Ciencia y Tecnología

Secretaría de Ciencia, Humanidades, Tecnología e Innovación

CIMAT
UNIDAD MÉRIDA

EJEMPLOS USANDO CUDA

Dr. Francisco J. Hernández López
SECIHTI – CIMAT-Mérida
fcoj23@cimat.mx, www.cimat.mx/~fcoj23

HOLA MUNDO

```cpp
#include <stdio.h>

// printf() is only supported
// for devices of compute capability 2.0 and higher

__global__ void helloCUDA(float e){
    printf("Hello, I am thread %d of block %d with value e=%f\n", threadIdx.x,blockIdx.x, e);
}

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

    helloCUDA<<<3, 4>>>(2.71828f);

    cudaDeviceReset();//is called to reinitialize the device.
    system("pause");
    return(0);
}
```

Compilación:

Windows

nvcc HelloCUDA.cu –o run_HelloCUDA.exe

Linux

nvcc HelloCUDA.cu –o run_HelloCUDA

```
Hello, I am thread 0 of block 1 with value e=2.718280
Hello, I am thread 1 of block 1 with value e=2.718280
Hello, I am thread 2 of block 1 with value e=2.718280
Hello, I am thread 3 of block 1 with value e=2.718280
Hello, I am thread 0 of block 0 with value e=2.718280
Hello, I am thread 1 of block 0 with value e=2.718280
Hello, I am thread 2 of block 0 with value e=2.718280
Hello, I am thread 3 of block 0 with value e=2.718280
Hello, I am thread 0 of block 2 with value e=2.718280
Hello, I am thread 1 of block 2 with value e=2.718280
Hello, I am thread 2 of block 2 with value e=2.718280
Hello, I am thread 3 of block 2 with value e=2.718280
Presione una tecla para continuar . . .
```

CONOCIENDO LOS ÍNDICES DE LOS HILOS

```
1  #include <stdio.h>
2  #include <stdlib.h>
3  #include <cuda_runtime.h>
4  #include <device_launch_parameters.h>
5
6  __global__ void idThreads_kernel(int *block_dev,
7      int *threadLocal_dev,
8      int *warp_dev,
9      int *threadGlobal_dev){
10     int threadGlobal_idx = (blockIdx.x*blockDim.x) + threadIdx.x;
11
12     block_dev[threadGlobal_idx] = blockIdx.x;
13     threadLocal_dev[threadGlobal_idx] = threadIdx.x;
14     warp_dev[threadGlobal_idx] = threadGlobal_idx / warpSize;
15     threadGlobal_dev[threadGlobal_idx] = threadGlobal_idx;
16 }
```

Función Kernel

```
21 #define TAM 128
22 //#define TAM 140
23
24 int main(void){
25     int num_blocks = 4;
26     int num_threads = 32;
27     //int num_threads = 35;
28
29     /*Arreglos estáticos en el host*/
30     int block_host[TAM];
31     int threadLocal_host[TAM];
32     int warp_host[TAM];
33     int threadGlobal_host[TAM];
```

Se propone un malla unidimensional

CONOCIENDO LOS ÍNDICES DE LOS HILOS

```
35      /*Arreglos dinámicos en el device*/
36      int *block_dev;
37      int *threadLocal_dev;
38      int *warp_dev;
39      int *threadGlobal_dev;
40      /*Crear memoria dinámica en el device*/
41      size_t TAM_Bytes_int=TAM*sizeof(int);
42      cudaMalloc((void**)&block_dev, TAM_Bytes_int);
43      cudaMalloc((void**)&threadLocal_dev, TAM_Bytes_int);
44      cudaMalloc((void**)&warp_dev, TAM_Bytes_int);
45      cudaMalloc((void**)&threadGlobal_dev, TAM_Bytes_int);
46
47      /*Inicializar con -1 en el device*/
48      cudaMemset(threadGlobal_dev,-1,TAM_Bytes_int);
49      cudaMemset(threadLocal_dev, -1, TAM_Bytes_int);
50      cudaMemset(warp_dev, -1, TAM_Bytes_int);
51      cudaMemset(block_dev, -1, TAM_Bytes_int);
52
53      /*Ejecutar Kernel*/
54      idThreads_kernel<<<num_blocks, num_threads>>>(block_dev, threadLocal_dev, warp_dev,threadGlobal_dev);
55
56      /*Copiar memoria desde el device al host*/
57      cudaMemcpy(block_host, block_dev, TAM_Bytes_int, cudaMemcpyDeviceToHost);
58      cudaMemcpy(threadLocal_host, threadLocal_dev, TAM_Bytes_int, cudaMemcpyDeviceToHost);
59      cudaMemcpy(warp_host, warp_dev, TAM_Bytes_int, cudaMemcpyDeviceToHost);
60      cudaMemcpy(threadGlobal_host, threadGlobal_dev, TAM_Bytes_int, cudaMemcpyDeviceToHost);
61
62      /*Liberar memoria*/
63      cudaFree(block_dev);
64      cudaFree(threadLocal_dev);
65      cudaFree(warp_dev);
66      cudaFree(threadGlobal_dev);
```

Se ejecuta la malla

x86_x64 Cross Tools Command Prompt for VS 2019 - idThreads.exe

```
Thread Global: 0 -- Bloque: 0 -- Warp: 0 -- Thread Local: 0
Thread Global: 1 -- Bloque: 0 -- Warp: 0 -- Thread Local: 1
Thread Global: 2 -- Bloque: 0 -- Warp: 0 -- Thread Local: 2
Thread Global: 3 -- Bloque: 0 -- Warp: 0 -- Thread Local: 3
Thread Global: 4 -- Bloque: 0 -- Warp: 0 -- Thread Local: 4
Thread Global: 5 -- Bloque: 0 -- Warp: 0 -- Thread Local: 5
Thread Global: 6 -- Bloque: 0 -- Warp: 0 -- Thread Local: 6
Thread Global: 7 -- Bloque: 0 -- Warp: 0 -- Thread Local: 7
Thread Global: 8 -- Bloque: 0 -- Warp: 0 -- Thread Local: 8
Thread Global: 9 -- Bloque: 0 -- Warp: 0 -- Thread Local: 9
Thread Global: 10 -- Bloque: 0 -- Warp: 0 -- Thread Local: 10
Thread Global: 11 -- Bloque: 0 -- Warp: 0 -- Thread Local: 11
Thread Global: 12 -- Bloque: 0 -- Warp: 0 -- Thread Local: 12
Thread Global: 13 -- Bloque: 0 -- Warp: 0 -- Thread Local: 13
Thread Global: 14 -- Bloque: 0 -- Warp: 0 -- Thread Local: 14
Thread Global: 15 -- Bloque: 0 -- Warp: 0 -- Thread Local: 15
Thread Global: 16 -- Bloque: 0 -- Warp: 0 -- Thread Local: 16
Thread Global: 17 -- Bloque: 0 -- Warp: 0 -- Thread Local: 17
Thread Global: 18 -- Bloque: 0 -- Warp: 0 -- Thread Local: 18
Thread Global: 19 -- Bloque: 0 -- Warp: 0 -- Thread Local: 19
Thread Global: 20 -- Bloque: 0 -- Warp: 0 -- Thread Local: 20
Thread Global: 21 -- Bloque: 0 -- Warp: 0 -- Thread Local: 21
Thread Global: 22 -- Bloque: 0 -- Warp: 0 -- Thread Local: 22
Thread Global: 23 -- Bloque: 0 -- Warp: 0 -- Thread Local: 23
Thread Global: 24 -- Bloque: 0 -- Warp: 0 -- Thread Local: 24
Thread Global: 25 -- Bloque: 0 -- Warp: 0 -- Thread Local: 25
Thread Global: 26 -- Bloque: 0 -- Warp: 0 -- Thread Local: 26
Thread Global: 27 -- Bloque: 0 -- Warp: 0 -- Thread Local: 27
Thread Global: 28 -- Bloque: 0 -- Warp: 0 -- Thread Local: 28
Thread Global: 29 -- Bloque: 0 -- Warp: 0 -- Thread Local: 29
Thread Global: 30 -- Bloque: 0 -- Warp: 0 -- Thread Local: 30
Thread Global: 31 -- Bloque: 0 -- Warp: 0 -- Thread Local: 31
Thread Global: 32 -- Bloque: 0 -- Warp: 1 -- Thread Local: 32
Thread Global: 33 -- Bloque: 0 -- Warp: 1 -- Thread Local: 33
Thread Global: 34 -- Bloque: 0 -- Warp: 1 -- Thread Local: 34
Thread Global: 35 -- Bloque: 1 -- Warp: 1 -- Thread Local: 0
Thread Global: 36 -- Bloque: 1 -- Warp: 1 -- Thread Local: 1
Thread Global: 37 -- Bloque: 1 -- Warp: 1 -- Thread Local: 2
Thread Global: 38 -- Bloque: 1 -- Warp: 1 -- Thread Local: 3
Thread Global: 39 -- Bloque: 1 -- Warp: 1 -- Thread Local: 4
Thread Global: 40 -- Bloque: 1 -- Warp: 1 -- Thread Local: 5
```

```
Thread Global: 100 -- Bloque: 2 -- Warp: 3 -- Thread Local: 30
Thread Global: 101 -- Bloque: 2 -- Warp: 3 -- Thread Local: 31
Thread Global: 102 -- Bloque: 2 -- Warp: 3 -- Thread Local: 32
Thread Global: 103 -- Bloque: 2 -- Warp: 3 -- Thread Local: 33
Thread Global: 104 -- Bloque: 2 -- Warp: 3 -- Thread Local: 34
Thread Global: 105 -- Bloque: 3 -- Warp: 3 -- Thread Local: 0
Thread Global: 106 -- Bloque: 3 -- Warp: 3 -- Thread Local: 1
Thread Global: 107 -- Bloque: 3 -- Warp: 3 -- Thread Local: 2
Thread Global: 108 -- Bloque: 3 -- Warp: 3 -- Thread Local: 3
Thread Global: 109 -- Bloque: 3 -- Warp: 3 -- Thread Local: 4
Thread Global: 110 -- Bloque: 3 -- Warp: 3 -- Thread Local: 5
Thread Global: 111 -- Bloque: 3 -- Warp: 3 -- Thread Local: 6
Thread Global: 112 -- Bloque: 3 -- Warp: 3 -- Thread Local: 7
Thread Global: 113 -- Bloque: 3 -- Warp: 3 -- Thread Local: 8
Thread Global: 114 -- Bloque: 3 -- Warp: 3 -- Thread Local: 9
Thread Global: 115 -- Bloque: 3 -- Warp: 3 -- Thread Local: 10
Thread Global: 116 -- Bloque: 3 -- Warp: 3 -- Thread Local: 11
Thread Global: 117 -- Bloque: 3 -- Warp: 3 -- Thread Local: 12
Thread Global: 118 -- Bloque: 3 -- Warp: 3 -- Thread Local: 13
Thread Global: 119 -- Bloque: 3 -- Warp: 3 -- Thread Local: 14
Thread Global: 120 -- Bloque: 3 -- Warp: 3 -- Thread Local: 15
Thread Global: 121 -- Bloque: 3 -- Warp: 3 -- Thread Local: 16
Thread Global: 122 -- Bloque: 3 -- Warp: 3 -- Thread Local: 17
Thread Global: 123 -- Bloque: 3 -- Warp: 3 -- Thread Local: 18
Thread Global: 124 -- Bloque: 3 -- Warp: 3 -- Thread Local: 19
Thread Global: 125 -- Bloque: 3 -- Warp: 3 -- Thread Local: 20
Thread Global: 126 -- Bloque: 3 -- Warp: 3 -- Thread Local: 21
Thread Global: 127 -- Bloque: 3 -- Warp: 3 -- Thread Local: 22
Thread Global: 128 -- Bloque: 3 -- Warp: 4 -- Thread Local: 23
Thread Global: 129 -- Bloque: 3 -- Warp: 4 -- Thread Local: 24
Thread Global: 130 -- Bloque: 3 -- Warp: 4 -- Thread Local: 25
Thread Global: 131 -- Bloque: 3 -- Warp: 4 -- Thread Local: 26
Thread Global: 132 -- Bloque: 3 -- Warp: 4 -- Thread Local: 27
Thread Global: 133 -- Bloque: 3 -- Warp: 4 -- Thread Local: 28
Thread Global: 134 -- Bloque: 3 -- Warp: 4 -- Thread Local: 29
Thread Global: 135 -- Bloque: 3 -- Warp: 4 -- Thread Local: 30
Thread Global: 136 -- Bloque: 3 -- Warp: 4 -- Thread Local: 31
Thread Global: 137 -- Bloque: 3 -- Warp: 4 -- Thread Local: 32
Thread Global: 138 -- Bloque: 3 -- Warp: 4 -- Thread Local: 33
Thread Global: 139 -- Bloque: 3 -- Warp: 4 -- Thread Local: 34
```

SUMA DE VECTORES

$$\vec{c} = \vec{a} + \vec{b}$$

- Creamos los vectores "a_h", "b_h" y "c_h" en el CPU
- Inicializamos con cualquier valor los vectores "a_h" y "b_h"
- Creamos los vectores "a_d", "b_d" y "c_d" en el GPU
- Copiamos el contenido de los vectores a y b del CPU al GPU
- Sumamos "a_d" y "b_d" y el resultado lo guardamos en c_d
- Copiamos el resultado del GPU al CPU
- Desplegamos la suma de los vectores a y b

SUMA DE VECTORES (C1)

```
// Código principal que se ejecuta en el Host
int main(void){
    float *a_h,*b_h,*c_h; //Punteros a arreglos en el Host
    float *a_d,*b_d,*c_d;  //Punteros a arreglos en el Device
    const int N = 24;  //Número de elementos en los arreglos  (probar 1000000)

    size_t size=N * sizeof(float);

    a_h = (float *)malloc(size); // Pedimos memoria en el Host
    b_h = (float *)malloc(size);
    c_h = (float *)malloc(size);//También se puede con cudaMallocHost

    //Inicializamos los arreglos a,b en el Host
    srand(time(NULL));
    for (int i=0; i<N; i++){
        //a_h[i] = (float)i;b_h[i] = (float)(i+1);
        a_h[i] = rand() % 100 + 1.0;
        b_h[i] = rand() % 100 + 1.0;
    }

    printf("\nArreglo a:\n");
    for (int i=0; i<N; i++) printf("%f ", a_h[i]);
    printf("\n\nArreglo b:\n");
    for (int i=0; i<N; i++) printf("%f ", b_h[i]);

    cudaMalloc((void **) &a_d,size);   // Pedimos memoria en el Device
    cudaMalloc((void **) &b_d,size);
    cudaMalloc((void **) &c_d,size);

    //Pasamos los arreglos a y b del Host al Device
    cudaMemcpy(a_d, a_h, size, cudaMemcpyHostToDevice);
    cudaMemcpy(b_d, b_h, size, cudaMemcpyHostToDevice);
```

SUMA DE VECTORES (C2)

```
//Realizamos el cálculo en el Device
int block_size =8;
int n_blocks = N/block_size + (N%block_size == 0 ? 0:1);

Suma_vectores <<< n_blocks, block_size >>> (c_d,a_d,b_d,N);


//Pasamos el resultado del Device al Host
cudaMemcpy(c_h, c_d, size,cudaMemcpyDeviceToHost);

//Resultado
printf("\n\nArreglo c:\n");
for (int i=0; i<N; i++) printf("%f ", c_h[i]);


// Liberamos la memoria del Host
free(a_h);
free(b_h);
free(c_h);

// Liberamos la memoria del Device
cudaFree(a_d);
cudaFree(b_d);
cudaFree(c_d);
return(0);
}
```

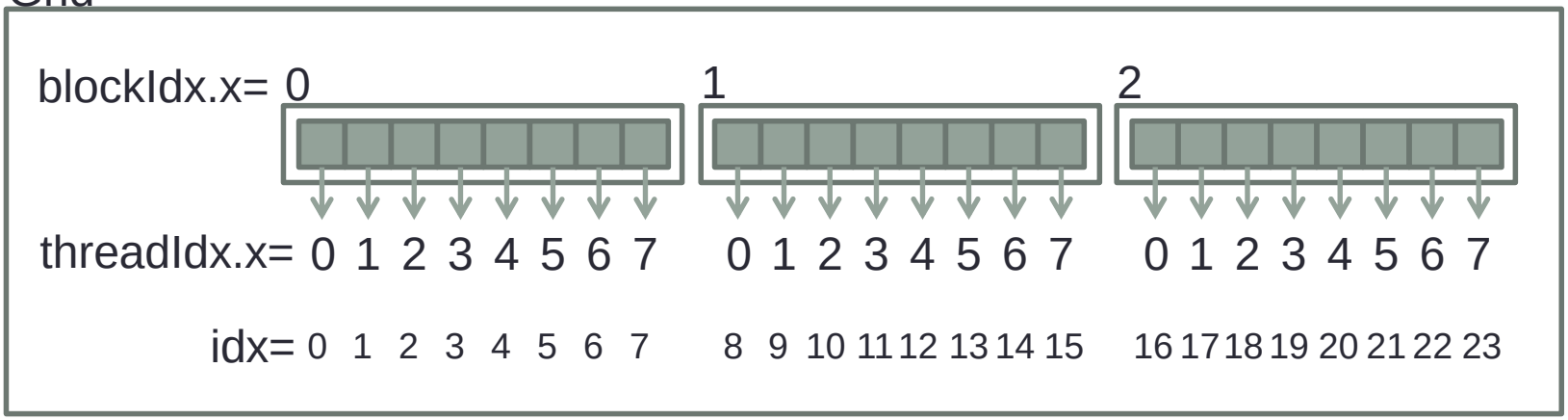
```
// Función Kernel que se ejecuta en el Device.
__global__ void Suma_vectores(float *c_d,float *a_d,float *b_d, int N)
{
  int idx = blockIdx.x * blockDim.x + threadIdx.x;
  if (idx<N){
      c_d[idx] = a_d[idx] + b_d[idx];
  }
}
```

SUMA DE VECTORES (C3)

idx = blockIdx.x * blockDim.x + threadIdx.x

N=24 y blockDim.x= 8

Grid



Compilación:

nvcc Suma_Vectores.cu –o run_Suma_Vectores

MULTIPLICACIÓN DE MATRICES

```
// Código principal que se ejecuta en el Host
int main(void){
    float *A_h,*B_h,*C_h; //Punteros a matrices en el Host
    float *A_d,*B_d,*C_d;  //Punteros a matrices en el Device
    int nfil = 12;  //Número de filas
    int ncol = 12;  //Número de columnas
    int N=nfil*ncol;  //Número de elementos de la matriz

    //GPU Time
    cudaEvent_t start, stop;
    float time;

    size_t size=N * sizeof(float);

    A_h = (float *)malloc(size); // Pedimos memoria en el Host
    B_h = (float *)malloc(size);
    C_h = (float *)malloc(size);//También se puede con cudaMallocHost

    //Inicializamos las matrices a,b en el Host
    for (int i=0; i<nfil; i++){
        for(int j=0;j<ncol;j++){
            A_h[i*ncol+j] = 1.0f;
            B_h[i*ncol+j] = 2.0f;
        }
    }

    cudaMalloc((void **) &A_d,size);   // Pedimos memoria en el Device
    cudaMalloc((void **) &B_d,size);
    cudaMalloc((void **) &C_d,size);

   //Pasamos las matrices a y b del Host al Device
   cudaMemcpy(A_d, A_h, size, cudaMemcpyHostToDevice);
   cudaMemcpy(B_d, B_h, size, cudaMemcpyHostToDevice);
```

MULTIPLICACIÓN DE MATRICES (C1)

```
//Realizamos el cálculo en el Device
dim3 block_size(BLOCK_SIZE,BLOCK_SIZE);
dim3 n_blocks(div_up(ncol,block_size.x),div_up(nfil,block_size.y)) ;
Multiplica_Matrices_GM<<< n_blocks, block_size >>> (C_d,A_d,B_d,nfil,ncol);

//Pasamos el resultado del Device al Host
cudaMemcpy(C_h, C_d, size,cudaMemcpyDeviceToHost);

//Resultado
printf("\n\nMatriz c:\n");
for (int i=0; i<10; i++){
    for(int j=0;j<10;j++){
        printf("%.2f ", C_h[i*ncol+j]);
    }
    printf("\n");
}

// Liberamos la memoria del Host
free(A_h);
free(B_h);
free(C_h);

// Liberamos la memoria del Device
cudaFree(A_d);
cudaFree(B_d);
cudaFree(C_d);
return(0);
```

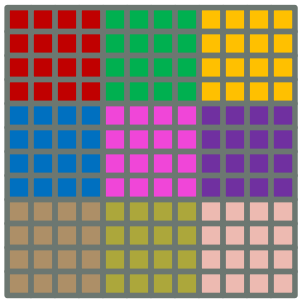
```
//Multiplicacion de Matrices en Memoria Global (GM)
__global__ void Multiplica_Matrices_GM(float *C,float *A,float *B,
                                       int nfil,int ncol)
{
    int idx = blockIdx.x * blockDim.x + threadIdx.x;
    int idy = blockIdx.y * blockDim.y + threadIdx.y;
    int index=idy*ncol+idx;
    if (idy<nfil && idx<ncol){
        float sum=0.0f;
        for(int k=0;k<ncol;k++){
            sum+=A[idy*ncol+k]*B[k*ncol+idx];
        }
        C[index] = sum;
    }
}
```

MULTIPLICACIÓN DE MATRICES (C2)

idx = blockIdx.x * blockDim.x + threadIdx.x
idy = blockIdx.y * blockDim.y + threadIdx.y

nfil=12, ncol=12, BLOCK_SIZE=4

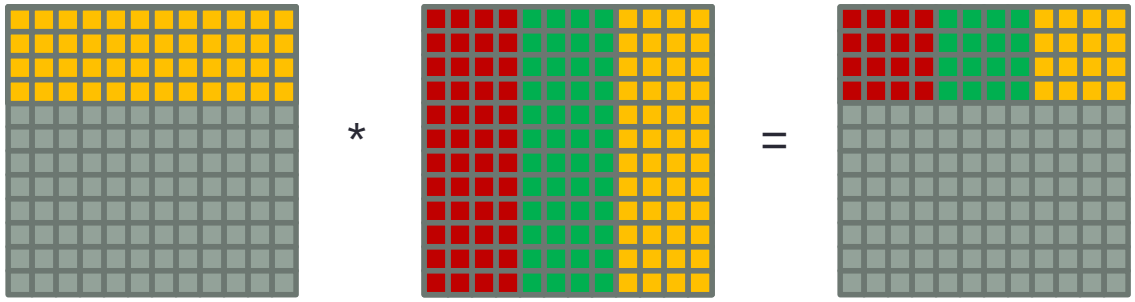
Grid



blockIdx.x={0,1,2}
blockIdx.y={0,1,2}
threadIdx.x={0,1,2,3}
threadIdx.y={0,1,2,3}
idx={0,1,2,…,11}
idy={0,1,2,…,11}

```
//Multiplicacion de Matrices en Memoria Global (GM)
__global__ void Multiplica_Matrices_GM(float *C,float *A,float *B,
                                       int nfil,int ncol)
{
    int idx = blockIdx.x * blockDim.x + threadIdx.x;
    int idy = blockIdx.y * blockDim.y + threadIdx.y;
    int index=idy*ncol+idx;
    if (idy<nfil && idx<ncol){
        float sum=0.0f;
        for(int k=0;k<ncol;k++){
            sum+=A[idy*ncol+k]*B[k*ncol+idx];
        }
        C[index] = sum;
    }
}
```

A B C



GRACIAS POR SU ATENCIÓN

Francisco J. Hernández-López

fcoj23@cimat.mx

WebPage:

www.cimat.mx/~fcoj23