

Escribir y lanzar kernels en CUDA

Tipos de memoria

Dra. María Guadalupe Sánchez Cervantes

- Para GPUs con capacidad mayor o igual a 2.0 se puede imprimir dentro de la GPU solo con especificar al momento de compilar con nvcc lo siguiente:

```
nvcc nombrearchivo.cu -o nombrearchivoexe -  
arch=compute_20
```

Reparto

- ¿Cómo se identifica un bloque, un hilo y un grid?

Ejemplo "Hola mundo"

```
#include <stdio.h>

__global__ void holaCUDA(float e) {

    printf("Hola, soy el hilo %d del bloque %d con valor pi->%f\n",
threadIdx.x,blockIdx.x,e);

}

int main(int argc, char **argv){

    holaCUDA<<<3,4>>>(3.1416);

    cudaDeviceReset(); //Esta llamada reinicializa el device

    return 0;

}
```

Ejemplo: Analizando las propiedades de nuestra tarjeta GPU

```
void DisplayProperties( cudaDeviceProp* pDeviceProp ){

    if( !pDeviceProp )

        return;

    printf( "\n*****\n" );
    printf( "\nDevice Name \t - %s ", pDeviceProp->name );
    printf( "\n*****\n" );
    printf( "\nTotal Global Memory\t\t -> %ld KB", pDeviceProp->totalGlobalMem/1024 );
    printf( "\nShared memory available per block \t -> %ld KB", pDeviceProp->sharedMemPerBlock/1024 );
    printf( "\nNumber of registers per thread block \t -> %d", pDeviceProp->regsPerBlock );
    printf( "\nWarp size in threads \t -> %d", pDeviceProp->warpSize );
    printf( "\nMemory Pitch \t -> %lu bytes", pDeviceProp->memPitch );
```

```
printf( "\nMaximum threads per block \t -> %d", pDeviceProp->maxThreadsPerBlock );

printf( "\nMaximum Thread Dimension (block) \t -> %d %d %d", pDeviceProp->maxThreadsDim[0],
pDeviceProp->maxThreadsDim[1], pDeviceProp->maxThreadsDim[2] );

printf( "\nMaximum Thread Dimension (grid) \t -> %d %d %d", pDeviceProp->maxGridSize[0],
pDeviceProp->maxGridSize[1], pDeviceProp->maxGridSize[2] );

printf( "\nTotal constant memory \t -> %lu bytes", pDeviceProp->totalConstMem );

printf( "\nCUDA ver \t -> %d.%d", pDeviceProp->major, pDeviceProp->minor );

printf( "\nClock rate \t -> %d KHz", pDeviceProp->clockRate );

printf( "\nTexture Alignment \t -> %lu bytes", pDeviceProp->textureAlignment );

printf( "\nDevice Overlap \t -> %s", pDeviceProp-> deviceOverlap?"Allowed":"Not Allowed" );

printf( "\nNumber of Multi processors \t -> %d\n", pDeviceProp->multiProcessorCount );

}
```

```
int main(void){

    cudaDeviceProp deviceProp;

    int nDevCount = 0;      cudaGetDeviceCount( &nDevCount );

    printf( "Total Device found: %d", nDevCount );

    for (int nDeviceIdx = 0; nDeviceIdx < nDevCount; ++nDeviceIdx )

    {

        memset( &deviceProp, 0, sizeof(deviceProp));

        if( cudaSuccess == cudaGetDeviceProperties(&deviceProp, nDeviceIdx))

            DisplayProperties( &deviceProp );

        else

            printf( "\n%s", cudaGetErrorString(cudaGetLastError()));

    }

}
```

Ejemplo de propiedades

Total Device found: 3

Device Name - Tesla M2050

Total Global Memory -> 2751936 KB

Shared memory available per block -> 48 KB

Number of registers per thread block -> 32768

Warp size in threads -> 32

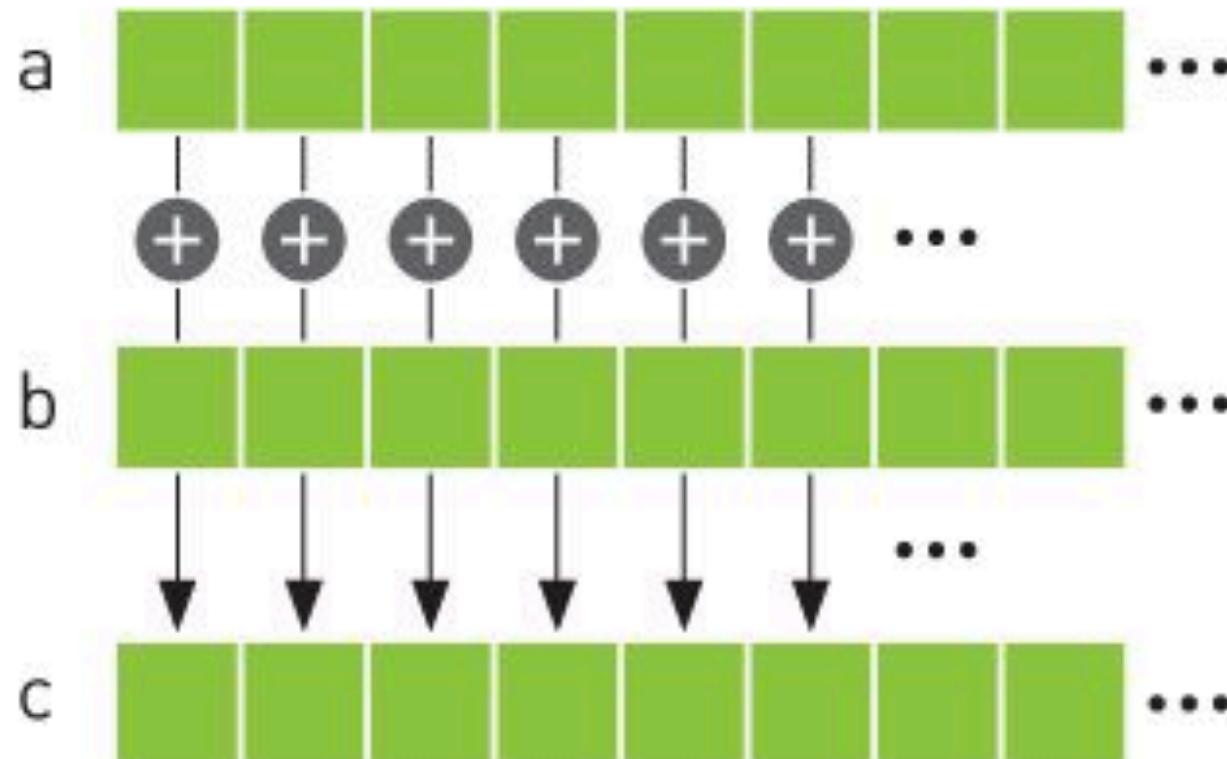
Memory Pitch -> 2147483647 bytes

Maximum threads per block -> 1024

Maximum Thread Dimension (block) -> 1024 1024 64
Maximum Thread Dimension (grid)
65535 -> 65535 65535
Total constant memory -> 65536 bytes
CUDA ver -> 2.0
Clock rate -> 1147000 KHz
Texture Alignment -> 512 bytes
Device Overlap -> Allowed
Number of Multi processors -> 14

- ¿Qué hace la función cudaMalloc()?
¿Cuál es su sintaxis?
- ¿Qué hace la función cudaMemcpy()?
¿Cuál es la sintaxis?
- ¿Cuál es el propósito de la función cudaFree()?

Resumen de dos vectores



Ejemplo: Suma de vectores de longitud 10. (bloques)

```
#include<stdio.h>

#define N 10

__global__ void addvec(int *a, int *b, int *c)
{
    int tid=blockIdx.x; //manejar los datos a este índice
    if(tid<N)
        c[tid]=a[tid]+b[tid];
}
```

```
//función principal  
int main(void){  
  
    int a[N], b[N], c[N];  
  
    int *dev_a, *dev_b, *dev_c;  
  
    //asignar memoria en la GPU  
    cudaMalloc((void**)&dev_a,N*sizeof(int));  
    cudaMalloc((void**)&dev_b,N*sizeof(int));  
    cudaMalloc((void**)&dev_c,N*sizeof(int));
```

```
//Llenar datos a los arreglos 'a' y 'b' en la CPU  
for(int i=0; i<N;i++){  
    a[i]=i;  
    b[i]=i+1;  
}  
  
//copiar el arreglo 'a' y 'b' en la GPU  
cudaMemcpy(dev_a,a,N*sizeof(int),cudaMemcpyHostToDevice);  
cudaMemcpy(dev_b,b,N*sizeof(int),cudaMemcpyHostToDevice);
```

```
//lanzar el kernel  
addvec<<<N,1>>>(dev_a,dev_b,dev_c);  
  
//copiar el arreglo 'c' de la GPU a la CPU  
cudaMemcpy(c,dev_c,N*sizeof(int),cudaMemcpyDeviceToHost);
```

```
//Desplegar el resultado  
for(int i=0; i<N; i++){  
    printf("%d+%d=%d\n",a[i],b[i],c[i]);  
}  
cudaFree(dev_a);  
cudaFree(dev_b);  
cudaFree(dev_c);  
return 0;  
}
```

Suma de vectores de longitud 10. (hilos)

- Cambiar el código anterior para que en lugar de lanzar N bloques de un hilo se puedan lanzar N hilos, todos dentro de un bloque.

Solución...

Cambiar la llamada al kernel:

```
//lanzar el kernel
```

```
addvecHilos<<<1,N>>>(dev_a,dev_b,dev_c);
```

Dentro del kernel indexamos la entrada y salida con el índice del bloque:

```
int tid= blockIdx.x
```

Ahora indexamos como:

```
int tid=threadIdx.x
```

Acomodo bidimensional de una combinación de bloques e hilos

Block 0	Thread 0	Thread 1	Thread 2	Thread 3
Block 1	Thread 0	Thread 1	Thread 2	Thread 3
Block 2	Thread 0	Thread 1	Thread 2	Thread 3
Block 3	Thread 0	Thread 1	Thread 2	Thread 3

- Si el hilo representa columnas y los bloques representan renglones, se puede obtener un único índice tomando el producto del índice bloque con el número de hilos en cada bloques y sumando el índice del hilo dentro del bloque.

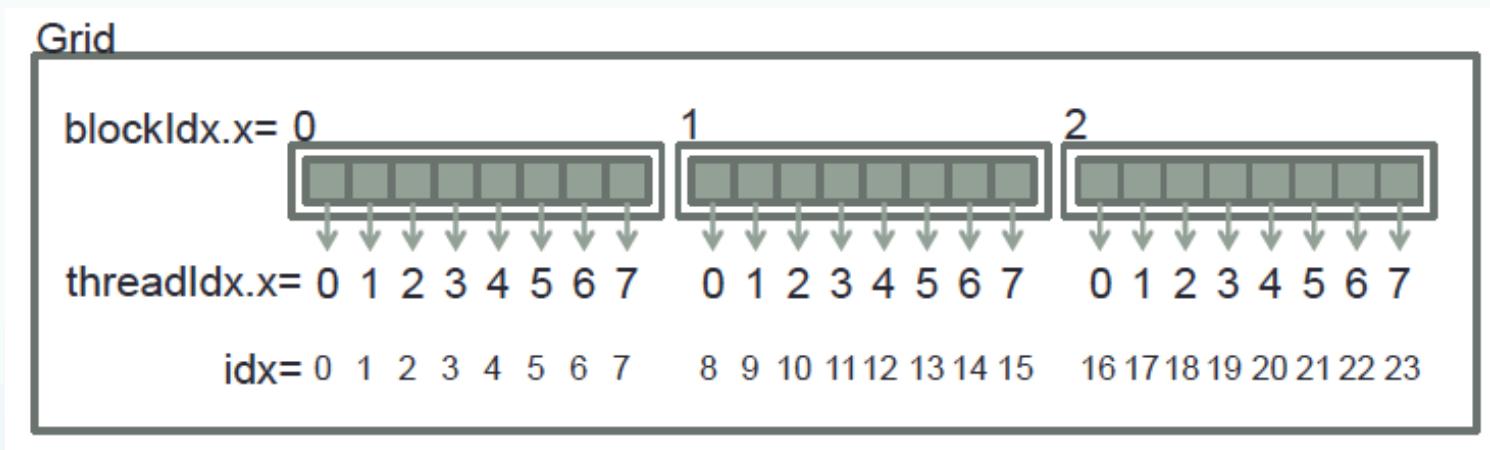
Cambiar el código para la suma grande de datos

```
__global__ void addVecGrande(int *a, int *b, int *c){  
int tid=threadIdx.x+blockIdx.x*blockDim.x;  
  
if(tid<N)  
{  
    c[tid]=a[tid]+b[tid];  
}  
}
```

```
//lanzar el kernel
```

```
addVecGrande<<<128,128>>>(dev_a,dev_b,dev_c);
```

¿Cómo sería gráficamente si N=24 y blockDim.x=8?



Ejercicio: Probar con N=50000 y 128 de bloques con 256 hilos cada uno.

Suma de matrices (dos dimensiones)

¿Cómo harías la suma de matrices con dimensiones x y y?

Pista:

```
int col = blockIdx.x * blockDim.x + threadIdx.x;
```

```
int fil = blockIdx.y * blockDim.y + threadIdx.y;
```

```
int indice = fil * N + col;
```

```
#define T 10 // max threads x bloque  
  
#define N 300  
  
*****  
  
// cada bloque en dimensión x y y tendrá un tamaño de T Threads  
  
dim3 ThreadsBloque(T, T);  
  
// Calculando el número de bloques en 1D  
  
float BFloat = (float) N / (float) T;  
  
int B = (int) ceil(BFloat);  
  
// El grid tendrá B número de bloques en x y y  
  
dim3 Bloques(B, B);  
  
// Llamando a ejecutar el kernel  
  
sumaMatrices<<<Bloques, ThreadsBloque>>>(matriz1, matriz2, matriz3);
```

Multiplicación de matrices

- Realizar la multiplicación de matrices utilizando memoria global