Multi-GPU parallelization for 3D tomographic reconstruction

Algorithm Architecture Adequacy for solving inverse problems

Nicolas GAC (Ass. Prof. Université Paris-sud)
Groupe Problèmes Inverses (GPI)
L2S (CentraleSupélec/CNRS/Univ Paris Sud)

actuellement en délégation CNRS (6 mois) au laboratoire Lagrange
Travaux sur le projet SKA avec André Ferrari

Séminaire du laboratoire Lagrange, site Valrose, 28 novembre 2017
The beauty, reconstruction algorithms for SKA
GPU (Graphic Processing Units): hardware and software
Solving (ill-posed) inverse Problems with big dataset
[Tomo3D] Parallelization on the many cores of each GPU board
[Tomo3D] Parallelization on the GPU boards of the server

...and the beast, multi GPU servers
1. **GPU (Graphic Processing Units): hardware and software**
   - GPU (re)designed as a many core architecture
   - Programming in CUDA
   - A toy example: acceleration of matrix multiplication

2. **Solving (ill-posed) inverse Problems with big dataset**
   - Iterative (bayesian) algorithm
   - Applications

3. **[Tomo3D] Parallelization on the many cores of each GPU board**
   - Hardware acceleration of $H_f$ and $H^t$ operators
   - Projection on GPU
   - Backprojection on GPU

4. **[Tomo3D] Parallelization on the GPU boards of the server**
   - multi-GPU Parallelization
   - CUDA Streams
   - CUDA Half float
   - Distribution/Centralization of Data
1. **GPU (Graphic Processing Units): hardware and software**
   - GPU (re)designed as a many core architecture
   - Programming in CUDA
   - A toy example: acceleration of matrix multiplication

2. **Solving (ill-posed) inverse Problems with big dataset**

3. **[Tomo3D] Parallelization on the many cores of each GPU board**

4. **[Tomo3D] Parallelization on the GPU boards of the server**
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)
GPU (Graphic Processing Units) : hardware and software
Solving (ill-posed) inverse Problems with big dataset
[Tomo3D] Parallelization on the many cores of each GPU board
[Tomo3D] Parallelization on the GPU boards of the server

GPU (re)designed as a many core architecture
Programming in CUDA
A toy example : acceleration of matrix multiplication

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 Architecture (prog. en C)
GPU (Graphic Processing Units) : hardware and software
Solving (ill-posed) inverse Problems with big dataset
[Tomo3D] Parallelization on the many cores of each GPU board
[Tomo3D] Parallelization on the GPU boards of the server

GPU (re)designed as a many core architecture
Programming in CUDA
A toy example : acceleration of matrix multiplication

Avant CUDA : pipeline graphique

Vertex
Shader
Transformation géométrique

Rasterization
Polygon → Fragments

Fragment
Shader
Calcul sur les Pixels
GPU (Graphic Processing Units): hardware and software
Solving (ill-posed) inverse Problems with big dataset
[Tomo3D] Parallelization on the many cores of each GPU board
[Tomo3D] Parallelization on the GPU boards of the server

GPU (re)designed as a many core architecture
Programming in CUDA
A toy example: acceleration of matrix multiplication

Puissance de calcul
Débit mémoire

![Graph showing theoretical peak GB/s over time for different types of processors: GeForce GPU, Tesla GPU, and Intel CPU. The graph shows a significant increase in performance over the years, with the Tesla GPU and GeForce GPU showing more rapid growth compared to the Intel CPU.](image-url)
GPU (Graphic Processing Units): hardware and software
Solving (ill-posed) inverse Problems with big dataset
[Tomo3D] Parallelization on the many cores of each GPU board
[Tomo3D] Parallelization on the GPU boards of the server

A toy example: acceleration of matrix multiplication

### Découpage en threads

<table>
<thead>
<tr>
<th>Matériel</th>
<th>Logiciel</th>
<th>Exécution</th>
</tr>
</thead>
<tbody>
<tr>
<td><img src="image" alt="Diagram" /></td>
<td><strong>un thread</strong></td>
<td>séquentielle</td>
</tr>
<tr>
<td><img src="image" alt="Diagram" /></td>
<td><strong>un bloc de threads</strong></td>
<td>parallèle (SIMT)</td>
</tr>
<tr>
<td><img src="image" alt="Diagram" /></td>
<td><strong>une grille de threads</strong></td>
<td>parallèle (MIMT)</td>
</tr>
</tbody>
</table>

- **(a)**
  - un Streaming Processor (SP)
- **(b)**
  - un Streaming MultiProcessor (SM)
  - (plusieurs warps)
- **(c)**
  - une carte GPU (device)
  - (kernel)
GPU (Graphic Processing Units) : hardware and software
Solving (ill-posed) inverse Problems with big dataset
Tomo3D Parallelization on the many cores of each GPU board
Tomo3D Parallelization on the GPU boards of the server

GPU (re)designed as a many core architecture
Programming in CUDA
A toy example : acceleration of matrix multiplication

Un id par thread et un id par bloc de threads
Hiérarchie mémoire

GPU (Graphic Processing Units) : hardware and software
Solving (ill-posed) inverse Problems with big dataset
[Tomo3D] Parallelization on the many cores of each GPU board
[Tomo3D] Parallelization on the GPU boards of the server

GPU (re)designed as a many core architecture
Programming in CUDA
A toy example : acceleration of matrix multiplication
GPU (Graphic Processing Units): hardware and software
Solving (ill-posed) inverse Problems with big dataset
[Tomo3D] Parallelization on the many cores of each GPU board
[Tomo3D] Parallelization on the GPU boards of the server

GPU (re)designed as a many core architecture
Programming in CUDA
A toy example: acceleration of matrix multiplication
GPU (Graphic Processing Units) : hardware and software
Solving (ill-posed) inverse Problems with big dataset
{Tomo3D] Parallelization on the many cores of each GPU board
{Tomo3D] Parallelization on the GPU boards of the server

Supercalculateur personnel

GPU (re)designed as a many core architecture
Programming in CUDA
A toy example : acceleration of matrix multiplication
### GPU (Graphic Processing Units) : hardware and software

Solving (ill-posed) inverse Problems with big dataset

[Tomo3D] Parallelization on the many cores of each GPU board

[Tomo3D] Parallelization on the GPU boards of the server

A toy example : acceleration of matrix multiplication

---

Découpage en threads et en grilles (kernels)

<table>
<thead>
<tr>
<th>Matériel</th>
<th>Logiciel</th>
<th>Exécution</th>
</tr>
</thead>
<tbody>
<tr>
<td>un Stream Processor (SP)</td>
<td>un thread</td>
<td>séquentielle</td>
</tr>
<tr>
<td>un processeur SIMT</td>
<td>un bloc de threads</td>
<td>parallèle (SIMT)</td>
</tr>
<tr>
<td>une carte GPU (device)</td>
<td>une grille de threads (kernel)</td>
<td>parallèle (MIMD) mémoire centralisée</td>
</tr>
<tr>
<td>PC multi-carte</td>
<td>threads du PC hôte via librairie pthread (un thread CPU = un kernel GPU)</td>
<td>parallèle (MIMD) mémoire distribuée</td>
</tr>
</tbody>
</table>
GPU (Graphic Processing Units): hardware and software
Solving (ill-posed) inverse Problems with big dataset
[Tomo3D] Parallelization on the many cores of each GPU board
[Tomo3D] Parallelization on the GPU boards of the server

GPU (re)designed as a many core architecture
Programming in CUDA
A toy example: acceleration of matrix multiplication

Flot de développement logiciel

CUDA Optimized Libraries: math.h, FFT, BLAS, ...
Integrated CPU + GPU C Source Code
NVIDIA C Compiler
NVIDIA Assembly for Computing (PTX)
CPU Host Code
CUDA Driver
Standard C Compiler
Profiler
CPU
GPU
GPU (re)designed as a many core architecture
Programming in CUDA
A toy example: acceleration of matrix multiplication

Programmation GPU

1. Parallélisation de l’algorithme

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

<table>
<thead>
<tr>
<th>n coeurs (1 Ghz) vs 1 coeur (3 Ghz)</th>
</tr>
</thead>
<tbody>
<tr>
<td>taux de parallélisation</td>
</tr>
<tr>
<td>100 %</td>
</tr>
<tr>
<td>99 %</td>
</tr>
<tr>
<td>95 %</td>
</tr>
<tr>
<td>90 %</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Accélération</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
</tr>
<tr>
<td>300</td>
</tr>
<tr>
<td>150</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Nombres de coeurs</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
</tr>
<tr>
<td>0</td>
</tr>
<tr>
<td>50</td>
</tr>
<tr>
<td>code parallèle à 100%</td>
</tr>
<tr>
<td>code parallèle à 99%</td>
</tr>
<tr>
<td>code parallèle à 95%</td>
</tr>
<tr>
<td>code parallèle à 90%</td>
</tr>
</tbody>
</table>
Programming GPU

1. Parallélisation de l’algorithme

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

2. Implémentation GPU

Selon l’intensité arithmétique du code (puissance de calcul exploitée / débit des données), l’exécution sera soit memory bound soit computation bound (ex : calcul $X^k$ [?])

- optimisation du code portera alors soit sur les accès mémoire ou soit sur la complexité arithmétique
Parallélisation du calcul matriciel

\[ C = A \cdot B \]

un thread : \[ c_{ij} = \sum_k a_{ik} b_{kj} \]
GPU (Graphic Processing Units) : hardware and software
Solving (ill-posed) inverse Problems with big dataset
{Tomo3D] Parallelization on the many cores of each GPU board
{Tomo3D] Parallelization on the GPU boards of the server

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

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

  float C_sum;
  int i_first, j_first;
  int i, j;

  i_first = blockIdx.x*BLOCK_SIZE;
  j_first = blockIdx.y*BLOCK_SIZE;

  i = i_first + threadIdx.x;
  j = j_first + threadIdx.y;

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

  C[i][j] = C_sum;
}
```
Lancement du kernel depuis le PC hôte

```c
#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_kernel<<< grid, threads >>>(C_device, A_device, B_device, N);
...
}
```
### Gestion de la mémoire GPU via le PC hôte

```c
#define BLOCK_SIZE 16
void matrixMul_host(int N) {
    // allocate host memory
    int mem_size = N * N * 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_kernel<<<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

<table>
<thead>
<tr>
<th>Matrices de taille 1024·1024</th>
<th>Processeur</th>
<th>Temps d’exécution</th>
<th>Transfert mémoire</th>
</tr>
</thead>
<tbody>
<tr>
<td>C non optimisé</td>
<td>Xeon Quad core 2.7 Ghz</td>
<td>9.35 s</td>
<td></td>
</tr>
<tr>
<td>Cuda</td>
<td>Tesla C1060 240 PE @1,3 Ghz</td>
<td>1.35 s (*6,9)</td>
<td>&lt; 1%</td>
</tr>
</tbody>
</table>
Accès séquentiels à la mémoire globale

Accès non séquentiels en mémoire globale

Accès séquentiels en mémoire globale

Accès par le thread 1 :

Accès par le thread 2 :

Accès par le thread 3 :

temps
### Matrices de taille 1024 x 1024

<table>
<thead>
<tr>
<th></th>
<th>Processeur</th>
<th>Temps d’exécution</th>
<th>Transfert mémoire</th>
</tr>
</thead>
<tbody>
<tr>
<td>C</td>
<td>Xeon Quad core 2.7 Ghz</td>
<td>9.35 s</td>
<td></td>
</tr>
<tr>
<td>non optimisé</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Cuda</td>
<td>Testla C1060 240 PE @1,3 Ghz</td>
<td>1.35 s (*6,9)</td>
<td>&lt; 1%</td>
</tr>
<tr>
<td>Cuda accès seq.</td>
<td>Testla C1060 240 PE @1,3 Ghz</td>
<td>124 m s (*10,9)</td>
<td>5%</td>
</tr>
</tbody>
</table>
Optimisation des accès mémoire

\[ C = A \cdot B \]

- un bloc : \[ C_{ij} = A_i B_j \]
- un thread : \[ c_{ij} = \sum_k a_{ik} b_{kj} \]

\[ \text{en mémoire shared} \]
Optimisation des accès mémoire

\( C = A \cdot B \)

un bloc : \( C_{IJ} = A_I B_J \)

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

\( A \)

\( B \)

\( C \)

en mémoire shared
Variable type qualifiers

__device__
- en mémoire globale
- durée de vie de l’application
- accessible par tous les threads de la grille et par le hôte via la librairie runtime

__constant__
- en mémoire globale (accès via cache constante)
- durée de vie de l’application
- accessible par tous les threads de la grille et par le hôte via la librairie runtime

__shared__
- en mémoire shared (locale à un coeur SIMT)
- durée de vie du bloc de threads
- seulement accessible par les threads d’un même bloc
### Temps GPU optimisé

<table>
<thead>
<tr>
<th>Matrices de taille 1024·1024</th>
<th>Processeur</th>
<th>Temps d’exécution</th>
<th>Transfert mémoire</th>
</tr>
</thead>
<tbody>
<tr>
<td>C non optimisé</td>
<td>Xeon Quad core 2.7 Ghz</td>
<td>9.35 s</td>
<td></td>
</tr>
<tr>
<td>Cuda</td>
<td>Testla C1060 240 PE @1,3 Ghz</td>
<td>1.35 s (*6,9)</td>
<td>&lt; 1%</td>
</tr>
<tr>
<td>Cuda acces seq.</td>
<td>Testla C1060 240 PE @1,3 Ghz</td>
<td>124 m s (*10,9)</td>
<td>5%</td>
</tr>
<tr>
<td>Cuda shared mem.</td>
<td>Testla C1060 240 PE @1,3 Ghz</td>
<td>17,5 m s (*7,1)</td>
<td>34%</td>
</tr>
</tbody>
</table>
Librairie CUBLAS : CUda Basic Linear Algebra Subprograms

```c
#include <cublas.h>
#include <cutil.h>

int main(void) {
    float alpha = 1.0f, beta = 0.0f;
    int N = 1024;
    int mem_size = 1024*1024*sizeof(float);

    // Allocate host memory
    float* A_host = (float*) malloc(mem_size);
    float* B_host = (float*) malloc(mem_size);
    float* C_host = (float*) malloc(mem_size);

    cublasInit();

    // Allocate device memory
    float* A_device,B_device,C_device;
    cublasAlloc(N*N, sizeof(float), (void **)&A_device);
    cublasAlloc(N*N, sizeof(float), (void **)&B_device);
    cublasAlloc(N*N, sizeof(float), (void **)&C_device);

    // copy host memory to device
    cublasSetMatrix(N,N, sizeof(float), A_host, N, A_device, N);
    cublasSetMatrix(N,N, sizeof(float), B_host, N, B_device, N);

    // Calcul matriciel sur le GPU
    cublasSgemm('n', 'n', N, N, N, alpha, A_device, N,B_device, N, beta, C_device, N);

    // Récupération du résultat sur le PC hôte
    cublasGetMatrix(N,N, sizeof(float), C_device,N, C_host, N);
}
```
### GPU (Graphic Processing Units) : hardware and software

Solving (ill-posed) inverse Problems with big dataset

[Tomo3D] Parallelization on the many cores of each GPU board

[Tomo3D] Parallelization on the GPU boards of the server

#### A toy example : acceleration of matrix multiplication

<table>
<thead>
<tr>
<th>Matrices de taille 1024·1024</th>
<th>Processeur</th>
<th>Temps d’exécution</th>
<th>Transfert mémoire</th>
</tr>
</thead>
<tbody>
<tr>
<td>C non optimisé</td>
<td>Xeon Quad core 2.7 Ghz</td>
<td>9.35 s</td>
<td></td>
</tr>
<tr>
<td>Cuda</td>
<td>Testla C1060 240 PE @1,3 Ghz</td>
<td>1.35 s (*6,9)</td>
<td>&lt; 1%</td>
</tr>
<tr>
<td>Cuda acces seq.</td>
<td>Testla C1060 240 PE @1,3 Ghz</td>
<td>124 m s (*10,9)</td>
<td>5%</td>
</tr>
<tr>
<td>Cuda shared mem.</td>
<td>Testla C1060 240 PE @1,3 Ghz</td>
<td>17,5 m s (*7,1)</td>
<td>34%</td>
</tr>
<tr>
<td>CUBLAS</td>
<td>Testla C1060 240 PE @1,3 Ghz</td>
<td>12,8 m s (*1,4)</td>
<td>43%</td>
</tr>
</tbody>
</table>
1. GPU (Graphic Processing Units): hardware and software

2. Solving (ill-posed) inverse Problems with big dataset
   - Iterative (bayesian) algorithm
   - Applications

3. [Tomo3D] Parallelization on the many cores of each GPU board

4. [Tomo3D] Parallelization on the GPU boards of the server
Without bayesian regularisation

\[ g = Hf + \epsilon \]

\( f \): volume
\( g \): tomograph data
\( H \): acquisition model
\( \epsilon \): noise

Criterion: Mean Square

\[ J(f) = ||g - Hf||^2 \]

\[ f^{n+1} = f^n - \alpha \cdot \nabla J(f^n) \]

\[ \nabla J(f) = -2 \cdot H^t(g - Hf) \]
Without bayesian regularisation

\[ f^n : \text{Estimée du volume} \]

\[ g = Hf + \epsilon \]

- **g**: tomograph data
- **f**: volume
- **\( H \)**: acquisition model
- **\( \epsilon \)**: noise

**Criterion : Mean Square**

\[ J(f) = \| g - Hf \|^2 \]
\[ f^{n+1} = f^n - \alpha \cdot \nabla J(f^n) \]
\[ \nabla J(f) = -2 \cdot H^t(g - Hf) \]
Without bayesian regularisation

\[ \hat{g} : \text{Estimée des données} \]

**Descente de gradient**

\[ f^n + 1 = f^n - \alpha \cdot \nabla J(f^n) \]

---

**Criterion : Mean Square**

\[ J(f) = \|g - Hf\|^2 \]

\[ f^{n+1} = f^n - \alpha \cdot \nabla J(f^n) \]

\[ \nabla J(f) = -2 \cdot H^t(g - Hf) \]
Without bayesian regularisation

\[ \delta g : \text{Correction des données} \]

\[ g = Hf + \epsilon \]

\[ f : \text{volume} \]
\[ g : \text{tomograph data} \]
\[ H : \text{acquisition model} \]
\[ \epsilon : \text{noise} \]

**Criterion : Mean Square**

\[ J(f) = \| g - Hf \|^2 \]
\[ f^{n+1} = f^n - \alpha \cdot \nabla J(f^n) \]
\[ \nabla J(f) = -2 \cdot H^t (g - Hf) \]
Without bayesian regularisation

\[ g = Hf + \epsilon \]

- \( f \): volume
- \( g \): tomograph data
- \( H \): acquisition model
- \( \epsilon \): noise

\[ \delta f : \text{Correction du volume} \]

\[ J(f) = \| g - Hf \|^2 \]

\[ f^{n+1} = f^n - \alpha \cdot \nabla J(f^n) \]

\[ \nabla J(f) = -2 \cdot H^t(g - Hf) \]
Solving (ill-posed) inverse Problems with big dataset

[Tomo3D] Parallelization on the many cores of each GPU board
[Tomo3D] Parallelization on the GPU boards of the server

Without Bayesian regularisation

\[ f^{n+1} : \text{Nouvelle estimée du volume} \]

\[ g = Hf + \epsilon \]

- \( g \) : tomograph data
- \( f \) : volume
- \( H \) : acquisition model
- \( \epsilon \) : noise

**Criterion : Mean Square**

\[ J(f) = \| g - Hf \|^2 \]

\[ f^{n+1} = f^n - \alpha \cdot \nabla J(f^n) \]

\[ \nabla J(f) = -2 \cdot H^t(g - Hf) \]
GPU (Graphic Processing Units): hardware and software
Solving (ill-posed) inverse Problems with big dataset
[Tomo3D] Parallelization on the many cores of each GPU board
[Tomo3D] Parallelization on the GPU boards of the server

With bayesian regularisation

\[ g = Hf + \epsilon \]
\[ f : \text{volume} \]
\[ g : \text{tomograph data} \]
\[ H : \text{acquisition model} \]
\[ \epsilon : \text{noise} \]

Criterion: Mean Square + Quadratic Regularisation (MSQR)
\[ J(f) = J_1(f) + J_2(f) \]
\[ J_1(f) = \|g - Hf\|^2 \]
\[ J_2(f) = \lambda \|Df\|^2 \]
\[ f^{n+1} = f^n - \alpha \cdot (\nabla J_1(f^n) + \nabla J_2(f^n)) \]
GPU (Graphic Processing Units) : hardware and software
Solving (ill-posed) inverse Problems with big dataset
[Tomo3D] Parallelization on the many cores of each GPU board
[Tomo3D] Parallelization on the GPU boards of the server

Iterative algorithm : Mean square + quadratic reg
Applications

[Planéto] Correction de vibrations mécaniques

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

Instrument PFS (Planetary Fourier Spectrum) de la mission MARS EXPRESS
**GPU (Graphic Processing Units) : hardware and software**

Solving (ill-posed) inverse Problems with big dataset

[Tomo3D] Parallelization on the many cores of each GPU board

[Tomo3D] Parallelization on the GPU boards of the server

**Applications**

**[Planéto] Correction de vibrations mécaniques**

**Instrument modélisé par une convolution 1D**

\[
x \text{ (spectre réel)} \times h \text{ (instrument PFS)} = y \text{ (spectre mesuré)}
\]

Taille gigantesque des données

Des années d’enregistrements de la mission MARS EXPRESS (2003) donc potentiellement 1 milliard de spectres (de 8192 échantillons) !
GPU (Graphic Processing Units): hardware and software
Solving (ill-posed) inverse Problems with big dataset
[Tomo3D] Parallelization on the many cores of each GPU board
[Tomo3D] Parallelization on the GPU boards of the server

Iterative algorithm: Mean square + quadratic reg

Applications

[Astro] Méthode de reconstruction en astronomie
GPU (Graphic Processing Units) : hardware and software
Solving (ill-posed) inverse Problems with big dataset
[Tomo3D] Parallelization on the many cores of each GPU board
[Tomo3D] Parallelization on the GPU boards of the server
Iterative algorithm : Mean square + quadratic reg
Applications

[Tomo3D] Algorithmes de reconstruction tomographique

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

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

② RECONSTRUCTION
- Algorithme itératif
- Accélération sur GPU

Objet à imager

Tomographe

GPU
$1K^3$ volume from 1K projections with $1K^2$ pixels (SAFRAN data set)

Work done in collaboration with SAFRAN (Post-doc Thomas Boulay)
1. GPU (Graphic Processing Units): hardware and software

2. Solving (ill-posed) inverse Problems with big dataset

3. [Tomo3D] Parallelization on the many cores of each GPU board
   - Hardware acceleration of $Hf$ and $H^t$ operators
   - Projection on GPU
   - Backprojection on GPU

4. [Tomo3D] Parallelization on the GPU boards of the server
Hf and $H^t \delta g$ computation

1. **Matrix multiplication**

- reading $h_{ij}$ coefficients in SDRAM memory
- volume $2048^3 \rightarrow$ matrix $H = 1$ Exa Bytes!
**Hf and H^t \delta g computation**

1. **Matrix multiplication**
   - reading $h_{ij}$ coefficients in SDRAM memory
   - \text{volume} $2048^3 \rightarrow \text{matrix } H = 1 \text{ Exa Bytes}!

2. **Geometric operators**
   - on line computation of $h_{ij}$ coefficients

---

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

- \text{déTECTEURS du tomographe}
- \text{Objet imagé}
- \text{projection}
- \text{Objet reconstruit}
- \text{réTROprojection}

\text{Projection on GPU}
\text{Backprojection on GPU}
GPU (Graphic Processing Units) : hardware and software
Solving (ill-posed) inverse Problems with big dataset
[Tomo3D] Parallelization on the many cores of each GPU board
[Tomo3D] Parallelization on the GPU boards of the server

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)
GPU (Graphic Processing Units) : hardware and software solving (ill-posed) inverse problems with big dataset.

[Tomo3D] Parallelization on the many cores of each GPU board.
[Tomo3D] Parallelization on the GPU boards of the server.

Thesis conclusions:

- Hardware acceleration of $H_f$ and $H_t$ operators.
- Projection on GPU.
- Backprojection on GPU.

CPU/GPU/FPGA comparison:

<table>
<thead>
<tr>
<th></th>
<th>CPU</th>
<th>GPU</th>
<th>FPGA</th>
</tr>
</thead>
<tbody>
<tr>
<td>Time</td>
<td>3ème (*4 P4)</td>
<td>1ère (*50 P4)</td>
<td>2ème (*5 P4)</td>
</tr>
<tr>
<td>Efficacy</td>
<td>2ème (7 C/op)</td>
<td>1ème (14 C/Op)</td>
<td>1ère (2 C/Op)</td>
</tr>
</tbody>
</table>

- GPU is the hardware accelerator the most performant.
- FPGA is the hardware accelerator the most efficient in term of cycles/op (thanks to our cache 3D).
GPU quickly adopted by the tomography community

Publications in Fully 3D

- 2007 : 1st Workshop HPIR (High Performance Image Reconstruction)
- 2011 : Keyword Multi GPU first appeared

<table>
<thead>
<tr>
<th>Hardware</th>
<th>2003</th>
<th>2005</th>
<th>2007</th>
<th>2009</th>
<th>2011</th>
<th>2013</th>
<th>2015</th>
<th>2017</th>
</tr>
</thead>
<tbody>
<tr>
<td>Cluster (MPI/Open MP)</td>
<td>2</td>
<td>3</td>
<td>5</td>
<td>6</td>
<td>3</td>
<td>2</td>
<td>1</td>
<td>2</td>
</tr>
<tr>
<td>GPU (NVIDIA)</td>
<td>10</td>
<td>14</td>
<td>17</td>
<td>7</td>
<td>10</td>
<td>20</td>
<td></td>
<td></td>
</tr>
<tr>
<td>GPU (AMD)</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Cell (IBM)</td>
<td>3</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>FPGA</td>
<td>4</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>1</td>
<td></td>
<td></td>
</tr>
<tr>
<td>DSP</td>
<td>2</td>
<td>1</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Intel (Larabee, Xeon phi)</td>
<td>2</td>
<td></td>
<td>2</td>
<td>2</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
2D projector “regular sampling”

```plaintext
for (un, phi) in Projection do
    for xn = 0 to xn_{max} - 1 do
        // coordinates computation
        yn(xn, un, phi) = ...
        // bi-linear interpolation
        f_{interp} = ...
        // accumulation
        g^*(un, phi) + = f_{interp}
    end for
end for
```
2D backprojection : algorithm

```plaintext
for (xn, yn) in Volume do
    for phi = 0 to phi_{max} - 1 do
        // coordinates computation
        u(\phi, xn, yn) = ...
        // accumulation
        f^*(xn, yn) + = g(u, \phi)
    end for
end for
```
2D backprojection : linear interpolation

\begin{align*}
\text{for } (xn, yn) \text{ in Volume do} \\
\quad \text{for } \phi = 0 \text{ to } \phi_{\text{max}} - 1 \text{ do} \\
\qquad \text{// coordinates computation} \\
\qquad u(\phi, xn, yn) = \ldots \\
\qquad \text{// linear interpolation} \\
\qquad g_{\text{interp}} = (1 - \epsilon_u) \cdot g(\phi, u_e) + \\
\qquad \qquad \epsilon_u \cdot g(\phi, u_e + 1) \\
\qquad \text{// accumulation} \\
\qquad f^*(xn, yn) + = g_{\text{interp}} \\
\text{end for} \\
\text{end for}
\end{align*}
2D backprojection: scattered data access

\[
\text{for } (x_n, y_n) \text{ in Volume do}
\]

\[
\text{for } \phi_i = 0 \text{ to } \phi_{\text{max}} - 1 \text{ do}
\]

// coordinates computation
\[
u(\phi, x_n, y_n) = \ldots
\]

// linear Interpolation
\[
g_{\text{interp}} = (1 - \epsilon_u) \cdot g(\phi_i, u_e) + \\
\epsilon_u \cdot g(\phi_i, u_e + 1)
\]

// accumulation
\[
f^*(x_n, y_n) + = g_{\text{interp}}
\]

end for

end for
2D backprojection: scattered data access

for \((x_n, y_n)\) in Volume do
  for \(\phi = 0\) to \(\phi_{\text{max}} - 1\) do
    // coordinates computation
    \(u(\phi, x_n, y_n) = \ldots\)
    // linear interpolation
    \(g_{\text{interp}} = (1 - \epsilon_u) \cdot g(\phi, u_e) + \epsilon_u \cdot g(\phi, u_e + 1)\)
    // accumulation
    \(f^*(x_n, y_n) + = g_{\text{interp}}\)
  end for
end for
2D backprojection : scattered data access

\[
\text{for } (x_n, y_n) \text{ in Volume do}
\]
\[
\text{for } \phi = 0 \text{ to } \phi_{\text{max}} - 1 \text{ do}
\]
// coordinates computation
\[
u(\phi, x_n, y_n) = \ldots
\]
// linear interpolation
\[
g_{\text{interp}} = (1 - \epsilon_u) \cdot g(\phi, u_e) + \epsilon_u \cdot g(\phi, u_e + 1)
\]
// accumulation
\[
f^*(x_n, y_n) + = g_{\text{interp}}
\]
end for
end for
2D backprojection by blocks: localized data access

\[
\text{for } (Bx, By) \text{ in Volume do}
\]
\[
\text{for } \phi_i = 0 \text{ to } \phi_{\text{max}} - 1 \text{ do}
\]
\[
\text{for } (xn, yn) \text{ in Bloc do}
\]
\[
// \text{ coordinates computation}
\]
\[
u(\phi_i, xn, yn) = \ldots
\]
\[
// \text{ linear interpolation}
\]
\[
g_{\text{interp}} = (1 - \epsilon_u) \cdot g(\phi_i, u_e) + \epsilon_u \cdot g(\phi_i, u_e + 1)
\]
\[
// \text{ accumulation}
\]
\[
f^*(xn, yn) + = g_{\text{interp}}
\]
\[
\text{end for}
\]
\[
\text{end for}
\]
\[
\text{end for}
\]
3D backprojection parallelization

(a) Sequential computation on processor element
- Loop on $z$
- Loop on $\phi$

(b) Parallel computation on a block of processors (SIMT)
- Loop on $(x,y)$

(c) Parallel computation on one card
- Loop on blocks $(Bx,By,Bz)$
3D backprojection parallelization

(a) Sequential computation on processor element
- Loop on \( z \)
- Loop on \( \phi \)

(b) Parallel computation on a block of processors (SIMT)
- Loop on \((x,y)\)

(c) Parallel computation on one card
- Loop on blocks \((B_x,B_y,B_z)\)
3D backprojection parallelization

(a) Sequential computation on processor element
- Loop on $z$
- Loop on $\phi$

(b) Parallel computation on a block of processors (SIMT)
- Loop on $(x, y)$

(c) Parallel computation on one card
- Loop on blocks $(Bx, By, Bz)$
3D backprojection parallelization

(a) Sequential computation on processor element
- Loop on z
- Loop on $\phi$

(b) Parallel computation on a block of processors (SIMT)
- Loop on $(x,y)$

(c) Parallel computation on one card
- Loop on blocks $(B_x,B_y,B_z)$
GPU (Graphic Processing Units): hardware and software
Solving (ill-posed) inverse Problems with big dataset
[Tomo3D] Parallelization on the many cores of each GPU board
[Tomo3D] Parallelization on the GPU boards of the server
Multi GPUs server (Carri Systems)
3D backprojection multi GPU parallelization
3D projection multi-GPU parallelization

Source X

Volume

GPU

1024

1024

256

256

N_\phi/2

N_\phi/2
Multi-GPU reconstruction time

Volume $1K^3$ (float) with 1024 projections on 1 to 8 Titans X (3072 cores at 1,075 Ghz)

<table>
<thead>
<tr>
<th></th>
<th>1 GPU</th>
<th>2 GPUs</th>
<th>4 GPUs</th>
<th>8 GPUs</th>
</tr>
</thead>
<tbody>
<tr>
<td>Proj (ms)</td>
<td>14416</td>
<td>8183 1,76</td>
<td>4610 3,13</td>
<td>2659 5,42</td>
</tr>
<tr>
<td>Back (ms)</td>
<td>7604</td>
<td>5181 1,47</td>
<td>3027 2,51</td>
<td>1929 3,94</td>
</tr>
<tr>
<td>Conv (ms)</td>
<td>3062</td>
<td>2987 1,02</td>
<td>2438 1,26</td>
<td>1668 1,84</td>
</tr>
</tbody>
</table>
GPU (Graphic Processing Units): hardware and software
Solving (ill-posed) inverse Problems with big dataset
[Tomo3D] Parallelization on the many cores of each GPU board
[Tomo3D] Parallelization on the GPU boards of the server

**Goal of streams:** hide PC/GPU memory transfer

---

**Différence entre synchrone et asynchrone**

[Diagram showing the difference between synchronous and asynchronous streams]
CUDA streams for mono GPU backprojection (1024 angles $1024^2$ plan)

1 stream

- [0] GeForce GTX TITAN X
- Context 1 (CUDA)
  - MemCpy (HtoD)
  - MemCpy (DtoH)
  - Compute
  - Streams
    - Default
    - Stream 13
    - Stream 14

GPU (Graphic Processing Units): hardware and software
Solving (ill-posed) inverse Problems with big dataset
[Tomo3D] Parallelization on the many cores of each GPU board
[Tomo3D] Parallelization on the GPU boards of the server
multi-GPU Parallelization
CUDA Streams
CUDA Half float
Distribution/Centralization of Data
CUDA streams for mono GPU backprojection (1024 angles $1024^2$ plan)
single GPU time with streams

$1K^3$ Volume (float) with 1024 projections on 1 Titan X (3072 cores at 1,075 Ghz)

<table>
<thead>
<tr>
<th></th>
<th>compute</th>
<th>upload</th>
<th>download</th>
<th>w/o stream</th>
<th>w/ streams</th>
<th>Acc.</th>
</tr>
</thead>
<tbody>
<tr>
<td>Proj (ms)</td>
<td>88 %</td>
<td>6 %</td>
<td>6 %</td>
<td>14416</td>
<td>11551</td>
<td>1,25</td>
</tr>
<tr>
<td>Rétro (ms)</td>
<td>71,1 %</td>
<td>16,9 %</td>
<td>12,1 %</td>
<td>7604</td>
<td>5358</td>
<td>1,42</td>
</tr>
<tr>
<td>Conv (ms)</td>
<td>5 %</td>
<td>28,1 %</td>
<td>66,9 %</td>
<td>3062</td>
<td>3072</td>
<td>0,99</td>
</tr>
</tbody>
</table>
GPU (Graphic Processing Units): hardware and software
Solving (ill-posed) inverse Problems with big dataset
[Tomo3D] Parallelization on the many cores of each GPU board
[Tomo3D] Parallelization on the GPU boards of the server

multi-GPU Parallelization
CUDA Streams
CUDA Half float
Distribution/Centralization of Data

multi-GPU time with streams

<table>
<thead>
<tr>
<th></th>
<th>1 GPU</th>
<th>2 GPUs</th>
<th>4 GPUs</th>
<th>8 GPUs</th>
</tr>
</thead>
<tbody>
<tr>
<td>Proj (ms) w/o streams</td>
<td>14416</td>
<td>8183 1.76</td>
<td>4610 3.13</td>
<td>2659 5.42</td>
</tr>
<tr>
<td>Proj (ms) w/ streams</td>
<td>11551</td>
<td>5783 2.0</td>
<td>3142 3.68</td>
<td>1756 6.58</td>
</tr>
<tr>
<td>Back (ms) w/o streams</td>
<td>7604</td>
<td>5181 1.47</td>
<td>3027 2.51</td>
<td>1929 3.94</td>
</tr>
<tr>
<td>Back (ms) w/ streams</td>
<td>5358</td>
<td>2609 2.0</td>
<td>1672 3.20</td>
<td>1731 3.10</td>
</tr>
<tr>
<td>Conv (ms) w/o streams</td>
<td>3062</td>
<td>2987 1.02</td>
<td>2438 1.26</td>
<td>1668 1.84</td>
</tr>
<tr>
<td>Conv (ms) w/ streams</td>
<td>3072</td>
<td>2482 1.24</td>
<td>2340 1.31</td>
<td>1674 1.83</td>
</tr>
</tbody>
</table>

Limitations due to PCI express gen2 bandwith (2 to 4 GB/s)
Half float data storage

CUDA 7.5 allows half float storage of data on GPU memory

- 16 bits format: sign (1bit), exponent (5bits), mantissa (10bits)
- Assembler instructions allow the conversion half/float and float/half in CUDA kernels
- Advantage (i): reduction of data volume to store on the GPU board
- Advantage (ii): reduction of memory transfer
- Advantage (iii): reduction of SDRAM GPU memory access by the GPU cores
single GPU time with streams and half-float storage

<table>
<thead>
<tr>
<th></th>
<th>float</th>
<th>half float</th>
<th>Acc.</th>
</tr>
</thead>
<tbody>
<tr>
<td>Proj (ms)</td>
<td>11551</td>
<td>8970</td>
<td>1.29</td>
</tr>
<tr>
<td>Back (ms)</td>
<td>5358</td>
<td>4252</td>
<td>1.26</td>
</tr>
<tr>
<td>Conv (ms)</td>
<td>3072</td>
<td>1608</td>
<td>1.91</td>
</tr>
</tbody>
</table>

Additional acceleration with half float storage for projection and backprojection

→ Reduction of SDRAM GPU memory access time by the GPU cores
multi-GPU Time with streams and half-float storage

<table>
<thead>
<tr>
<th></th>
<th>1 GPU</th>
<th>2 GPUs</th>
<th>4 GPUs</th>
<th>8 GPUs</th>
</tr>
</thead>
<tbody>
<tr>
<td>Proj (ms) f</td>
<td>11551</td>
<td>5783 2,0</td>
<td>3142 3,68</td>
<td>1756 6,58</td>
</tr>
<tr>
<td>Proj (ms) hf</td>
<td>8970</td>
<td>4620 1,94</td>
<td>2357 3,80</td>
<td>1265 7,09</td>
</tr>
<tr>
<td>Back (ms) f</td>
<td>5358</td>
<td>2609 2,0</td>
<td>1672 3,20</td>
<td>1731 3,10</td>
</tr>
<tr>
<td>Back (ms) hf</td>
<td>4252</td>
<td>2164 1,96</td>
<td>1229 3,46</td>
<td>876 4,83</td>
</tr>
<tr>
<td>Conv (ms) f</td>
<td>3072</td>
<td>2482 1,24</td>
<td>2340 1,31</td>
<td>1674 1,83</td>
</tr>
<tr>
<td>Conv (ms) hf</td>
<td>1608</td>
<td>1267 1,27</td>
<td>1171 1,37</td>
<td>843 1,91</td>
</tr>
</tbody>
</table>

Limitations due to PCI express gen2 bandwidth (2 to 4 GB/s)
## Data storage during the iterative loop

### CPU centralisation

All the data ($f^n$ and $f^{n+1}$ volume, real $g$ and estimated $\hat{g}$ sinograms...) could not stay on the GPU board (true from $1K^3$ volumes)

**Because of the cone beam geometry, data could not easily cut in independant block of data**

$\rightarrow$ Data need to be backed up on the CPU at least one time after each iteration

### (single)GPU centralization

All the data ($f^n$ and $f^{n+1}$ volume, real $g$ and estimated $\hat{g}$ sinograms...) could stay on the GPU board (true up to $512^3$ volumes)

$\rightarrow$ All the iterative loop could be done on the GPU

### (multi)GPU centralisation

All the data (n and n+1 volume, real and estimate sinograms...) could be distributed on the different GPU boards (true up to $2K^3$ volumes)

$\rightarrow$ All the iterative loop could be done without data storage on the CPU
CPU centralization

Current strategy: result of each operator (proj, back, conv) is backed up on the CPU

- Advantage: operators (proj, back, conv) are independants (usefull for utilization with Matlab and mex function)
- Disadvantage: several synchronizations CPU/GPU and memory transfer time cost

Solutions to avoid these multiples synchronizations and its impact on reconstruction time

- Use of only one synchronization per iteration by merging operators working on subbblock of data (need of a reduction step)
- Hide memory transfer time thanks to streams and half float data storage.
Reconstruction time (per iteration with computation of the optimized gradient step) with CPU centralization

### $1K^3$ volume (float) with 1024 projections on Titans X (3072 cores at 1,075 Ghz)

<table>
<thead>
<tr>
<th></th>
<th>proj (*2)</th>
<th>retro</th>
<th>conv(*3)</th>
<th>autres</th>
<th>total</th>
<th>Acc.</th>
</tr>
</thead>
<tbody>
<tr>
<td>1 GPU</td>
<td>49,6 %</td>
<td>20,2 %</td>
<td>30,1 %</td>
<td>28,1%</td>
<td>47,1 s</td>
<td>1,45</td>
</tr>
<tr>
<td>2 GPUs</td>
<td>36,6 %</td>
<td>7,5%</td>
<td>14,9 %</td>
<td>40,9%</td>
<td>32,4 s</td>
<td>1,69</td>
</tr>
<tr>
<td>4 GPUs</td>
<td>23,6 %</td>
<td>7,5 %</td>
<td>21,6 %</td>
<td>47,2 %</td>
<td>27,9 s</td>
<td>1,99</td>
</tr>
<tr>
<td>8 GPUs</td>
<td>15,9 %</td>
<td>6,6%</td>
<td>21,6%</td>
<td>55,9%</td>
<td>23,6 s</td>
<td>1,99</td>
</tr>
</tbody>
</table>

### $2K^3$ volume (float) with 2048 projections on Titans X (3072 cores at 1,075 Ghz)

<table>
<thead>
<tr>
<th></th>
<th>proj (*2)</th>
<th>retro</th>
<th>conv(*3)</th>
<th>autres</th>
<th>total</th>
<th>Acc.</th>
</tr>
</thead>
<tbody>
<tr>
<td>4 GPUs</td>
<td>36,27 %</td>
<td>20,65%</td>
<td>10,38 %</td>
<td>32,69%</td>
<td>5,4 mn</td>
<td>1,4</td>
</tr>
<tr>
<td>8 GPUs</td>
<td>26,38 %</td>
<td>13,31%</td>
<td>15,22 %</td>
<td>45,09%</td>
<td>3,8 mn</td>
<td>1,4</td>
</tr>
</tbody>
</table>
Reconstruction time (per iteration with computation of the optimized gradient step) with CPU centralization

Limitations of this CPU centralization

→ The "little" operations (norm L2, subtraction...) are becoming preponderant.

Solutions:

- Parallelization on the CPU cores (the minimum to do...)
- Merge the operators (break the frontier between each operator)
- Use of half float storage to get a GPU centralization (code 100% GPU)
Reconstruction time (per iteration with computation of the optimized gradient step) with **GPU centralization**

**1K³ volume (float) with 1024 projections on one Titan X (3072 cores at 1,075 Ghz)**

<table>
<thead>
<tr>
<th>proj (*2)</th>
<th>back</th>
<th>conv(*3)</th>
<th>others</th>
<th>total</th>
<th>Acc.</th>
</tr>
</thead>
<tbody>
<tr>
<td><em>CPU centralization</em></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>1 GPU</td>
<td>49,7 %</td>
<td>9,8 %</td>
<td>12,7 %</td>
<td>27,0 %</td>
<td>43,9 s</td>
</tr>
<tr>
<td><em>GPU centralization and half float</em></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>1 GPU</td>
<td>78,3 %</td>
<td>18,3 %</td>
<td>2,2 %</td>
<td>1,2 %</td>
<td>21,9 s</td>
</tr>
</tbody>
</table>

76/78
Conclusions

Towards an efficient computation on GPU for each operator

- Local and spatial memory locality
- Threads/Blocks “optimal” definition (thread parallelism)
- Unrolling loop (instruction parallelism)
- Incremental computation

Use of streams to hide CPU/GPU memory transfer time

Half-float data storage on GPU

- Reduction of CPU/GPU memory transfer
- Reduction of SDRAM GPU/coeurs GPU memory transfer
- Reduction of storage on SDRAM GPU

- > A significant acceleration factor (1.2/1.3) on a single GPU and a more efficient multi-GPU parallelization
- > A 100 % GPU code for $1K^3$ volume is becoming possible

iterative reconstruction of $2K^3$ volume
Perspective for SKA project

Short term perspectives

- Acceleration of the convolution (H and Ht)
- Multi-GPU parallelization with CPU Centralisation of data

Median/long term perspectives

- Multi-GPU parallelization with multi-GPU distribution of data
- Use of FPGA Architecture with HLS (High Level Synthesis) tools