ARQUITECTURA DE COMPUTADORS
PRÀCTIQUES DE SUPERCOMPUTACIÓ

PRÀCTICA 2: INTRODUCCIÓ A LA PROGRAMACIÓ AMB CUDA

Objectius:

- Introduir els conceptes bàsics sobre la programació amb CUDA.
- Crear el nostre primer programa fent ús del CPU i del GPU.
- Fer una primera aproximació pràctica a la supercomputació.

Requisits:

Per a poder realitzar aquesta pràctica, cal disposar d'una gràfica NVIDIA que suporti CUDA. Actualment podeu comprovar si la vostra NVIDIA és compatible a:

https://developer.nvidia.com/cuda-gpus

La pràctica es farà amb Linux.

Es pot consultar la documentació de CUDA a http://docs.nvidia.com/cuda/index.html

1) Instruccions per instal·lar el CUDA SDK

NOTA: Als ordinadors de les aules de la UPF ja està instal·lat.

A la pàgina https://developer.nvidia.com/cuda-downloads heu de descarregar l'opció *.run* per a Ubuntu (el que tingueu a la vostra màquina).

Un cop està baixat el fitxer, canviem els permisos per a poder executar-lo:

chmod +x cuda_5.5.22_linux_64.run

I l'executem:

sudo sh cuda_5.5.22_linux_64.run

2) Configurem l'entorn per a CUDA

Ara cal que el sistema operatiu sàpiga on trobar els executables de CUDA i les seves llibreries. Per tant, cal que li ho indiquem amb les variables d'entorn.

Al vostre fitxer .bashrc o .profile, indiqueu on es troben les llibreries de CUDA afegint aquestes linies al final del fitxer:

```
export PATH=/home/usuari/cuda/bin/:$PATH
export LD_LIBRARY_PATH=/home/usuari/cuda/lib64:/home/usuari/cuda/lib
```

Un cop fet això, ja podem començar a utilitzar CUDA.

3) El nostre primer programa

Obriu el fitxer p2_ex0.cu (L'extensió dels vostres programes en CUDA serà .cu) I observeu el codi. Aquest és un programa senzill que realitza la suma de 2 nombres desde el vostre *"device"*.

Compileu el programa amb el compilador de CUDA:

```
nvcc p2_ex0.cu -o p2_ex0
```

I executem-lo:

```
./p2_ex0
```

Veurem que ens retorna per pantalla el resultat d'una suma:

```
2 + 3 = 5
```

4) Què fa aquest programa?

Anem a analitzar les parts importants del programa:

```
__global__ void add( int a, int b, int *c ) {
    *c = a + b;
}
```

Aquesta funció és la principal i és la que realment s'executarà a la tarja d'NVIDIA. La marca __global__ ens indica que és un codi que s'executarà al dispositiu CUDA i que es cridarà des del programa principal que s'executarà a la CPU de l'ordinador.
Aquesta funció té dos paràmetres d'entrada (a i b) i un de sortida (c). El paràmetre de sortida està passat amb un apuntador per poder-ne modificar el seu contingut.

```
int main( void ) {
    int c;
    int *dev_c;
```

Comencem el programa principal definint una variable c i un apuntador a la variable c que estarà al dispositiu. Tota la informació de l'ordinador al dispositiu i a l'inrevés es passarà a través de transferències de memòria. Per això ens cal saber l'apuntador de la variable al dispositiu.

```
cudaMalloc( (void**)&dev_c, sizeof(int) );
```

Reservem la memòria necessària per a una variable del tipus enter al dispositiu. Si ho féssim directament amb C al nostre ordinador hauríem fet servir la funció malloc per reservar la memòria a l'ordinador principal.. Per reservar-la al dispositiu, fem servir cudaMalloc.

```
    add<<<1,1>>>( 2, 3, dev_c );
```

Cridem la funció add definida abans i que s'executarà al dispositiu. Les marques <<< i >>> indiquen el nivell de paral·lelisme que necessitem. En aquest cas, només s'executarà una còpia de la funció.

```
    cudaMemcpy( &c, dev_c, sizeof(int), cudaMemcpyDeviceToHost );
```

Llegim el resultat de la funció. Aquest resultat es grava a la memòria del dispositiu. Amb la funció cudeMemcpy en copiem el contingut de l'apuntador dev_c (que apunta al dispositiu) a la zona de memòria de la variable c (que és a la memòria principal de l'ordinador).

```
    printf( "2 + 3 = %d\n", c );
```

Escrivim el resultat per pantalla.

```
    cudaFree( dev_c );
```

Alliberem la memòria del dispositiu que havíem reservat amb cudaMalloc.

```
    return 0;
}
```

5) Suma de Vectors amb el CPU

Compileu i executeu el programa p2_vector.c amb:

g++ p2_vector.c -o p2_vector_cpu

Comproveu que fa una suma senzilla de vectors.

6) Suma de vectors amb el GPU

Obriu el fitxer p2_vector.cu

EXERCICI 1:

Ompliu el codi que hi falta amb la informació donada a la classe de pràctiques de manera que:

1.1. S'utilitzi 1 sol bloc amb tants threads com elements ténen els vectors.
1.2. S'utilitzin tants blocs d'un sol thread com elements ténen els vectors.

Només caldrà entregar 1 sol fitxer amb aquestes dues implementacions, de manera que una quedi comentada (//).

a) Digueu quines són les diferencies entre aquestes dues implementacions.

EXERCICI 2:

Obriu i executeu el fitxer info.cu amb:

nvcc info.cu -o info

Veureu que sortirà per pantalla la informació relativa a la vostra NVIDIA. Màxim nombre de threads per bloc, màxim nombre de blocs, etc. Responeu a les següents preguntes:

a) Quin és el nom/model de la vostra NVIDIA?
b) Quin és el màxim nombre de threads per bloc que podeu utilitzar?
c) Quin és el màxim nombre de blocs?

EXERCICI 3:

Torneu al codi que heu implementat a l'exercici 1.1 i modifiqueu el nombre d'elements del vector per un valor superior al màxim nombre de threads que la vostra NVIDIA pot fer servir.

Torneu al codi que heu implementat a l'exercici 1.2 i modifiqueu el nombre d'elements del vector per un valor superior al màxim nombre de blocs que la vostra NVIDIA pot fer servir.

a) Que és el que passa en els 2 casos? A que creieu que és degut?

EXERCICI 4:

Creeu un nout fitxer anomenat p2_vector_long.cu que permeti fer sumes de vectors amb un nombre d'elements superior al nombre de blocs i nombre de threads màxims. Per a fer això, haureu de fer servir una combinació de blocs i threads de manera que:

Index del Thread + Index del Block * Dimensió del bloc

També haureu de modificar la crida al kernel.

Adjunteu el vostre codi a l'entrega.

7) Multiplicació de dues matrius

Obriu el fitxer p2_matrix.cu i mireu el codi. Cada thread que es llança dins d'un bloc està identificat amb les variables CUDA threadIdx.x, threadIdx.y i threadIdx.z. D'aquesta manera podem crear un bloc de procés i executar cada codi per separat. Per exemple, el programa p2_matrix.cu emplena una matriu bidimensional amb valors.

EXERCICI 5:

Modifiqueu el codi de manera que s'utilitzi la informació del grid enlloc de la informació del bloc.

EXERCICI 6:

Amb la modificació que heu fet a l'exercici 5:

a) Complementeu el programa de l'exercici anterior amb una funció que s'executarà al dispositiu:

*__global__ void matrix_mult_device(int *Ma, int *Mb, int *Mc, int width)*

b) Escriviu el codi necessari al programa principal per reservar la memòria al dispositiu, copiar-hi les dues matrius, fer-hi la multiplicació i copiar el resultat final al host.

Feu servir un grid de dimensió (1,1) i feu que els blocs tinguin la mateixa dimensió que les matrius.

c) Afegiu el codi que verifiqui que el resultat de la multiplicació al host i al device són idèntics.

Adjunteu el codi fet a l'entrega de la pràctica.

ENTREGA DE LA PRÀCTICA

Les pràctiques es realitzaran en grups de 3 persones.

La pràctica s'haurà de lliurar a l'aula global i haureu d'entregar un fitxer comprimit que contingui els fitxers de codi demanats amb el codi que heu afegit i amb els fitxers que heu creat i una memòria amb les vostres respostes en format pdf. Només un dels tres membres del grup ha de pujar la pràctica.

El nom del fitxer que pengeu a l'aula global:

GP_p2_NIA1_NIA2_NIA3.zip