CUDA & OpenCV
Engineering school

2 places

- Noisy-le-grand (East of Paris)
- Amiens (North of France)

About 30 student clubs/associations

1500 students

5 years program

- 3 years of common courses (maths, physic, mechanic, electronic, computer science, management)
- 2 years of specialisation
ESIEE-engineering (Paris)
- Computer Science
- Electronic and Microelectronic Systems
- Embedded Systems
- Telecommunications and Signal Processing
- Computer network architect
- Digital design 3D
- Electronic system for transportation

ESIEE-AMIENS
- Electronic and sustainable Development
- production systems
- Telecommunications and computer networks
- Building trade energy
Summary

- Introduction
- CUDA
- OpenCV + GPU support
- results
- tips & tricks
Introduction
about OpenCV

- Open Computer Vision library

- Image processing, feature detection, Object detection, Video analysis, Machine learning .......

```cpp
Mat image_dst, image_tmp image_src = imread("color.jpg", CV_LOAD_IMAGE_COLOR);
/*convert image_src to a 1 channel image of 8 byte per pixel, result stored in image_tmp*/
cvtColor(image_src, image_tmp, CV_8U, 1);
/*threshold image_tmp and store the result in image_dst, maxVal if < thres, 0 if > thres */
threshold(image_tmp, image_dst, thresh, maxVal, THRESH_BINARY);
```

- Started by Intel (1999), now open source project (Willow Garage co)

- OOP support since 2.0

- CUDA support since September 2010
about CUDA

- Compute Unified Device Architecture
  - GPGPU technology (General Purpose computing on Graphics Processing Unit)
  - C language extended
  - Windows, Linux, Mac OS

- Previous methods:
  - shading languages for Real-time rendering
  - CTM
  - BrookGPU
  - Then CUDA since 2007
Processing flow on CUDA

1. Copy processing data
2. Instruct the processing
3. Execute parallel in each core
4. Copy the result
Motivations

- Architecture massively parallel
  => SIMT (Single Instruction Multiple Thread)
  => High speed linear algebra

- Cheap and accessible to everyone

- 'Easy' to set up and to program

- But limited (see later)
### Motivations

**Benchmark:**

<table>
<thead>
<tr>
<th>Example Applications</th>
<th>URL</th>
<th>Application Speedup</th>
</tr>
</thead>
<tbody>
<tr>
<td>Seismic Database</td>
<td><a href="http://www.headwave.com">http://www.headwave.com</a></td>
<td>66x to 100x</td>
</tr>
<tr>
<td>Mobile Phone Antenna Simulation</td>
<td><a href="http://www.acceleware.com">http://www.acceleware.com</a></td>
<td>45x</td>
</tr>
<tr>
<td>Molecular Dynamics</td>
<td><a href="http://www.ks.uiuc.edu/Research/vmd">http://www.ks.uiuc.edu/Research/vmd</a></td>
<td>21x to 100x</td>
</tr>
<tr>
<td>Neuron Simulation</td>
<td><a href="http://www.evolvedmachines.com">http://www.evolvedmachines.com</a></td>
<td>100x</td>
</tr>
<tr>
<td>MRI processing</td>
<td><a href="http://bic-test.beckman.uiuc.edu">http://bic-test.beckman.uiuc.edu</a></td>
<td>245x to 415x</td>
</tr>
<tr>
<td>Atmospheric Cloud Simulation</td>
<td><a href="http://www.cs.clemson.edu/~jesteel/clouds.html">http://www.cs.clemson.edu/~jesteel/clouds.html</a></td>
<td>50x</td>
</tr>
</tbody>
</table>
CUDA
vocabulary

- Host & Devices
- What is a Kernel?
- What is a Thread?
Kernel hierarchy

- Grid
- Block
- Thread
### Software

- Thread

### Hardware

- Thread Processor
  - Threads are executed by thread processors

- Multiprocessor
  - Thread blocks are executed on multiprocessors
  - Thread blocks do not migrate
  - Several concurrent thread blocks can reside on one multiprocessor - limited by multiprocessor resources (shared memory and register file)

- Device
  - A kernel is launched as a grid of thread blocks
  - Only one kernel can execute on a device at one time
Kernel creation

- **Definition**: `__type__` void myKernel(params)
  - `__host__`
  - `__global__`
  - `__device__`

- **Call**:
  - Define blocks dimensions: `dim3 nbThread(x,y,z)`
  - Define grid dimensions: `dim3 nbBlock(x,y,1)`
  - `myKernel<<<nbBlock, nbThread>>>(params)`
  - `dimension limited`
In the Kernels

- `dim3 blockIdx` : block index in the grid

- `dim3 threadIdx` : thread index in the block

- `dim3 blockDim` : number of thread in each block

- `__syncthreads()` : wait until all the thread of the kernel reach this point
Data management

- Allocation
  - `cudaMalloc(&source, size)`

- Transfer
  - `cudaMemcpy(destination, source, size, direction)`
  - `cudaMemcpyHostToDevice` or `cudaMemcpyDeviceToHost`

- `cudaFree(source)`
<table>
<thead>
<tr>
<th>name</th>
<th>prefixe</th>
<th>R/W</th>
<th>Fast/slow</th>
<th>size</th>
<th>Accessibility</th>
<th>Life time</th>
</tr>
</thead>
<tbody>
<tr>
<td>global</td>
<td><strong>device</strong></td>
<td>RW</td>
<td>slow</td>
<td>Some GBytes</td>
<td>all</td>
<td>application</td>
</tr>
<tr>
<td>Constant</td>
<td><strong>constant</strong></td>
<td>RO</td>
<td>fast</td>
<td>64 KB/mp</td>
<td>all</td>
<td>application</td>
</tr>
<tr>
<td>texture</td>
<td></td>
<td>RO</td>
<td></td>
<td>all</td>
<td>application</td>
<td></td>
</tr>
<tr>
<td>shared</td>
<td><strong>shared</strong></td>
<td>RW</td>
<td>fast when no bank conflict</td>
<td>16 KB/mp</td>
<td>One block</td>
<td>block</td>
</tr>
<tr>
<td>register</td>
<td>(Default)</td>
<td>RW</td>
<td>fast</td>
<td>8-16 KB/mp</td>
<td>One thread</td>
<td>thread</td>
</tr>
<tr>
<td>local</td>
<td></td>
<td>RW</td>
<td></td>
<td>One thread</td>
<td>thread</td>
<td></td>
</tr>
</tbody>
</table>
Other CUDA function

- Error handling:
  - cudaError_t code = cudaGetLastError()
  - char* cudaGetErrorString(cudaError_t code)

- Synchronisation:
  - cudaThreadSynchronize()

- Mathematical functions
  - Sqrt, exp, cos, floor....

- See CUDA documentation:
Increment Array Example

**CPU program**

```c
void inc_cpu(int *a, int N)
{
    int idx;

    for (idx = 0; idx < N; idx++)
        a[idx] = a[idx] + 1;
}

int main()
{
    ...
    inc_cpu(a, N);
}
```

**CUDA program**

```c
__global__ void inc_gpu(int *a, int N)
{
    int idx = blockIdx.x * blockDim.x
               + threadIdx.x;

    if (idx < N)
        a[idx] = a[idx] + 1;
}

int main()
{
    ...
    dim3 dimBlock (blocksize);
    dim3 dimGrid( ceil( N / (float)blocksize ) )
    inc_gpu<<<dimGrid, dimBlock>>>(a, N);
}
```
limits

- Memory
  - Size
  - Bandwidth

- Not efficient for all algorithms

- Only for Nvidia cards

- Language limitation
OpenCV & CUDA support
Presentation:

- OpenCV 2.2 or 2.3
- Set WITH_CUDA flag in Cmake
- Requirement:
  - CUDA toolkit 4.0 (OpenCV 2.3)
  - CUDA toolkit 3.2 (OpenCV 2.2)
  - G++ or Visual Studio 2008/2010
- More informations: see openCV website, gpu section
Presentation:

- Based on GpuMat,
  similar as cv::Mat for Cpu

- Documentation:

- Possibility:
  - geometrical image transform
  - color conversion
  - corner detection
  - filter engine
  - histograms
  - feature detection
  - .....
Performances:

<table>
<thead>
<tr>
<th></th>
<th>CPU, ms</th>
<th>GPU, ms</th>
<th>SPEEDUP</th>
<th>DESCRIPTION</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>dft:</strong></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>305</td>
<td>13</td>
<td>x22.4</td>
<td>size 1000, 32FC2, complex-to-complex</td>
</tr>
<tr>
<td></td>
<td>983</td>
<td>34</td>
<td>x28.3</td>
<td>size 2000, 32FC2, complex-to-complex</td>
</tr>
<tr>
<td></td>
<td>4998</td>
<td>74</td>
<td>x66.9</td>
<td>size 4000, 32FC2, complex-to-complex</td>
</tr>
<tr>
<td><strong>cornerHarris:</strong></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>328</td>
<td>53</td>
<td>x6.12</td>
<td>size 2000, 32F</td>
</tr>
<tr>
<td></td>
<td>1610</td>
<td>195</td>
<td>x8.25</td>
<td>size 4000, 32F</td>
</tr>
<tr>
<td><strong>threshold:</strong></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>4</td>
<td>1</td>
<td>x3.36</td>
<td>size 2000, 8U, THRESH_TRUNC</td>
</tr>
<tr>
<td></td>
<td>13</td>
<td>1</td>
<td>x12.6</td>
<td>size 3000, 8U, THRESH_TRUNC</td>
</tr>
<tr>
<td></td>
<td>24</td>
<td>1</td>
<td>x13.8</td>
<td>size 4000, 8U, THRESH_TRUNC</td>
</tr>
<tr>
<td></td>
<td>6</td>
<td>0</td>
<td>x9.53</td>
<td>size 2000, 8U, THRESH_BINARY</td>
</tr>
<tr>
<td></td>
<td>13</td>
<td>2</td>
<td>x5.48</td>
<td>size 3000, 8U, THRESH_BINARY</td>
</tr>
<tr>
<td></td>
<td>22</td>
<td>2</td>
<td>x9.65</td>
<td>size 4000, 8U, THRESH_BINARY</td>
</tr>
<tr>
<td></td>
<td>27</td>
<td>2</td>
<td>x16.6</td>
<td>size 2000, 32F, THRESH_TRUNC</td>
</tr>
<tr>
<td></td>
<td>44</td>
<td>3</td>
<td>x13</td>
<td>size 3000, 32F, THRESH_TRUNC</td>
</tr>
<tr>
<td></td>
<td>81</td>
<td>6</td>
<td>x13.5</td>
<td>size 4000, 32F, THRESH_TRUNC</td>
</tr>
<tr>
<td><strong>solvePnPRansac:</strong></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>557</td>
<td>301</td>
<td>x1.85</td>
<td>num_points 5000</td>
</tr>
<tr>
<td></td>
<td>1015</td>
<td>309</td>
<td>x3.29</td>
<td>num_points 18000</td>
</tr>
<tr>
<td></td>
<td>4175</td>
<td>349</td>
<td>x11.9</td>
<td>num_points 70688</td>
</tr>
<tr>
<td><strong>log:</strong></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>57</td>
<td>1</td>
<td>x33.8</td>
<td>size 2000, 32F</td>
</tr>
<tr>
<td></td>
<td>128</td>
<td>3</td>
<td>x37.8</td>
<td>size 3000, 32F</td>
</tr>
<tr>
<td></td>
<td>226</td>
<td>5</td>
<td>x38.3</td>
<td>size 4000, 32F</td>
</tr>
<tr>
<td><strong>exp:</strong></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>57</td>
<td>1</td>
<td>x34.1</td>
<td>size 2000, 32F</td>
</tr>
<tr>
<td></td>
<td>128</td>
<td>3</td>
<td>x37.1</td>
<td>size 3000, 32F</td>
</tr>
<tr>
<td><strong>BruteForceMatcher:</strong></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>10438</td>
<td>199</td>
<td>x52.2</td>
<td>match</td>
</tr>
<tr>
<td></td>
<td>9872</td>
<td>462</td>
<td>x21.3</td>
<td>knnMatch</td>
</tr>
<tr>
<td></td>
<td>9215</td>
<td>590</td>
<td>x15.6</td>
<td>radiusMatch</td>
</tr>
</tbody>
</table>
Results:
OpenCV + my own kernels
Problem & solution

Problem:
Impossible to include CUDA and OpenCV in the same file => conflict?

Solution:
hardware

- CPU Intel Pentium 4 3.6 GHz
- GPU : GeForce GT 240
  - Compute capability : 1.2
  - 12 Mp x 8 CUDA Cores/MP
  - 512 threads max
  - 65 535 blocks max
  - Clock speed : 1.34 GHz

software

- CUDA 4.0
- Visual Studio 2008
- OpenCV 2.3
```c
int y = blockIdx.y * blockDim.y + threadIdx.y;  
int x = blockIdx.x * blockDim.x + threadIdx.x;  
int red,blue,green;  
if(y < height && x<width){
    unsigned char *src_pointer = in + y*in_step+x*3;  
    unsigned char *dst_pointer = out + y*out_step+x;  
    blue = *src_pointer;  
    src_pointer++;  
    green = *src_pointer;  
    src_pointer++;  
    red = *src_pointer;  
    dst_pointer = 
        gpu_lut_data[ blue + sizeX*green + sizeX*sizeY*red ];  
    *dst_pointer = 
        gpu_lut_data[ blue + sizeX*green + sizeX*sizeY*red ];  
    src_pointer++;  
    dst_pointer++;  
}  
for(int y=0; y<height; y++){  
    unsigned char *src_pointer = in + y*in_step;  
    unsigned char *dst_pointer = out + y*out_step;  
    for(int x=0; x<width; x++){  
        int blue = *src_pointer;  
        src_pointer++;  
        int green = *src_pointer;  
        src_pointer++;  
        int red = *src_pointer;  
        src_pointer++;  
        *dst_pointer =  
            gpu_lut_data[ blue + sizeX*green + sizeX*sizeY*red ];  
        dst_pointer++;  
    }  
```
apply mask

GPU

```c
int y = blockIdx.y * blockDim.y + threadIdx.y;
int x = blockIdx.x * blockDim.x + threadIdx.x;

if(y < height && x<width){
    unsigned char *src_pointer = in + y*in_step+x;
    unsigned char *dst_pointer = out + y*out_step+x;
    unsigned char *mask_pointer = mask + y*mask_step;
    *dst_pointer = (*src_pointer) * (*mask_pointer);
    dst_pointer++;
    src_pointer++;
    *dst_pointer = (*src_pointer) * (*mask_pointer);
    dst_pointer++;
    src_pointer++;
    *dst_pointer = (*src_pointer) * (*mask_pointer);
    dst_pointer++;
    src_pointer++;
    *dst_pointer = (*src_pointer) * (*mask_pointer);
    dst_pointer++;
    src_pointer++;
}
```

CPU

```c
for(int y=0; y<height; y++){
    unsigned char *src_pointer = in + y*in_step;
    unsigned char *dst_pointer = out + y*out_step;
    unsigned char *mask_pointer = mask + y*mask_step;
    for(int x=0; x<width; x++){
        *dst_pointer = (*src_pointer) * (*mask_pointer);
        dst_pointer++;
        src_pointer++;
        *dst_pointer = (*src_pointer) * (*mask_pointer);
        dst_pointer++;
        src_pointer++;
        *dst_pointer = (*src_pointer) * (*mask_pointer);
        dst_pointer++;
        src_pointer++;
        *dst_pointer = (*src_pointer) * (*mask_pointer);
        dst_pointer++;
        src_pointer++;
    }
}
```
Kernels call

```c

dim3 nbThread (16,16);
dim3 nbBlock (ceil((float)width/nbThread.x),ceil((float)height/nbThread.x));
applyMaskColor<<nbBlock,nbThread>>>
    (input_image, output_image, mask,width,height,
     input_image_step, output_image_step, mask_step);
```

- Fix the number of thread (< 512)
- Deduce the number of Blocks
Kernel representation

1 block

1 pixel = 1 thread

image = Grid
benchmark

```
thread : 16x16, blocks : 90 x 160, total number of threads : 3686400
image : 1440 x 2553, 3 channel(s), 3676320 pixels, 11028960 bytes

load image cpu : 519.000000 ms
load image gpu : 2382.000000 ms
 cpu is 4.5x faster

generate lut cpu : 1443.000000 ms
generate mask cpu : 237.000000 ms
apply mask cpu : 70.990000 ms
generate lut gpu : 1224.000000 ms
generate mask gpu : 10.000000 ms
apply mask gpu : 5.410000 ms
 lut generation : gpu is 1.2x faster
mask generation : gpu is 23x faster
mask application : gpu is 14x faster
	save image cpu : 1085.000000 ms
save image gpu : 1579.000000 ms
cpu is 1.45x faster
```
tips & tricks
Create a project for CUDA

- Installation, all you need is here: http://developer.nvidia.com/cuda-toolkit-40

- Include:
  - <cuda_runtime.h>
  - <cuda.h>
  - NVIDIA_gpu_Toolkit\CUDA\version\inc

- Library:
  - cuda.lib
  - cudart.lib
  - NVIDIA_gpu_Toolkit\CUDA\version\lib\Win32

- Ignore library:
  - libcmt.lib
  - libcmtld.lib
Configure visual studio

- .cu files

- Nvcc configuration:
  - custom build rules

- Syntax highlighting:
  - copy usertype.dat from C:\ProgramData\NVIDIA Corporation\NVIDIA GPU Computing SDK 4.0\C\doc\syntax_highlighting\visual_studio_8
  - To C:\Program Files\Microsoft Visual Studio 9.0\Common7\IDE
Some tips

- Pay attention with Cmake!
- Mostly gpuMat::step ≠ Mat::step
- Performance measure
- gpuMat <=> Mat:
  - gpuMat gpu_image = gpuMat(cpu_image)
  - Or gpu_image.upload(cpu_image)
  - Mat cpu_image = gpu_image
optimisation

- Warp of 32 threads
- If/else
- Kernel call is asynchronous
  - CudaMemcpy()
  - cudaThreadSynchronize()
- Avoid data transfer cpu => gpu
  - See sample “bandwidth test”
- Overlapping data transfer
References

  - NVIDIA CUDA C Programming Guide
  - CUDA API REFERENCE MANUAL
- CUDA tutorial by Cyril Zeller
- Optimizing CUDA, Nvidia tutorial
- Developpez.com (fr)
  - Une introduction à CUDA