

# GPU Computing Projects

---

E. Carlinet, J. Chazalon `{firstname.lastname@lrde.epita.fr}`

Nov. 2021

EPITA Research & Development Laboratory (LRDE)



## Overview

---

The goals of the project are to:

- apply data-parallelism concepts
- practice with CUDA
- set up a benchmark with a sound evaluation procedure
- present your results in a clear and convincing way

# Possible Subjects

## Standard Assignment

We propose 1 subject, that most of you should work on:

**Implementation a barcode detector in CUDA**

*More details later in this presentation.*

## Custom Assignment

For students who are at ease with CUDA, and want to investigate a particular question:

**Implementation and performance analysis of *SOME INTERESTING* algorithm in *YOUR PARALLEL PROGRAMMING TECHNOLOGY OF CHOICE***

If you choose this assignment, you must **validate your subject with us**.

*Contact us by email ASAP.*

## Important dates

1. Group composition (**teams of 4**) on Moodle **by Nov. 7th**
2. Final project submission on Moodle (**the day before your defense, before 23:59**)
3. Oral defense (either Teams or in presence) (SCIA: Dec. 16th, GISTRE: Dec. 17th)

## Deliverables

---

### 1. Implementation

- Source code for C++ CPU reference
- Source code for CUDA implementation(s)
- Source code for benchmark tools
- Build scripts (GNU Make, CMake...)

**We must be able to reproduce your results**

### 2. Report

- Description of the problem
  - Detailed if custom subject
  - Quick summary otherwise
- Quick description of the baseline CPU implementation (paper reference, parallel or not, etc.)
- Quick description of the baseline GPU implementation (changes from CPU version)
- **Which performance indicators you have used and why**
- **Identification of performance bottlenecks** (with measured indicators, graphs, etc.)
- **For each improvement over the GPU baseline** (implementations):
  - justification of this work regarding performance analysis
  - description of the improvement (e.g. used output privatization instead of global atomics)
  - comparison of the performance with and without this implementation
- **Table with summary** of the benchmark of all variants implemented
- **Summary of the contributions of each team member**



### 3. A live lecture / defense

- 15' presentation
- 5' demo
- 5' discussion
- Defenses will be held on Teams (opt. in presence)
- **All** the team members must be there

We will share with you some images/videos to process during the defense.

*Links to each meeting will be shared when all groups are formed.*

Submit code + report + slides on Moodle

- GISTRE: `https://moodle.cri.epita.fr/mod/assign/view.php?id=10526`
- SCIA: `https://moodle.cri.epita.fr/mod/assign/view.php?id=10754`

# Grade Sheet Used for Last Session

	<b>Travail effectif</b>	<b>Présentation</b>	<b>Note individuelle</b>
<b>Total</b>	<b>10</b>	<b>5</b>	<b>5</b>
<b>Critères</b>	Difficulté technique (bonus/malus) Qualité de l'implémentation (au regard des notions vues en cours) Qualité de l'évaluation, du benchmark (2 implém. CPU, 1+ GPU...) Prise de recul, analyse (identification des goulots d'étranglement...)	Qualité du support Clarté de l'exposé – Pédagogie Mise en contexte Positionnement et état de l'art	Prise de parole Réponse aux questions Implication dans le groupe
<b>Repères</b>			
A [16-20]	Benchmarks cohérents Au moins 3 implémentations (dont 2 variantes GPU) Utilisation de nvprof et/ou nsight (présentation graphique) Description des techniques utilisées Explication des différences de performance Identification des pistes d'amélioration	Oral de qualité, apprécié de l'auditoire Structure claire Illustrations claires Bonne maîtrise du temps	Maîtrise globale du sujet Réponses pertinentes aux questions Leader de groupe (A+)
B [12-16]	Bonne implémentation (versions CPU et GPU avec gain) Au moins un benchmark entre les 2 implémentations Identification d'au moins un facteur de ralentissement (mais pas forcément de solutions) MAIS une seule version GPU	Présentation retenant l'attention (B+) Présentation orale honnête (B-)	Participation honnête au projet Capable de faire appel aux notions de cours
C [8-12]	Minimum attendu Version GPU fonctionnelle MAIS manque de prise de recul sur les gains possibles MAIS manque d'analyse de la performance OU pas de version CPU	Présentation passable (C+) Présentation décevante (C-)	Présentation confuse Passif lors des questions Maîtrise incomplète du sujet
D [5-8]	Clairement en dessous du travail attendu Implémentation GPU non fonctionnelle	Mauvaise présentation orale	Travail très faible Mauvaise compréhension du sujet
E [0-5]	Pas de travail significatif Pas d'implémentation GPU	Inacceptable / absent / ...	Inacceptable / absent / ...

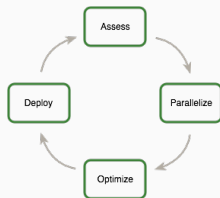
How you should work

---

# Our Expectations

We expect your implementation to be:

- running on GPU;
- correct, i.e. it produces an acceptable result.



**Do not try to make it fast at first, just make it work.**

Then, try to apply NVidia's Assess, Parallelize, Optimize, Deploy (APOD) design cycle as described in the CUDA C++ Best Practices Guide:

1. identify the part of the code which is responsible for the bulk of the execution time;
2. get a parallel version of the code (assumed to be sequential at first);
3. optimize the performance of the parallel code;
4. measure the performance of the new code.

## Project Outline for Standard Performance Analysis

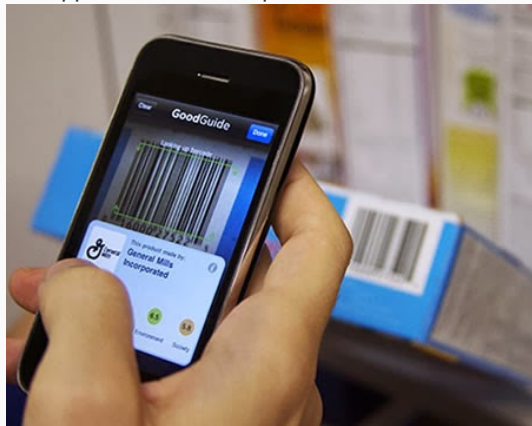
Broad Outline	Concrete Example
Choose an application	Mandelbrot
Determine the most time-consuming part of the app	Global atomics
Determine one or more data-parallel approaches to solve the problem	Tiling...
Create multiple implementations of the approach	One naive version, one version with shared memory...
Benchmark the implementations	Record memory transfer time, kernel time, utilization, FLOPS, etc.
Relate results to course concepts	Identify the cause of the bottleneck (memory or compute bounding)

## Detect Barcodes with CUDA

---

# Objective

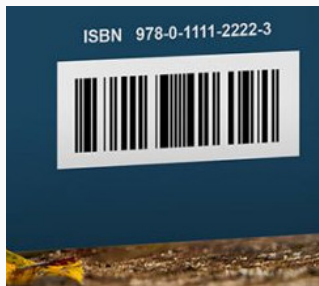
End to end application for smartphone real-time barcode detector.



*In practice: at least a working CUDA implementation which can process 1 image*



Barcodes exhibit a particular texture made of highly contrasted lines.



We will try to detect perfectly horizontal barcodes which have perfectly vertical lines.

Such region must therefore exhibit:

- a high density of strong horizontal gradients
- a low density of strong vertical gradients

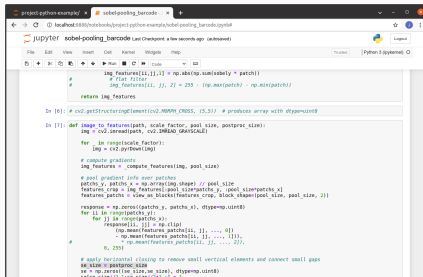
*Image horizontal/vertical gradients are computed by looking at the variation of the luminance when moving left-right/up-down.*

## Recommended approach

We encourage you to follow these steps:

1. Compute (absolute) horizontal and vertical derivatives for each pixel 3x3 neighborhood using Sobel.
2. Divide the image into patches and compute the mean gradient values (H and V) over each patch.
3. Compute some simple patch “barcodeness” indicator by taking the difference of the mean horizontal gradients and the mean vertical gradients: good patch candidates have a large amount of horizontal gradients and a little amount of vertical gradients.
4. Apply some post-processing to remove small activations and connect large ones horizontally using a (grayscale) morphological closing using a horizontal rectangular (3x5 for instance) structuring element.
5. Threshold the final activation map, using 50% of the maximal value of the activation map.
6. Process the activation map on CPU to detect barcodes (hint: use connected components detection).
7. Package everything in a nice CLI tool, so you can process our images during the demo.
8. Find performance bottlenecks, fix them, document the improvements.
9. Go further: try “hit or miss” filters instead of Sobel, try various parameter values (Sobel aperture, patch size, post-processing...), experiment with various CUDA parameters (block size, work per thread), try to detect connected components on the GPU, or even rectangle coordinates, package everything in a nice demo which processes webcam frames that we will showcase during JPOs...

# Sample Python notebook



```
project-python-examples/ x sobel-pooling_barcode x  
localhost:8888/notebooks/project-python-examples/sobel-pooling_barcode.ipynb  
jupyter sobel-pooling_barcode Last Opened a few seconds ago (Idle) Logout  
File Edit View Insert Cell Kernel Help Python 3 (ipykernel)  
In [6]: # cv2.getStructuringElement(cv2.MORPH_CROSS, (2,2)) # produces array with dtype=uint8  
In [7]: def image_to_features(path, scale_factor, pool_size, postproc_size):  
    img = cv2.imread(path, cv2.IMREAD_GRAYSCALE)  
    for _ in range(scale_factor):  
        img = cv2.pyrDown(img)  
        # compute gradients  
        img_features = compute_features(img, pool_size)  
        # pool gradient info over patches  
        patches_y, patches_x = np.arange(img.shape[0] // pool_size)  
        features_crop = img_features[pool_size*patches_y, :pool_size*patches_x]  
        features_patches = view_as_blocks(features_crop, block_shape=(pool_size, pool_size, 2))  
        responses = np.zeros((patches_y, patches_x, dtype=np.uint8))  
        for ii in range(patches_y):  
            for jj in range(patches_x):  
                responses[ii, jj] = np.clip(  
                    np.mean(features_patches[ii, :pool_size, :]), 0, 1),  
                    np.mean(features_patches[ii, :pool_size, :]), 1),  
                    np.mean(features_patches[ii, :pool_size, :]), 2),  
                    0, 255)  
    # apply horizontal closing to remove small vertical elements and connect small gaps  
    np_size = postproc_size  
    cv = np.zeros((n, n, n), dtype=np.uint8)  
    return cv
```

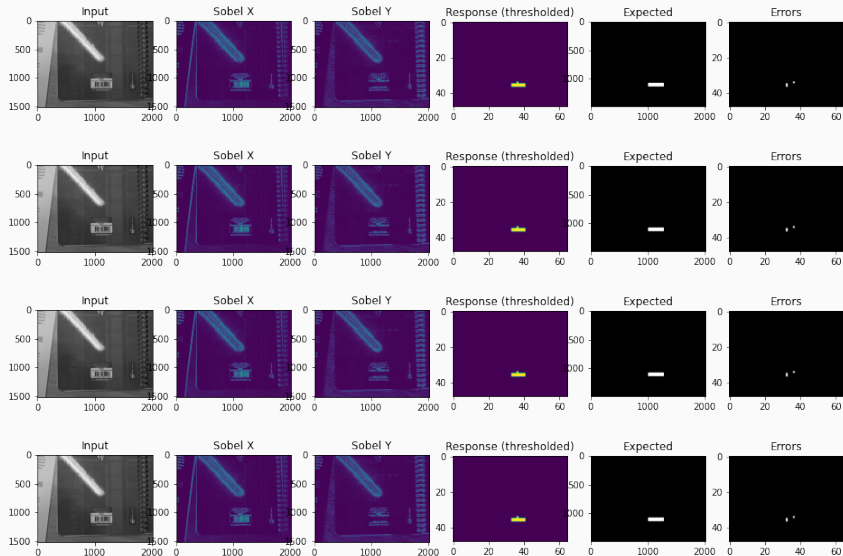
ipynb version:

[https://www.lrde.epita.fr/~carlinet/cours/GPGPU/barcode\\_sobel/sobel-pooling\\_barcode.ipynb](https://www.lrde.epita.fr/~carlinet/cours/GPGPU/barcode_sobel/sobel-pooling_barcode.ipynb)

HTML version:

[https://www.lrde.epita.fr/~carlinet/cours/GPGPU/barcode\\_sobel/sobel-pooling\\_barcode.html](https://www.lrde.epita.fr/~carlinet/cours/GPGPU/barcode_sobel/sobel-pooling_barcode.html)

## Full pipeline

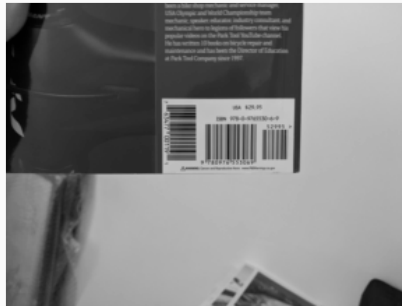


## Inputs

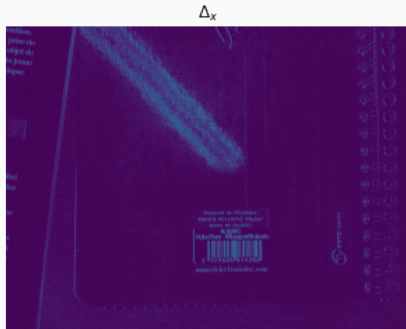
Sample input 1



Sample input 2



## Horizontal gradients



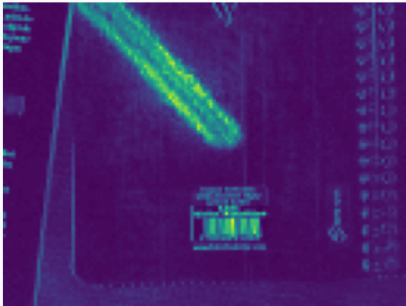
## Vertical gradients





Pooling of horizontal gradients over patches

$\langle \Delta_x \rangle$

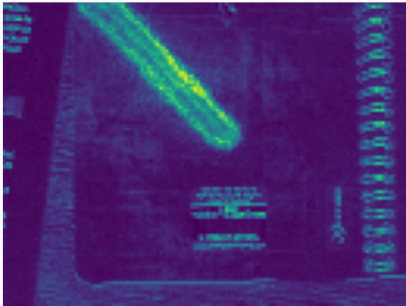


$\langle \Delta_x \rangle$

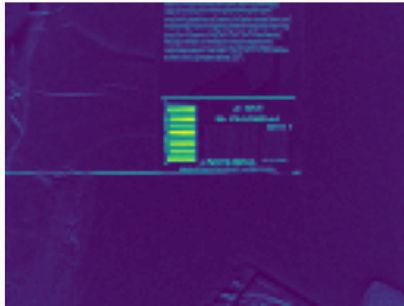


Pooling of vertical gradients over patches

$\langle \Delta_x \rangle$

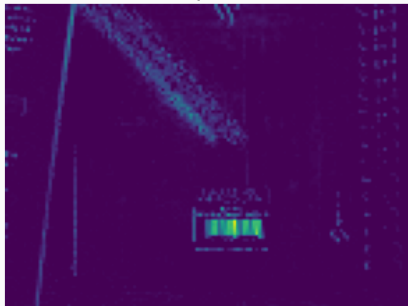


$\langle \Delta_x \rangle$

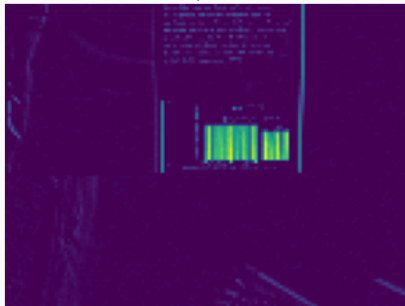


$$\text{Response } R = \Delta_x - \Delta_y$$

Response

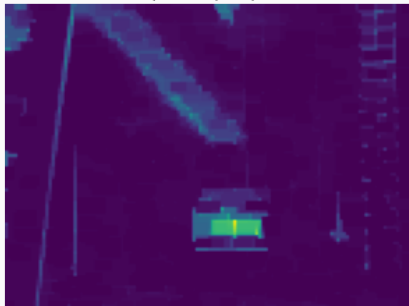


Response

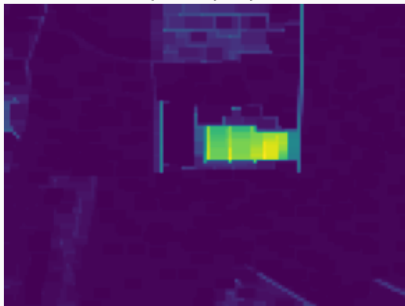


Post-processing  $R' = \epsilon(\delta(R))$

Response (postproc)



Response (postproc)



Final result  $R' > 0.5 * \max(R')$

Response (thresholded)



Response (thresholded)



## BONUS: End to end pipeline with Streams



1. Have an end to end, real-time video processing pipeline that detects barcodes (JPO demo).
2. Capture a video with your smartphone/webcam and process it (show the framerate!)

1. Get a working reference CPU (C++ or Python) version
2. Have a simple/slow port of your baseline to GPU/CUDA
3. Benchmark their respective speed and compute relevant indicators (occupancy, L1/L2 cache hit rates...)
4. Identify the performance bottleneck(s)
  - **Minimum expected work reached at this point** —
5. Perform more CUDA optimizations, measures, etc.
6. Experiment with algorithm variants, packaging, etc.

Collaborative dataset for photos and videos:

[\*https://cloud.lrde.epita.fr/s/Y36oXiYJ77ezytC\*](https://cloud.lrde.epita.fr/s/Y36oXiYJ77ezytC)

Each **group** have to upload video/photos of barcodes:

1. 1 video (1080p or 4K, < 20s)
2. 5 photos (16-24 Mpix, JPEG)
3. 5 ground truth files for photos (PNG with white color for regions with a barcode, black elsewhere)

You should try your algorithm on the simplest possible data to begin.



## Implementation Hints (Final Reminders)

- Have a working (*slow*) C++ reference implementation first (and keep it forever)
- Tag (*git tag*) the versions of your program before any optimization (useful to track and benchmark ideas)
- Try optimizations step by step so that you can tell which ones are the most important