Laboratorio uso de CUDA

En este laboratorio (cuenta como ejercicio) vamos a implementar dos programas acelerados con CUDA. La idea es que se familiaricen con esta tecnología para programar tarjetas de video.

Para la primera parte del laboratorio, sigan los pasos en An Even Easier Introduction to CUDA: https://developer.nvidia.com/blog/even-easier-introduction-cuda/

Para la segunda parte del laboratorio, vamos a implementar el algoritmo de cálculo de pi usando el algoritmo de Montecarlo.

En ambos casos, suban screenshots de su programa corriendo. Midan el tiempo usando time.

Cálculo de Pi

El método de Monte Carlo para el cálculo de Pi consiste en generar puntos aleatorios en un cuadrado de lado 2, y contar cuántos de estos puntos caen dentro de un círculo de radio 1. La proporción de puntos dentro del círculo respecto al total de puntos generados se puede usar para aproximar el valor de Pi. El código presentado está basado en el escrito por Barry Wilkinson y Patrick Rogers.

Algoritmo en C++

Primero vamos a empezar con el algoritmo en C++ básico.

```
#include <stdlib.h>
#include <stdio.h>
#include <cuda.h>
#include <math.h>
#include <time.h>

#define PI 3.1415926535  // known value of pi

float host_monte_carlo(long trials) {
    float x, y;
    long points_in_circle;
    for(long i = 0; i < trials; i++) {
        x = rand() / (float) RAND_MAX;
        y = rand() / (float) RAND_MAX;
        points_in_circle += (x*x + y*y <= 1.0f);
    }
    return 4.0f * points_in_circle / trials;
}
```

```
int main (int argc, char *argv[]) {
    clock_t start, stop;

    start = clock();
    float pi_cpu = host_monte_carlo(BLOCKS * THREADS *
TRIALS_PER_THREAD);
    stop = clock();
    printf("CPU pi calculated in %f s.\n", (stop-
start)/(float)CLOCKS_PER_SEC);

    printf("CPU estimate of PI = %f [error of %f]\n", pi_cpu,
pi_cpu - PI);

    return 0;
}
```

Compile el programa usando GCC y asegúrese que funcione.

Ahora, investiguen cómo generar números aleatorios en CUDA e implementen el cálculo de PI usando CUDA. Básense en el siguiente código:

```
// Written by Barry Wilkinson, UNC-Charlotte. Pi.cu  December 22,
2010.
//Derived somewhat from code developed by Patrick Rogers, UNC-C

#include <stdlib.h>
#include <stdio.h>
#include <cuda.h>
#include <math.h>
#include <time.h>
#include <curand_kernel.h>

#define TRIALS_PER_THREAD 4096
#define BLOCKS 256
#define THREADS 256
#define PI 3.1415926535  // known value of pi

__global__ void gpu_monte_carlo(float *estimate, curandState
*states) {
     unsigned int tid = threadIdx.x + blockDim.x * blockIdx.x;
     int points_in_circle = 0;
     float x, y;

      // Initialize CURAND
```

```
     for(int i = 0; i < TRIALS_PER_THREAD; i++) {
          // TODO: genere valores aleatorios uniformes para x, y
          x = 1;
          y = 1;
          points_in_circle += (x*x + y*y <= 1.0f); // count if x &
y is in the circle.
     }
     estimate[tid] = 4.0f * points_in_circle / (float)
TRIALS_PER_THREAD; // return estimate of pi
}

float host_monte_carlo(long trials) {
     float x, y;
     long points_in_circle;
     for(long i = 0; i < trials; i++) {
          x = rand() / (float) RAND_MAX;
          y = rand() / (float) RAND_MAX;
          points_in_circle += (x*x + y*y <= 1.0f);
     }
     return 4.0f * points_in_circle / trials;
}

int main (int argc, char *argv[]) {
     clock_t start, stop;
     float host[BLOCKS * THREADS];
     float *dev;
     curandState *devStates;

     printf("# of trials per thread = %d, # of blocks = %d, # of
threads/block = %d.\n", TRIALS_PER_THREAD,
BLOCKS, THREADS);

     start = clock();

     // TODO: use cudaMalloc para pedir un bloque de memoria de
tamaño BLOCKS * THREADS * sizeof(float). Guárdelo en dev.
     // TODO: use cudaMalloc para pedir un bloque de memoria de
tamaño THREADS * BLOCKS * sizeof(curandState). Será usado para
guardar el estado del generador de números aleatorios. Guardelo en
devStates.

     gpu_monte_carlo<<<BLOCKS, THREADS>>>(dev, devStates);

     // TODO: Use cudaMemcpy para copiar los datos de dev a host.

     float pi_gpu;
```

```
     for(int i = 0; i < BLOCKS * THREADS; i++) {
          pi_gpu += host[i];
     }

     pi_gpu /= (BLOCKS * THREADS);

     stop = clock();

     printf("GPU pi calculated in %f s.\n", (stop-
start)/(float)CLOCKS_PER_SEC);

     start = clock();
     float pi_cpu = host_monte_carlo(BLOCKS * THREADS *
TRIALS_PER_THREAD);
     stop = clock();
     printf("CPU pi calculated in %f s.\n", (stop-
start)/(float)CLOCKS_PER_SEC);

     printf("CUDA estimate of PI = %f [error of %f]\n", pi_gpu,
pi_gpu - PI);
     printf("CPU estimate of PI = %f [error of %f]\n", pi_cpu,
pi_cpu - PI);

     return 0;
}
```