PRÁCTICA 3.

PROGRAMACIÓN de GPUs UTILIZANDO CUDA

El propósito de este laboratorio es adquirir una experiencia mínima en la utilización del entorno de programación CUDA y desarrollar programas sencillos para sistemas heterogéneos compuestos por CPU+GPU.

Los equipos de los laboratorios tienen instalada una versión de CUDA donde podéis realizar esta práctica.

En estos equipos, en el directorio NVIDIA_CUDA-11.1_Samples/0_Simple, hay diversos ejemplos que podéis compilar y comprobar su funcionamiento (vectorAdd).

En el aula virtual también tenéis disponible una guía de programación de GPUs y los códigos de los ejemplos implementados en clase de problemas. Podéis descargar y comprobar su funcionamiento.

El desarrollo de la práctica consiste en la implementación de diferentes versiones de código para ejecutar sobre una GPU una operación básica del álgebra matricial, el producto matriz-vector. Se pretende implementar utilizando diferentes estrategías de programación de CUDA.

$$y = A * x,$$

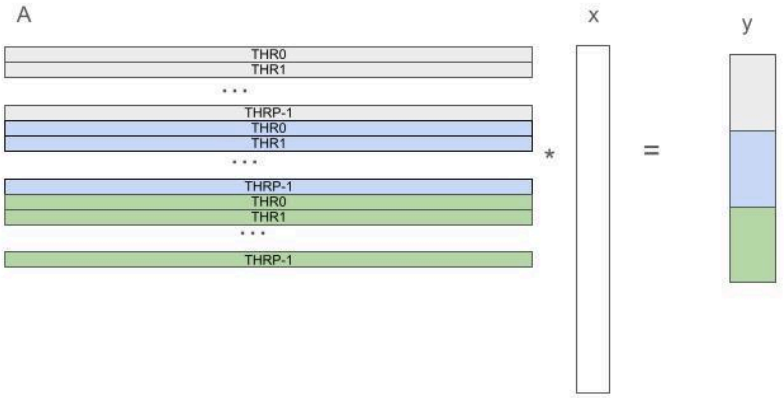
donde A es una matriz de n x m, x e y son vectores de m elementos que pueden tomar valores genéricos y establecidos en el momento de la ejecución. .

Código en secuencial y en procesador escalar:

```
for (i=0;i<n; i++)
    for (j=0;j<m;j++)
        y[i] = y[i] +x[j]* a[i*m+j];
```

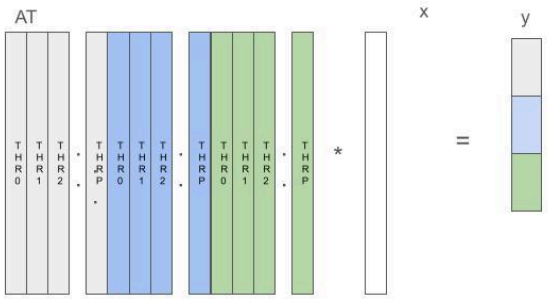
Ejercicios a implementar:

1. Implemente una versión básica donde el grid de bloques y los bloques son unidimensionales. La matriz, por tanto, se particiona en bloques de filas y se creen tantos threads como filas componen la matriz. Cada thread de cada uno de los bloques de threads que se crean calcula un elemento del vector final.

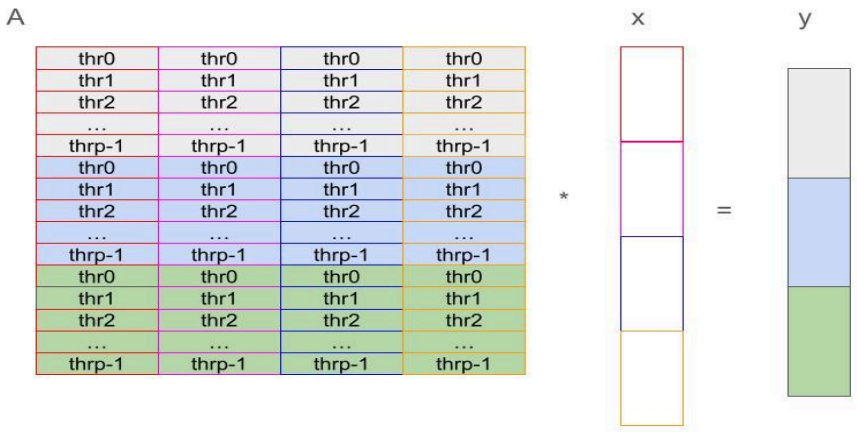


2. Modifica la versión anterior para que sea más eficiente utilizando memoria compartida de la GPU para reducir los accesos a memoria global que se realizan por parte de los threads de un bloque y, que estos accesos sean alineados (coalasced). Idea: trabajar con la matriz transpuesta.

.



3. Desarrolla una nueva versión, donde se implementa un paralelismo de grano más fino:
    a. El grid de bloques es bidimensional y los bloques son unidimensionales. La matriz se particiona en bloques.
    b. Cada thread de cada uno de los bloques calcula parte de la suma total de un elemento del vector resultante y. Para calcular el valor final habrá que realizar la suma de todas esas sumas parciales de forma atómica.



Cada elemento de y se calcula como una suma parcial en cada uno de los bloques de la dimensión x. Por ejemplo para el cálculo del elemento y[0], se calculará por partes entre todos los threads 0 de los bloques (0,0), (1,0), (2,0), y (3,0):
Suma de las sumas parciales calculadas en cada uno de los bloques de la matriz:

y[0] = sum_parcial(B[0,0]thr[0]) + sum_parcial[B[1,0]thr[0])+ sum_parcial(B[2,0]thr[0])+ suma_parcial(B[3,0]thr[0])

4. Modifica la versión anterior para hacer un uso más eficiente de la memoria, utilizando memoria compartida de la GPU para reducir los accesos a memoria global que se realizan por parte de los threads de un bloque y, que estos accesos sean alineados (coalasced).