Multi-GPU parallelization of 3D X-Ray Reconstruction

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

Workshop Imag’in, laboratoire GEOPS, Orsay, 15 décembre 2016
Our multiGPU server

- CPU
  - Coeur 1
  - Coeur 2
  - Coeur 3
  - Coeur 4

- Memoire SDRAM
- Carte graphique 1
- Carte graphique 4
- Memoire globale
  - (4 Go)
  - (32 Go)
  - (100 Go/s)

- Memoire cache
  - (16 Ko)
  - (8 Ko)

- Cache 2D
- SIMT
  - Shared
  - (8 Ko)

- Chipset
  - Memoire SDRAM
  - Memoire cache
  - Memoire globale

- Bus externe
  - (100 Mo/s)

- Bus PCI Express
  - (1 Go/s)

- Bus memoire
  - (10 Go/s)
1. Inverse Problem
   - Iterative algorithm: mean square minimisation + quadratic regularisation
   - Real/big data reconstruction
   - Projector/backprojector pair
   - Hardware acceleration

2. GPU projector/backprojector
   - Projection on GPU
   - Backprojection on GPU

3. Parallelization optimisation 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

5. Conclusions and Perspectives
Without bayesian regularisation

Descente de gradient

\[ \hat{g} = Hf^n \]

\[ H^t \delta g \]

\[ g = Hf + \epsilon \]

\[ f : \text{volume} \]

\[ g : \text{tomograph data} \]

\[ H : \text{acquisition model} \]

\[ \epsilon : \text{noise} \]

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

**Iterative algorithm**: Mean square + quadratic reg

- **Real/big data reconstruction**
- **Projector/backprojector pair**
- **Hardware acceleration**

**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

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

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

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

**Descente de gradient**
\[ \rightarrow N \text{ iterations} \]

\[ \delta f = H^t \delta g \]

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

**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

\[ g = Hf + \epsilon \]

\( f \): 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 baysian regularisation

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

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

**Criterion : Mean Square**

\[ J(f) = ||g - Hf||^2 \]
\[ f^{n+1} = f^n - \alpha \cdot \nabla J(f^n) \]
\[ \nabla J(f) = -2 \cdot H^t(g - Hf) \]
Inverse Problem
GPU projector/backprojector
Parallelization optimization on a server (PC/GPUs)
Iterative loop parallelization
Conclusions and Perspectives

With bayesian regularisation

\[ \delta g = g - \hat{g} \]

\[ \delta_{\text{MC}} f = H^t \delta g \]

\[ \delta_{\text{reg}} f = 2 \lambda D^t D f \]

\[ f^{n+1} = f^n - \alpha \cdot (\nabla J_1(f^n) + \nabla J_2(f^n)) \]

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

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

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

\[ g = Hf + \epsilon \]

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

Criterion: Mean Square + Quadratic Regularisation (MSQR)
$1K^3$ volume from $1K$ projections with $1K^2$ pixels (SAFRAN data set)

Work done in collaboration with SAFRAN (Post-doc Thomas Boulay)
Inverse Problem
GPU projector/backprojector
Parallelization optimization on a server (PC/GPUs)
Iterative loop parallelization
Conclusions and Perspectives

Iterative algorithm: Mean square + quadratic reg
Real/big data reconstruction
Projector/backprojector pair
Hardware acceleration

$Hf$ and $H^t\delta g$ computation

1. Matrix multiplication

- Reading $h_{ij}$ coefficients in SDRAM memory
- Volume $2048^3 \rightarrow$ matrix $H = 1$ To!
**Hf and \( H^t \delta g \) computation**

1. **Matrix multiplication**
   - Reading \( h_{ij} \) coefficients in SDRAM memory
   - ! Volume \( 2048^3 \) \( \rightarrow \) matrix \( H = 1 \) To!

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

---

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

- Détecteurs du tomographe
- Objet imagé
- 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)
**Inverse Problem**

GPU projector/backprojector
Parallelization optimization on a server (PC/GPUs)
Iterative loop parallelization

**Conclusions and Perspectives**

Iterative algorithm: Mean square + quadratic reg
Real/big data reconstruction
Projector/backprojector pair
Hardware acceleration

### Thesis conclusions

![Images of brain scans]

**CPU/GPU/FPGA comparison**

<table>
<thead>
<tr>
<th></th>
<th>CPU</th>
<th>GPU</th>
<th>FPGA</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Time</strong></td>
<td>3&lt;sup&gt;eme&lt;/sup&gt; (*4 P4)</td>
<td>1&lt;sup&gt;er&lt;/sup&gt; (*50 P4)</td>
<td>2&lt;sup&gt;eme&lt;/sup&gt; (*5 P4)</td>
</tr>
<tr>
<td><strong>Efficacy</strong></td>
<td>2&lt;sup&gt;eme&lt;/sup&gt; (7 C/op)</td>
<td>1&lt;sup&gt;eme&lt;/sup&gt; (14 C/Op)</td>
<td>1&lt;sup&gt;er&lt;/sup&gt; (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)
Acquisition speed // Reconstruction speed

- Inverse Problem
- GPU projector/backprojector
- Parallelization optimization on a server (PC/GPUs)
- Iterative loop parallelization
- Conclusions and Perspectives

**Iterative algorithm**: Mean square + quadratic reg
**Real/big data reconstruction**
**Projector/backprojector pair**
**Hardware acceleration**

- **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)**

<table>
<thead>
<tr>
<th>Année de production</th>
<th>1</th>
<th>10</th>
<th>100</th>
<th>1000</th>
<th>10000</th>
</tr>
</thead>
<tbody>
<tr>
<td>Toshiba Prototype</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>256 slices/0.33s</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>64 slices/0.33s</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>16 slices/0.42s</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>4 ASICs [Terarecon]</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>8 CPUs</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>1 ASIC [Terarecon]</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>1 CPU [Exxim07]</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>1 FPGA [Godard02]</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>1 FPGA en simulation [Li04]</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>8 FPGAs [Heigl07]</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>2 Cells [Scherl07]</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>1 GPU [Xu07]</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

**16 slices/0.42s**
**1 FPGA [Exxim07]**
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 Accelerator</th>
<th>2003</th>
<th>2005</th>
<th>2007</th>
<th>2009</th>
<th>2011</th>
<th>2013</th>
<th>2015</th>
</tr>
</thead>
<tbody>
<tr>
<td>Cluster/Cloud (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>
</tr>
<tr>
<td>GPU (NVIDIA)</td>
<td></td>
<td></td>
<td>10</td>
<td>14</td>
<td>17</td>
<td>7</td>
<td>10</td>
</tr>
<tr>
<td>GPU (AMD)</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>Xeon phi (Intel)</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>2</td>
<td>2</td>
<td></td>
</tr>
<tr>
<td>FPGA</td>
<td>4</td>
<td></td>
<td></td>
<td></td>
<td>1</td>
<td></td>
<td></td>
</tr>
<tr>
<td>DSP</td>
<td></td>
<td>2</td>
<td></td>
<td></td>
<td></td>
<td>1</td>
<td></td>
</tr>
<tr>
<td>Cell (IBM)</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>3</td>
</tr>
<tr>
<td>Larabee (Intel)</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>2</td>
</tr>
</tbody>
</table>
1. Inverse Problem

2. GPU projector/backprojector
   - Projection on GPU
   - Backprojection on GPU

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

4. Iterative loop parallelization

5. Conclusions and Perspectives
2D projector “regular sampling”

```python
for (un, phi) in Projection do
    for xn = 0 to xn_max - 1 do
        // coordinates computation
        y(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

\textbf{CALCUL DES COORDONNEES}

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: scattered data access

### Calcul des coordonnées

\[ u(x) \]

\[ u_e+1 \]

\[ \epsilon_u \cdot g(\phi, u_e+1) + (1 - \epsilon_u) \cdot g(\phi, u_e) \]

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

### Acces aux données \( g(\phi, u) \)

\[ \text{for } (x_n, y_n) \text{ in Volume do} \]
\[ \text{for } \phi = 0 \text{ to } \phi_{\text{max}} - 1 \text{ do} \]
\[ // \text{ coordinates computation} \]
\[ u(\phi, x_n, y_n) = ... \]
\[ // \text{ linear Interpolation} \]
\[ g_{\text{interp}} = (1 - \epsilon_u) \cdot g(\phi, u_e) + \epsilon_u \cdot g(\phi, u_e + 1) \]
\[ // \text{ accumulation} \]
\[ f^*(x_n, y_n) + = g_{\text{interp}} \]

end for

end for
2D backprojection: scattered data access

\[
\text{CALCUL DES COORDONNEES}
\]

\[
\text{ACCES AUX DONNEES } g(\phi, u)
\]

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

\[
\text{for } \text{phi} = 0 \text{ to } \text{phi}_{\text{max}} - 1 \text{ do}
\]

\[
// \text{ coordinates computation}
\]

\[
\text{u(\text{phi}, x_n, y_n) = ...}
\]

\[
// \text{ linear interpolation}
\]

\[
g_{\text{interp}} = (1 - \epsilon_u) \cdot g(\text{phi}, u_e) + \epsilon_u \cdot g(\text{phi}, u_e + 1)
\]

\[
// \text{ accumulation}
\]

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

\[
\text{end for}
\]

\[
\text{end for}
\]

\[
@ = u + \text{phi} \times W_u
\]

Données 1D dans la mémoire
2D backprojection: scattered data access

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

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

Source X
Volume
Plan de détecteurs
3D projection multi-GPU parallelization
### 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 <strong>1,76</strong></td>
<td>4610 <strong>3,13</strong></td>
<td>2659 <strong>5,42</strong></td>
</tr>
<tr>
<td>Back (ms)</td>
<td>7604</td>
<td>5181 <strong>1,47</strong></td>
<td>3027 <strong>2,51</strong></td>
<td>1929 <strong>3,94</strong></td>
</tr>
<tr>
<td>Conv (ms)</td>
<td>3062</td>
<td>2987 <strong>1,02</strong></td>
<td>2438 <strong>1,26</strong></td>
<td>1668 <strong>1,84</strong></td>
</tr>
</tbody>
</table>
Goal of streams: hide PC/GPU memory transfer

**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 3</td>
<td>Upload Image 3</td>
<td></td>
</tr>
</tbody>
</table>

5 cycles

*Différence entre synchrone et asynchrone*
CUDA streams for mono GPU backprojection (1024 angles $1024^2$ plan)
CUDA streams for mono GPU backprojection (1024 angles 1024² plan)
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>

1K³ Volume (float) with 1024 projections on 1 Titan X (3072 cores at 1,075 Ghz)
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 bandwidth (2 to 4 GB/s)
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
**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**

1$K^3$ 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><strong>Proj (ms) f</strong></td>
<td>11551</td>
<td>5783 2,0</td>
<td>3142 3,68</td>
<td>1756 6,58</td>
</tr>
<tr>
<td><strong>Proj (ms) hf</strong></td>
<td>8970</td>
<td>4620 1,94</td>
<td>2357 3,80</td>
<td>1265 7,09</td>
</tr>
<tr>
<td><strong>Back (ms) f</strong></td>
<td>5358</td>
<td>2609 2,0</td>
<td>1672 3,20</td>
<td>1731 3,10</td>
</tr>
<tr>
<td><strong>Back (ms) hf</strong></td>
<td>4252</td>
<td>2164 1,96</td>
<td>1229 3,46</td>
<td>876 4,83</td>
</tr>
<tr>
<td><strong>Conv (ms) f</strong></td>
<td>3072</td>
<td>2482 1,24</td>
<td>2340 1,31</td>
<td>1674 1,83</td>
</tr>
<tr>
<td><strong>Conv (ms) hf</strong></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 bandwith (2 to 4 GB/s)
1. Inverse Problem

2. GPU projector/backprojector

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

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

5. Conclusions and Perspectives
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 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

### 1K³ 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>

### 2K³ 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**

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

<table>
<thead>
<tr>
<th></th>
<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>
<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>
<td></td>
</tr>
<tr>
<td><strong>GPU centralization and half float</strong></td>
<td></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>
<td>2</td>
</tr>
</tbody>
</table>
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?

In Adequation with GPI methodological developments

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