# Kernel Validation

Verify just once that the kernel operations are correctly performed, which is crutial when working with large amounts of data. This consists in a comparison between the result of the same operation done both in CPU and GPU.

We would need to add a function that does that same pixel operation in the host, to continue the same examples with image processing, for example. This is basically making the kernel function but in the host.

Validation: execute the same kernel operations in the host and compare the results given by the CPU and GPU.

## Exercise

Execute a kernel validation for the last kernel where we made the complement of an RGB image. The kernel validation includes two functions: one for performing the complement in the CPU and one for comparing the CPU image and the GPU image.

### Solution

```
1  #include "cuda_runtime.h"
2  #include "device_launch_parameters.h"
3
4  #include <stdio.h>
5  #include <stdlib.h>
6  #include <opencv2/opencv.hpp>
7
8  using namespace cv;
9
10 __host__ void checkCUDAError(const char* msg) {
11     cudaError_t error;
12     cudaDeviceSynchronize();
13     error = cudaGetLastError();
14     if (error != cudaSuccess) {
15         printf("ERROR %d: %s (%s)\n", error,
               cudaGetErrorString(error), msg);
16     }
17 }
18
19 __global__ void complement(uchar* RGB) {
20
21     // locate my current block row
22     int threads_per_block = blockDim.x * blockDim.y;
23     int threads_per_row = threads_per_block * gridDim.x;
24     int row_offset = threads_per_row * blockIdx.y;
```

```
25
26      // locate my current block column
27      int block_offset = blockIdx.x * threads_per_block;
28      int threadId_inside = blockDim.x * threadIdx.y + threadIdx
            .x;
29
30      // locate my current grid row
31      int thread_per_grid = (gridDim.x * gridDim.y *
            threads_per_block);
32      int gridOffset = blockIdx.z * thread_per_grid;
33
34      int gId = gridOffset + row_offset + block_offset +
            threadId_inside;
35      RGB[gId] = 255 - RGB[gId];
36  }
37
38  __host__ void complementCPU(Mat* original, Mat* comp) {
39      for (int i = 0; i < original->rows; i++) {
40          for (int j = 0; j < original->cols; j++) {
41              comp->at<Vec3b>(i, j)[0] = 255 - original->at<
                    Vec3b>(i, j)[0];
42              comp->at<Vec3b>(i, j)[1] = 255 - original->at<
                    Vec3b>(i, j)[1];
43              comp->at<Vec3b>(i, j)[2] = 255 - original->at<
                    Vec3b>(i, j)[2];
44          }
45      }
46  }
47
48  __host__ bool validationKernel(Mat img1, Mat img2) {
49      Vec3b* pImg1, * pImg2;
50      for (int k = 0; k < 3; k++) {
51          for (int i = 0; i < img1.rows; i++) {
52              pImg1 = img1.ptr<Vec3b>(i);
53              pImg2 = img2.ptr<Vec3b>(i);
54              for (int j = 0; j < img1.cols; j++) {
55                  if (pImg1[j][k] != pImg2[j][k]) {
56                      printf("Error at kernel validation\n");
57                      return true;
58                  }
59              }
60          }
61      }
62      printf("Kernel validation successful\n");
63      return false;
64  }
65
66  int main() {
67
68      Mat img = imread("antenaRGB.jpg");
69
```

```
70      const int R = img.rows;
71      const int C = img.cols;
72
73      Mat imgComp(img.rows, img.cols, img.type());
74      Mat imgCompCPU(img.rows, img.cols, img.type());
75      uchar* host_rgb, * dev_rgb;
76      host_rgb = (uchar*)malloc(sizeof(uchar) * R * C * 3);
77
78      cudaMalloc((void**)&dev_rgb, sizeof(uchar) * R * C * 3);
79      checkCUDAError("Error at malloc dev_r1");
80
81      // matrix as vector
82      for (int k = 0; k < 3; k++) {
83          for (int i = 0; i < R; i++) {
84              for (int j = 0; j < C; j++) {
85                  Vec3b pix = img.at<Vec3b>(i, j);
86
87                  host_rgb[i * C + j + (k * R * C)] = pix[k];
88
89              }
90          }
91      }
92      cudaMemcpy(dev_rgb, host_rgb, sizeof(uchar) * R * C * 3,
           cudaMemcpyHostToDevice);
93      checkCUDAError("Error at memcpy host_rgb -> dev_rgb");
94
95      dim3 block(32, 32);
96      dim3 grid(C / 32, R / 32, 3);
97
98      complement << < grid, block >> > (dev_rgb);
99      cudaDeviceSynchronize();
100     checkCUDAError("Error at kernel complement");
101
102     cudaMemcpy(host_rgb, dev_rgb, sizeof(uchar) * R * C * 3,
           cudaMemcpyDeviceToHost);
103     checkCUDAError("Error at memcpy host_rgb <- dev_rgb");
104
105     for (int k = 0; k < 3; k++) {
106         for (int i = 0; i < R; i++) {
107             for (int j = 0; j < C; j++) {
108                 imgComp.at<Vec3b>(i, j)[k] = host_rgb[i * C +
                       j + (k * R * C)];
109             }
110         }
111     }
112
113     complementCPU(&img, &imgCompCPU);
114     bool error = validationKernel(imgCompCPU, imgComp);
115
116     if (error) {
117         printf("Check kernel operations\n");
```

```
118             return 0;
119         }
120 
121 
122         imshow("Image", img);
123         imshow("Image Complement CPU", imgCompCPU);
124         imshow("Image Complement GPU", imgComp);
125         waitKey(0);
126 
127         free(host_rgb);
128         cudaFree(dev_rgb);
129 
130         return 0;
131 }
```