HIGH PERFORMANCE COMPUTING

CHAPTER 7: MULTI-CPU/MULTI-GPU PROCESSING

APPLICATION FOR IMAGE AND VIDEO PROCESSING

Sidi Ahmed Mahmoudi

sidi.mahmoudi@umons.ac.be
Introduction

I. GPU Presentation

II. GPU Programming

III. Exploitation of Multi-CPU-GPU Architectures

IV. Application for Multimedia processing

1. Multi-CPU/Multi-GPU based image processing

2. Multi-GPU based video processing in Real Time

V. Multi-CPU/Multi-GPU based framework for HD image and video processing

VI. Experimental Results

Conclusion
Introduction

I. GPU Presentation

II. GPU Programming

III. Exploitation of Multi-CPU-GPU Architectures

IV. Application for Multimedia processing
   1. Multi-CPU/Multi-GPU based image processing
   2. Multi-GPU based video processing in Real Time

V. Multi-CPU/Multi-GPU based framework for HD image and video processing

VI. Experimental Results

Conclusion
Introduction

Moore’s Law : 1968

The number of transistors in an integrated circuit doubles approximately every two years

✓ Law well verified : the CPU power has doubled every 18 months

✗ CPU power capped to 4 GHZ : the law is no more verified
Introduction

- Multiplication of computing units in CPU: Multi-CPU, Multi-core, GPU
- GPU: initially used in 3D and video games.

CPU Intel Core i7 3770K (2012)

- 4 cores/8 threads
- 320 Euros

GPU GeForce GTX 580 (2011)

- 512 CUDA cores
- 350 Euros

Birth of GPGPU: General Purpose Graphic Processing Unit
Introduction

I. GPU Presentation

II. GPU Programming

III. Exploitation of Multi-CPU-GPU Architectures

IV. Application for Multimedia processing
   1. Multi-CPU/Multi-GPU based image processing
   2. Multi-GPU based video processing in Real Time

V. Multi-CPU/Multi-GPU based framework for HD image and video processing

VI. Experimental Results

Conclusion
GPU Presentation

- Graphic processing units
- Integrated within graphic cards
- Initially used for 3D rendering and video games
GPU Presentation

- GPU specialized for massively parallel treatments
- More transistors dedicated for computing
- Less number of control and cache units.

- **GPU**: Multiplication of computing units
GPU Presentation

INTRODUCTION | GPU PRESENTATION | GPU PROGRAMMING | MULTI-CPU-GPU PROGRAMMING | MULTIMEDIA PROCESSING APPLICATIONS | EXPERIMENTATIONS | CONCLUSION

TPC

Réseau d'interconnexion

Mémoire (cache)

Mémoire (cache)

Mémoire (cache)

SM

Textures

Caches

Unité d'Instruction

Mémoire Partagée

Fichier Registre

SFU

SM

SP

SP

SP

SFU

Sidi Ahmed Mahmoudi

Computer Science Department. Faculté polytechnique
**GPU Presentation**

**Power evolution of CPU Intel vs. GPU Nvidia**

![Graph showing theoretical peak floating point operations per clock cycle, double precision for various CPU and GPU models from 2008 to 2016.](image)

- **INTEL Xeon CPUs**
- **NVIDIA Tesla GPUs**
- **AMD Radeon GPUs**
- **INTEL Xeon Phis**

---

Université de Mons

Sidi Ahmed Mahmoudi

Computer Science Department. Faculté polytechnique
## GPU Presentation


<table>
<thead>
<tr>
<th>Rank</th>
<th>Site</th>
<th>System</th>
<th>Cores</th>
<th>Rmax (TFlop/s)</th>
<th>Rpeak (TFlop/s)</th>
<th>Power (kW)</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>DOE/SC/Oak Ridge National Laboratory United States</td>
<td><strong>Summit</strong> - IBM Power System AC922, IBM POWER9 22C 3.07GHz, NVIDIA Volta GV100, Dual-rail Mellanox EDR Infiniband IBM</td>
<td>2,414,592</td>
<td>148,600.0</td>
<td>200,794.9</td>
<td>10,096</td>
</tr>
<tr>
<td>2</td>
<td>DOE/NNSA/LLNL United States</td>
<td><strong>Sierra</strong> - IBM Power System AC922, IBM POWER9 22C 3.1GHz, NVIDIA Volta GV100, Dual-rail Mellanox EDR Infiniband IBM / NVIDIA / Mellanox</td>
<td>1,572,480</td>
<td>94,640.0</td>
<td>125,712.0</td>
<td>7,438</td>
</tr>
<tr>
<td>3</td>
<td>National Supercomputing Center in Wuxi China</td>
<td><strong>Sunway TaihuLight</strong> - Sunway MPP, Sunway SW26010 260C 1.45GHz, Sunway NRCPC</td>
<td>10,649,600</td>
<td>93,014.6</td>
<td>125,435.9</td>
<td>15,371</td>
</tr>
</tbody>
</table>
Introduction

I. GPU Presentation

II. GPU Programming

III. Exploitation of Multi-CPU-GPU Architectures

IV. Application for Multimedia processing
   1. Multi-CPU/Multi-GPU based image processing
   2. Multi-GPU based video processing in Real Time

V. Multi-CPU/Multi-GPU based framework for HD image and video processing

VI. Experimental Results

Conclusion
GPU Programming

- **Brook GPU**: since 2004
- **ATI Stream**: for ATI graphic cards
- **DirectX 11, OpenGL**: GPGPU shaders
- **OpenCL**: compatible with both NVIDIA and ATI
- **CUDA**: for NVIDIA cards only
OpenCL

- Launched by Apple in order to unify the use of multi-cores and GPUs
- Joint by AMD/ATI and NVIDIA
- First version in 2009
- Compatible with both NVIDIA and Ati cards
- Newer than CUDA: less powerful !!
- New GPU data types (vector, image, etc.).
CUDA

- Compatible with GPU NVIDIA.
- Syntax similar to C.
- Grid: set of blocs.
- Bloc: set of threads.

### Table: GPU Memory Utilization

<table>
<thead>
<tr>
<th>Type of Memory</th>
<th>Utility</th>
<th>Size</th>
<th>Latency (horloge cycles)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Global</td>
<td>Principal Memory</td>
<td>1Go</td>
<td>400 to 600</td>
</tr>
<tr>
<td>Shared</td>
<td>Cooperation between threads</td>
<td>16 Ko</td>
<td>4</td>
</tr>
<tr>
<td>Registers</td>
<td>Local memory to each thread</td>
<td>100 byte per thread</td>
<td>1</td>
</tr>
</tbody>
</table>
CUDA

- Example of vectors addition:

\[
\begin{pmatrix}
A[1] \\
\vdots \\
A[N]
\end{pmatrix} +
\begin{pmatrix}
B[1] \\
B[2] \\
\vdots \\
B[N]
\end{pmatrix} =
\begin{pmatrix}
C[1] \\
C[2] \\
\vdots \\
C[N]
\end{pmatrix}
\]

```c
__global__ void vecAdd(float* A, float* B, float* C)
{
    int i = threadIdx.x;
    C[i] = A[i] + B[i];
}

int main()
{
    // Kernel invocation
    vecAdd<<<1, N>>>(A, B, C);
}
```
\[
\begin{pmatrix}
A[1] \\
\vdots \\
A[N]
\end{pmatrix} +
\begin{pmatrix}
B[1] \\
B[2] \\
\vdots \\
B[N]
\end{pmatrix} =
\begin{pmatrix}
C[1] \\
C[2] \\
\vdots \\
C[N]
\end{pmatrix}
\]

\[
\text{vecAdd}^{<<1, N>>}(A, B, C);
\]

\[
\text{vecAdd}^{<<2, N/2>>}(A, B, C);
\]

\[
\text{vecAdd}^{<<100, N/100>>}(A, B, C);
\]

→ 1 Bloc of N threads

→ 2 Blocs of N/2 threads

→ 100 Blocs of N/100 threads
A CUDA program consists of five steps:

1. Memory allocation on GPU
2. Data transfer from CPU to GPU memory
3. Definition of the number of CUDA threads
4. Launching of CUDA functions in parallel
5. Data (result) transfer from GPU to CPU memory
CUDA

Example of matrix addition: Memory allocation

- Memory allocation on GPU
- The allocated memory will receive data from central (CPU) memory.

```c
float *Ad, *Bd, *Cd;
const int size = N*N*sizeof(float);   // matrix size

cudaMalloc((void **)&A_d, size);      // Allocate matrix A_d

cudaMalloc((void **)&B_d, size);      // Allocate matrix B_d

cudaMalloc((void **)&C_d, size);      // Allocate matrix C_d
```
Example of matrix addition: data transfer (CPU Mem to GPU Mem)

- Data transfer from CPU memory to GPU memory
- Matrix initialized on central memory
- Precise the type of transfer: HostToDevice

```c
cudaMemcpy(A_d, A, size, cudaMemcpyHostToDevice);
cudaMemcpy(B_d, B, size, cudaMemcpyHostToDevice);

// A and B represent the matrix initialized on CPU memory
```
CUDA

Example of matrix addition : define the number of threads

- Define the number of GPU threads :
  - Define the grid size
  - Define the bloc size

- The threads number depends from the size of data to treat

```c
// Bloc size
dim3 dimBlock( blocksize, blocksize );

// Grid size
dim3 dimGrid( N/dimBlock.x, N/dimBlock.y );
```
CUDA

Example of matrix addition: launching of CUDA kernels

- Define the CUDA function (kernel)
- Call and execute the CUDA kernel
- Specify the threads number (already defined)

```c
__global__ void add_matrix( float* a, float *b, float *c, int N )
{
    int i = blockIdx.x * blockDim.x + threadIdx.x;
    int j = blockIdx.y * blockDim.y + threadIdx.y;
    int index = i + j*N;
    if ( i < N && j < N )
        c[index] = a[index] + b[index];
}

add_matrix<<<dimGrid, dimBlock>>>( A_d, B_d, C_d, N );
```
Example of matrix addition: Results transfer (GPU Mem to CPU Mem)

- Transfer of results from GPU to CPU memory
- Precise the type of transfer: `DeviceToHost`
- Release the allocated memory on GPU

```c
cudaMemcpy( C, C_d, size, cudaMemcpyDeviceToHost );
cudaFree( A_d );
cudaFree( B_d );
cudaFree( C_d );

// Results will be saved on CPU memory « C ».
```
PLAN

I. GPU Presentation

II. GPU Programming

III. Exploitation of Multi-CPU-GPU Architectures

IV. Application for Multimedia processing
   1. Multi-CPU/Multi-GPU based image processing
   2. Multi-GPU based video processing in Real Time

V. Multi-CPU/Multi-GPU based framework for HD image and video processing

VI. Experimental Results

Conclusion
Exploitation of Multi-CPU/Multi-GPU Architectures

**StarPU**

- Developed in LABRI laboratory. Bordeaux, France
- Simultaneous exploitation of multiple CPUs and GPUs
- Exploit functions implemented with C, CUDA or OpenCL
- Efficient scheduling strategies

![Survey of StarPU][1]
Exploitation of Multi-CPU/Multi-GPU Architectures

StarSS

- Developed at the University of Catalonia (Barcelona)
- Flexible programming model for multicore
- Based essentially on:
  - **CellSs**: programming Cell processors
  - **SPMSs**: programming SMPs processors
  - **ClearSpeedSs**: programming Clear Speed processors
  - **GPUSS**: programming Multi-GPUs
  - **ClusterSs**: programming of clusters
  - **GridSs**: programming resources in grids
Exploitation des architectures hétérogènes

OpenACC

- API for high level programming
- Supported by CAPS enterprise, CRAY Inc, The Portland Group Inc (PGI) and NVIDIA
- A collection of compilation directives
- Takes into account the tasks of initialization, launching and stopping of accelerators
- Provides information for compilers
- Equivalent to Open MP for CPU programming
Introduction

I. GPU Presentation

II. GPU Programming

III. Exploitation of Multi-CPU-GPU Architectures

IV. Application for Multimedia processing
   1. Multi-CPU/Multi-GPU based image processing
   2. Multi-GPU based video processing in Real Time

V. Multi-CPU/Multi-GPU based framework for HD image and video processing

VI. Experimental Results

Conclusion
INTRODUCTION | GPU PRESENTATION | GPU PROGRAMMING | MULTI-CPU-GPU PROGRAMMING | MULTIMEDIA PROCESSING APPLICATIONS | EXPERIMENTATIONS | CONCLUSION

Intensive treatments
Large volumes
HD, Full HD, 4K, etc.

Computing time?
Real time?
Cost?

Image processing
Video processing
Medical imaging

Multimedia processing

Multi-CPU

GPU or Multi-GPU

Acceleration
Parallel computing
Hybrid computing
CPU or/and GPU?
Application for Multimedia processing

Hardware

- High computing power of GPUs.
- Heterogeneous architectures (Multi-CPU/Multi-GPU)
Application for Multimedia processing

Hardware
- High computing power of GPUs.
- Heterogeneous architectures (Multi-CPU/Multi-GPU)

Applications
- Intensive processing of images and videos
- Intensity: HD, Full HD or 4K of images and videos (real time)

200 images
05 minutes

Vertebra detection [Lecron2011]
## Application for Multimedia processing

<table>
<thead>
<tr>
<th>Hardware</th>
<th>Applications</th>
</tr>
</thead>
<tbody>
<tr>
<td>• High computing power of GPUs.</td>
<td></td>
</tr>
<tr>
<td>• Heterogeneous architectures (Multi-CPU/Multi-GPU)</td>
<td></td>
</tr>
<tr>
<td>• Intensive processing of videos: motion tracking and recognition.</td>
<td></td>
</tr>
<tr>
<td>• intensity: HD, Full HD or 4K videos to treat in real time</td>
<td></td>
</tr>
</tbody>
</table>

### Example:

- **Images indexation:** MediaCycle [Siebert2009]

- **2000 images** → **10 minutes**
Application for Multimedia processing

**Hardware**
- High computing power of GPUs.
- Heterogeneous architectures (Multi-CPU/Multi-GPU)

**Applications**
- Intensive processing of videos: motion tracking and recognition.
- Intensity: HD, Full HD or 4K videos to treat in real time

<table>
<thead>
<tr>
<th>Video Resolution</th>
<th>Frames Per Second</th>
</tr>
</thead>
<tbody>
<tr>
<td>720x640</td>
<td>19 FPS</td>
</tr>
<tr>
<td>640x376</td>
<td>20 FPS</td>
</tr>
<tr>
<td>720x640</td>
<td>8 FPS</td>
</tr>
</tbody>
</table>
## Application for Multimedia processing

<table>
<thead>
<tr>
<th>Hardware</th>
<th>Applications</th>
<th>Constraints</th>
<th>Objectives</th>
</tr>
</thead>
<tbody>
<tr>
<td>• High computing power of GPUs.</td>
<td>• Intensive processing of videos: motion tracking and recognition.</td>
<td>• Transfer times between CPU and GPU memories.</td>
<td>• Efficient management of memories: significant reduction of transfer times</td>
</tr>
<tr>
<td>• Heterogeneous architectures (Multi-CPU/Multi-GPU)</td>
<td>• intensity: HD, Full HD or 4K videos to treat in real time</td>
<td>• Efficient distribution of tasks between multiple GPUs</td>
<td>• Efficient image and video processing on GPU and Multi-GPU platforms</td>
</tr>
</tbody>
</table>
Introduction

I. GPU Presentation

II. GPU Programming

III. Exploitation of Multi-CPU-GPU Architectures

IV. Application for Multimedia processing
   1. Multi-CPU/Multi-GPU based image processing
   2. Multi-GPU based video processing in Real Time

V. Multi-CPU/Multi-GPU based framework for HD image and video processing

VI. Experimental Results

Conclusion
Scheme development: image processing

1. N Multiple Images
2. Image i (i ≤ N) on GPU → CUDA Parallel Treatment
3. Output Images

1. Image Unique on GPU → CUDA Parallel Treatment
2. OpenGL Visualization

CPU

GPU

MULTI-CPU-GPU PROGRAMMING | MULTIMEDIA PROCESSING APPLICATIONS | EXPERIMENTATIONS | CONCLUSION

INTRODUCTION | GPU PRESENTATION | GPU PROGRAMMING | MULTI-CPU-GPU PROGRAMMING | MULTIMEDIA PROCESSING APPLICATIONS | EXPERIMENTATIONS | CONCLUSION

Université de Mons

Sidi Ahmed Mahmoudi
Computer Science Department. Faculté polytechnique
GPU optimizations

Single image processing
- Texture Memory: fast access to pixel values
- Shared Memory: fast access to neighbors' values

Multiple image processing
- Texture and shared memories
- Overlapping of data transfers by kernels executions on GPU: CUDA streams
Implemented algorithms on GPU

Classic image processing algorithms:

- Geometrical transformations:
  - rotation, translation, homography
- Parallel treatments between image pixels
- GPU acceleration: from 10x to 100x

Points of interest detection on GPU:

- Preliminary step of several computer vision methods
- GPU implementation based on Harris technic
- Efficiency: invariant to rotation, brightness, etc.

Contours detection on GPU:

- GPU implementation based on Deriche-Canny approach
- Efficiency: noise robustness, reduced number of operations
- Good quality of detected contours
Results: Single image processing on GPU

Comparaison des performances CPU/GPU

Temperature of treatments (ms)

CPU       GPU
512x512   11,82x
1024x1024 14,65x
1472x1760
3936x3936

Taille de l'image

Partition du traitement sur GPU

visualisation OpenCV
kernels
chargement

512x512
58%
1024x1024
51%
1472x1760
3936x3936

Edges and corners detection on GPU: Global Memory
Results: Single image processing on GPU

Edges and corners detection on GPU:

OpenGL Visualization
Results: Single image processing on GPU

**Edges and corners detection on GPU:**

**Texture & Shared memories**
Results : Multiple images processing on GPU

Images resolution : 2048x2048

Comparaison des performances CPU/GPU

Partition du traitement sur GPU

Edges and corners detection on GPU : Texture & Shared memories
Results: Multiple images processing on GPU

Images resolution: 2048x2048

Comparaison des performances CPU/GPU

- Low accelerations: costs of data transfers between CPU and GPU memories
- Full exploitation of computing power within heterogeneous (Multi-CPU-GPU) platforms

Partition du traitement sur GPU

Edges and corners detection on GPU: CUDA streaming
Multiple images processing on heterogeneous platforms

StarPU: runtime for exploiting Heterogeneous platforms.

Multiple images

StarPU buffers

Dynamic scheduling

INTRODUCTION | GPU PRESENTATION | GPU PROGRAMMING | MULTI-CPU-GPU PROGRAMMING | MULTIMEDIA PROCESSING APPLICATIONS | Experimentations | Conclusions

4 CUDA streams

Copy to CPU Memory

44
Heterogeneous treatment: results

Edge and corners detection on heterogeneous platforms
Heterogeneous treatment: results

Edge and corners detection on heterogeneous platforms

Dynamic scheduling
Heterogeneous treatment : results

<table>
<thead>
<tr>
<th>Nombre d'images</th>
<th>2CPU</th>
<th>8CPU</th>
<th>1GPU</th>
<th>1GPU-2CPU</th>
<th>2GPU</th>
<th>2GPU-4CPU</th>
<th>4GPU</th>
<th>4GPU-8CPU</th>
</tr>
</thead>
<tbody>
<tr>
<td>10</td>
<td>2</td>
<td>3</td>
<td>5</td>
<td>4</td>
<td>3</td>
<td>3</td>
<td>5</td>
<td>5</td>
</tr>
<tr>
<td>50</td>
<td>4</td>
<td>5</td>
<td>7</td>
<td>6</td>
<td>4</td>
<td>4</td>
<td>7</td>
<td>7</td>
</tr>
<tr>
<td>100</td>
<td>6</td>
<td>7</td>
<td>9</td>
<td>8</td>
<td>6</td>
<td>6</td>
<td>9</td>
<td>9</td>
</tr>
<tr>
<td>200</td>
<td>8</td>
<td>9</td>
<td>10</td>
<td>9</td>
<td>8</td>
<td>8</td>
<td>10</td>
<td>10</td>
</tr>
</tbody>
</table>

Edge and corners detection on heterogeneous platforms

Dynamic scheduling + CUDA streaming
Introduction

I. GPU Presentation

II. GPU Programming

III. Exploitation of Multi-CPU-GPU Architectures

IV. Application for Multimedia processing

1. Multi-CPU/Multi-GPU based image processing

2. Multi-GPU based video processing in Real Time

V. Multi-CPU/Multi-GPU based framework for HD image and video processing

VI. Experimental Results

Conclusion
Video Processing on GPU: scheme development

1. Image i on GPU
2. CUDA Processing
3. OpenGL Visualization

Flowchart:
- Image i (i ≤ N) on CPU
- Input video
- i = i+1
- If i = N, STOP, END
- Otherwise, go back to 1.

Introduction | GPU Presentation | GPU Programming | Multi-CPU-GPU Programming | Multimedia Processing Applications | Experimentations | Conclusion

Sidi Ahmed Mahmoudi
Computer Science Department, Faculté polytechnique
GPU based motion tracking

INTRODUCTION | GPU PRESENTATION | GPU PROGRAMMING | MULTI-CPU-GPU PROGRAMMING | MULTIMEDIA PROCESSING APPLICATIONS | EXPERIMENTATIONS | CONCLUSION

Video → OpenCV decoding → frames on GPU

Corners detection → Optical flow computation → Noise elimination → OpenGL Visualization
Multi-GPU based motion tracking

**Introduction**

- **Video**
  - **OpenCV decoding**

**GPU**

- **Frames on GPU**
  - **GPU 1**
    - **Sub-frame N° 1**
    - **Corners detection**
    - **Optical flow computing**
    - **Noise elimination**
    - **OpenGL Visualization**
  - **GPU 2**
    - **Sub-frame N° 2**
    - **Corners detection**
    - **Optical flow computing**
    - **Noise elimination**
  - **GPU 3**
    - **Sub-frame N° 3**
    - **Corners detection**
    - **Optical flow computing**
    - **Noise elimination**
  - **GPU 4**
    - **Sub-frame N° 4**
    - **Corners detection**
    - **Optical flow computing**
    - **Noise elimination**

**Conclusion**

- **GPU 1 (output video)**
CPU based motion tracking

CPU based motion tracking in Full HD videos (1920x1080)
GPUCV based motion tracking

GPUCV based motion tracking in Full HD videos (1920x1080)
GPU based motion tracking in Full HD videos (1920x1080)
Introduction

I. GPU Presentation

II. GPU Programming

III. Exploitation of Multi-CPU-GPU Architectures

IV. Application for Multimedia processing
   1. Multi-CPU/Multi-GPU based image processing
   2. Multi-GPU based video processing in Real Time

V. Multi-CPU/Multi-GPU based framework for HD image and video processing

VI. Experimental Results

Conclusion
The proposed framework

Multimedia objects

- Single Image
  - Compute fc
    - YES: GPU Processing
    - NO: Multi-CPU-GPU Processing

- Multiple Images

- Multiple Videos

- Video in real time

Computer vision examples

- Medical applications
- Motion tracking
- Videos indexation
- Real time Event detection

The Framework

GPU based image and video primitive functions
Introduction

I. GPU Presentation

II. GPU Programming

III. Exploitation of Multi-CPU-GPU Architectures

IV. Application for Multimedia processing
   1. Multi-CPU/Multi-GPU based image processing
   2. Multi-GPU based video processing in Real Time

V. Multi-CPU/Multi-GPU based framework for HD image and video processing

VI. Experimental Results

Conclusion
1st use case: Vertebra segmentation

- **N** X-ray images
  - Histogram equalization
  - Edges detection
  - Corners detection
  - Vertebra localization

Multi-CPU → Multi-CPU/Multi-GPU → Multi-CPU

(a) Original Image
(b) Detection of the vertebrae
1st use case: Vertebra segmentation

<table>
<thead>
<tr>
<th>Etapes</th>
<th>1CPU</th>
<th>8CPU</th>
<th>1GPU/8CPU</th>
<th>4GPU/8CPU</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Temps(T)</td>
<td>T</td>
<td>Acc</td>
<td>T</td>
</tr>
<tr>
<td>Egalisation histogramme</td>
<td>62.10 s</td>
<td>15.44 s</td>
<td>4.02×</td>
<td>/</td>
</tr>
<tr>
<td>Detection contours</td>
<td>135.8 s</td>
<td>39.06 s</td>
<td>3.48×</td>
<td>15.80 s</td>
</tr>
<tr>
<td>Detection coins (poly)</td>
<td>46.12 s</td>
<td>11.51 s</td>
<td>4.01×</td>
<td>/</td>
</tr>
<tr>
<td>Temps total</td>
<td>244.02 s</td>
<td>66.01 s</td>
<td>3.70×</td>
<td>42.75 s</td>
</tr>
</tbody>
</table>

Performances of heterogeneous vertebra detection using 200 images (1476x1680)
2nd use case: Movement detection within mobile camera
2nd use case: Movement detection within mobile camera

GPU performance: movement detection within mobile camera
3rd use case: real time event detection
4\textsuperscript{th} use case: eyes, nose and frontal face detection
4th use case: eyes, nose and frontal face tracking
Conclusion

- Parallel processing between image pixels
- Parallel and heterogeneous processing between images
- Efficient exploitation of GPU memories
- Multi-GPU treatments for HD/Full HD videos
- Efficient selection of resources (CPUs or/and GPUs)
THANK YOU