Ejercicio 2   Parte 2/2 (Código en examen)

En total son 753,664 hilos

Y la máxima capacidad de hilos por bloque es 1024
Por lo que tendríamos 736 bloques en total.

Agregaríamos al código:
num_SM * max_blocks_per_SM y max_threads_per_block

```
#include "cuda_runtime.h"
#include "device_launch_parameters.h"
#include <stdio.h>
#include <stdlib.h>

#define GPUErrorAssertion(ans) { gpuAssert((ans), __FILE__, __LINE__); }
inline void gpuAssert(cudaError_t code, const char* file, int line, bool abort = true) {
   if (code != cudaSuccess) {
      fprintf(stderr, "GPUassert: %s %s %d\n", cudaGetErrorString(code), file, line);
      if (abort) exit(code);
   }
}

__global__ void imageThreshold(unsigned char* a, unsigned char* b, unsigned char*
c, int N) {
   int idx = threadIdx.x + blockIdx.x * blockDim.x;
   if (idx < N) {
      c[idx] = a[idx] | b[idx];
   }
}

int main() {
   const int N = 1024;
   const int dataSize = N * sizeof(unsigned char);

   unsigned char* A_cpu;
   unsigned char* B_cpu;
   unsigned char* C_cpu;

   unsigned char* A_gpu;
   unsigned char* B_gpu;
   unsigned char* C_gpu;

   A_cpu = (unsigned char*)malloc(dataSize);
```

```
    B_cpu = (unsigned char*)malloc(dataSize);
    C_cpu = (unsigned char*)malloc(dataSize);

    for (int i = 0; i < N; i++) {
        A_cpu[i] = rand() % 127;
        B_cpu[i] = rand() % 127;
    }

    GPUErrorAssertion(cudaMalloc((void**)&A_gpu, dataSize));
    GPUErrorAssertion(cudaMalloc((void**)&B_gpu, dataSize));
    GPUErrorAssertion(cudaMalloc((void**)&C_gpu, dataSize));

    GPUErrorAssertion(cudaMemcpy(A_gpu, A_cpu, dataSize,
cudaMemcpyHostToDevice));
    GPUErrorAssertion(cudaMemcpy(B_gpu, B_cpu, dataSize,
cudaMemcpyHostToDevice));

    int maxThreadsPerBlock;
    cudaDeviceProp prop;
    cudaGetDeviceProperties(&prop, 0);
    maxThreadsPerBlock = prop.maxThreadsPerBlock;

    int threadsPerBlock = maxThreadsPerBlock;
    int blocksPerGrid = (N + threadsPerBlock - 1) / threadsPerBlock;

    imageThreshold<<<blocksPerGrid, threadsPerBlock>>>(A_gpu, B_gpu, C_gpu,
N);
    GPUErrorAssertion(cudaDeviceSynchronize());

    GPUErrorAssertion(cudaMemcpy(C_cpu, C_gpu, dataSize,
cudaMemcpyDeviceToHost));

    for (int i = 0; i < 10; i++) {
        printf("Result[%d] = %d\n", i, C_cpu[i]);
    }

    cudaFree(A_gpu);
    cudaFree(B_gpu);
    cudaFree(C_gpu);

    free(A_cpu);
    free(B_cpu);
    free(C_cpu);
    return 0;
}
```