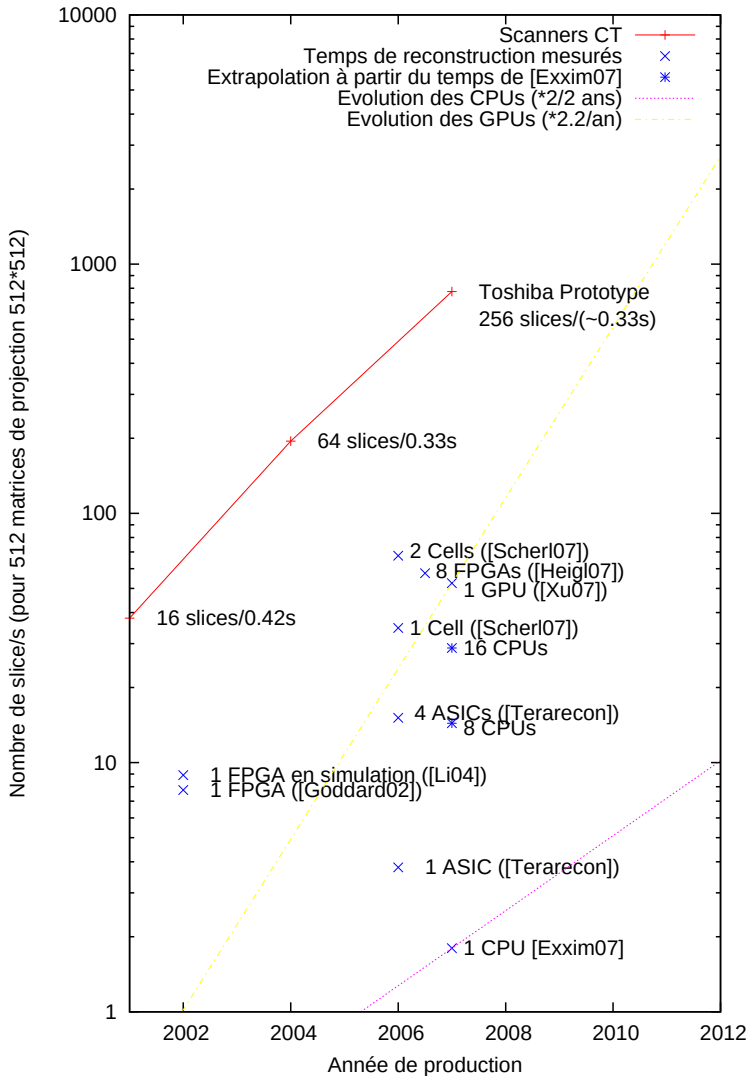
FPGA
(itératif)

Comparaison CPU/GPU/FPGA

	CPU	**GPU**	**FPGA**
Temps	3^{eme} (*4 P4)	1^{er} (*50 P4)	2^{eme} (*5 P4)
Efficacité	2^{eme} (7 C/op)	1^{eme} (14 C/Op)	1^{er} (2 C/Op)

- **GPU est l'accélérateur matériel le plus performant**
- **FPGA est le plus efficace en cycles/op (grace au cache 3D)**

Vitesse d'acquisition // Vitesse de reconstruction



Le GPU très vite adopté peu après son apparition en 2006

Publications dans la conférence Fully3D, spécialisée dans la reconstruction tomographique

- 2007 : 1er Workshop HPIR (High Performance Image Reconstruction)
- 2011 : Le mot clef Multi GPU apparait

	2003	2005	2007	2009	2011	2013
Cluster (MPI, Open MP)	2	3	5	6	3	?
GPU (NVIDIA)			10	14	17	?
GPU (ATI)				1	1	?
FPGA			4		1	?
DSP			2	1		?
Cell (IBM)			3			
Larabee (Intel)				2		
MIC (intel)						?

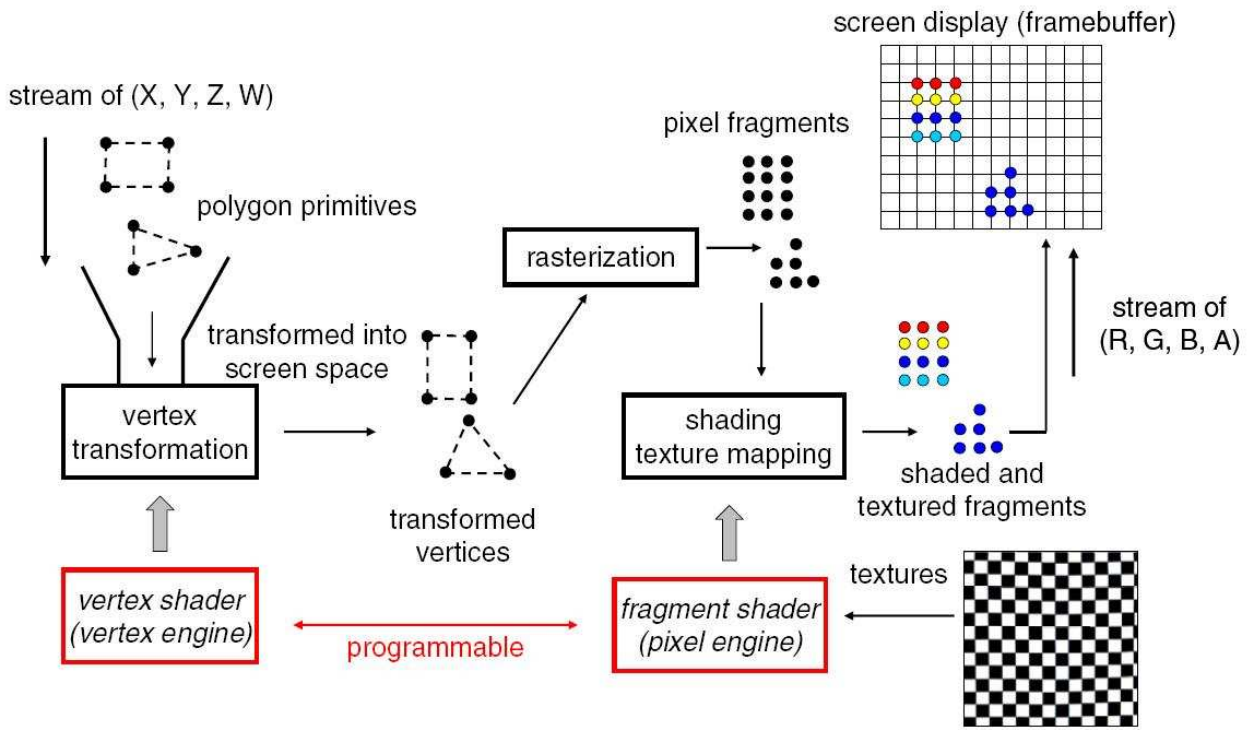
GPU : Graphic Processing Unit

Evolution vers une architecture *many core*

- A l'origine, architecture dédiée pour le rendu de volume
  ➲ Pipeline graphique (prog. en OpenGL/Cg)
- Depuis 2006, architecture adaptée à la parallélisation de divers calculs scientifiques
  ➲ CUDA : Common Unified Device Architecure (prog. en C)

source : [programming guide NVIDIA 2012]

Avant CUDA : pipeline graphique



source : [?]

Vertex Shader

Transformation géométrique

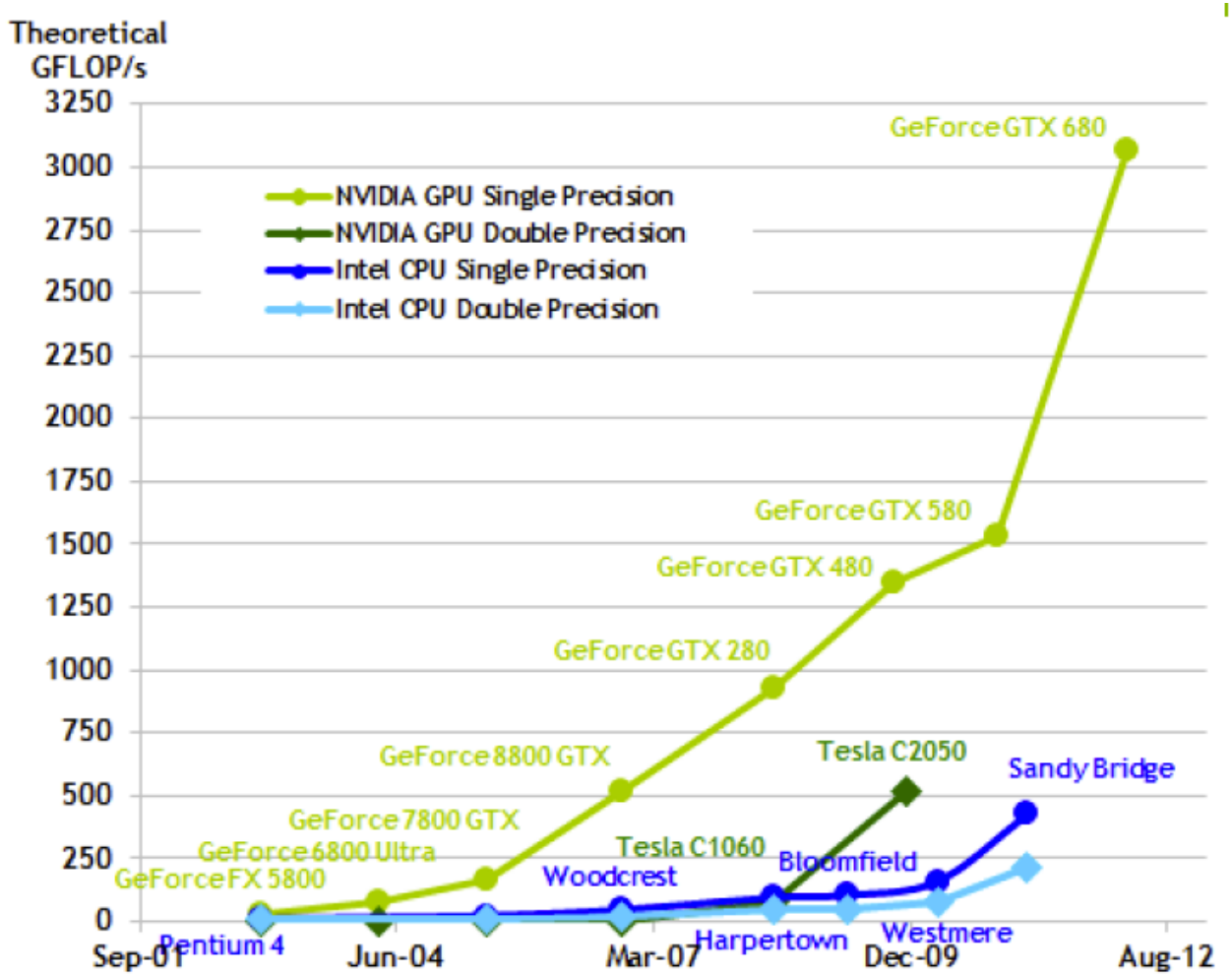
Rasterization

Polygon $->$ Fragments

Fragment Shader

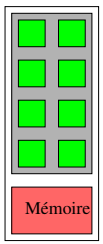
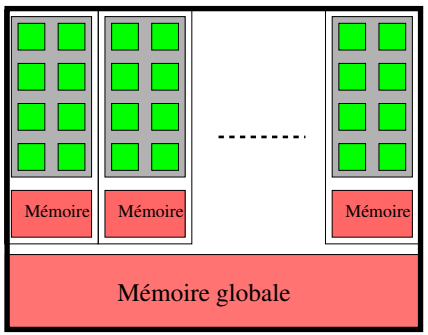
Calcul sur les Pixels

Le GPU, un moteur de calcul

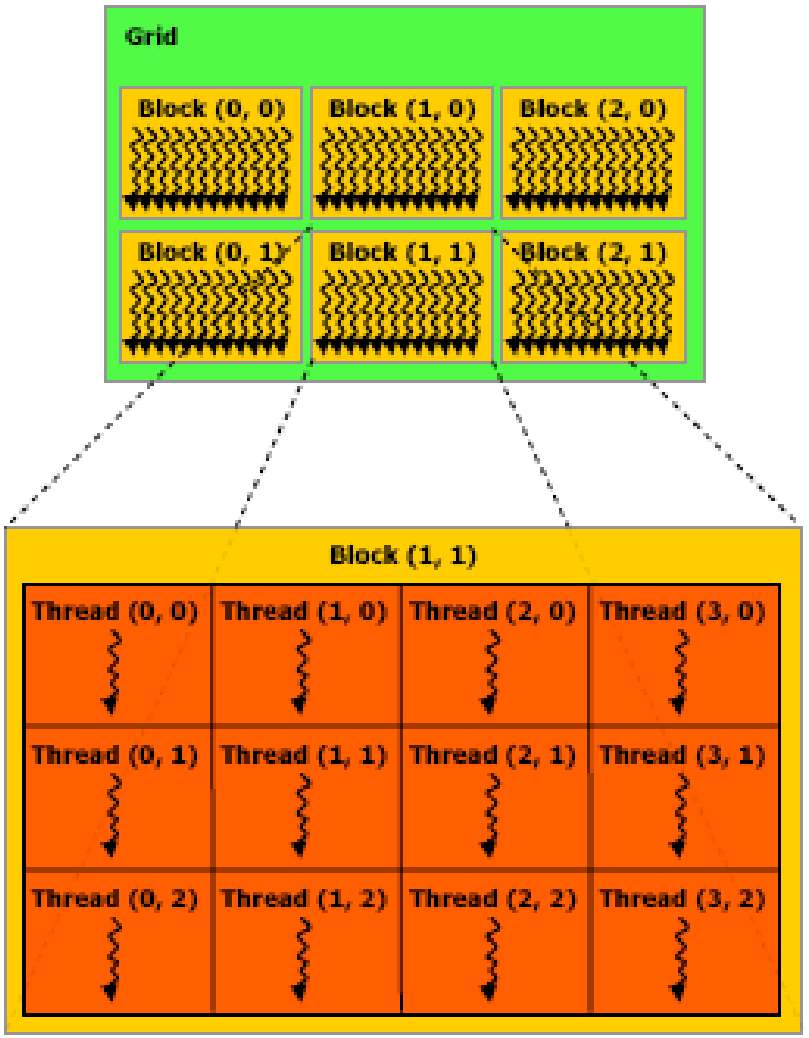


source : [programming guide NVIDIA 2012]

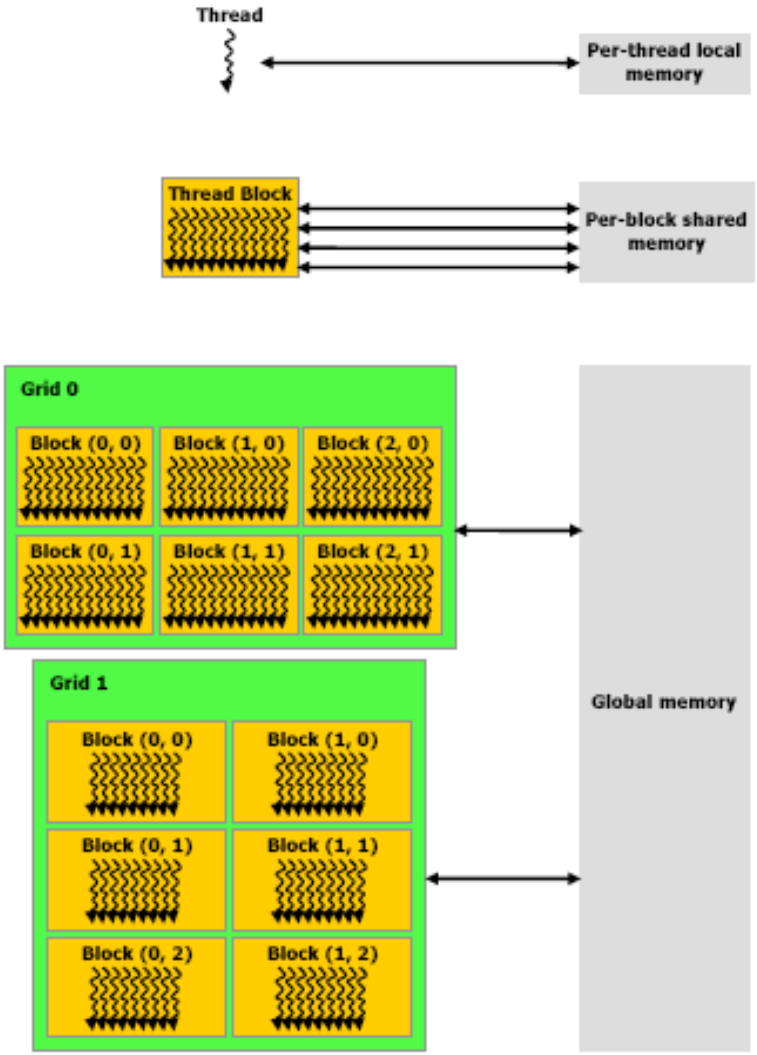
Découpage en threads

	Matériel	Logiciel	Exécution	
	un Streaming Processor (SP)	**un thread**	séquentielle	**(a)**
	un Streaming MultiProcessor (SM)	**un bloc de threads** (plusieurs warps)	parallèle (SIMT)	**(b)**
	une carte GPU (device)	**une grille de threads** (kernel)	parallèle (MIMT)	**(c)**

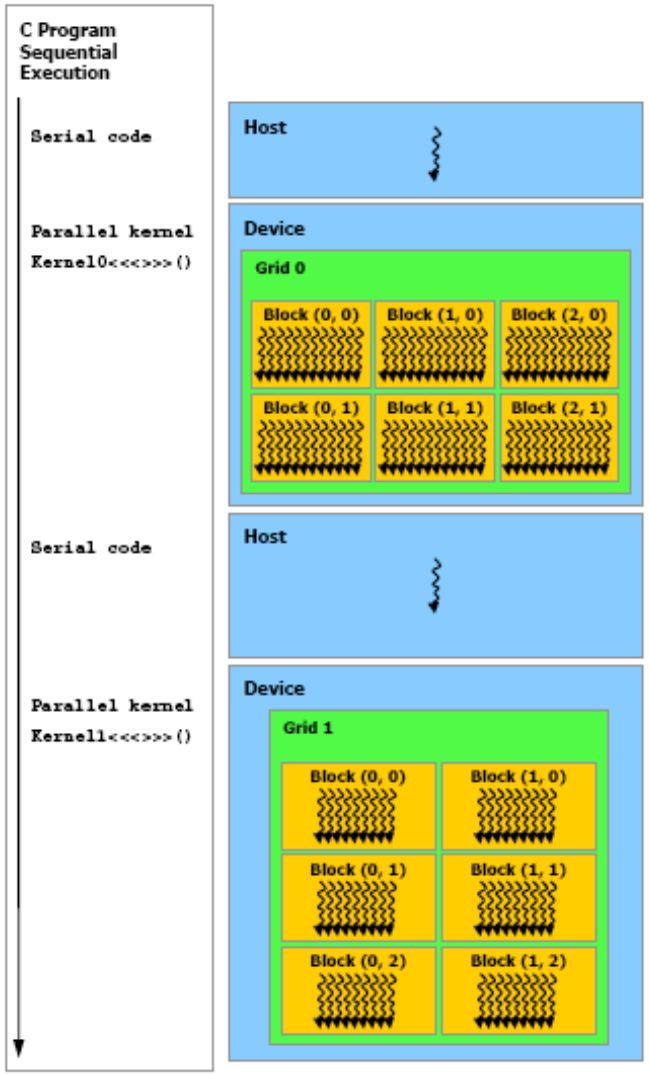
Un **id** par thread et un **id** par bloc de threads



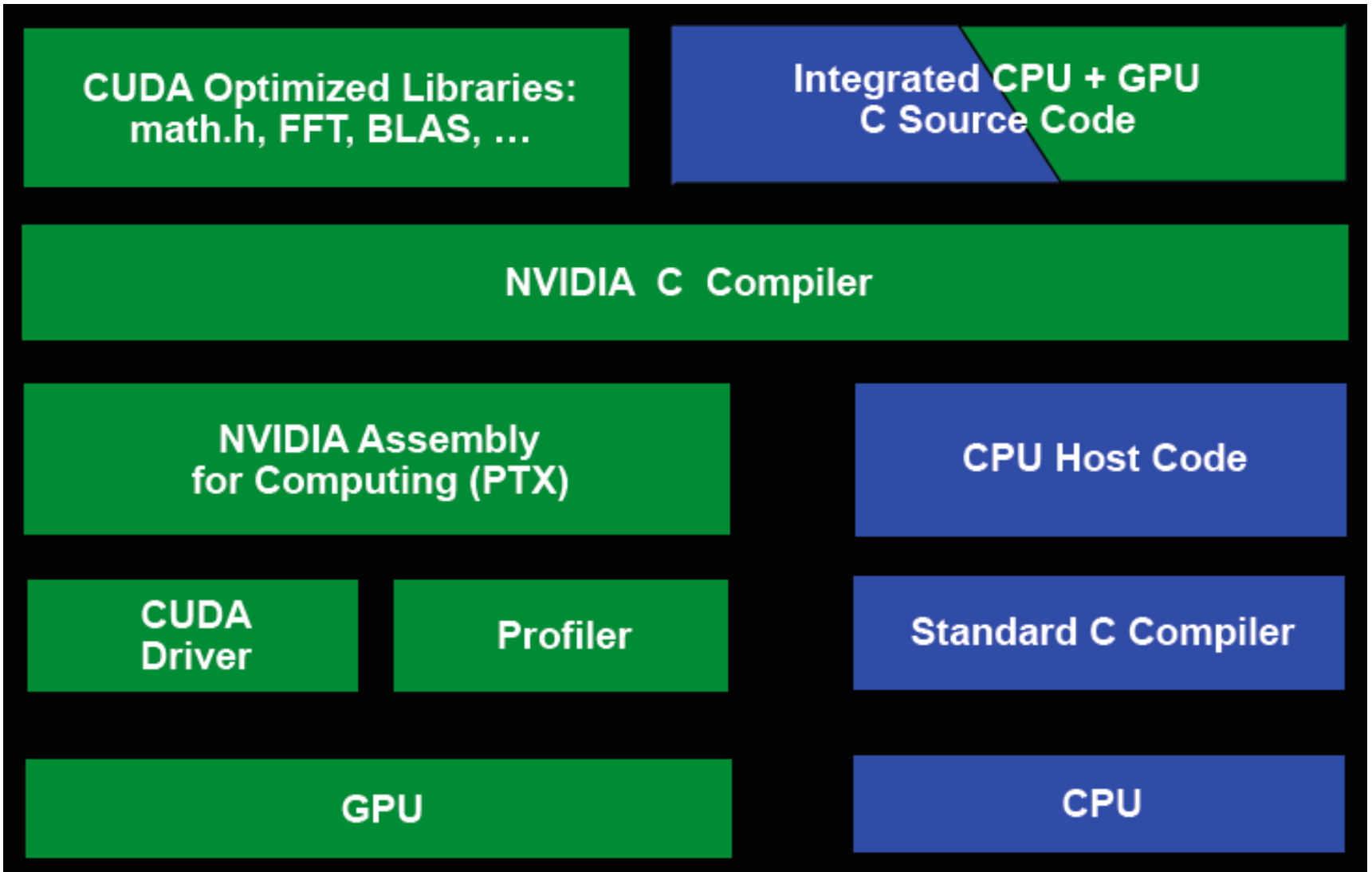
Hiérarchie mémoire



PC hote et carte graphique



Flot de développement logiciel

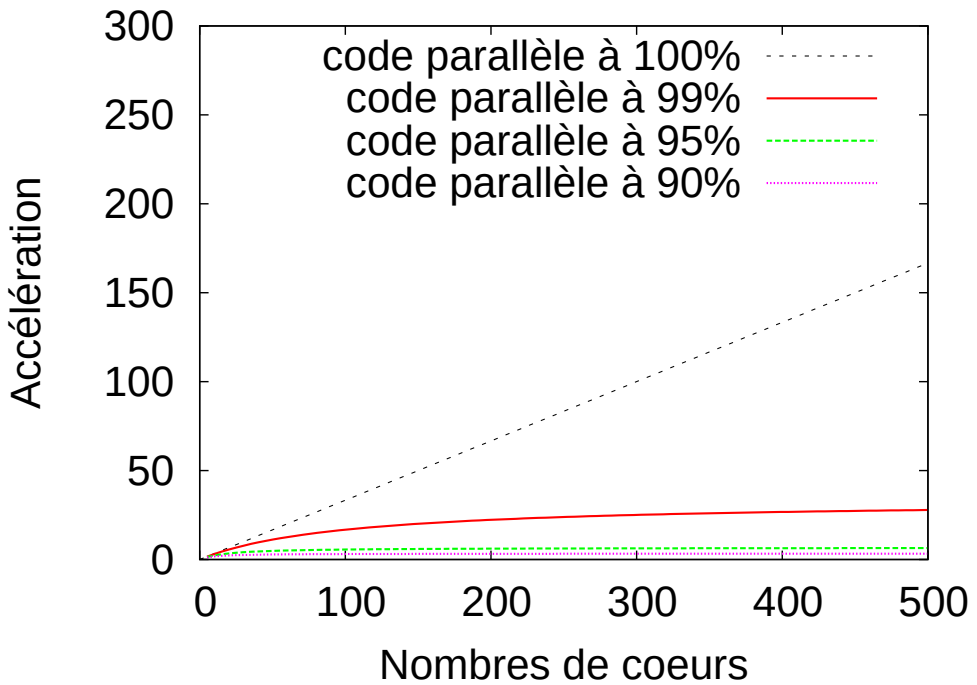


Programmation GPU (1/2)

❶ Parallélisation de l'algorithme

➲ nourrir en threads (plus ou moins indépendants) le GPU

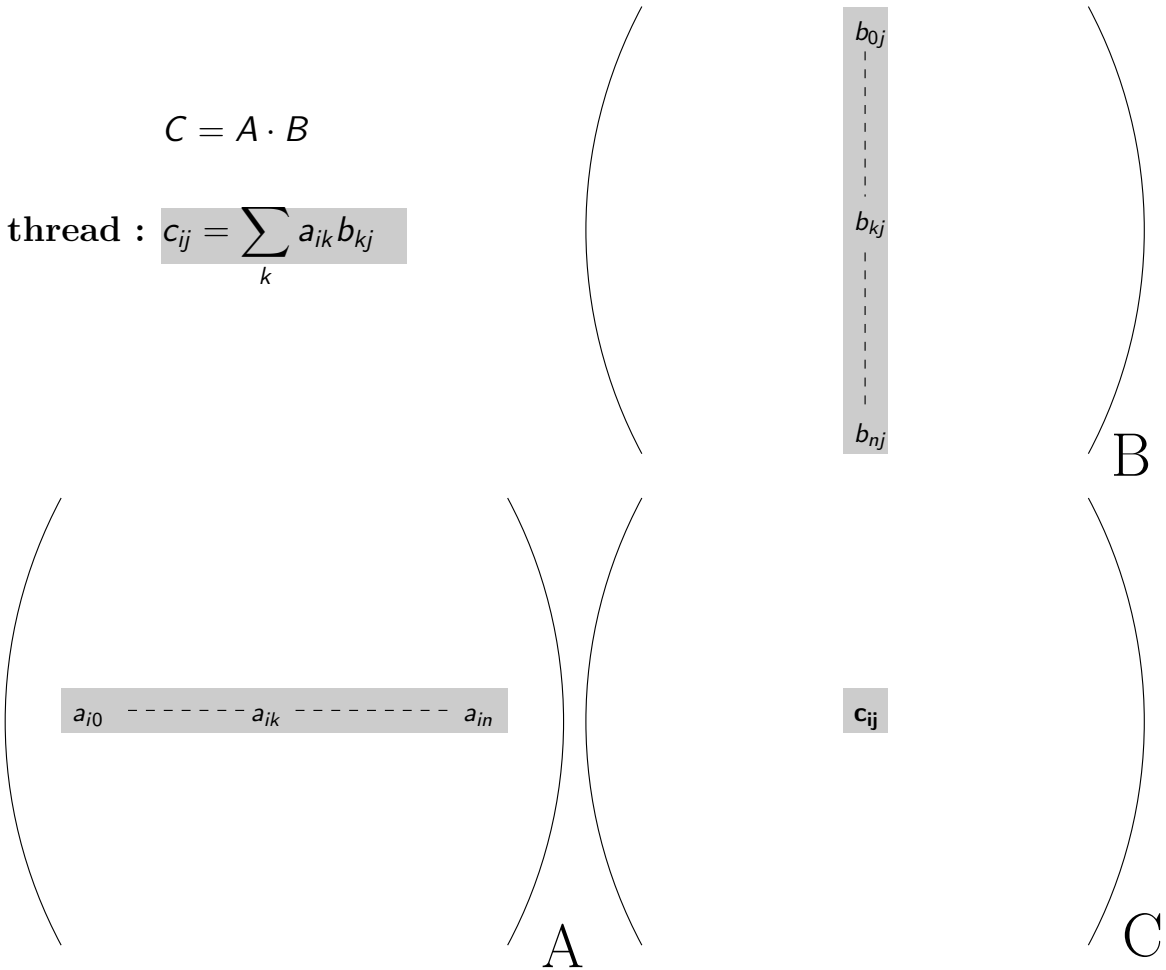
n coeurs (1 Ghz) vs 1 coeur (3 Ghz)	
taux de parallélisation	accélération GTX 200 (240 coeurs)
100 %	80
99 %	24
95 %	6
90 %	3



Parallélisation du calcul matriciel

$$C = A \cdot B$$

un thread : $c_{ij} = \sum_k a_{ik} b_{kj}$

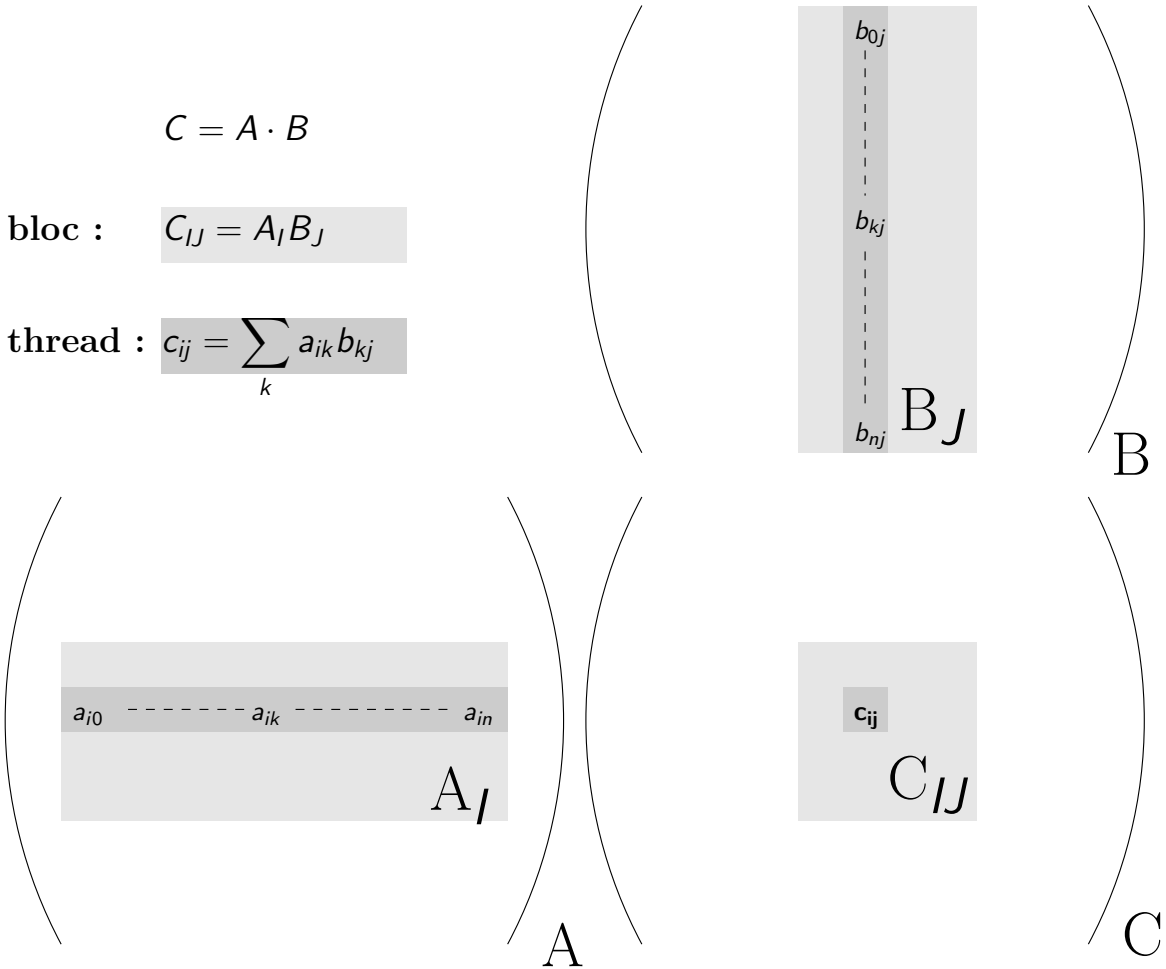


Découpage en blocs de threads

$$C = A \cdot B$$

un bloc : $C_{IJ} = A_I B_J$

un thread : $c_{ij} = \sum_k a_{ik} b_{kj}$



kernel = code des threads executés sur le GPU

```
__global__ void matrixMul_kernel( float* C, float* A, float* B,int matrix_size) {

float Csum;
int ifirst,jfirst;
int i,j;

ifirst=blockIdx.x*BLOCK_SIZE;
jfirst=blockIdx.y*BLOCK_SIZE;

i=ifirst+threadIdx.x;
j=jfirst+threadIdx.y;

for (k = 0; k < matrix_size; k++)
   Csum += A[i][k] * B[k][j];

C[i][j] = Csum;

}
```

Lancement du kernel depuis le PC hôte

```
#define BLOCK_SIZE 16

void matrixMul_host(int N) {

...

//setup execution parameters
dim3 threads(BLOCK_SIZE, BLOCK_SIZE);
dim3 grid(N /BLOCK_SIZE , N /BLOCK_SIZE );

//execute the kernel
matrixMul<<< grid, threads >>>(C_device, A_device, B_device, N);

...

}
```

Gestion de la mémoire GPU via le PC hôte

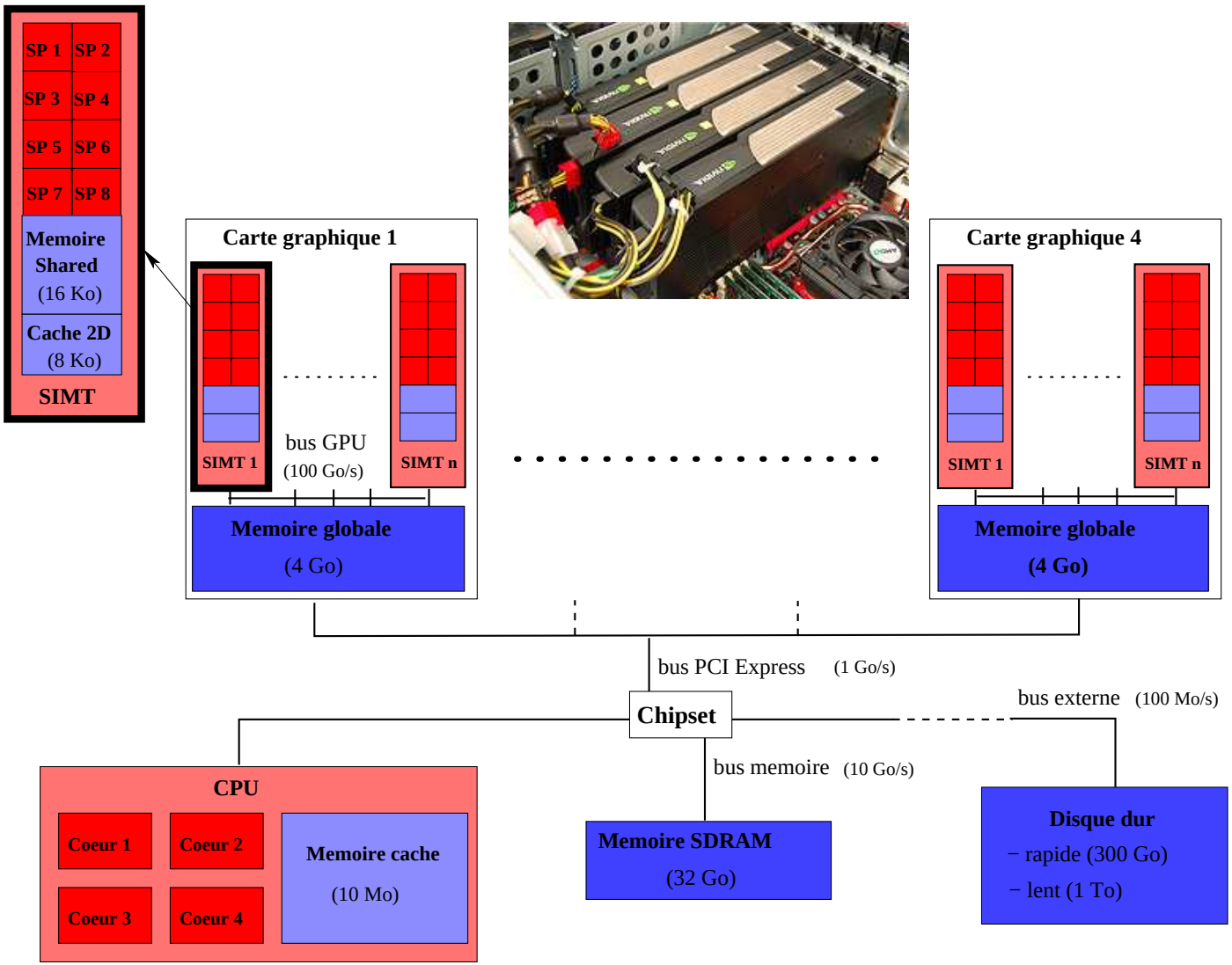
```
#define BLOCK_SIZE 16

void matrixMul_host(int N) {

// allocate host memory int mem_size=N^2*sizeof(float);
float* A_host = (float*) malloc(mem_size);
float* B_host = (float*) malloc(mem_size);
float* C_host = (float*) malloc(mem_size);

// allocate device memory
float* A_device,B_device,C_device;
cudaMalloc((void**) &A_device, mem_size);
cudaMalloc((void**) &B_device, mem_size);
cudaMalloc((void**) &C_device, mem_size);

// copy host memory to device cudaMemcpy(A_device, A_host, mem_size,cudaMemcpyHostToDevice);
cudaMemcpy(B_device, B_host, mem_size,cudaMemcpyHostToDevice);

//setup execution parameters
dim3 threads(BLOCK_SIZE, BLOCK_SIZE);
dim3 grid(N /BLOCK_SIZE , N /BLOCK_SIZE );

// execute the kernel
matrixMul<<< grid, threads >>>(C_device, A_device, B_device, N);

// copy result from device to host
cudaMemcpy(C_host, C_device, mem_size,cudaMemcpyDeviceToHost) ;


}
```

Temps GPU

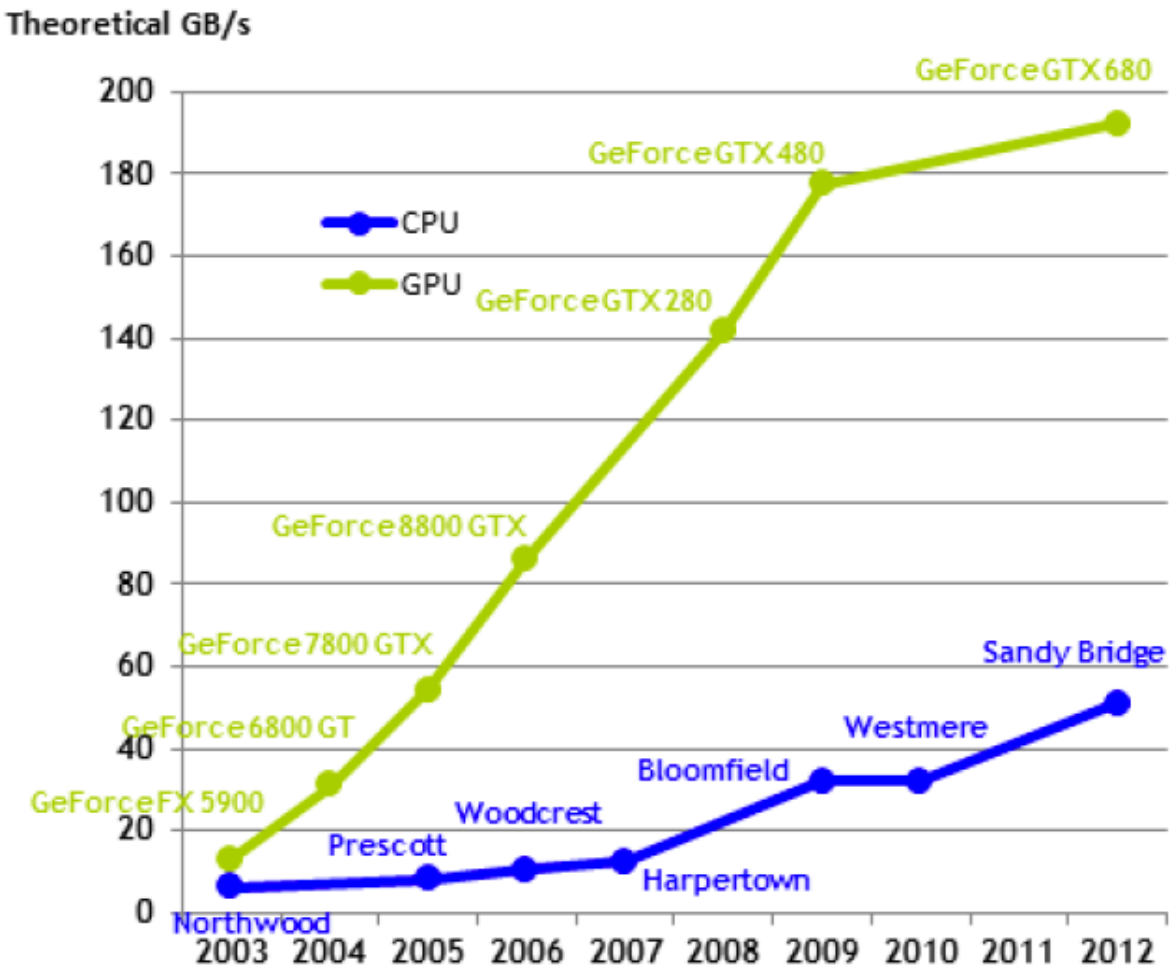
Matrices de taille 1024·1024

	Processeur	Temps d'exécution	Transfert mémoire
C ⚠ code "naif" non optimisé	Intel i5 3.33 Ghz	7.4 s	
Cuda	Geforce GT330 96 PE @1,34 Ghz	1.7 s (***4.5**)	$< 1\%$

Le GPU, un consommateur de données



Le GPU, un consommateur de données



source : [programming guide NVIDIA 2012]

Programmation GPU (2/2)

❶ Parallélisation de l'algorithme

➲ nourrir en threads (plus ou moins indépendants) le GPU

❷ Implémentation GPU

Selon l'intensité arithmétique du code (puissance de calcul exploitée / débit des données), l'execution sera soit *memory bound* soit *computation bound*

➲ optimisation du code portera alors soit sur les **accès mémoire**, soit sur la **complexité arithmétique**

Temps GPU

Matrices de taille 1024·1024

	Processeur	Temps d'exécution	Transfert mémoire
C ⚠ code "naif" non optimisé	Intel i5 3.33 Ghz	7.4 s	
Cuda	Geforce GT330 96 PE @1,34 Ghz	1.7 s (***4.5**)	$< 1\%$
Cuda acces seq.	Geforce GT330 96 PE @1,34 Ghz	283 ms s (***5.9**)	2.3%

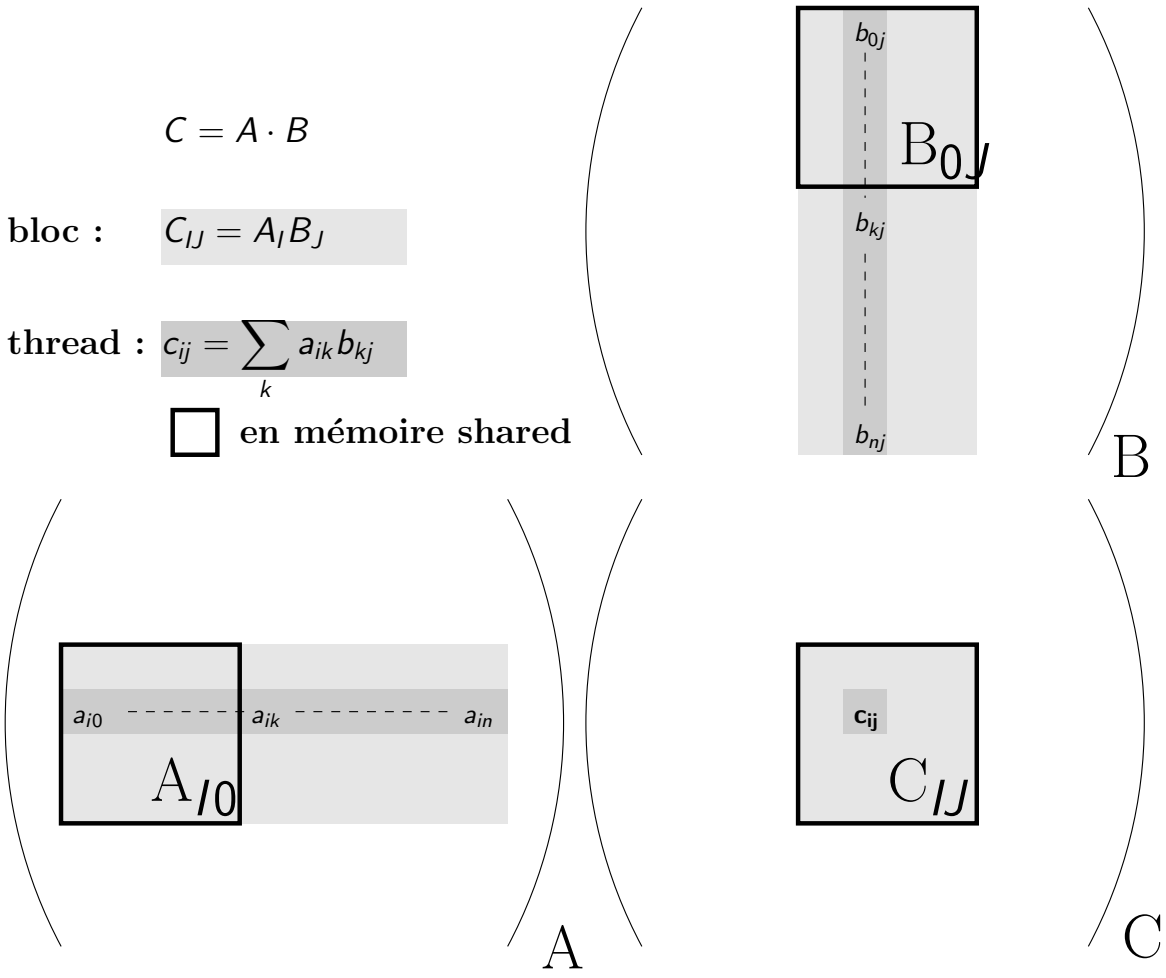
Optimisation des accès mémoire

$C = A \cdot B$

un bloc : $C_{IJ} = A_I B_J$

un thread : $c_{ij} = \sum_k a_{ik} b_{kj}$

en mémoire shared



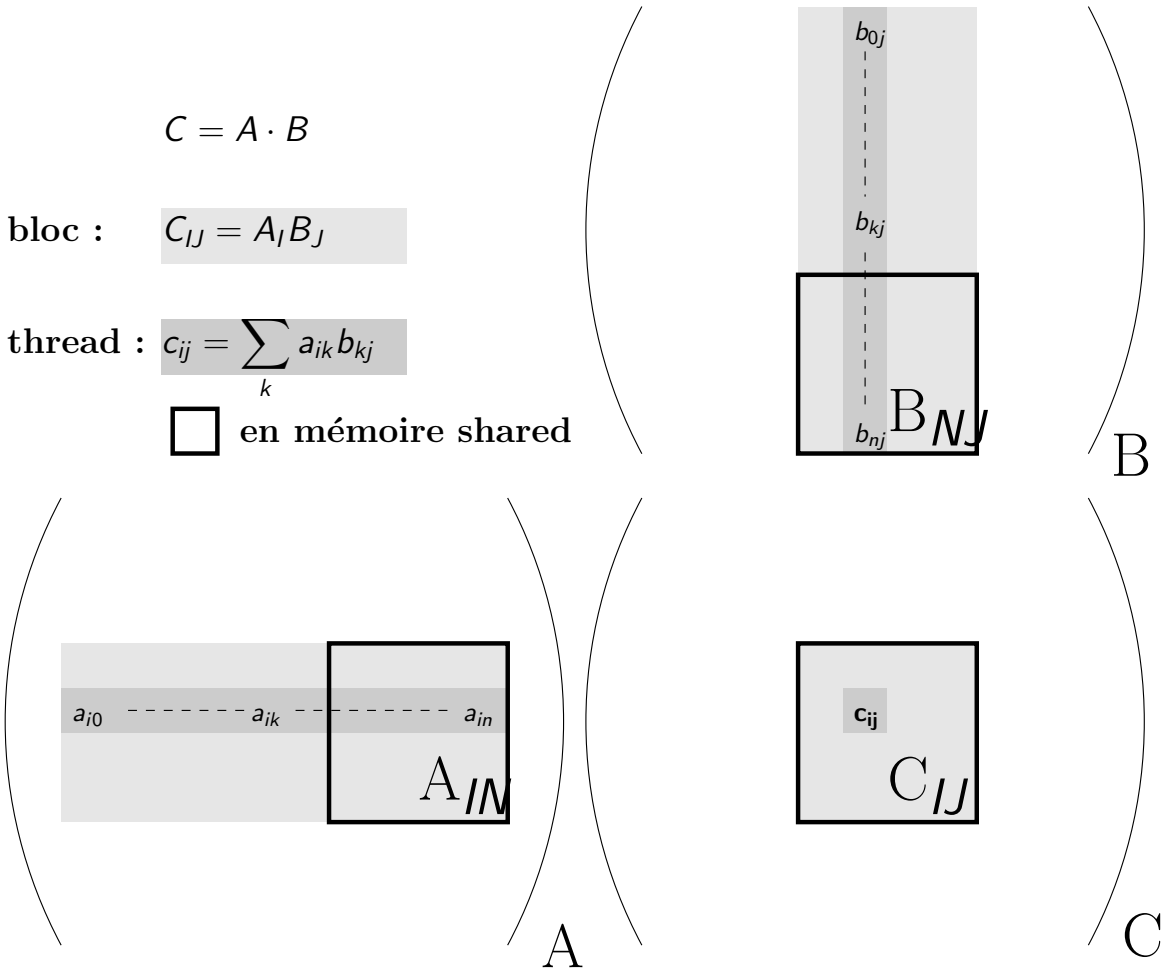
Optimisation des accès mémoire

$C = A \cdot B$

un bloc : $C_{IJ} = A_I B_J$

un thread : $c_{ij} = \sum_k a_{ik} b_{kj}$

en mémoire shared

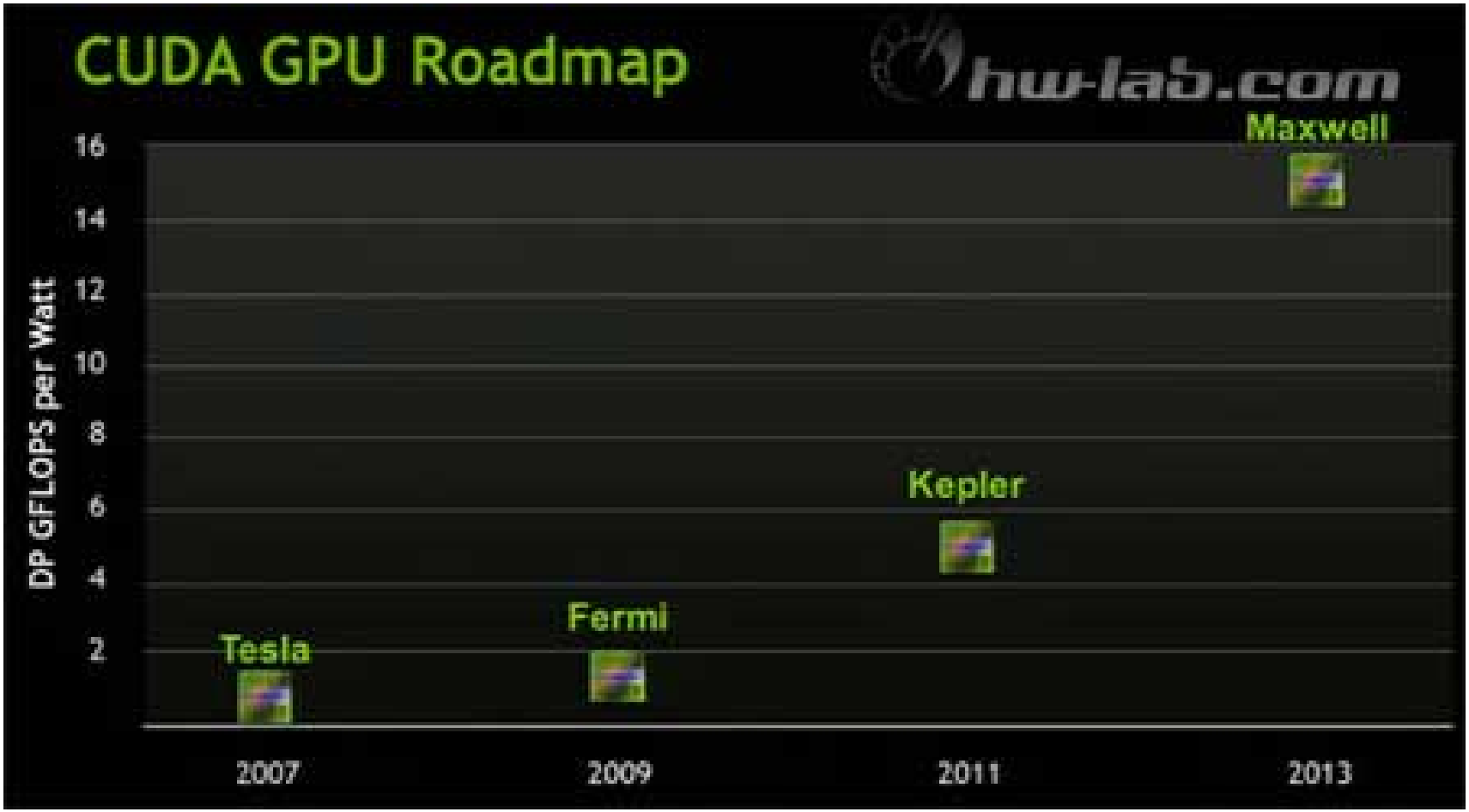


Temps GPU

Matrices de taille 1024·1024

	Processeur	Temps d'exécution	Transfert mémoire
C ⚠ code "naif" non optimisé	Intel i5 3.33 Ghz	7.4 s	
Cuda	Geforce GT330 96 PE @1,34 Ghz	1.7 s (***4.5**)	$< 1\%$
Cuda acces seq.	Geforce GT330 96 PE @1,34 Ghz	283 ms (***5.9**)	2.3%
Cuda shared mem	Geforce GT330 96 PE @1,34 Ghz	38 ms (***9.2**)	22%

Le GPU, une architecture en constante évolution

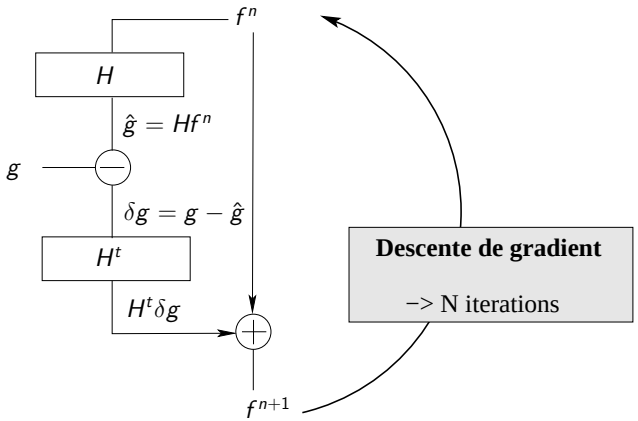


Temps GPU avec une architecture FERMI (4.5 fois plus de coeurs + cache mémoire !)

Matrices de taille 1024·1024

	Processeur	Temps d'exécution
C ⚠ code "naif" non optimisé	Intel i5 3 Ghz	9,68 s
Cuda	Tesla 2050 448 PE @1,15 Ghz	164 ms (***59**)
Cuda acces seq.	Tesla 2050 448 PE @1,15 Ghz	31 ms s (***5.3**)
Cuda shared mem	Tesla 2050 448 PE @1,15 Ghz	12 ms s (***2.6**)

Algorithme itératif : moindres carrés



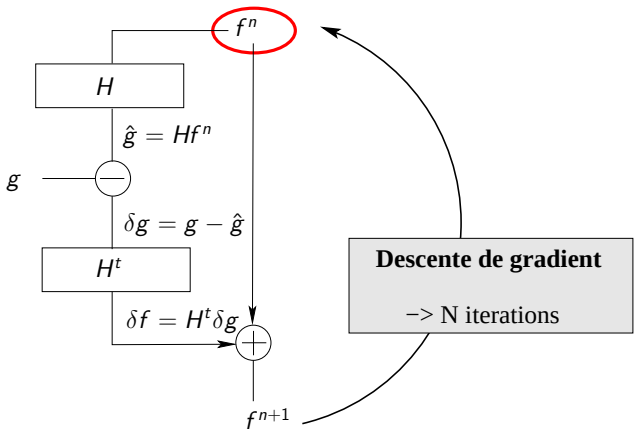
$\mathbf{g = Hf + \epsilon}$

f : objet observé
g : mesure de l'instrument
H : modèle d'acquisition
ϵ : bruit

Descente de gradient

$$J(f) = ||g - Hf||^2$$
$$f^{n+1} = f^n - \alpha \cdot \nabla J(f^n)$$
$$\nabla J(f) = -2 \cdot H^t(g - Hf)$$

Algorithme itératif : moindres carrés



f^n : **Estimée du volume**

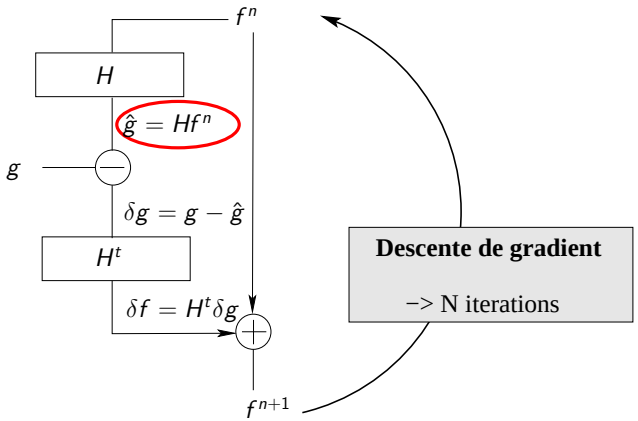
$\mathbf{g = Hf + \epsilon}$

f : objet observé
g : mesure de l'instrument
H : modèle d'acquisition
ϵ : bruit

Descente de gradient

$$J(f) = ||g - Hf||^2$$
$$f^{n+1} = f^n - \alpha \cdot \nabla J(f^n)$$
$$\nabla J(f) = -2 \cdot H^t(g - Hf)$$

Algorithme itératif : moindres carrés



$\hat{g}$: **Estimée des données**

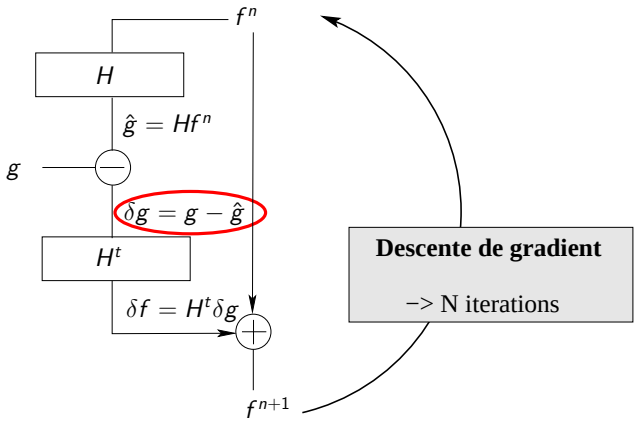
$\mathbf{g = Hf + \epsilon}$

f : objet observé
g : mesure de l'instrument
H : modèle d'acquisition
ϵ : bruit

Descente de gradient

$$J(f) = ||g - Hf||^2$$
$$f^{n+1} = f^n - \alpha \cdot \nabla J(f^n)$$
$$\nabla J(f) = -2 \cdot H^t(g - Hf)$$

Algorithme itératif : moindres carrés



δg : **Correction des données**

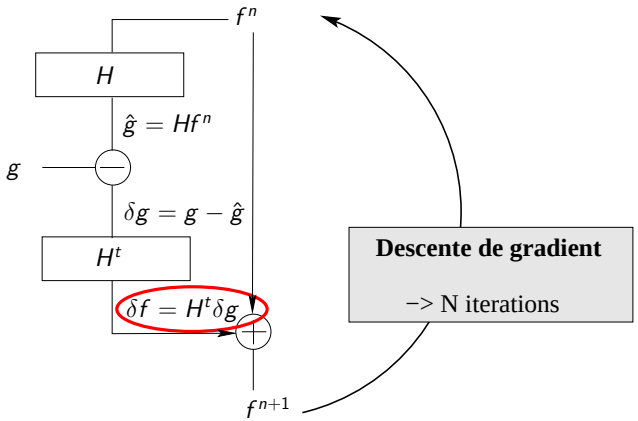
$g = Hf + \epsilon$

f : objet observé
g : mesure de l'instrument
H : modèle d'acquisition
ϵ : bruit

Descente de gradient

$$J(f) = ||g - Hf||^2$$
$$f^{n+1} = f^n - \alpha \cdot \nabla J(f^n)$$
$$\nabla J(f) = -2 \cdot H^t(g - Hf)$$

Algorithme itératif : moindres carrés



δf : **Correction du volume**

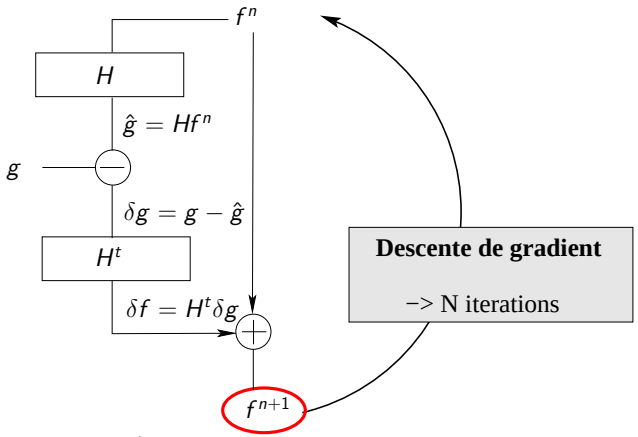
$\mathbf{g = Hf + \epsilon}$

f : objet observé
g : mesure de l'instrument
H : modèle d'acquisition
ϵ : bruit

Descente de gradient

$$J(f) = ||g - Hf||^2$$
$$f^{n+1} = f^n - \alpha \cdot \nabla J(f^n)$$
$$\nabla J(f) = -2 \cdot H^t(g - Hf)$$

Algorithme itératif : moindres carrés



f^{n+1} : **Nouvelle estimée du volume**

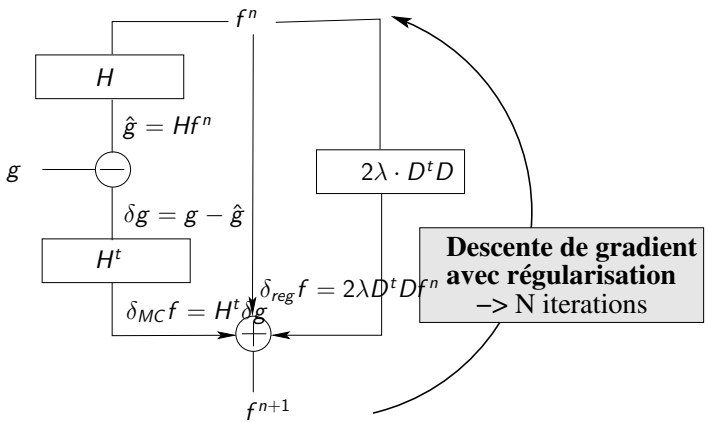
$g = Hf + \epsilon$

f : objet observé
g : mesure de l'instrument
H : modèle d'acquisition
ϵ : bruit

Descente de gradient

$$J(f) = ||g - Hf||^2$$
$$f^{n+1} = f^n - \alpha \cdot \nabla J(f^n)$$
$$\nabla J(f) = -2 \cdot H^t(g - Hf)$$

Algorithme itératif : moindres carrés + régularisation



$\mathbf{g = Hf + \epsilon}$

f : objet observé
g : mesure de l'instrument
H : modèle d'acquisition
ϵ : bruit

Descente de gradient régularisée

$$J(f) = J_{MC} f) + J_{reg}(f)$$
$$J_{MC}(f) = ||g - Hf||^2$$
$$J_{reg}(f) = \lambda ||Df||^2$$
$$f^{n+1} = f^n - \alpha \cdot (\nabla J_{MC}(f^n) + \nabla J_{reg}(f^n))$$
$$\nabla J_{MC}(f) = -2 \cdot H^t(g - Hf)$$
$$\nabla J_{reg}(f) = 2\lambda \cdot D^t Df$$

Correction de vibrations mécaniques par déconvolution 1D

Collaboration avec l'IDES de l'Univ. Paris-Sud (F. Schmidt)

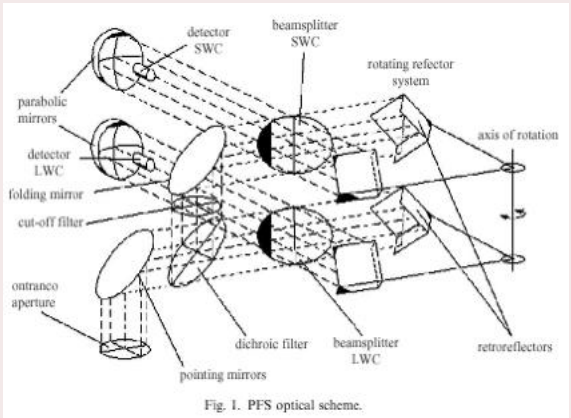
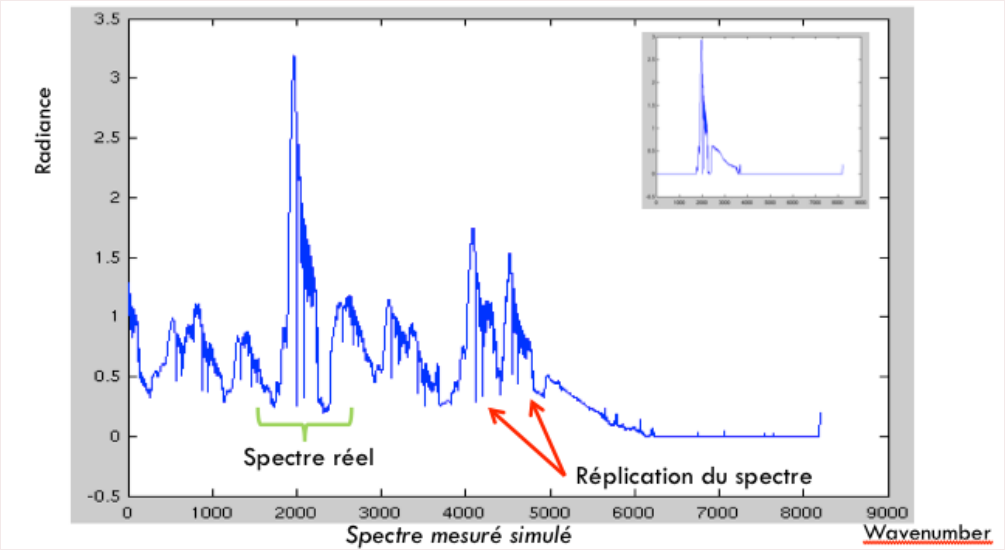


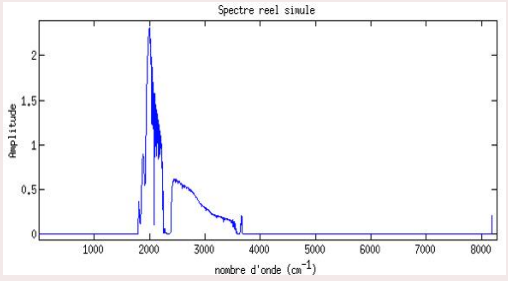
Fig. 1. PFS optical scheme.

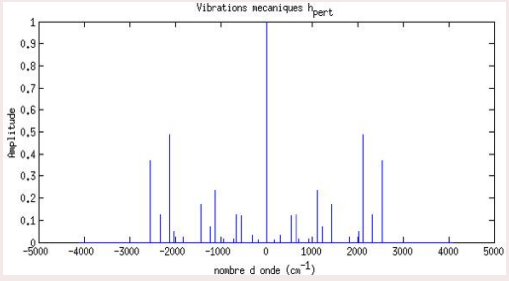
Instrument PFS (Planetary Fourier Spectrum) de la mission MARS EXPRESS



Correction de vibrations mécaniques par déconvolution 1D

Instrument modélisé par une convolution 1D

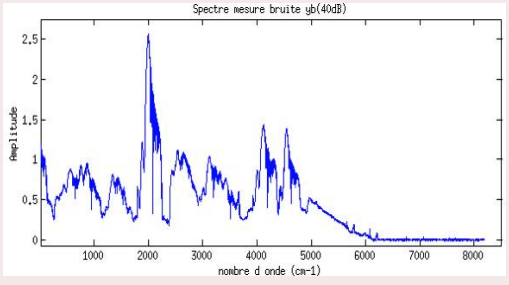




x (spectre réel) * h (instrument PFS)

=

y (spectre mesuré)



Taille gigantesque des données

Des années d'enregistrements de la mission MARS EXPRESS (2003) donc potentiellement 1 milliard de spectres (de 8192 échantillons) !

Déconvolution 2D

Restauration d'image (instrument modélisé par une convolution2D)

Image source
f

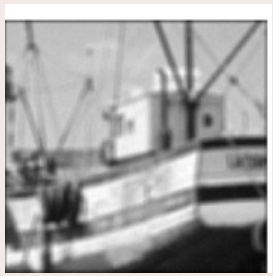
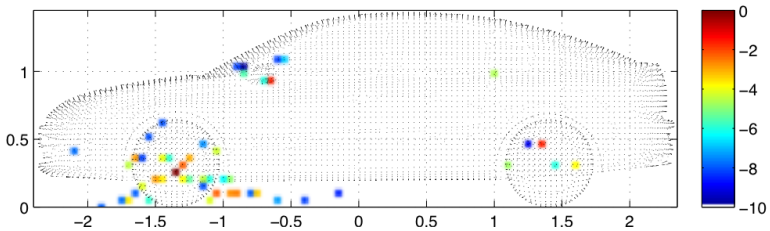


Image acquise
$g = Hf + \epsilon$

Image deconvoluée
$\hat{f}$

Localisation de sources acoustiques (structure itérative et H = convolution 2D)

Thèse Ning CHU au GPI/L2S (direction : A.Djafari) en collaboration avec Supélec (José Picheral)



Algorithmes de reconstruction tomographique

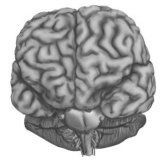
Objet à imager

APPLICATIONS
- **Controle non destructif** (réacteurs, pipelines, pièces industrielles...)
- **Imagerie medicale**
- **Transport**
- **Agro–alimentaire**
- **Micro–électronique** (controle qualité)

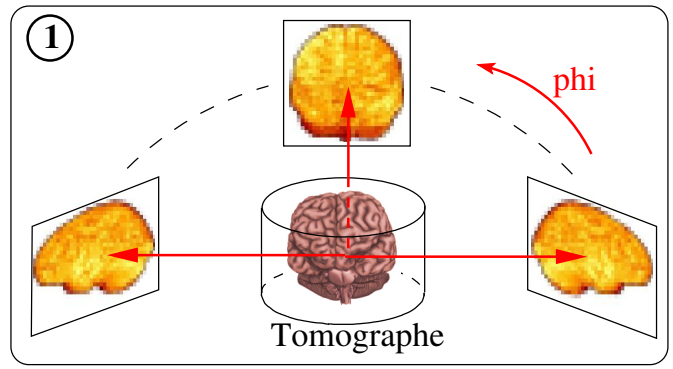
① ACQUISITION
- Scanner (X–ray, TEP..)

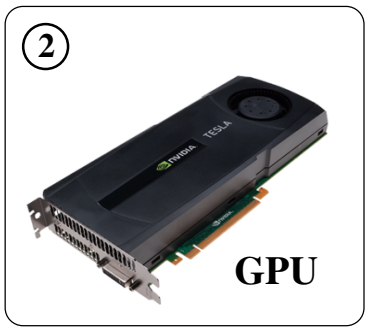
Projections

② RECONSTRUCTION
- Algorithme itératif
- Accélération sur GPU



Structure interne de l'objet





Algorithmes de reconstruction tomographique

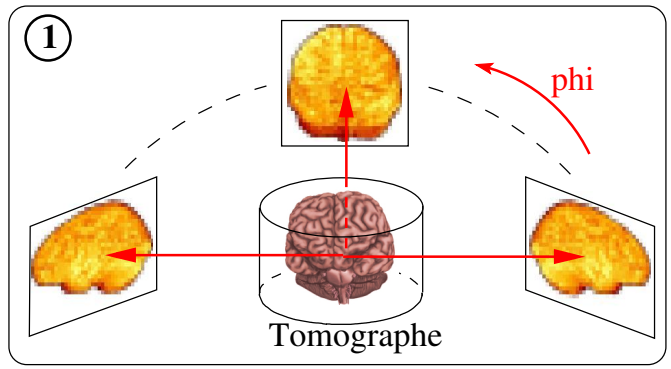
Objet à imager

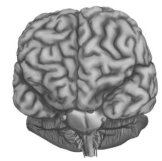
APPLICATIONS
- **Controle non destructif** (réacteurs, pipelines, pièces industrielles...)
- **Imagerie medicale**
- **Transport**
- **Agro–alimentaire**
- **Micro–électronique** (controle qualité)

① ACQUISITION
– Scanner (X–ray, TEP..)

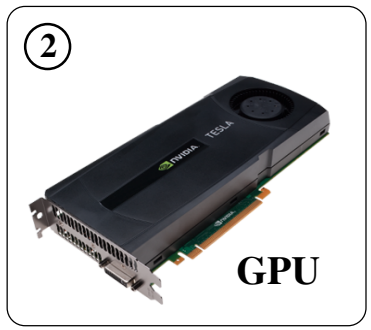
Projections

② RECONSTRUCTION
– Algorithme itératif
– Accélération sur GPU





Structure interne de l'objet



Reconstruction Tomographique 3D pour le Dentaire

Thèse de Long Chen au GPI/L2S (direction : Thomas Rodet) en collaboration avec **Carestream Dental**

Calcul de Hf et $H^t \delta g$: choix de la méthode

❶ Calcul matriciel

➲ lecture des coefficients h_{ij} dans la mémoire SDRAM
⚠volume $2048^3 -> $ matrice H $= 1$ To !

Calcul de Hf et $H^t\delta g$: choix de la méthode

❶ Calcul matriciel

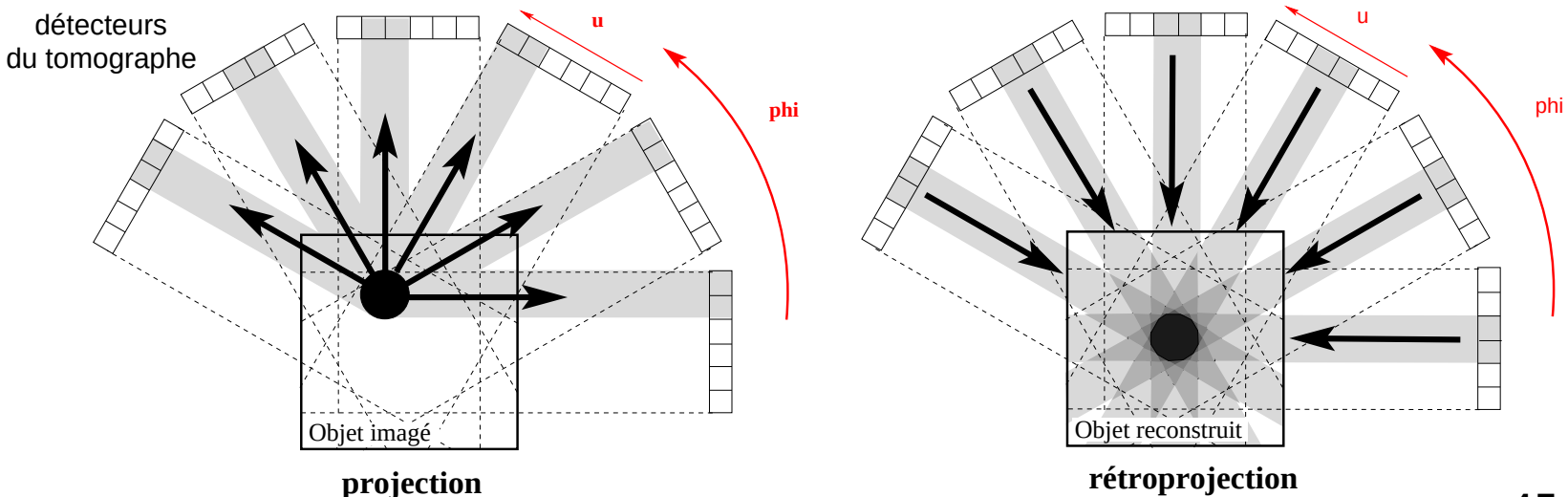
➲ lecture des coefficients h_{ij} dans la mémoire SDRAM

⚠volume $2048^3 -> $ matrice H $= 1$ To !

❷ Opérateurs géométriques

➲ calcul en ligne des coefficients h_{ij}

Paire de projection/rétroprojection en tomographie à émission (géométrie parallèle)



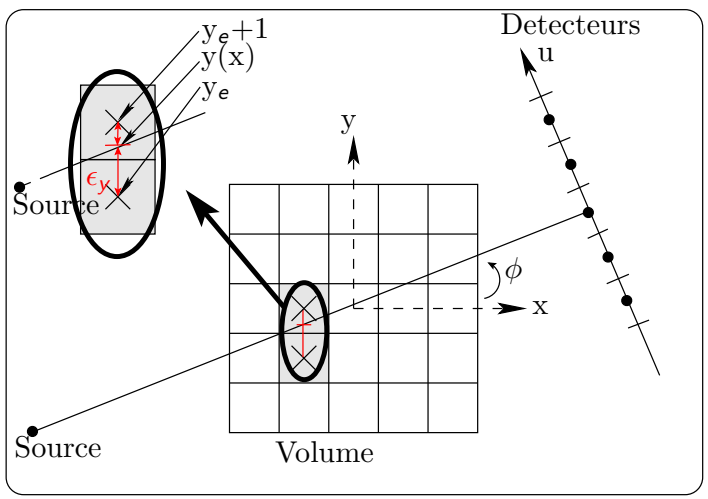
Deconvolution1D avec conv1D Matlab/C/GPU

Test sur un PC avec une Tesla 2050

- 100 itérations de l'algorithme des moindres carrés régularisés.
- Traitement de 512 vecteurs de 8192 échantillons.

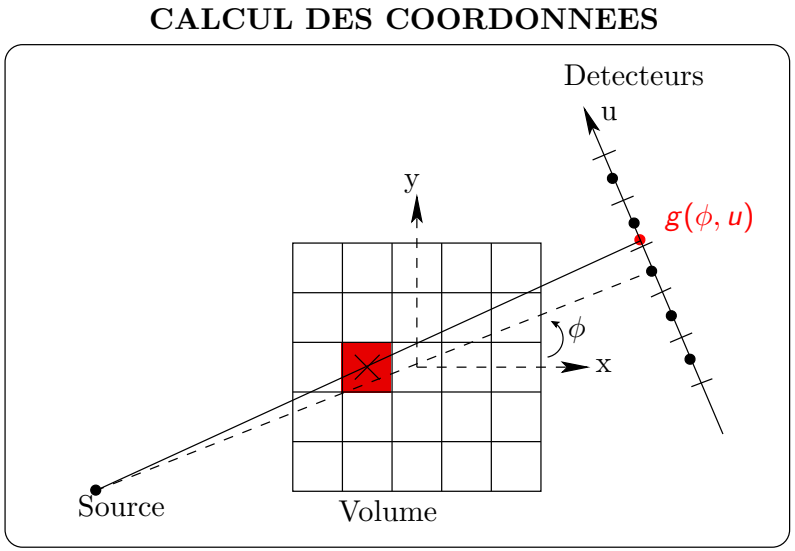
version	**H, Ht**	**boucle**	**Temps**	**Acc.**
v1	Matlab	Matlab	49,3h	
v2	C	Matlab	5,6h	***8,8**
v3	GPU	Matlab	27,3 mn	***12,3**
v4	GPU	C	20,5 mn	***1,33**

Projecteur 2D "échantilonnage régulier"



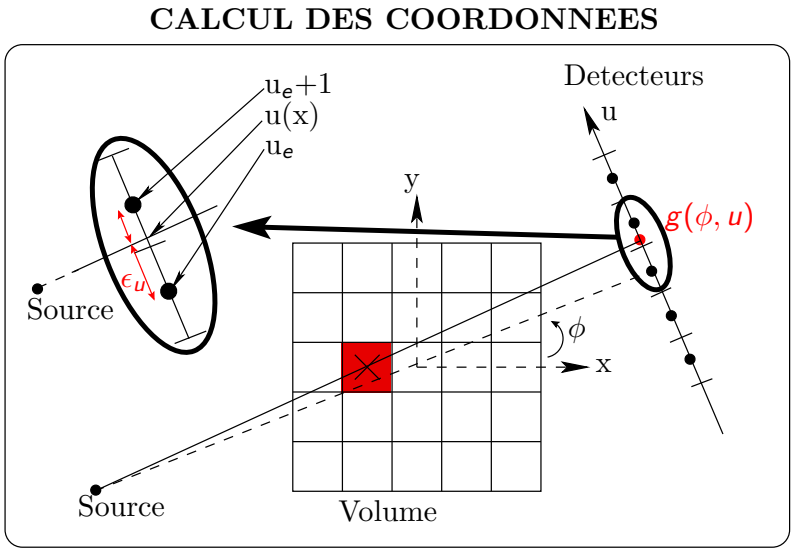
for $(\texttt{un}, \texttt{phi})$ in Projection **do**
  for $\texttt{xn} = 0$ to $\texttt{xn}_{\texttt{max}} - 1$ **do**
    // Calcul des coordonnées
    $\texttt{yn}(\texttt{xn}, \texttt{un}, \texttt{phi}) = ...$
    // Interpolation bi-linéaire
    $f_{interp} = ...$
    // Accumulation
    $g^*(\texttt{un}, \texttt{phi}) += f_{interp}$
  end for
end for

Rétroprojection 2D : algorithme



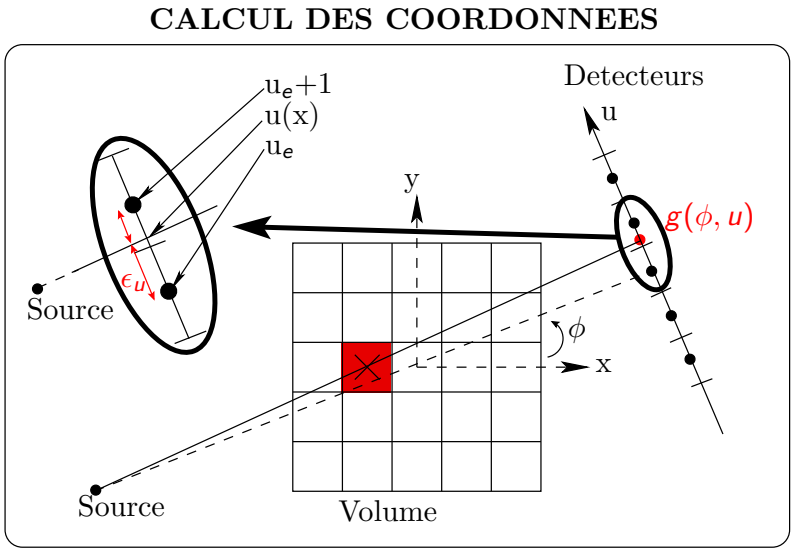
for $(\texttt{xn}, \texttt{yn})$ in Volume **do**
  for $\texttt{phi} = 0$ to $\texttt{phi}_{\texttt{max}} - 1$ **do**
    // Calcul des coordonnées
    $\texttt{u}(\texttt{phi}, \texttt{xn}, \texttt{yn}) = ...$
    // Accumulation
    $f^*(\texttt{xn}, \texttt{yn}) += g(u, \phi)$
  end for
end for

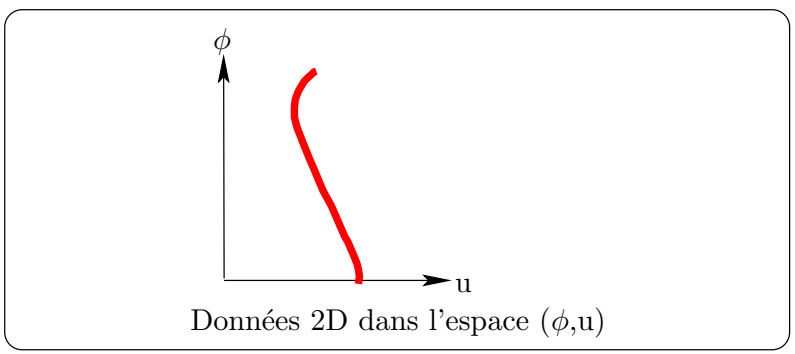
Rétroprojection 2D : interpolation linéaire



for $(\texttt{xn}, \texttt{yn})$ in Volume **do**
  for $\texttt{phi} = 0$ to $\texttt{phi}_{\texttt{max}} - 1$ **do**
    // Calcul des coordonnées
    $\texttt{u}(\texttt{phi}, \texttt{xn}, \texttt{yn}) = ...$
    // Interpolation linéaire
    $g_{interp} = (1 - \epsilon_u) \cdot g(\texttt{phi}, \texttt{u}_\texttt{e}) +$
    $\epsilon_u \cdot g(\texttt{phi}, \texttt{u}_\texttt{e} + 1)$
    // Accumulation
    $f^*(\texttt{xn}, \texttt{yn}) += g_{interp}$
  end for
end for

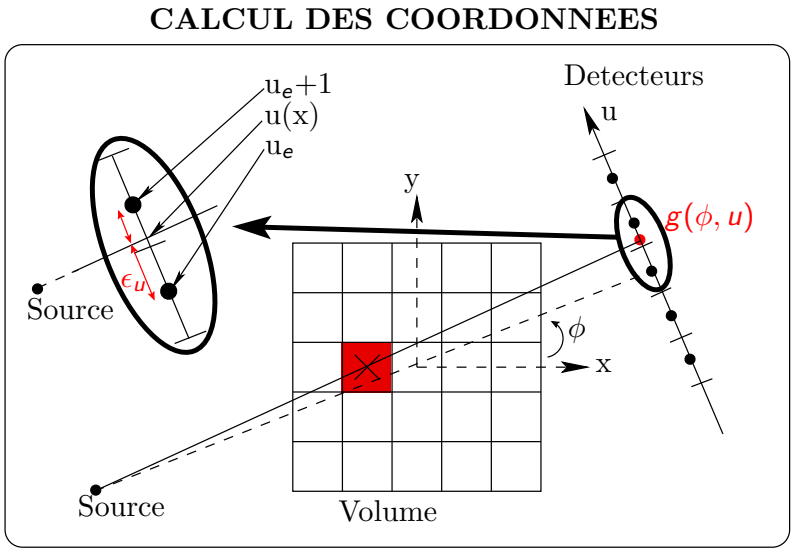
Rétroprojection 2D : accès aux données dispersés

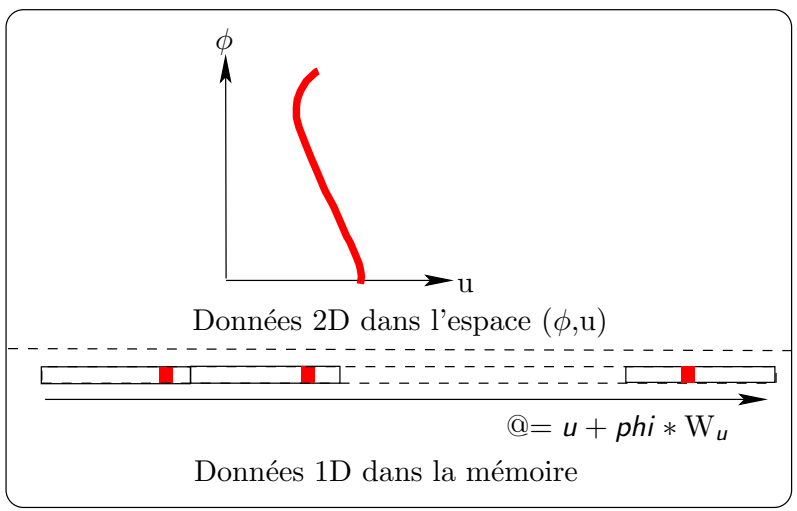




for $(\mathtt{xn}, \mathtt{yn})$ in Volume **do**
  for $\mathtt{phi} = 0$ to $\mathtt{phi}_{\mathtt{max}} - 1$ **do**
    // Calcul des coordonnées
    $\mathtt{u}(\mathtt{phi}, \mathtt{xn}, \mathtt{yn}) = \ldots$
    // Interpolation linéaire
    $g_{interp} = (1 - \epsilon_u) \cdot g(\mathtt{phi}, \mathtt{u_e}) +$
    $\epsilon_u \cdot g(\mathtt{phi}, \mathtt{u_e} + 1)$
    // Accumulation
    $f^*(\mathtt{xn}, \mathtt{yn}) += g_{interp}$
  end for
end for

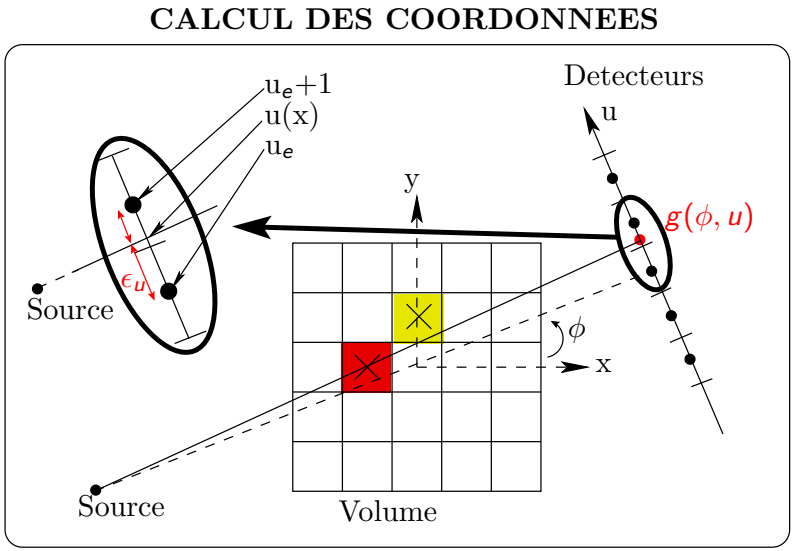
Rétroprojection 2D : accès aux données dispersés

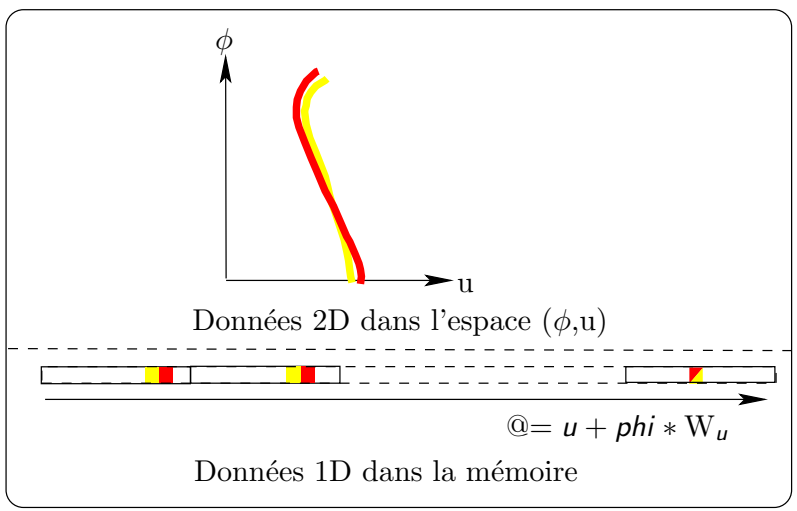




for $(\mathtt{xn}, \mathtt{yn})$ in Volume **do**
  for $\mathtt{phi} = 0$ to $\mathtt{phi}_{\mathtt{max}} - 1$ **do**
    // Calcul des coordonnées
    $\mathtt{u}(\mathtt{phi}, \mathtt{xn}, \mathtt{yn}) = \ldots$
    // Interpolation linéaire
    $g_{interp} = (1 - \epsilon_u) \cdot g(\mathtt{phi}, \mathtt{u_e}) +$
    $\epsilon_u \cdot g(\mathtt{phi}, \mathtt{u_e} + 1)$
    // Accumulation
    $f^*(\mathtt{xn}, \mathtt{yn}) += g_{interp}$
  end for
end for

Rétroprojection 2D : accès aux données dispersés

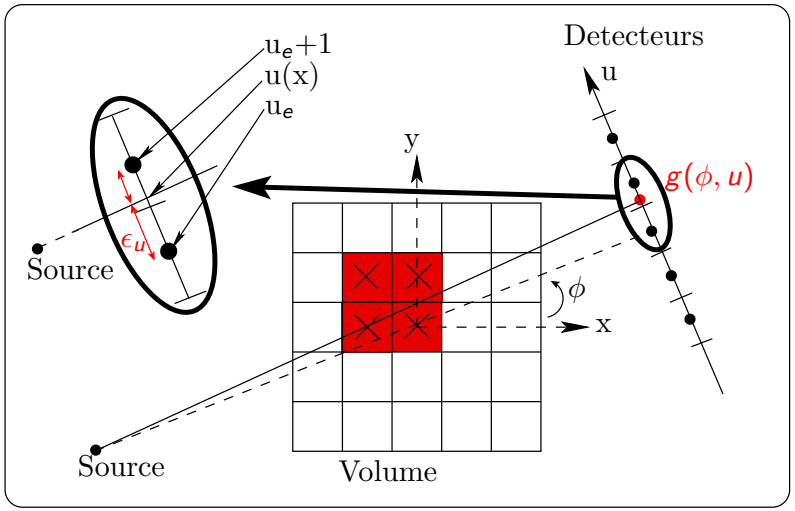




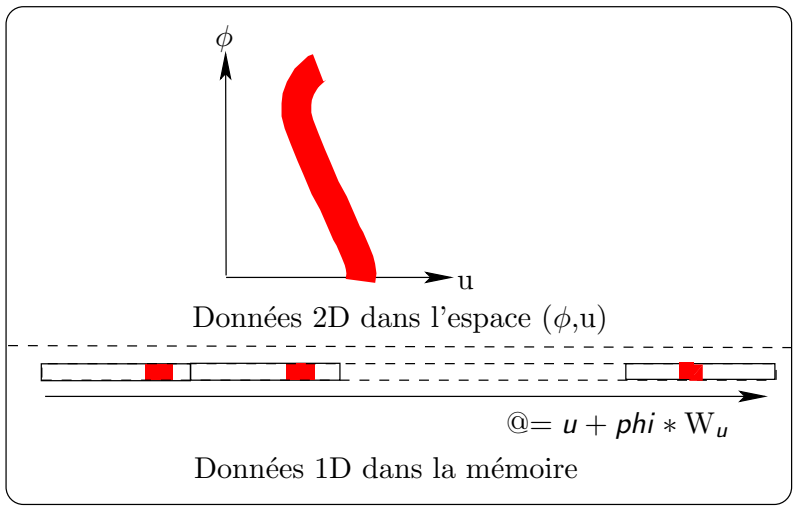
for $(\mathtt{xn}, \mathtt{yn})$ in Volume **do**
  for $\mathtt{phi} = 0$ to $\mathtt{phi}_{\mathtt{max}} - 1$ **do**
    // Calcul des coordonnées
    $\mathtt{u}(\mathtt{phi}, \mathtt{xn}, \mathtt{yn}) = \ldots$
    // Interpolation linéaire
    $g_{interp} = (1 - \epsilon_u) \cdot g(\mathtt{phi}, \mathtt{u_e}) +$
    $\epsilon_u \cdot g(\mathtt{phi}, \mathtt{u_e} + 1)$
    // Accumulation
    $f^*(\mathtt{xn}, \mathtt{yn}) += g_{interp}$
  end for
end for

Rétroprojection 2D par bloc : accès aux données localisés

CALCUL DES COORDONNEES

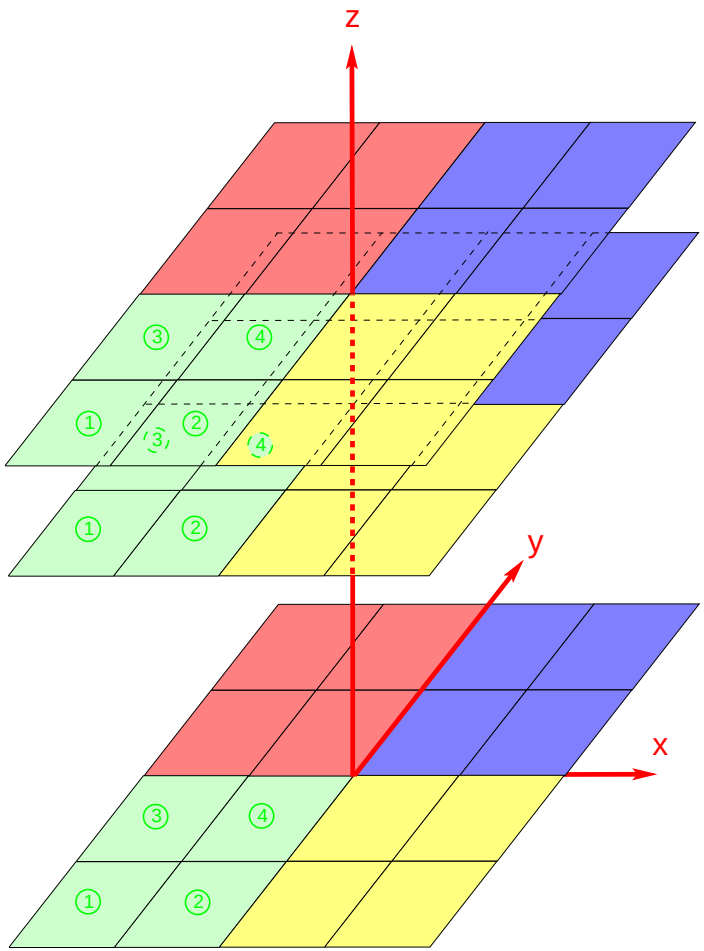


ACCES AUX DONNEES $g(\phi, u)$



for (Bx, By) in Volume **do**
  for phi $= 0$ to $\texttt{phi}_{\texttt{max}} - 1$ **do**
    for (xn, yn) in Bloc **do**
      // *Calcul des coordonnées*
      $\texttt{u}(\texttt{phi}, \texttt{xn}, \texttt{yn}) = \ldots$
      // *Interpolation linéaire*
      $g_{interp} = (1 - \epsilon_u) \cdot g(\texttt{phi}, \texttt{u}_\texttt{e}) + \epsilon_u \cdot g(\texttt{phi}, \texttt{u}_\texttt{e} + 1)$
      // *Accumulation*
      $f^*(\texttt{xn}, \texttt{yn}) += g_{interp}$
    end for
  end for
end for

Découpage en threads de la rétroprojection 3D



(a) Calcul séquentiel sur un PE

- Boucle sur z
- Boucle sur ϕ

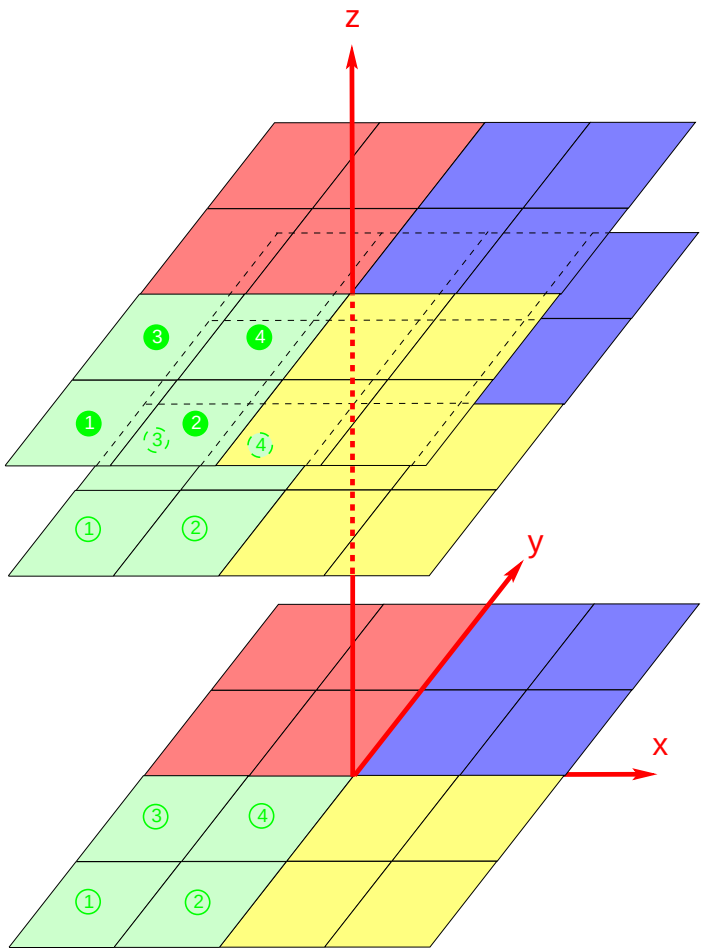
(b) Calcul parallèle sur un proc. SIMT

- Boucle sur (x,y)

(c) Calcul parallèle sur une carte

- Boucle sur les blocs (Bx,By,Bz)

Découpage en threads de la rétroprojection 3D



(a) Calcul séquentiel sur un PE

- Boucle sur z
- Boucle sur ϕ

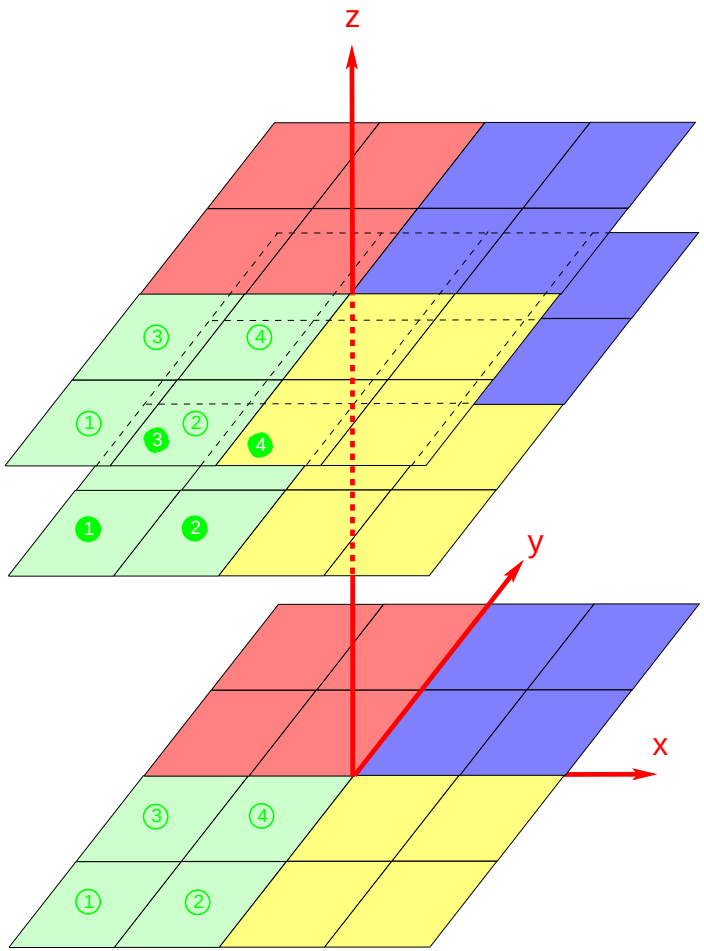
(b) Calcul parallèle sur un proc. SIMT

- Boucle sur (x,y)

(c) Calcul parallèle sur une carte

- Boucle sur les blocs (Bx,By,Bz)

Découpage en threads de la rétroprojection 3D



(a) Calcul séquentiel sur un PE

- Boucle sur z
- Boucle sur ϕ

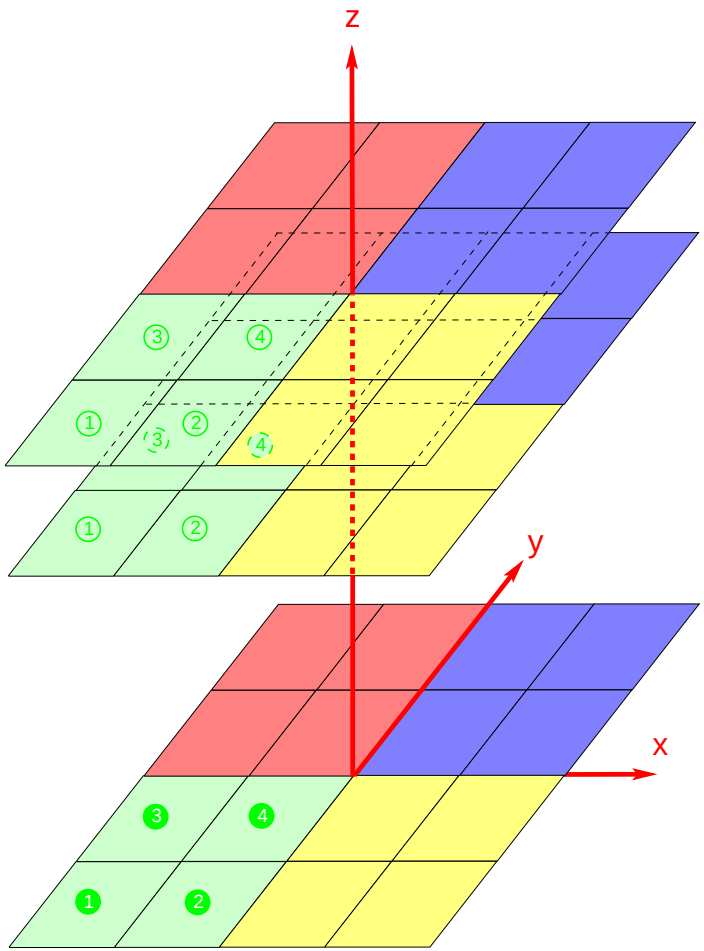
(b) Calcul parallèle sur un proc. SIMT

- Boucle sur (x,y)

(c) Calcul parallèle sur une carte

- Boucle sur les blocs (Bx,By,Bz)

Découpage en threads de la rétroprojection 3D



(a) Calcul séquentiel sur un PE

- Boucle sur z
- Boucle sur ϕ

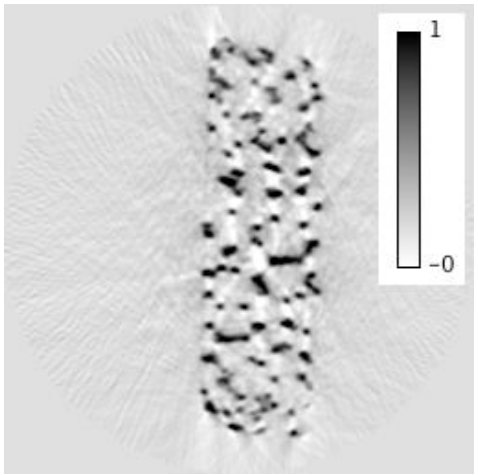
(b) Calcul parallèle sur un proc. SIMT

- Boucle sur (x,y)

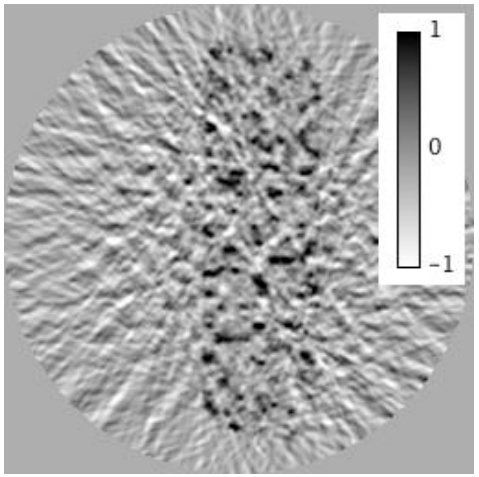
(c) Calcul parallèle sur une carte

- Boucle sur les blocs (Bx,By,Bz)

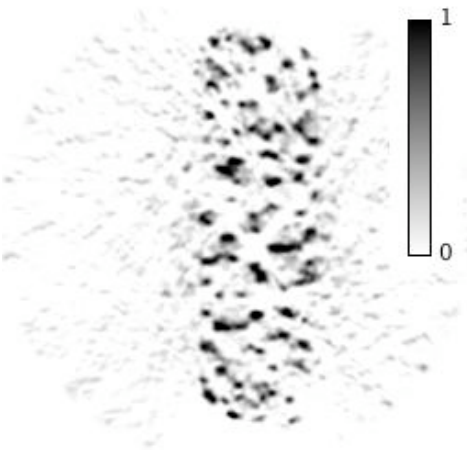
mousses de nickel 256**3 (données CEA-LIST)



(a)



(b)



(c)

Reconstructions d'une mousse 256^3 à partir de N projections sur le plan 256^2 de détecteurs : méthode directe FDK avec N=256 (a) et N=32 (b); méthode MCRQ avec N=32 (c).

Travaux réalisés dans le cadre du projet DIGITEO TOMOX

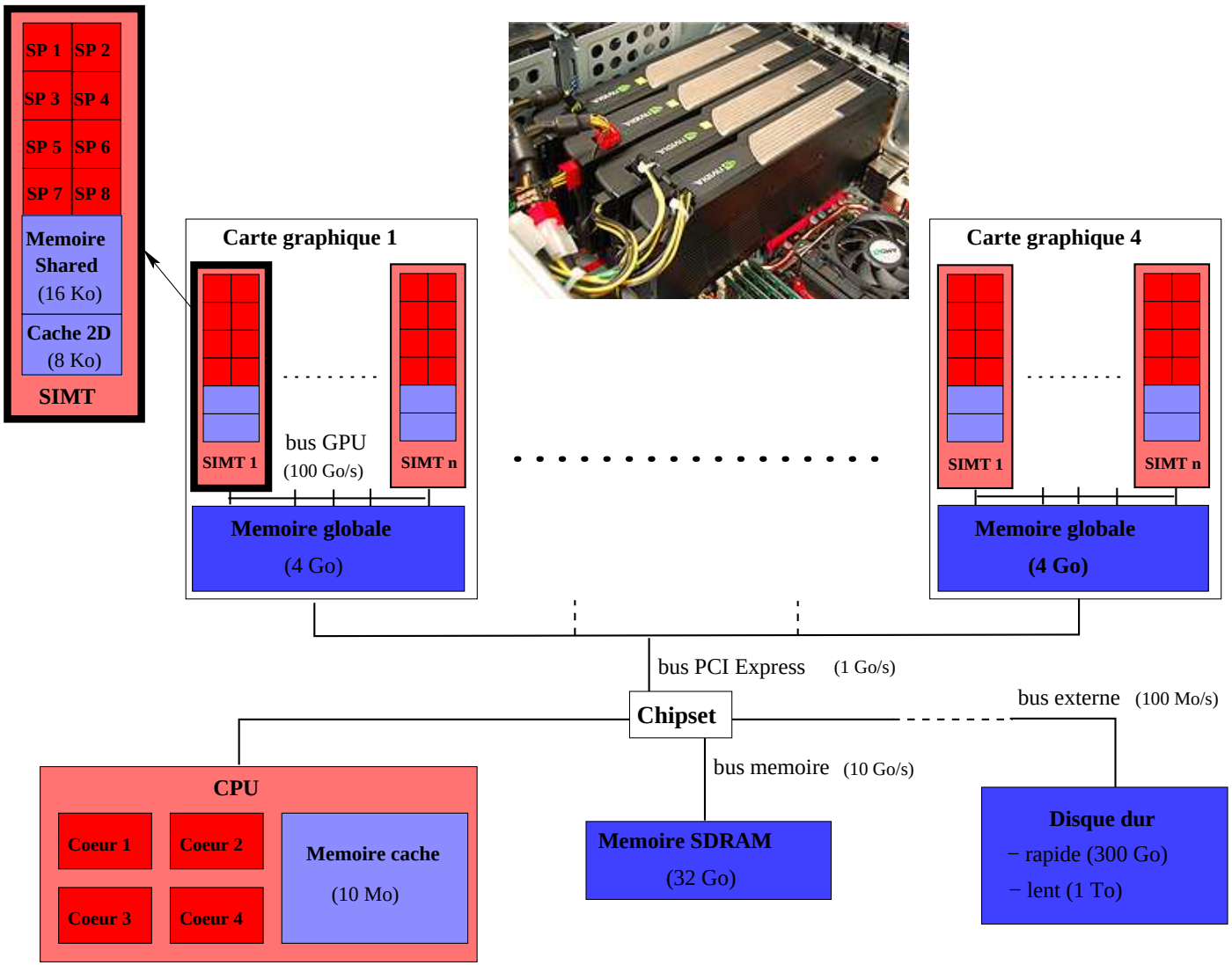
Collaboration avec le CEA-LIST (Alexandre Vabre)

Temps de reconstruction mono-GPU

Opérateurs	**Temps de calcul**	
	v1	v2
Projection $2 \times H_P$	4.1 h *(42.5 %)*	7.1 mn *(64.9 %)* $\rightarrow \times$ **35**
Rétroprojection H^t_{RP}	5.5 h *(56.9 %)*	21.8 s *(3.3 %)* $\rightarrow \times$ **908**
Convolution $3 \times D$	3.2 mn *(0.6 %)*	3.2 mn *(29.2 %)*
Autre	17 s *(0.0 %)*	17 s *(2.6 %)*
Total	9.7 h	10.9 mn $\rightarrow \times$ **53**

v1 : H_P, H^t_{RP} et D sur CPU (⚠ code "naif" non optimisé)
v2 : H_P et H^t_{RP} sur 1 GPU, D sur CPU
v3 : H_P et H^t_{RP} sur 8 GPUs, D sur CPU
v4 : H_P et H^t_{RP} sur 8 GPUs, D sur 1 GPU

Transfert mémoire PC-GPU



[Deconv1D] Tous les calculs sur le GPU

Test sur un PC avec une Tesla 2050

- 100 itérations de l'algorithme des moindres carrés régularisés.
- Traitement de 512 vecteurs de 8192 échantillons.

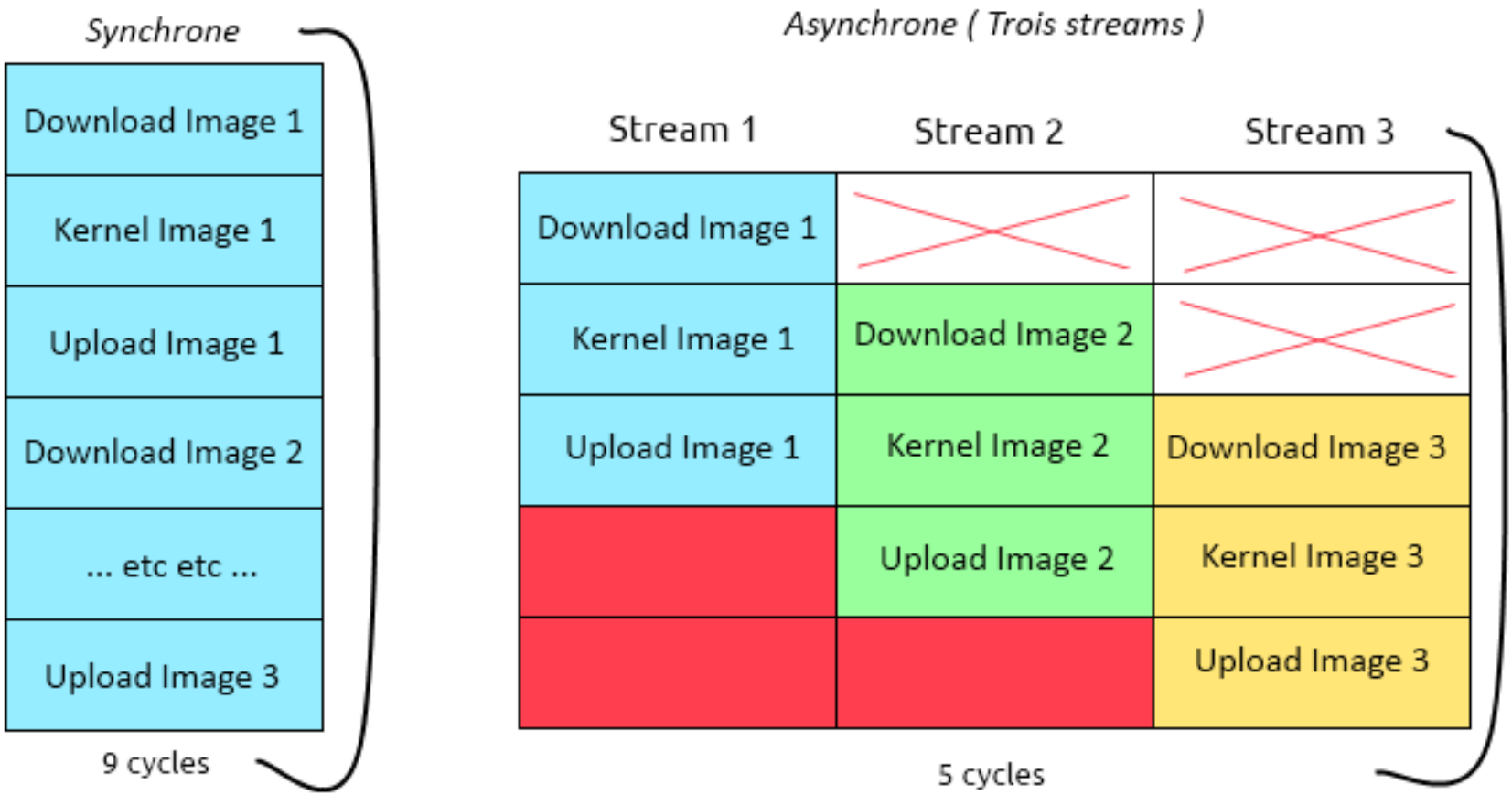
version	H, Ht	boucle	Temps	Acc.
v1	Matlab	Matlab	49,3h	
v2	C	Matlab	5,6h	***8,8**
v3	GPU	Matlab	27,3 mn	***12,3**
v4	GPU	C	20,5 mn	***1,8**
v5	GPU	GPU	11,3 mn	***1,33**

[Tomo3D] Temps de transfert mémoire PC - GPU

	1 GPU
Projecteur H_P	10 %
Rétroprojecteur H^t_{RP}	1.4 %
Convolution D	68.9 %

Proportion du temps de traitement consacré au transfert mémoire entre le PC et la carte graphique pour chaque opérateur lors de la reconstruction d'un volume de 1024^3 à partir de 256 projections.

Streams : masquage du temps de transfert PC-GPU



Différence entre synchrone et asynchrone

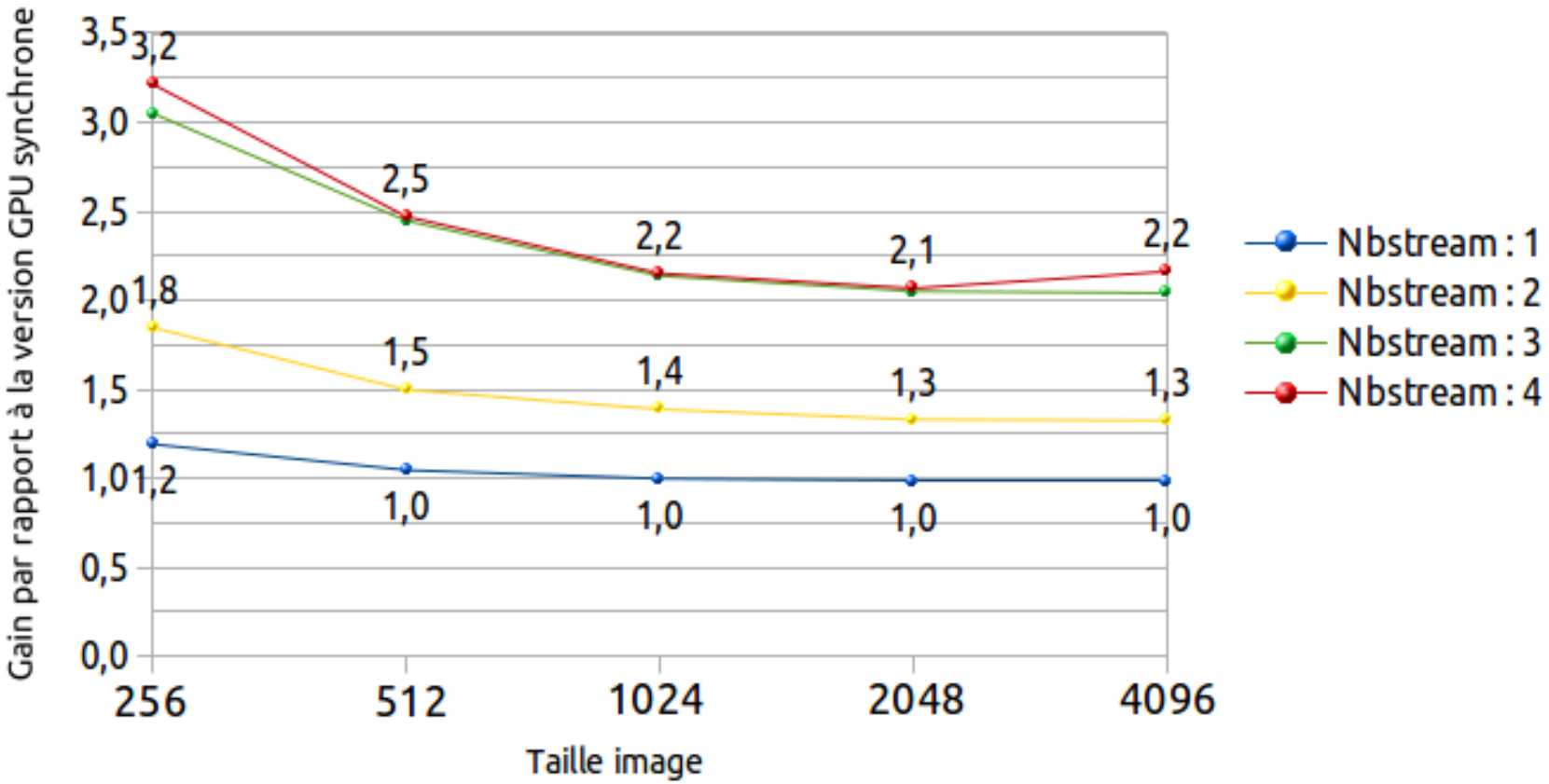
[Conv2D] Répartition temps de calcul / temps de transfert

Convolution avec un noyau 7^2 d'une séquence de 100 images 1024^2 sur une Tesla C2050

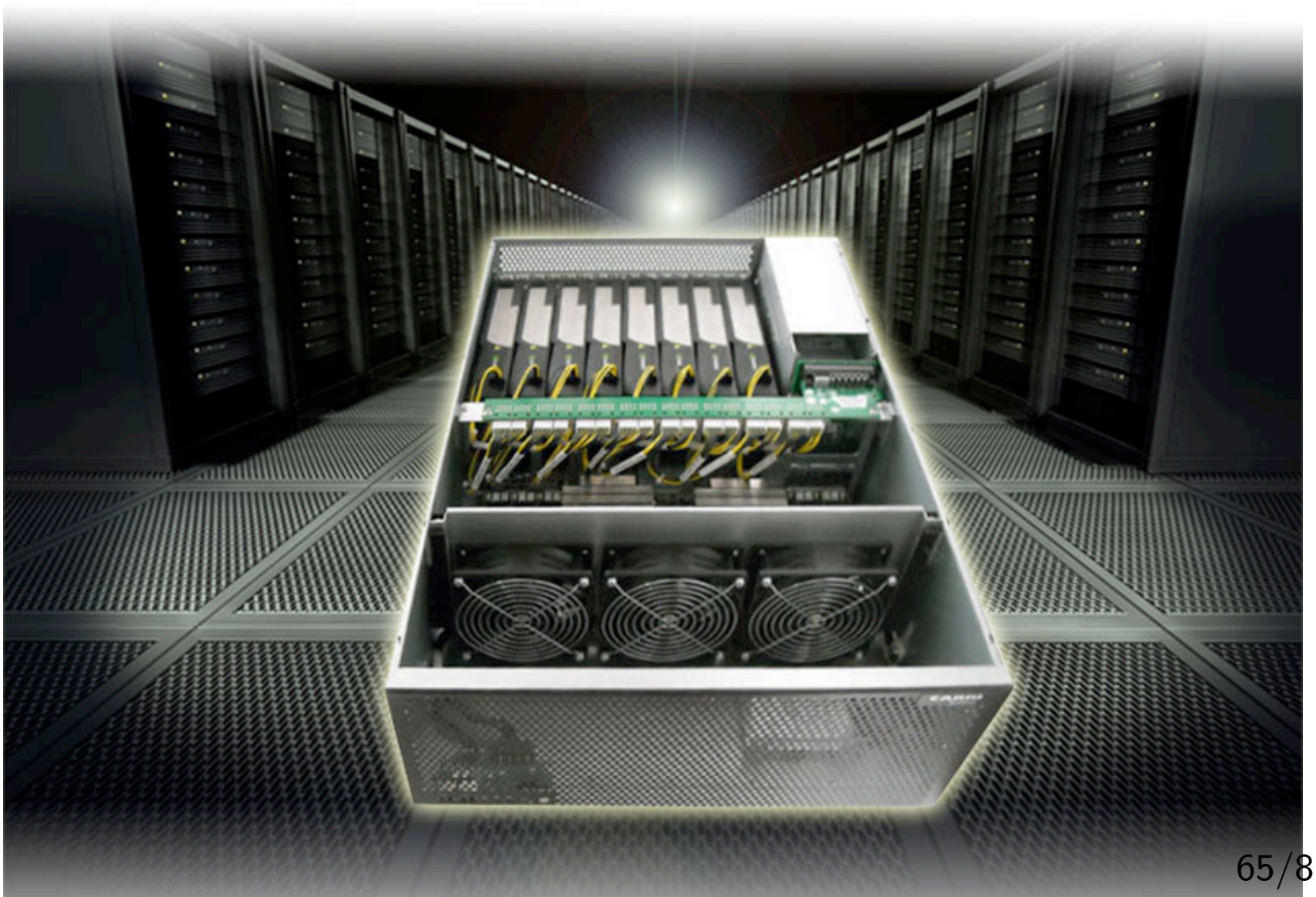
- Chargement mémoire (PC $->$ GPU) : 30 %
- Calcul du kernel CUDA (GPU) : 40 %
- Déchargement mémoire (GPU $->$ PC) : 30 %

[Conv2D] Accélération obtenue grâce aux streams (Tesla C2060)

Accélération de la convolution d 'une séquence d'images (noyau 7*7) par rapport à la version synchrone (Sans streams)



Serveur de calcul Carri - 8 Tesla C1060



[Deconv1D] Parallélisation sur un GPU selon les vecteurs (en plus des échantillons)

Test sur un PC avec une Tesla 2050

- 100 itérations de l'algorithme des moindres carrés régularisés.
- Traitement de 512 vecteurs de 8192 échantillons.

version	H, Ht	boucle	Temps	Acc.
v1	Matlab	Matlab	49,3h	
v2	C	Matlab	5,6h	***8,8**
v3	GPU	Matlab	27,3 mn	***12,3**
v4	GPU	C	20,5 mn	***1,8**
v5	GPU	GPU	11,3 mn	***1,33**
v6	GPU (vecteurs //)	GPU (vecteurs //)	8,1 mn	***1,4**

[Deconv1D] Parallélisation multi GPU selon les vecteurs

Temps extrapolé à partir de mesures faites sur un PC avec 8 Tesla 1060

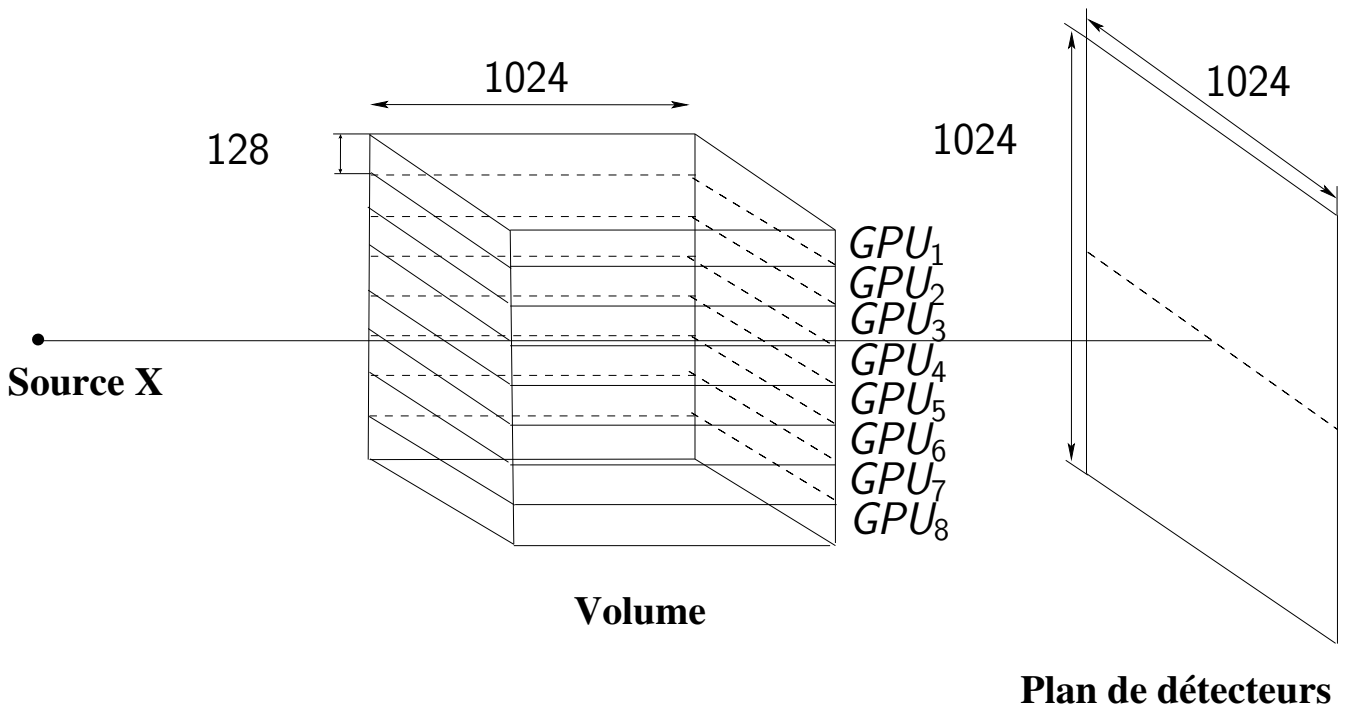
- 100 itérations de l'algorithme des moindres carrés régularisés.
- Traitement de 512 vecteurs de 8192 échantillons.

version	H, Ht	boucle	Temps	Acc.
v1	Matlab	Matlab	49,3h	
v2	C	Matlab	5,6h	***8,8**
v3	GPU	Matlab	27,3 mn	***12,3**
v4	GPU	C	20,5 mn	***1,8**
v5	GPU	GPU	11,3 mn	***1,33**
v6	GPU (vecteurs //)	GPU (vecteurs //)	8,1 mn	***1,4**
v7	8 GPUs (vecteurs //)	8 GPUs (vecteurs //)	1.0 mn	***7,9**

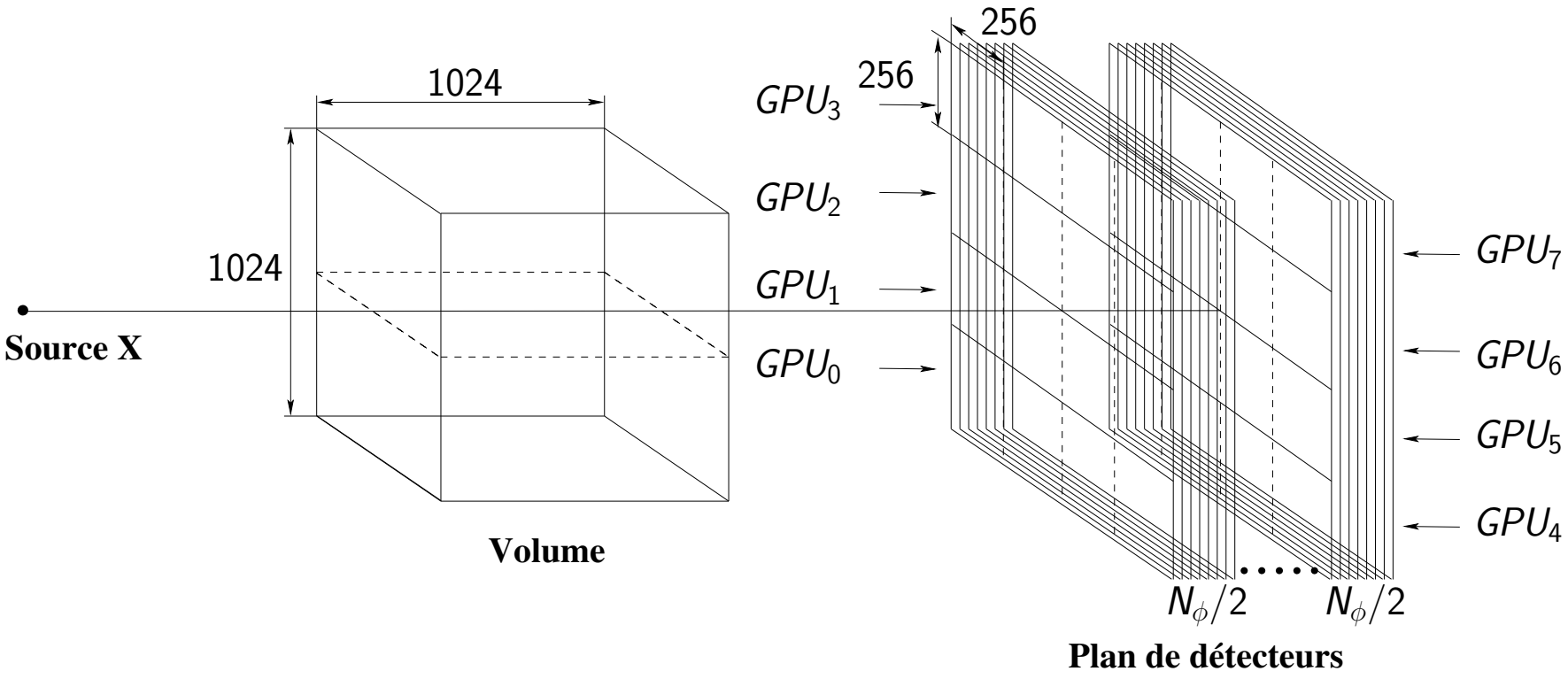
Traitement d'un milliard de spectres

- 4 ans seraient nécessaires
- Si les GPUs doublent leur puissance de calcul tous les deux ans, dans 30 ans le traitement se ferait en une heure (...mais quid de la consommation d'énergie ?)

Parallélisation multi-GPU de la rétroprojection 3D



Parallélisation multi-GPU de la projection 3D



Temps de reconstruction multi-GPUs

Opérateurs	**Temps de calcul**		
	v1	v2	v3
Projection $2 \times H_P$	4.1 h *(42.5 %)*	7.1 mn *(64.9 %)* $\rightarrow \times$ **35**	57 s *(21.1 %)* $\rightarrow \times$ **7**
Rétroprojection H^t_{RP}	5.5 h *(56.9 %)*	21.8 s *(3.3 %)* $\rightarrow \times$ **908**	4.0 s *(1.5 %)* $\rightarrow \times$ **5**
Convolution $3 \times D$	3.2 mn *(0.6 %)*	3.2 mn *(29.2 %)*	3.2 mn *(71.1 %)*
Autre	17 s *(0.0 %)*	17 s *(2.6 %)*	17 s *(6.3 %)*
Total	9.7 h	10.9 mn $\rightarrow \times$ **53**	4.5 mn $\rightarrow \times$ **2.4**

v1 : H_P, H^t_{RP} et D sur CPU (⚠ code "naif" non optimisé)
v2 : H_P et H^t_{RP} sur 1 GPU, D sur CPU
v3 : H_P et H^t_{RP} sur 8 GPUs, D sur CPU

Temps de reconstruction multi-GPUs + conv3D GPU

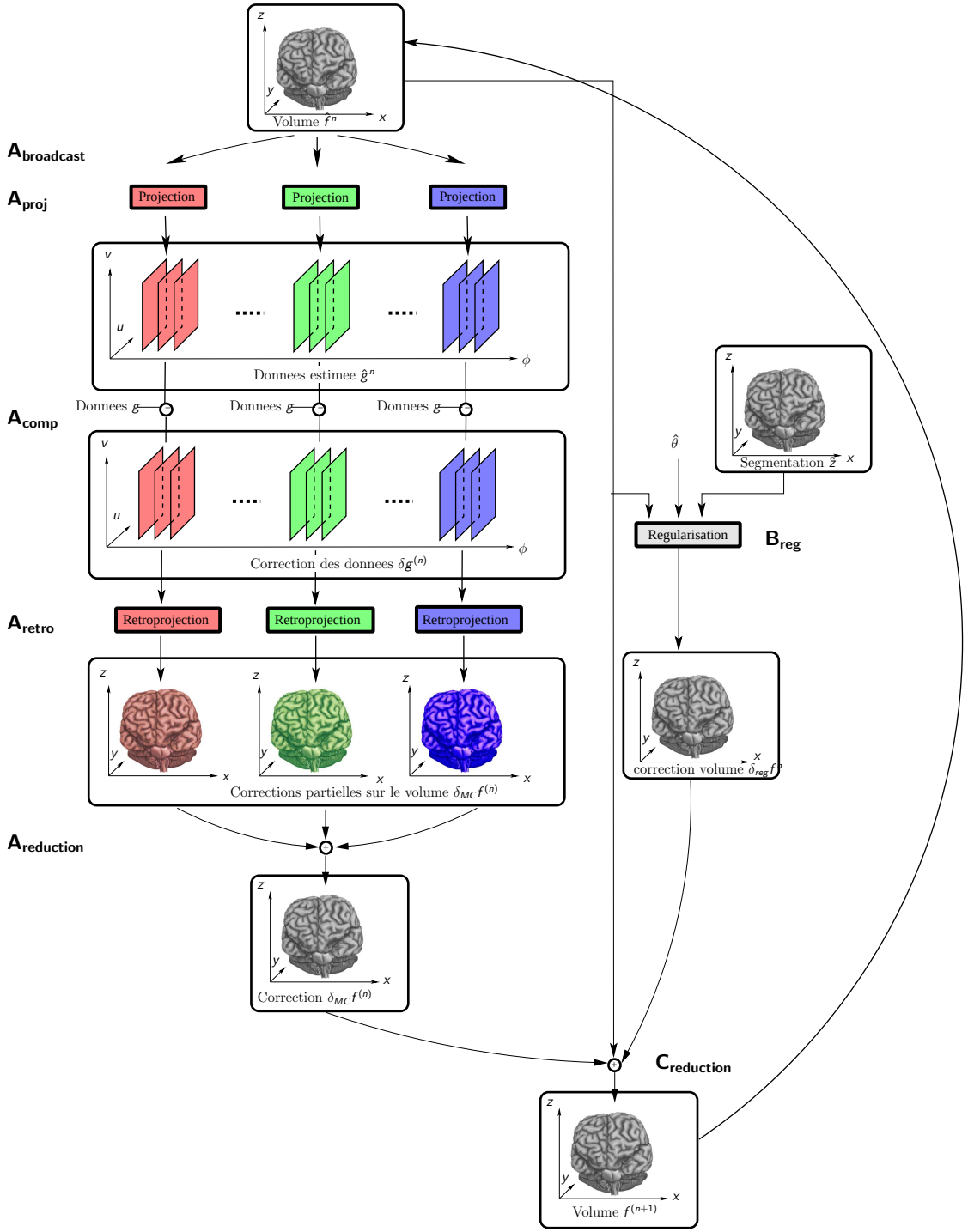
Opérateurs	**Temps de calcul**			
	v1	v2	v3	v4
Projection $2 \times H_P$	4.1 h *(42.5 %)*	7.1 mn *(64.9 %)* $\rightarrow \times$ **35**	57 s *(21.1 %)* $\rightarrow \times$ **7**	57 s *(63.3 %)*
Rétroprojection H^t_{RP}	5.5 h *(56.9 %)*	21.8 s *(3.3 %)* $\rightarrow \times$ **908**	4.0 s *(1.5 %)* $\rightarrow \times$ **5**	4.0 s *(4.4 %)*
Convolution $3 \times D$	3.2 mn *(0.6 %)*	3.2 mn *(29.2 %)*	3.2 mn *(71.1 %)*	12.1 s *(13.4 %)* $\rightarrow \times$ **16**
Autre	17 s *(0.0 %)*	17 s *(2.6 %)*	17 s *(6.3 %)*	17 s *(18.9 %)*
Total	9.7 h	10.9 mn $\rightarrow \times$ **53**	4.5 mn $\rightarrow \times$ **2.4**	1.5 mn $\rightarrow \times$ **3.0**

v1 : H_P, H^t_{RP} et D sur CPU (⚠ code "naif" non optimisé)
v2 : H_P et H^t_{RP} sur 1 GPU, D sur CPU
v3 : H_P et H^t_{RP} sur 8 GPUs, D sur CPU
v4 : H_P et H^t_{RP} sur 8 GPUs, D sur 1 GPU

Temps de transfert mémoire PC - GPU

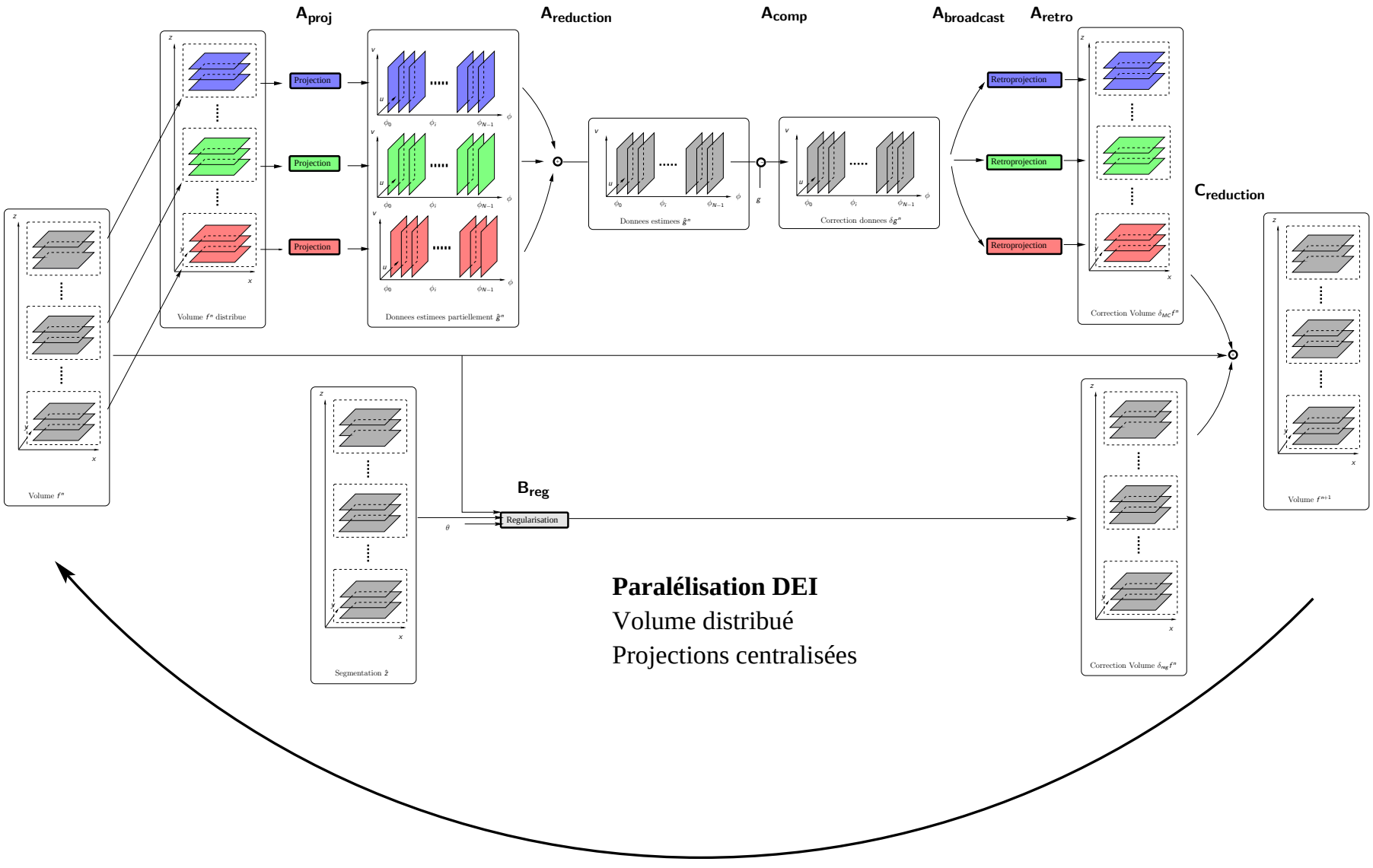
	1 GPU	**8 GPUs**
Projecteur H_P	10 %	37.5 %
Rétroprojecteur H_{RP}^t	1.4 %	6.8 %
Convolution D	68.9 %	

Proportion du temps de traitement consacré au transfert mémoire entre le PC et la carte graphique pour chaque opérateur lors de la reconstruction d'un volume de 1024^3 à partir de 256 projections.



Parallélisation DEP

Volume centralisé
Projections centralisées


A_{proj}
$A_{reduction}$
A_{comp}
$A_{broadcast}$
A_{retro}
$C_{reduction}$
B_{reg}
Projection
Projection
Projection
Retroprojection
Retroprojection
Retroprojection
Regularisation
Volume f^n distribue
Donnees estimees partiellement $\hat{g}^n$
Donnees estimees $\hat{g}^n$
Correction donnees δg^n
Correction Volume $\delta_{MC} f^n$
Volume f^n
Volume f^{n+1}
Segmentation $\hat{z}$
Correction Volume $\delta_{reg} f^n$
Paralélisation DEI
Volume distribué
Projections centralisées

Librairies

Les programmes du SDK de Nvidia (accès au code source)

- Calcul matriciel (multiplication, transpose, valeur propre...)
- Réduction somme, tri...
- Convolution 2D (seulement séparables, code assez obscure, pas paramétrisable facilement...)
- Monte Carlo (générateur de nombre aléatoire..)
- ...

Les librarires NVIDIA (pas d'accès au code source)

- CUBLAS, CUFFT, CURAND, CUSPARSE...
- NPP (Nvidia Performance Primitive)
- ...

Initiatives autres que Nvidia

- openCV GPU
- ...

Matlab et les GPUs

Les mexfunctions

- comme pour le C...

PCT : Parallel Computing Toolbox

- Fonctions Matlab parallélisées sur GPU (ex : FFT, conv2...)
- Programmation des kernels sous Matlab
- Performance versus Jacket d'AccelerEyes ?

Outils et formations/communautés GPU

Nsight : Debug, Profiling...

- Déjà existant sous Windows avec Visual studio
- Bientôt (CUDA 5.0) sous Linux avec Eclispe

formation/communauté GPU

- Ecole d'été GPU à Grenoble en été 2013 ? (Dominique Houzet/Vincent Fristot du Gipsa-lab)
- Formation continue (Dominque Houzet à Grenoble INP, Stephane Vialle à Supélec...)
- Meet-up HPC/GPU (coordinateur : Jérome Piat)
- ...

Conclusion

Pré-requis : Avoir connaissance des forces et faibles l'architecture

3 niveaux de parallélisation avec calcul distribué :

- sur les n coeurs d'un SM
- sur les p SM d'un GPU
- sur les q GPUs d'un serveur

3 niveaux d'accès mémoire

- mémoire "on chip" : registre, mémoire locale, cache (quelques Ko voire Mo)
- mémoire "off chip" SDRAM de la carte GPU (1...6 Go)
- mémoire SDRAM du PC (4...64 Go)

Conclusion

Le but : Atteindre l'équilibre entre débit de calcul et débit mémoire

❶ Exploiter au maximum la puissance de calcul

- Parallélisation de l'algorithme (bon découpage en threads)
- Utilisation de l'interpolateur *hardware* de texture

❷ Favoriser la localité spatio-temporelle des données (accès à la mémoire globale)

- Utiliser un maximum les registres
- Lecture des données via le cache 2D de texture
  - réutilisation au maximum des données
  - accès à des données 2D
- Lecture/Ecriture dans la mémoire shared
  ➲ **Harmoniser l'ordre des boucles avec la structure des données**

❸ Attention au temps de transfert CPU/GPU

- avoir un algorithme avec une la plus forte intensité arithmétique possible
- avoir un jeu de données pouvant être stocké dans la SDRAM (taille $< 4/6$ Go)
  ➲ **parallélisation multi-GPU peut impliquer des broadcast, scatter, gather...**

Merci de votre attention !

Etudiants stagiaires ayant contribués aux résultats obtenus

Développements GPU :

- Asier Rabanal (stage Erasmus/ingénieur Digiteo) : projection/rétroprojection multi-GPU
- Benoit Pencrec'h (stage IUT Cachan) : convolution 2D sur GPU
- Alexandre Frizac (stage IUT Cachan) : Transfert asynchrone PC-GPU
- Rémi Navarre (stage IUT Cachan) : Déconvolution 1D multi-GPU

Problèmes inverses :

- Krayni Anis (stage ingénieur 3A) : reconstruction tomographique
- Samsophath Nhean (stage M1) : traitement de données de Mars