Práctica 4 – Multiplicación de vectores

Fecha: 03/5/2022

Grupo: 4CDV1

Equipo: NetPower

Integrantes:

Alcibar Zubillaga Julián
De Luna Ocampo Yanina
Salinas Velazquez Jacob

Objetivos

Parte 1: Con base en el código de la práctica 4, donde se suman dos vectores en el GPU, modificarlo para que realice un producto punto de dichos vectores en paralelo.

Escenario

Los hilos (threads) en CUDA constituyen la base de la ejecución de aplicaciones en paralelo. Cuando se lanza un kernel se crea una malla de hilos donde todos ellos ejecutan el mismo programa. Es decir, el kernel especifica las instrucciones que van a ser ejecutadas por cada hilo individual.

En esta práctica se va a hacer uso del paralelismo a nivel de hilo que proporciona CUDA lanzando varios hilos.

Nota: Se utilizará un servidor externo para realizar la práctica. Es necesario contar con un programa cliente ssh y credenciales de acceso al servidor externo.

Recursos necesarios para realizar la práctica

- 1 computadora personal
- Acceso a internet
- 1 programa cliente SSH* (Putty: https://www.chiark.greenend.org.uk/~sgtatham/putty/latest.html)

INTRODUCCIÓN

Los threads (hilos) constituyen la base de la ejecución de aplicaciones en paralelo. Cuando se lanza un kernel se crea una malla de hilos donde todos ellos ejecutan el mismo programa, es decir, el kernel específica las instrucciones que van a ser ejecutadas por cada hilo individual [2]. Recordemos que cada thread recorre una única fila de la malla de principio a fin ejecutando el algoritmo. La fila sobre la que se ejecuta cada thread (hilo) se determina por la variable interna threadidx.y, estos, se sincronizan al comenzar y finalizar el recorrido, su inconveniente es que los que comparten una celda vecina actualizan al mismo tipo de celdas y al mismo tiempo.

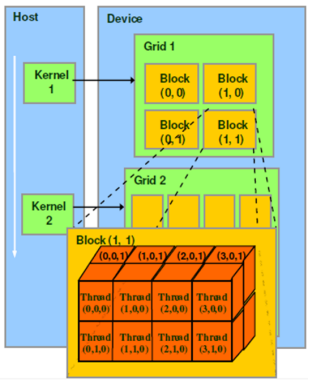
Actualmente la mayoría de los simuladores son modelos en donde cada celda lee el estado de las vecinas, computan el nuevo estado de la celda y lo actualizan en memoria auxiliar. Estos simuladores no tienen problemas de concurrencia al ser paralelizados, independientemente de la tecnología aplicada.

Se propuso una solución, la cual es que se implementan índices desfasados sobre memoria compartida que evita errores de concurrencia cuando varias threads escriben el mismo dato. Los threads empiezan a recorrer la fila desde un punto intermedio y al llegar al final saltan al inicio para completar el recorrido. Esta solución, es factible gracias a que los threads son estrictamente paralelos en una arquitectura SIMD y que el dominio se definió como una malla regular. [1]

Los hilos (threads) en CUDA constituyen la base de la ejecución de aplicaciones en paralelo. Cuando se lanza un kernel se crea una malla de hilos donde todos ellos ejecutan el mismo programa. Es decir, el kernel especifica las instrucciones que van a ser ejecutadas por cada hilo individual.

Cuando especificamos que queremos una malla formada por un único bloque y N hilos paralelos. Esto le dice al sistema que queremos una malla unidimensional. Los hilos en una malla se organizan en dos niveles. Por simplicidad se ha representado un número muy reducido de hilos. En el nivel superior los hilos se organizan en bloques. Cada malla está formada por uno o varios bloques de hilos. Todos los bloques de una malla tienen el mismo tamaño. En CUDA, cada bloque está unívocamente identificado mediante las palabras “blockIdx.x” y “blockIdx.y”. Por otro lado, todos los bloques deben contener el mismo número de hilos y además deben estar organizados de la misma forma.

Cada bloque se organiza en una matriz tridimensional con un total máximo de 512 hilos por bloque. Las coordenadas de los hilos en un bloque están definidas mediante los índices “threadIdx.x”, “threadIdx.y”, y “threadIdx.z”. Sin embargo, si uno de los índices fuera constante e igual a uno daría una matriz bidimensional. [2]



Parte 1: Correr y analizar el programa 04_memo.cu

Utiliza un cliente SSH para conectarte al servidor lambda 01.

(te puedes conectar al servidor lambda de la misma forma que lo hiciste en la práctica 01).

Dentro del servidor Lambda. Acceder a la carpeta “hpc_4AV1

Accedemos a nuestra carpeta de trabajo y hacemos una copia del programa 06_sumavect.cu a un nuevo archivo llamado 07_prodvect.cu, ejecuta el siguiente comando:

```
/hpc_4AV1/estudiantes/<tu_carpeta_de_trabajo>/u01/# cp 06_sumavect.cu 07_prodvect.cu
```

<tu_carpeta_de_trabajo> es el nombre de tu carpeta de trabajo. Este comando realizará una copia del programa “06_sumavect.cu” en un archivo llamado 07_prodvect.cu

Se usará el mismo programa de la práctica 04 pero, en lugar de sumar los valores dentro del arreglo se multiplicarán. También se deberá agregar una función al final para sumar todos los valores del arreglo resultante.

Modifica las siguientes opciones:

1 Producto en el CPU

Modificaremos la función que suma los vectores por la siguiente que multiplica los valores en el CPU

```
void productoCPU(int* a, int* b, int *c)
{
   int i;
   for(i=0; i<N; i++)
   {
       c[i] = a[i] * b[i];
   }
}
```

2 Producto en el GPU

Se hará lo mismo en la función que suma los dos vectores en el GPU:

```
__global__ void productoGPU(int* a, int* b, int* c)
{
  int id;
  id = threadIdx.x + (blockIdx.x*blockDim.x);

  c[id] = a[id] * b[id];

}
```

3 sumar los valores dentro del arreglo

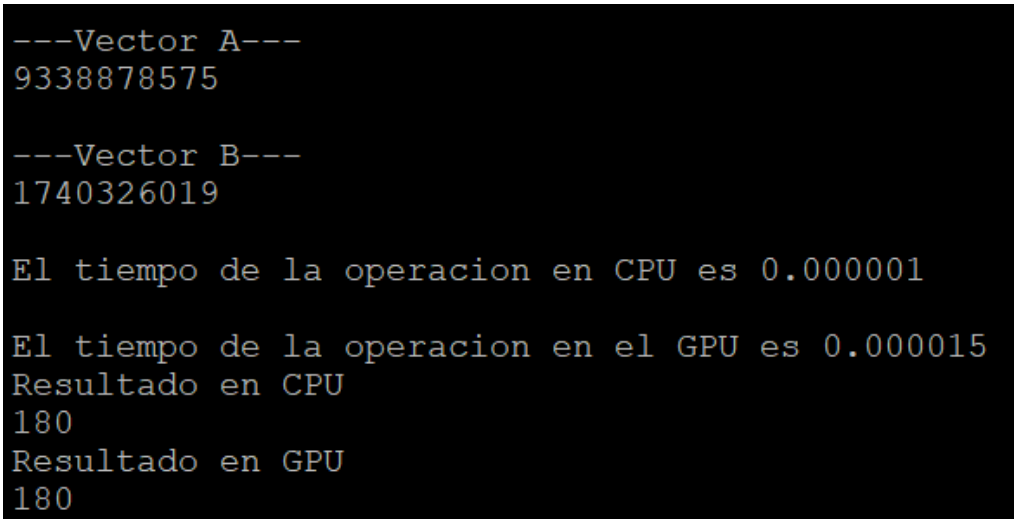
Finalmente realiza una función que sume todos los valores en el arreglo resultante, el resultado es un escalar.

Compila y ejecuta el programa, usa los siguientes comandos:

```
nvcc 06_prodvect.cu -o 06_prodvect.x
```

./06_prodvect.x

INCLUYE AQUÍ LA CAPTURA DE PANTALLA CON EL RESULTADO DEL PROGRAMA EJECUTÁNDOSE



Captura de pantalla con la ejecución del programa

Incluye aquí el código completo:

```
//Inicio del código
```

```
  GNU nano 4.8
// includes
#include <stdio.h>
#include <cassert>
#include <stdlib.h>
#include <ctime>
#include <time.h>
#include <cuda_runtime.h>
#define N 10

__global__ void productoGPU(int* a, int* b, int* c)
{
  int id;
  id = threadIdx.x + (blockIdx.x * blockDim.x);
  c[id] = a[id] * b[id];

}

int suma(int* c){
  int temp = 0;
  for(int i=0; i < N; i++){
    temp += c[i];
  }
  return temp;
}

int dotproductCPU(int* a, int* b)
{
   int i;
   int* c = (int*)malloc(sizeof(int)*N);

   for(i=0; i<N; i++)
   {
       c[i] = a[i] * b[i];
   }
   return suma(c);
}

void rellena(int* arreglo){
   int i;
   for(i = 0;i < N;i++){
        arreglo[i] = rand()%10;
   }
}
```

```
void imprime(int* arreglo){
   for(int i = 0; i < N; i++){
        printf("%d",arreglo[i]);
   }
   printf("\n");
}

// MAIN: rutina principal ejecutada en el  CPU
int main(int argc, char** argv)
{
// declaracion
unsigned t0, t1;
int bloque, procesos, productoc, productog;
int* s1, *s2, *s1_g, *s2_g, *suma_g, *suma_h;

//reserva memoria CPU
s1 = (int*) malloc(sizeof(int)*N);
s2 = (int*)malloc(sizeof(int)*N);
suma_h = (int*)malloc(sizeof(int)*N);
// reserva en el GPU
cudaMalloc( (void**)&s1_g, sizeof(int)*N);
cudaMalloc((void**)&s2_g, sizeof(int)*N);
cudaMalloc((void**)&suma_g, sizeof(int)*N);
//rellenamos los vectores
srand(time(0));
rellena(s1);
rellena(s2);
printf("\n---Vector A---\n");
imprime(s1);
printf("\n---Vector B---\n");
imprime(s2);
//Llamamos a la funcion en el CPU
t0 = clock();
productoc = dotproductCPU(s1,s2);
t1 = clock();
printf("\nEl tiempo de la operacion en CPU es %f\n", double(t1-t0)/CLOCKS_PER_SEC);
//enviar vectores al GPU
cudaMemcpy(s1_g,s1, sizeof(int) * N,cudaMemcpyHostToDevice);
cudaMemcpy(s2_g,s2, sizeof(int) * N,cudaMemcpyHostToDevice);
```

```
//Llamamos a la función en el GPU
procesos = 10;
bloque = N/procesos;
t0 = clock();
productoGPU<<<bloque,procesos>>>(s1_g,s2_g,suma_g);
t1 = clock();
printf("\nEl tiempo de la operacion en el GPU es %f\n",double(t1-t0)/CLOCKS_PER_SEC);

//copiamos el resultado de suma
cudaMemcpy(suma_h,suma_g, sizeof(int) * N,cudaMemcpyDeviceToHost);
productog = suma(suma_h);
//imprime resultado
printf("Resultado en CPU\n%d\n",productoc);
printf("Resultado en GPU\n%d\n",productog);

cudaFree(s1_g);
cudaFree(s2_g);
cudaFree(suma_g);

printf("\npulsa INTRO para finalizar...");
fflush(stdin);

char tecla = getchar();
return 0;

}

//Fin del código
```

Parte 2: Cuestionario

¿Podrías paralelizar la función para sumar los resultados del arreglo?

No

¿Por qué? (justifica la respuesta de la pregunta anterior)

Porque si analizamos la fórmula de suma, que recordándola es: temp = temp + c[i], inicializando previamente temp en 0, vemos que necesitamos el valor anterior para poder seguir iterándola.

Conclusiones

En esta práctica retomamos el tema de hilos y bloques, recordando que estos constituyen la base de la ejecución de aplicaciones en paralelo. En esta práctica aplicaremos el uso del paralelismo a nivel de hilo, recordamos que este es un procesador multi-hilo que los distintos hilos de ejecución comparten las unidades funcionales del procesador lo que hace necesaria su replicación.

De este se distinguen tres aproximaciones: multi-hilo de grano no, multi-hilo de grano grueso y multi-hilo simultáneo. En el multi-hilo de grano no se alterna entre hilos en cada una de las instrucciones. En el multi-hilo de grano grueso solamente se producen alternancias en las detenciones largas. El multi-hilo simultáneo permite la ejecución de instrucciones de varios hilos de forma simultánea.

En esta práctica pusimos en marcha este concepto de forma que completamos nuestro aprendizaje teórico con el práctico, logrando entender de forma completa el tema.

Bibliografía

1. "Threads y Bloques - Implementación paralela en CUDA". 1Library.Co - plataforma para compartir documentos. https://1library.co/article/threads-y-bloques-implementación-paralela-en-cuda.q0294rly (accedido el 23 de abril de 2022).
2. "Organización de hilos en CUDA - PRÁCTICA nº 4: Hilos y Bloques". 1Library.Co - plataforma para compartir documentos. https://1library.co/article/organización-hilos-cuda-práctica-nº-hilos-bloques.z1dxg53z (accedido el 23 de abril de 2022).

Consideraciones finales

Descarga el documento antes de llenarlo.

Este documento se debe llenar en equipo, aunque la práctica la deben hacer TODOS los integrantes de este.

Después de llenar el documento, guárdalo como PDF y envíalo a través de la plataforma TEAMS, en la pestaña de tareas correspondiente. Solamente lo tiene que subir uno de los integrantes. Pero deben incluir TODOS los nombres de los integrantes del equipo en la primera página.

Queda estrictamente prohibido cualquier tipo de plagio a otros equipos o grupos. En caso de que ocurra, se anulará la práctica y se descontarán dos puntos a los equipos involucrados.