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

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

Ecole GPU, Grenoble, 9 novembre 2017
Our multiGPU server

- Memoire SDRAM
- Carte graphique 1
- Disque dur
- Bus externe
- Bus PCI Express
- Memoire globale
- Memoire cache
- Simt
- CPU
- Coeur 1
- Coeur 2
- Coeur 3
- Coeur 4
- Chipset
- Bus memoire
- Bus PCI Express
- Memoire SDRAM
- Memoire globale
- Memoire cache
- Disque dur
- Rapide (300 Go)
- Lent (1 To)
1. Inverse Problem
   - Iterative algorithm: mean square minimisation + quadratic regularisation
   - Applications/Projects

2. 3D tomographic reconstruction accelerated on GPU
   - Projector/backprojector pair
   - Hardware acceleration
   - Projection on GPU
   - Backprojection on GPU

3. Parallelization optimization on a server (PC/GPUs)
   - multi-GPU Parallelization
   - CUDA Streams
   - CUDA Half float

4. Iterative loop parallelization
   - Distribution/Centralization of Data
   - Reconstruction time with CPU/GPU centralization
Without bayesian regularisation

\[ g = Hf + \epsilon \]

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

**Criterion : Mean Square**

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

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

\[ \nabla J(f) = -2 \cdot H^t(g - Hf) \]
Inverse Problem
3D tomographic reconstruction accelerated on GPU
Parallelization optimization on a server (PC/GPUs)
Iterative loop parallelization

Without bayesian regularisation

Descente de gradient
→ N iterations

\[ f^n : \text{Estimée du volume} \]

\[ g = Hf + \epsilon \]

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

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

\[ g = Hf + \epsilon \]

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

**Criterion : Mean Square**

\[ J(f) = \|g - Hf\|^2 \]
\[ f^{n+1} = f^n - \alpha \cdot \nabla J(f^n) \]
\[ \nabla J(f) = -2 \cdot H^t(g - Hf) \]
Inverse Problem
3D tomographic reconstruction accelerated on GPU
Parallelization optimization on a server (PC/GPUs)
Iterative loop parallelization

Without bayesian regularisation

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

Descente de gradient

\[
g = Hf + \epsilon
\]

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

Criterion : Mean Square

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

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

\[
\nabla J(f) = -2 \cdot H^t(g - Hf)
\]
Inverse Problem
3D tomographic reconstruction accelerated on GPU
Parallelization optimization on a server (PC/GPUs)
Iterative loop parallelization

Without bayesian regularisation

Descente de gradient

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

\[ g = Hf + \epsilon \]

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

Criterion : Mean Square

\[ J(f) = \| g - Hf \|^2 \]
\[ f^{n+1} = f^n - \alpha \cdot \nabla J(f^n) \]
\[ \nabla J(f) = -2 \cdot H^t (g - Hf) \]
Inverse Problem
3D tomographic reconstruction accelerated on GPU
Parallelization optimization on a server (PC/GPUs)
Iterative loop parallelization

Applications/Projects

Without Bayesian regularisation

Descente de gradient $\rightarrow$ N iterations

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

\[
\begin{align*}
g &= Hf + \epsilon \\
f^n \xrightarrow{\delta g = g - \hat{g}} g \\
f^n \xrightarrow{\delta f = H^t \delta g} f^{n+1}
\end{align*}
\]

\[
\begin{align*}
f^n + 1 &= f^n - \alpha \cdot \nabla J(f^n) \\
\nabla J(f) &= -2 \cdot H^t (g - Hf)
\end{align*}
\]

Criterion : Mean Square

\[ J(f) = \| g - Hf \|^2 \]
Inverse Problem
3D tomographic reconstruction accelerated on GPU
Parallelization optimization on a server (PC/GPUs)
Iterative loop parallelization

Applications/Projects

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)

\[ J(f) = J_1(f) + J_2(f) \]

\[ J_1(f) = ||g - Hf||^2 \]

\[ J_2(f) = \lambda ||Df||^2 \]

\[ f^{n+1} = f^n - \alpha \cdot (\nabla J_1(f^n) + \nabla J_2(f^n)) \]
Inverse Problem
3D tomographic reconstruction accelerated on GPU
Parallelization optimization on a server (PC/GPUs)
Iterative loop parallelization

Applications/Projects

Iterative algorithm: Mean square + quadratic reg

Collaboration avec l’IDES de l’Univ. Paris-Sud (F. Schmidt)
PhD fellowship financed by CDS (beginning autumn 2015)

Instrument PFS (Planetary Fourier Spectrum) de la mission MARS EXPRESS
Inverse Problem
3D tomographic reconstruction accelerated on GPU
Parallelization optimization on a server (PC/GPUs)
Iterative loop parallelization

Applications/Projects

[Planéto] Correction de vibrations mécaniques

Instrument modélisé par une convolution 1D

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

Taille gigantesque des données
Des années d’enregistrements de la mission MARS EXPRESS (2003) donc potentiellement 1 milliard de spectres (de 8192 échantillons) !
Inverse Problem
3D tomographic reconstruction accelerated on GPU
Parallelization optimization on a server (PC/GPUs)
Iterative loop parallelization

Iterative algorithm: Mean square + quadratic reg
Applications/Projects

[Astro] Méthode de reconstruction en astronomie
**Inverse Problem**

3D tomographic reconstruction accelerated on GPU
Parallelization optimization on a server (PC/GPUs)
Iterative loop parallelization

**Applications/Projects**

[Tomo3D] Algorithmes de reconstruction tomographique

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

**APPLICATIONS**

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

2. **RECONSTRUCTION**
   - Algorithme itératif
   - Accélération sur GPU

![Diagram](image_url)
$1K^3$ volume from $1K$ projections with $1K^2$ pixels (SAFRAN data set)

Work done in collaboration with SAFRAN (Post-doc Thomas Boulay)
1. Inverse Problem

2. 3D tomographic reconstruction accelerated on GPU
   - Projector/backprojector pair
   - Hardware acceleration
   - Projection on GPU
   - Backprojection on GPU

3. Parallelization optimization on a server (PC/GPUs)

4. Iterative loop parallelization
$Hf$ 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
   - Volume \( 2048^3 \) \( \rightarrow \) matrix \( H = 1 \) Exa Bytes!

2. **Geometric operators**
   - On line computation of \( h_{ij} \) coefficients

---

### Diagram

- **Projection**
  - \( u \) and \( \phi \)
  - Detecteurs du tomographe
  - Objet image

- **Retroprojection**
  - \( u \) and \( \phi \)
  - Objet reconstruit
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)
Thesis conclusions

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>$3^{eme}$ (*4 P4)</td>
<td>$1^{er}$ (*50 P4)</td>
<td>$2^{eme}$ (*5 P4)</td>
</tr>
<tr>
<td>Efficacy</td>
<td>$2^{eme}$ (7 C/op)</td>
<td>$1^{eme}$ (14 C/Op)</td>
<td>$1^{er}$ (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)
Inverse Problem
3D tomographic reconstruction accelerated on GPU
Parallelization optimization on a server (PC/GPUs)
Iterative loop parallelization

Acquisition speed // Reconstruction speed

Scanners CT
Temps de reconstruction mesurés
Extrapolation à partir du temps de [Exxim07]
Evolution des CPUs (*2/2 ans)
Evolution des GPUs (*2.2/an)

Nombre de slice/s (pour 512 matrices de projection 512*512)

Temps de reconstruction mesurés
Extrapolation à partir du temps de [Exxim07]
Evolution des CPUs (*2/2 ans)
Evolution des GPUs (*2.2/an)

1 CPU [Exxim07]
1 GPU ([Xu07])
1 Cell ([Scherl07])
2 Cells ([Scherl07])
8 FPGAs ([Heigl07])
1 ASIC ([Terarecon])
4 ASICs ([Terarecon])
1 FPGA en simulation ([Li04])
1 FPGA ([Goddard02])
1 FPGA ([Goddard02])
1 ASIC ([Terarecon])
1 CPU [Exxim07]

1
10
100
1000
10000
Année de production

16 slices/0.42s
64 slices/0.33s
256 slices/ (~0.33s)

16 slices/0.42s
64 slices/0.33s
Toshiba Prototype
256 slices/ (~0.33s)
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></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></td>
<td>10</td>
<td>14</td>
<td>17</td>
<td>7</td>
<td>10</td>
<td>20</td>
<td></td>
</tr>
<tr>
<td>GPU (AMD)</td>
<td>3</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>1</td>
<td></td>
<td></td>
<td>1</td>
</tr>
<tr>
<td>DSP</td>
<td>2</td>
<td></td>
<td></td>
<td></td>
<td>1</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Intel (Larabee, Xeon phi)</td>
<td>2</td>
<td></td>
<td></td>
<td></td>
<td>2</td>
<td>2</td>
<td>2</td>
<td></td>
</tr>
</tbody>
</table>
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

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

\[
\text{CALCUL DES COORDONNEES}
\]

\[
\begin{align*}
\text{Source} & \quad \text{Detecteurs} \\
u_e + 1 & \quad u(x) \\
& \quad \hat{u}_e \\
\end{align*}
\]

\[
\text{Volume} \
\begin{array}{c}
\text{Source} \\
\end{array}
\]

\[
g(\phi, u)
\]

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

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

```plaintext
for (xn, yn) in Volume do
    for phi = 0 to phi_max - 1 do
        // coordinates computation
        u(phi, xn, yn) = ...
        // linear interpolation
        g_interp = (1 - epsilon_u) \cdot g(phi, u_e) + epsilon_u \cdot g(phi, u_e + 1)
        // accumulation
        f^*(xn, yn) + = g_interp
    end for
end for
```
2D backprojection: scattered data access

for \((x_n, y_n)\) in Volume do
  for \(\phi_i = 0\) to \(\phi_{\text{max}} - 1\) do
    // coordinates computation
    \(u(\phi_i, x_n, y_n) = \ldots\)
    // 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 by blocks: localized data access

```
for (Bx, By) in Volume do
    for phi = 0 to phi_{max} - 1 do
        for (xn, yn) in Bloc do
            // coordinates computation
            u(phi, xn, yn) = ...
            // linear interpolation
            g_{interp} = (1 - \epsilon_u) \cdot g(phi, u_e) + \epsilon_u \cdot g(phi, u_e + 1)
            // accumulation
            f*(xn, yn) += g_{interp}
        end for
    end for
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 $(B_x, B_y, B_z)$
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)
Our 8 GPUs server (Carri Systems)
3D backprojection multi GPU parallelization

Source X

Volume

Plan de détecteurs
3D projection multi-GPU parallelization

- 1024 × 256
- GPU 0, 1, 2, 3, 4, 5, 6, 7
- Nφ/2
- Plan de détecteurs

Source X
Volume
**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</td>
<td>4610</td>
<td>2659</td>
</tr>
<tr>
<td>Back (ms)</td>
<td>7604</td>
<td>5181</td>
<td>3027</td>
<td>1929</td>
</tr>
<tr>
<td>Conv (ms)</td>
<td>3062</td>
<td>2987</td>
<td>2438</td>
<td>1668</td>
</tr>
</tbody>
</table>

The table shows the reconstruction times for different numbers of GPUs. The times are listed in milliseconds (ms) for three different steps: Projection (Proj), Back projection (Back), and Convolution (Conv). The times decrease as the number of GPUs increases, indicating improved performance with parallelization.
Goal of streams: hide PC/GPU memory transfer

**Différence entre synchrone et asynchrone**

*Synchrone*
- Download Image 1
- Kernel Image 1
- Upload Image 1
- Download Image 2
- ... etc etc ...
- Upload Image 3

9 cycles

*Asynchrone (Trois streams)*

<table>
<thead>
<tr>
<th>Stream 1</th>
<th>Stream 2</th>
<th>Stream 3</th>
</tr>
</thead>
<tbody>
<tr>
<td>Download Image 1</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Kernel Image 1</td>
<td>Download Image 2</td>
<td></td>
</tr>
<tr>
<td>Upload Image 1</td>
<td>Kernel Image 2</td>
<td>Download Image 3</td>
</tr>
<tr>
<td>Download Image 2</td>
<td>Upload Image 2</td>
<td>Kernel Image 3</td>
</tr>
<tr>
<td>Upload Image 2</td>
<td></td>
<td>Upload Image 3</td>
</tr>
</tbody>
</table>

5 cycles

*Goal of streams: hide PC/GPU memory transfer*
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
```
CUDA streams for mono GPU backprojection (1024 angles $1024^2$ plan)

4 streams

- Profiling overhead
  - [0] GeForce GTX TITAN X
  - Context 1 (CUDA)
    -Memcpy (HtoD)
    -Memcpy (DtoH)
  - Compute
  - Streams
    - Default
    - Stream 13
    - Stream 14
    - Stream 15
    - Stream 16
    - Stream 17

36/49
3D tomographic reconstruction accelerated on GPU
Parallelization optimization on a server (PC/GPUs)
Iterative loop parallelization

CUDA Streams
CUDA Half float

single GPU time with streams

<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>
**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</td>
<td>4610</td>
<td>2659</td>
</tr>
<tr>
<td>Proj (ms) w/ streams</td>
<td>11551</td>
<td>5783</td>
<td>3142</td>
<td>1756</td>
</tr>
<tr>
<td>Back (ms) w/o streams</td>
<td>7604</td>
<td>5181</td>
<td>3027</td>
<td>1929</td>
</tr>
<tr>
<td>Back (ms) w/ streams</td>
<td>5358</td>
<td>2609</td>
<td>1672</td>
<td>1731</td>
</tr>
<tr>
<td>Conv (ms) w/o streams</td>
<td>3062</td>
<td>2987</td>
<td>2438</td>
<td>1668</td>
</tr>
<tr>
<td>Conv (ms) w/ streams</td>
<td>3072</td>
<td>2482</td>
<td>2340</td>
<td>1674</td>
</tr>
</tbody>
</table>

**1K³ volume (float) with 1024 projections on 1 to 8 Titans X (3072 cores at 1,075 Ghz)**

Limitations due to PCI express gen2 bandwidth (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**

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

<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

$1K^3$ volume (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) f</td>
<td>11551</td>
<td>5783</td>
<td>2,0</td>
<td>3142</td>
</tr>
<tr>
<td>Proj (ms) hf</td>
<td>8970</td>
<td>4620</td>
<td>1,94</td>
<td>2357</td>
</tr>
<tr>
<td>Back (ms) f</td>
<td>5358</td>
<td>2609</td>
<td>2,0</td>
<td>1672</td>
</tr>
<tr>
<td>Back (ms) hf</td>
<td>4252</td>
<td>2164</td>
<td>1,96</td>
<td>1229</td>
</tr>
<tr>
<td>Conv (ms) f</td>
<td>3072</td>
<td>2482</td>
<td>1,24</td>
<td>2340</td>
</tr>
<tr>
<td>Conv (ms) hf</td>
<td>1608</td>
<td>1267</td>
<td>1,27</td>
<td>1171</td>
</tr>
</tbody>
</table>

Limitations due to PCI express gen2 bandwidth (2 to 4 GB/s)
1. Inverse Problem

2. 3D tomographic reconstruction accelerated on GPU

3. Parallelization optimization on a server (PC/GPUs)

4. Iterative loop parallelization
   - Distribution/Centralization of Data
   - Reconstruction time with CPU/GPU centralization
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*

− > 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)

− > 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)

− > 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 subblock 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

1$K^3$ 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></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>1,45</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>1,69</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>

2$K^3$ 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></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, substraction...) are becoming preponderants....

Solutions:

- Parallelization on the CPU cores (the minimum to do....)
- Merge the operators (break the frontier between each operators)
- 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**

$1K^3$ 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><strong>CPU centralization</strong></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><strong>GPU centralization and half float</strong></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>

Acc.: Accuracy
### 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
Short term perspectives

- Multi-GPU Centralisation of data
- Algorithmic acceleration with reduction of the number of iterations (preconditioner, conjugate gradient with hessian compute ...)

Median/long term perspectives

- Merge operators of each iteration to minimize the number of synchronization CPU/GPUs.
- Use of another projection/backprojection pair (matched ?)
- Futures Architectures : SDRAM stacks on the GPU chip ? Compute in half float ? Link between PC/GPU improved ?
- Post-Doc Mircea Dumitru working on integration of our methodological and software developments on opensource Tookit (Astra, RTK...)

In Adequation with GPI methodological developments

- PhD. Li WANG on bayesian hierarchical methods
- PhD. Camille CHAPDELAINE (SAFRAN) on iterative reconstruction algorithms for NDT of aeronautic pieces