Prueba GPU - 1

Fabián Trigo
Universidad de Valparaíso, Valparaíso
(Dated: Septiembre 12, 2023)

Prueba de GPU, se comeinza con unas preguntas sobre el codigo tratado

```
__global__ void vecAddKernel(float* A, float* B, float* C, int n){
    int i = threadIdx.x + blockDim.x * blockIdx.x * 2;
    if (i < n){ C[i] = A[i] + B[i] };
    i += blockDim.x;
    if (i < n){ C[i] = A[i] + B[i] };
}

int vecAdd(float* A, float* B, float* C, int n){
    // "n" es el n  umero  de elementos en A, B y C
    int size = n * sizeof(float);
    float* A_d; float* B_d; float* C_d;
    cudaMalloc((void **)&A_d, size);
    cudaMalloc((void **)&B_d, size);
    cudaMalloc((void **)&C_d, size);
    cudaMemcpy(A_d, A, size, cudaMemcpyHostToDevice);
    cudaMemcpy(B_d, B, size, cudaMemcpyHostToDevice);
    vecAddKernel<<<ceil(n / 2048.0), 1024>>>(A_d, B_d, C_d, n);
    cudaMemcpy(C, C_d, size, cudaMemcpyDeviceToHost);
}
```

Observando el codigo a simple vista, tenemos un vecAddKernel que toma los arrays A,B,C.

I. PREGUNTA 1

1. Si el tamaño de los arrays A, B y C es 50.000 elementos cada uno, ¿cuantos bloques de threads estan generados? Se generan en la linea: `vecAddKernel<<<ceil(n / 2048.0), 1024>>>(A_d, B_d, C_d, n);` Tengase en cuenta las cantidades `n=50000; ceil(n/2048) => 25` Osea que tendríamos asignados unos 25 bloques, cada bloque tendría unas 1024 threads por bloque.

II. PREGUNTA 2

2. Si el tama no de los arrays A, B y C es 50.000 elementos cada uno, ¿cu antos warps hay en cada bloque de threads?

En la arquitectura de Nvidia, cada bloque posee warps de 32 threads que se ejecutan de manera paralela Debido a que hay unos 1024 threads por bloque, tendriamos unos

$$\text{warp} = \frac{1024}{32} = 32$$

por bloque.

III. PREGUNTA 3

Si el tama no de los arrays A, B y C es 50.000 elementos cada uno, ¿cuantos threads en total tendr a la grilla?

La grilla es este conjunto completo de bloques y threads, para ello bastara con que multipliquemos la cantidad de threads por bloque por la cantidad de bloques

$$\text{threads totales} = \frac{\text{threads}}{\text{por bloque}} \cdot \frac{\text{bloques}}{=} 1024 \cdot 25 = 25600 \text{threads en grilla}$$

IV. PREGUNTA 4

¿Cuantos sumas calcula cada thread en el kernel?

Como podemos observar en las lineas 1 a 6 del codigo, al ingresar se encuentra el ‘threadIdx.x‘ asi como el ‘blockIdx.x‘, esto posiciona al thread en un lugar especifico de la memoria, es importante el hecho de que el ‘blockIdx.x‘ esta multiplicado por 2, quiere decir que es el tamaño de steps que damos en los bloques.

Cada thread ejecutara la funcion ‘voidAddKernel‘, lo que quiere decir que que tomara su index (threadId, blockDim) y el siguiente (threadId, blockDim+1), por tanto realizara 2 calculos como mucho.

V. PREGUNTA 5

Escribir otra version del codigo arriba tal que cada thread calcula 4 sumas y por lo tanto puede lanzar el kernel con la mitad del n umero de threads usado en el codigo arriba.

```
__global__ void vecAddKernel(float* A, float* B, float* C, int n){
    int i = threadIdx.x + blockDim.x * blockIdx.x * 4;
    if (i < n){ C[i] = A[i] + B[i] }; // (x,y)
    i += blockDim.x;
    if (i < n){ C[i] = A[i] + B[i] };} // (x,y+1)
    i += blockDim.x;
    if (i < n){ C[i] = A[i] + B[i] }; // (x,y+2)
    i += blockDim.x;
    if (i < n){ C[i] = A[i] + B[i] }; // (x,y+3)
}

int vecAdd(float* A, float* B, float* C, int n){
    int size = n * sizeof(float);
    float* A_d; float* B_d; float* C_d;
    cudaMalloc((void **)&A_d, size);
    cudaMalloc((void **)&B_d, size);
    cudaMalloc((void **)&C_d, size);
    cudaMemcpy(A_d, A, size, cudaMemcpyHostToDevice);
    cudaMemcpy(B_d, B, size, cudaMemcpyHostToDevice);
    // cada bloque se lanzaria con la mitad del numero de threads como es pedido
    // lo que equivaldria a tener 16 warps, con sus 32 threads corriendo en paralelo
    vecAddKernel<<<ceil(n / 2048.0), 512>>>(A_d, B_d, C_d, n);

    cudaMemcpy(C, C_d, size, cudaMemcpyDeviceToHost);
}
```

VI. PREGUNTA 6

El siguiente fragmento de c odigo contiene un error, indentifiquelo.

```
int main(){
    dim3 bloques (1024, 1024); // 1024 es el tama o maximo de bloques
    dim3 threads (1024, 1024); // el problema va aqui
    ...
    kernel2D<<<bloques, threads>>>();
    ...
}
```

Existe la limitacion de que por bloque no pueden haber mas de 1024 thread en total; osea que al multiplicar las dimensiones de los threads estas deben entregarnos un numero menor a 1024, en este caso la cantidad de threads por bloque

$$\text{threasd por bloque} = 1024 * 1024 = 1048576$$

para arreglar este problema hemos de cambiar el numero total de threads

```
int main(){
    dim3 bloques (1024, 1024); // 1024 es el tama o maximo de bloques
    dim3 threads (32, 32); // el total daria 1024
    ...
    kernel2D<<<bloques, threads>>>();
    ...
}
```