Programación GPGPU

Procesamiento de datos a gran escala

Alejandro Cabana Suárez y Óscar Gómez Borzdynski

MUCD20-21

# Suma de 2 matrices

## Suma de vectores

En primer lugar, ejecutamos el código en serie, sin intervención de la tarjeta gráfica definiendo la función de suma sin el modificador *\_\_global\_\_*, comprobando que no se genera ningún error.

Posteriormente, ejecutamos el proceso en la GPU con un solo thread colocando el modificador *\_\_global\_\_* en el kernel de suma. También debemos reservar memoria en la tarjeta gráfica mediante *cudaMalloc* y llamar al kernel con *<<<1,1>>>*. Con todo ello podemos ver que tampoco se realiza ningún error.

La suma ha tardado más de 120ms, pero se está ejecutando en serie en un dispositivo pensado para trabajar con múltiples hilos. Para acelerar la suma de los vectores deberemos paralelizar el kernel suma.

A continuación, utilizamos 256 threads para realizar la suma, pero sin modificar el kernel. Esto provoca carrera de datos ya que no estamos definiendo qué thread se va a encargar de qué parte del vector, por lo que se generarán errores. Realmente si ejecutásemos todos los hilos en serie realizaríamos la operación 256\*x + y.

Ofrecemos una alternativa sin carrera de datos y con uso de bloques y threads simultáneos. Para ello dividimos el vector en bloques donde cada thread se encargará únicamente de sumar una posición del vector. Definimos el número de threads por bloque a 1024 tras ver las propiedades de la gráfica que usaremos en Google Colab. Anticipándonos a un cambio en N, utilizaremos el techo de la división del tamaño del vector entre el número de threads por bloque. También debemos definir qué índice va a operar cada thread mediante *threads\_per\_block \* blockIdx.x + threadIdx.x.* Compilamos y ejecutamos viendo que no tenemos ningún error y que reducimos el tiempo de ejecución del kernel a 2.69ms.

## Modificación a matrices