



UNIVERSIDAD NACIONAL DE SAN AGUSTIN

CIENCIA DE LA COMPUTACIÓN

---

# Informe de la multiplicación de matrices usando *tilde*

---

***Alumnas :***

*Judith Escalante Calcina*

***Profesor:***

*Mg. Alvaro Henry Mamani*

*Aliaga*

## Índice

<b>1. Multiplicación en cpu</b>	<b>2</b>
<b>2. Multiplicación en gpu</b>	<b>2</b>
2.1. Multiplicación sin memoria compartida . . . . .	2
2.2. Multiplicación con memoria compartida . . . . .	3
<b>3. Resultados</b>	<b>5</b>
<b>4. Conclusión</b>	<b>7</b>

## 1. Multiplicación en cpu

La multiplicación de matrices es un proceso altamente utilizado y conocido , este funciona perfectamente con matrices pequeñas pero no con matrices de grandes dimensiones, en el siguiente código podemos ver la función principal de una multiplicación de matrices en general :

```
void cpu_matrix_mult(int *h_a, int *h_b, int *h_result ,
int m, int n, int k) {
    for (int i = 0; i < m; ++i)
    {
        for (int j = 0; j < k; ++j)
        {
            int tmp = 0.0;
            for (int h = 0; h < n; ++h)
            {
                tmp += h_a[i * n + h] * h_b[h * k + j];
            }
            h_result[i * k + j] = tmp;
        }
    }
}
```

## 2. Multiplicación en gpu

### 2.1. Multiplicación sin memoria compartida

La multiplicación realiza en la gpu de una computadora es mucho más eficiente que que la hecha en la cpu , por que utiliza varios thread para cada elemento de la matriz , que qa su vez es dividida en una cierta cantidad de bloques. Aún asi este no es uno de las ejecuciones más eficientes. Aqui podemos ver el código principal:

```
--global-- void matrix_mult(int *a,int *b, int *c, int m,
int n, int k)
{
    int row = blockIdx.y * blockDim.y + threadIdx.y;
    int col = blockIdx.x * blockDim.x + threadIdx.x;
    int sum = 0;
    if( col < k && row < m)
```

```
{
    for(int i = 0; i < n; i++)
    {
        sum += a[row * n + i] * b[i * k + col];
    }
    c[row * k + col] = sum;
}
```

## 2.2. Multiplicación con memoria compartida

La multiplicación utilizando memoria compartida y *tileds* es una versión optimizada del producto de dos matrices,  $A \times B$ , en la que cada bloque de hilos computa una submatriz o *tiled* de la matrix resultado  $C$ . Esto permite reducir el cuello de botella del ancho de banda de la memoria, puesto que varios elementos del bloque acceden a la misma fila de  $A$  y columna de  $B$ .

A su vez, esta localidad de acceso será aprovechada para utilizar la memoria compartida, lo que también nos obligará a realizar algunas sincronizaciones entre hilos dentro de un bloque. La memoria compartida dentro de cada multiprocesador se utilizará para almacenar cada submatriz antes de los cálculos, acelerando el acceso a memoria global. A continuación podemos ver el código principal:

```
--global-- void matrix_mult_tiled(int *d_a, int *d_b,
int *d_result, int n)
{
    --shared-- int Mds[TILED][TILED];
    --shared-- int Nds[TILED][TILED];
    int bx = blockIdx.x; int by = blockIdx.y;
    int tx = threadIdx.x; int ty = threadIdx.y;

    int Row = by * TILED + ty;
    int Col = bx * TILED + tx;
    int nuevo = 0;

    for (int g = 0; g < n/TILED; ++g)
    {
        Mds[ty][tx] = d_a[Row*n + g*TILED + tx];
    }
}
```

```

        Nds[ty][tx] = d_b[(g*TILED + ty)*n + Col];
        __syncthreads();
        for (int k = 0; k < TILED; ++k)
        {
            nuevo += Mds[ty][k] * Nds[k][tx];
        }
        __syncthreads();
    }
    d_result[Row*Width + Col] = nuevo;
}

```

Para que la multiplicación con memoria compartida resulte se debe tener la arquitectura del device :

Figura 1: Arquitectura del GPU

```

cs01@slave3:/usr/local/cuda-8.0/extras/demo_suite$ ./deviceQuery
./deviceQuery Starting...

  CUDA Device Query (Runtime API) version (CUDA static linking)

Detected 1 CUDA Capable device(s)

Device 0: "GeForce GT 620"
  CUDA Driver Version / Runtime Version      8.0 / 8.0
  CUDA Capability Major/Minor version number: 2.1
  Total amount of global memory:              963 MBytes (1010040832 bytes)
  ( 1) Multiprocessors, ( 48) CUDA Cores/MP: 48 CUDA Cores
  GPU Max Clock rate:                        1620 MHz (1.62 GHz)
  Memory Clock rate:                          897 Mhz
  Memory Bus Width:                           64-bit
  L2 Cache Size:                             65536 bytes
  Maximum Texture Dimension Size (x,y,z)      1D=(65536), 2D=(65536, 65535),
  3D=(2048, 2048, 2048)
  Maximum Layered 1D Texture Size, (num) layers 1D=(16384), 2048 layers
  Maximum Layered 2D Texture Size, (num) layers 2D=(16384, 16384), 2048 layers
  Total amount of constant memory:             65536 bytes
  Total amount of shared memory per block:     49152 bytes
  Total number of registers available per block: 32768
  Warp size:                                  32
  Maximum number of threads per multiprocessor: 1536
  Maximum number of threads per block:         1024
  Max dimension size of a thread block (x,y,z): (1024, 1024, 64)
  Max dimension size of a grid size    (x,y,z): (65535, 65535, 65535)
  Maximum memory pitch:                       2147483647 bytes
  Texture alignment:                          512 bytes
  Concurrent copy and kernel execution:        Yes with 1 copy engine(s)
  Run time limit on kernels:                   Yes
  Integrated GPU sharing Host Memory:          No
  Support host page-locked memory mapping:     Yes
  Alignment requirement for Surfaces:          Yes
  Device has ECC support:                     Disabled
  Device supports Unified Addressing (UVA):     Yes
  Device PCI Domain ID / Bus ID / location ID: 0 / 1 / 0
  Compute Mode:
    < Default (multiple host threads can use ::cudaSetDevice() with device simul-
    taneously) >

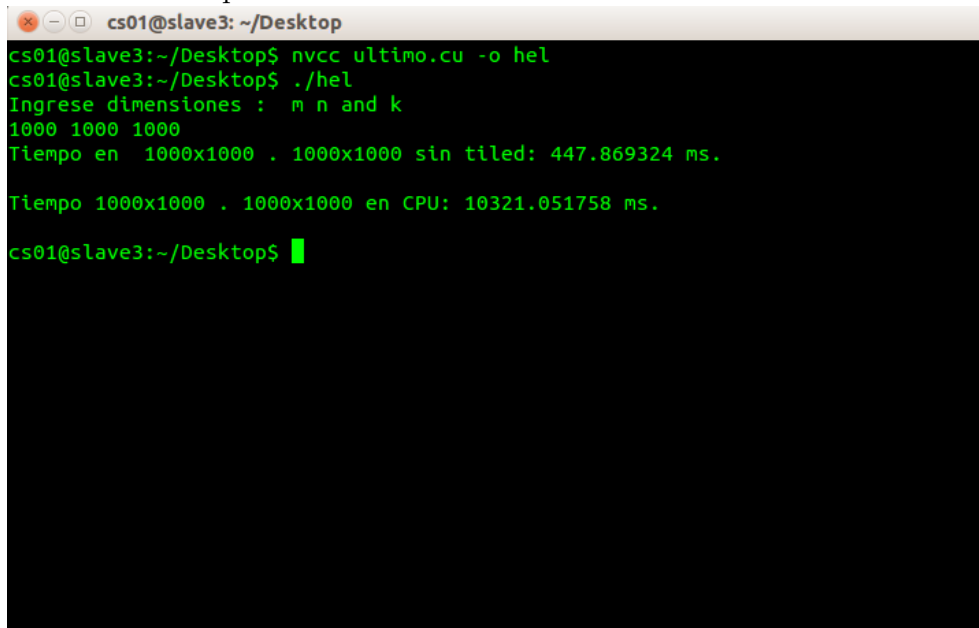
deviceQuery, CUDA Driver = CUDART, CUDA Driver Version = 8.0, CUDA Runtime Versi-
on = 8.0, NumDevs = 1, Device0 = GeForce GT 620
Result = PASS

```

### 3. Resultados

En las siguientes figuras podemos ver la multiplicación de diferentes matrices de dimensiones  $n \times m$ , el tiempo de ejecución es menor en la GPU en ambos casos, pero el menor tiempo obtenido es utilizando memoria compartida en GPU.

Figura 2: Tiempo de ejecución de la multiplicación de matrices ( $1000 \times 1000$ ) sin utilizar memoria compartida

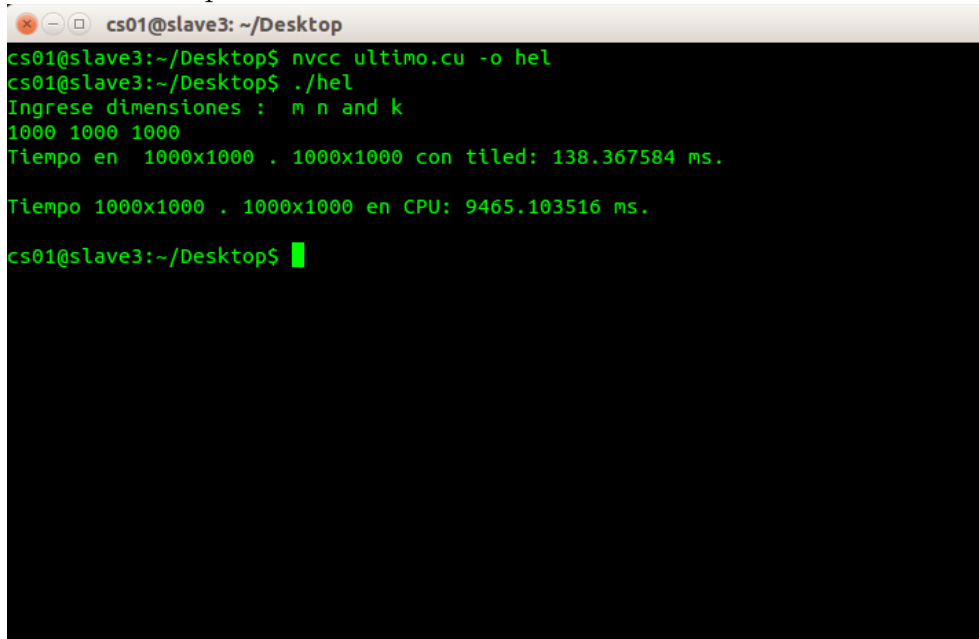
A terminal window titled 'cs01@slave3: ~/Desktop' with a black background and green text. The user enters 'nvcc ultimo.cu -o hel' and then './hel'. The program prompts for dimensions 'm n and k' and the user enters '1000 1000 1000'. The program outputs 'Tiempo en 1000x1000 . 1000x1000 sin tiled: 447.869324 ms.' and 'Tiempo 1000x1000 . 1000x1000 en CPU: 10321.051758 ms.'.

```
cs01@slave3:~/Desktop$ nvcc ultimo.cu -o hel
cs01@slave3:~/Desktop$ ./hel
Ingrese dimensiones : m n and k
1000 1000 1000
Tiempo en 1000x1000 . 1000x1000 sin tiled: 447.869324 ms.

Tiempo 1000x1000 . 1000x1000 en CPU: 10321.051758 ms.

cs01@slave3:~/Desktop$
```

Figura 3: Tiempo de ejecución de la multiplicación de matrices ( $1000 * 1000$ ) utilizando memoria compartida

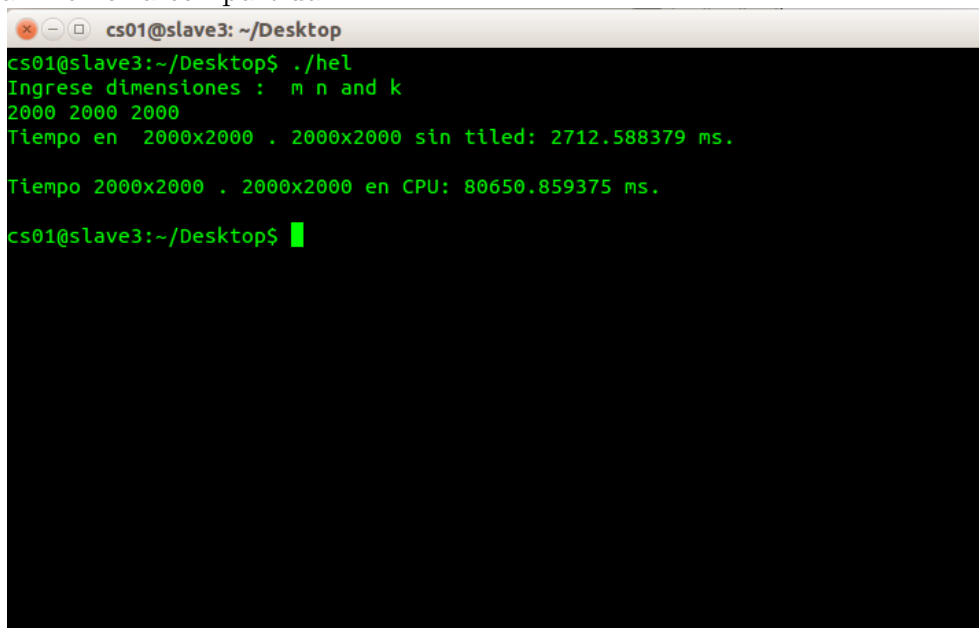


```
cs01@slave3: ~/Desktop
cs01@slave3:~/Desktop$ nvcc ultimo.cu -o hel
cs01@slave3:~/Desktop$ ./hel
Ingrese dimensiones : m n and k
1000 1000 1000
Tiempo en 1000x1000 . 1000x1000 con tiled: 138.367584 ms.

Tiempo 1000x1000 . 1000x1000 en CPU: 9465.103516 ms.

cs01@slave3:~/Desktop$
```

Figura 4: Tiempo de ejecución de la multiplicación de matrices ( $2000 * 2000$ ) sin utilizar memoria compartida

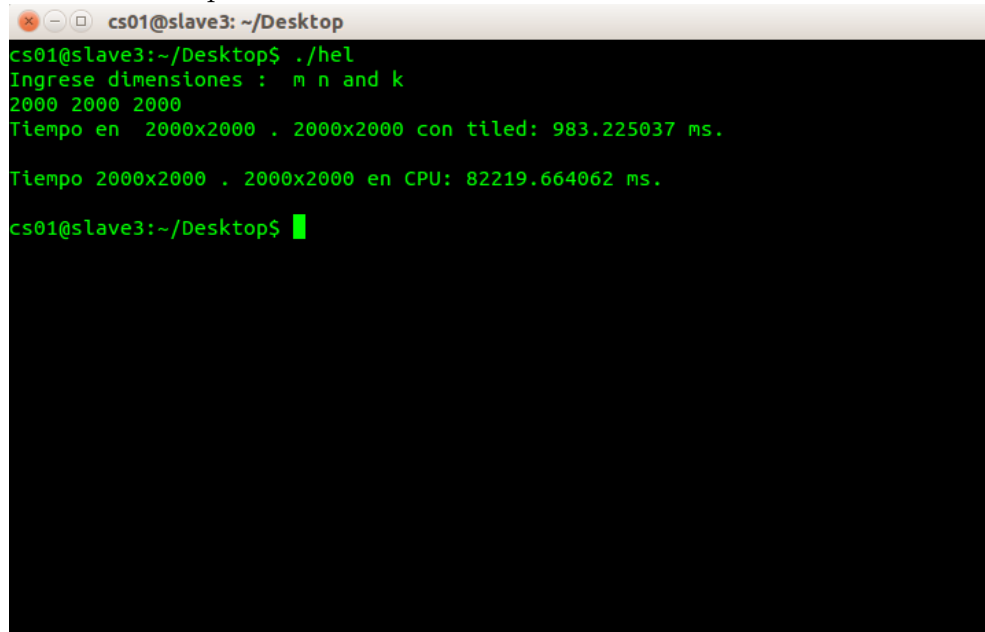


```
cs01@slave3: ~/Desktop
cs01@slave3:~/Desktop$ ./hel
Ingrese dimensiones : m n and k
2000 2000 2000
Tiempo en 2000x2000 . 2000x2000 sin tiled: 2712.588379 ms.

Tiempo 2000x2000 . 2000x2000 en CPU: 80650.859375 ms.

cs01@slave3:~/Desktop$
```

Figura 5: Tiempo de ejecución de la multiplicación de matrices ( $2000 * 2000$ ) utilizando memoria compartida



```
cs01@slave3: ~/Desktop
cs01@slave3:~/Desktop$ ./hel
Ingrese dimensiones : m n and k
2000 2000 2000
Tiempo en 2000x2000 . 2000x2000 con tiled: 983.225037 ms.

Tiempo 2000x2000 . 2000x2000 en CPU: 82219.664062 ms.

cs01@slave3:~/Desktop$
```

## 4. Conclusión

Como podemos ver en todos los casos el algoritmo ejecutado en GPU es mucho más rápido que en CPU pero aún así la misma GPU tiene algunas mejoras , como por ejemplo utilizar memoria compartida.