Practice

- The gId is calculated from our configs, and therefore inactive threads do not interfere in the calculation of gId's. If you launch 2 blocks with 40 threads, even if you use 4 warps, the thread gIds will go from 0 to 79, for example.

Exercise

Now we will process our image of 768 x 448 pixels, divided into 14,24 blocks. For now, the image is read as grayscale.

Solution

```
#include "cuda_runtime.h"
#include "device_launch_parameters.h"

#include <stdio.h>
#include <stdlib.h>
#include <opencv2/opencv.hpp>

__host__ void checkCUDAError(const char* msg) {
    cudaError_t error;
    cudaDeviceSynchronize();
    error = cudaGetLastError();
    if (error != cudaSuccess) {
        printf("ERROR %d: %s (%s)\n", error,
           cudaGetErrorString(error), msg);
    }
}

__global__ void complement(uchar* dev_a, uchar* dev_b) {
    // locate my current block row
    int threads_per_block = blockDim.x * blockDim.y;
    int threads_per_row = threads_per_block * gridDim.x;
    int row_offset = threads_per_row * blockIdx.y;

    // locate my current block column
    int block_offset = blockIdx.x * threads_per_block;
    int threadId_inside = blockDim.x * threadIdx.y + threadIdx
       .x;

    int gId = row_offset + block_offset + threadId_inside;
    dev_b[gId] = 255 - dev_a[gId];
}

using namespace cv;
int main() {
```

```
33
34      Mat img = imread("antenaParalelo.jpg", IMREAD_GRAYSCALE);
35
36      const int R = img.rows;
37      const int C = img.cols;
38
39      Mat imgResult(img.rows, img.cols, img.type());
40      uchar* host_a, * host_b, * dev_a, * dev_b, * pImg;
41      host_a = (uchar*)malloc(sizeof(uchar) * R * C);
42      host_b = (uchar*)malloc(sizeof(uchar) * R * C);
43      cudaMalloc((void**)&dev_a, sizeof(uchar) * R * C);
44      checkCUDAError("Error at malloc dev_a");
45      cudaMalloc((void**)&dev_b, sizeof(uchar) * R * C);
46      checkCUDAError("Error at malloc dev_b");
47
48      // matrix as vector
49      for (int i = 0; i < R; i++) {
50          pImg = img.ptr<uchar>(i); // points to a row each time
51          for (int j = 0; j < C; j++) {
52              host_a[i * C + j] = pImg[j];
53          }
54      }
55      cudaMemcpy(dev_a, host_a, sizeof(uchar) * R * C,
           cudaMemcpyHostToDevice);
56
57      dim3 block(32, 32);
58      dim3 grid(C / 32, R / 32);
59
60      complement << < grid, block >> > (dev_a, dev_b);
61      checkCUDAError("Error at kernel");
62
63      cudaMemcpy(host_b, dev_b, sizeof(uchar) * R * C,
           cudaMemcpyDeviceToHost);
64
65      for (int i = 0; i < R; i++) {
66          pImg = imgResult.ptr<uchar>(i);
67          for (int j = 0; j < C; j++) {
68              pImg[j] = host_b[i * C + j];
69          }
70      }
71
72      imshow("Image", img);
73      imshow("Image Result", imgResult);
74      waitKey(0);
75
76      free(host_a);
77      free(host_b);
78      cudaFree(dev_a);
79      cudaFree(dev_b);
80
81      return 0;
```

```
82 }
```