



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

1. Multiplicación en cpu	2
2. Multiplicación en gpu	2
2.1. Multiplicación sin memoria compartida	2
2.2. Multiplicación con memoria compartida	3
3. Resultados	4
4. Conclusión	6

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;

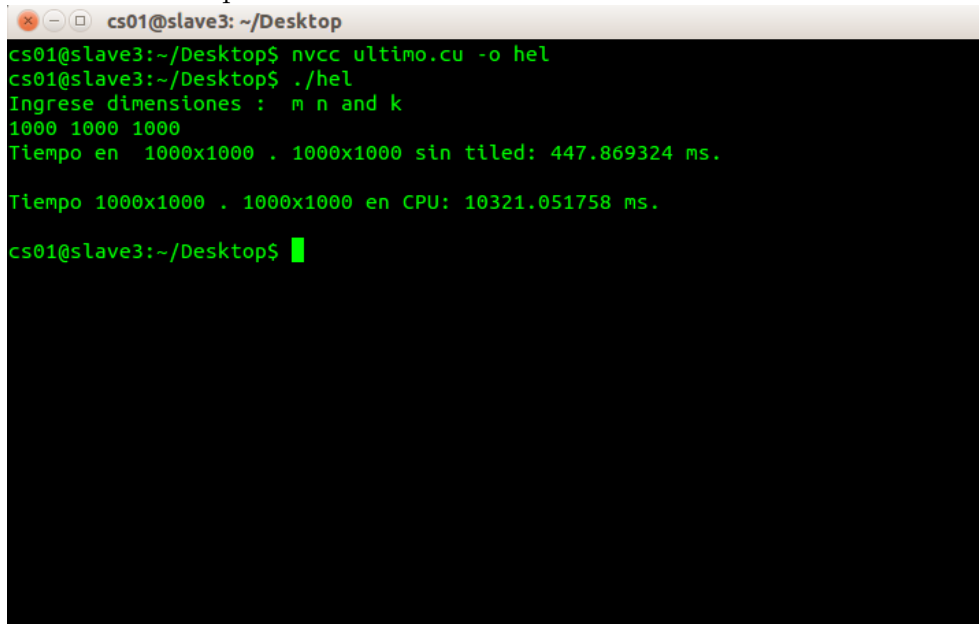
    for (int g = 0; g < n/TILED; ++g)
    {
        Mds[ty][tx] = d_a[Row*n + g*TILED + tx];
    }
}
```

```
        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;
}
```

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 1: Tiempo de ejecución de la multiplicación de matrices (1000×1000) sin utilizar memoria compartida

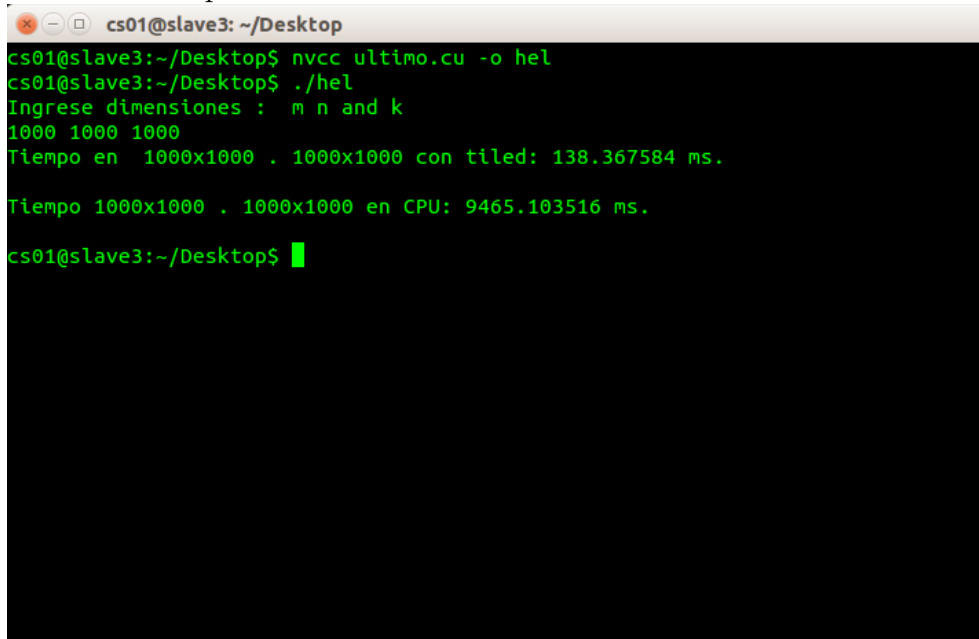


```
cs01@slave3: ~/Desktop
cs01@slave3:~/Desktop$ nvcc ultimo.cu -o hel
cs01@slave3:~/Desktop$ ./hel
Ingrese dimensiones : m n and k
1000 1000 1000
Tiempo en 1000x1000 . 1000x1000 sin tiled: 447.869324 ms.

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

cs01@slave3:~/Desktop$
```

Figura 2: Tiempo de ejecución de la multiplicación de matrices ($1000 * 1000$) utilizando memoria compartida

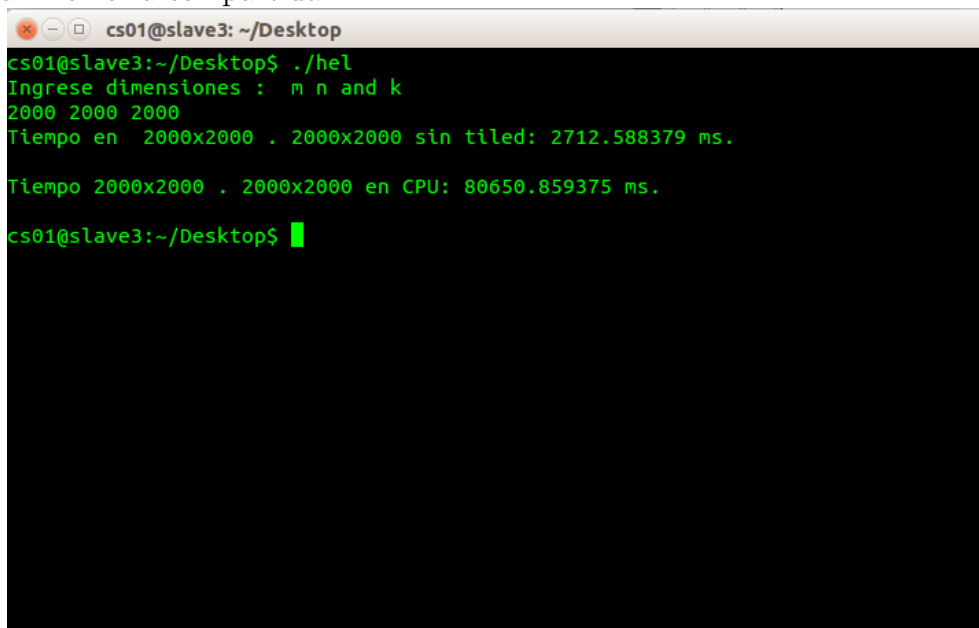


```
cs01@slave3: ~/Desktop
cs01@slave3:~/Desktop$ nvcc ultimo.cu -o hel
cs01@slave3:~/Desktop$ ./hel
Ingrese dimensiones : m n and k
1000 1000 1000
Tiempo en 1000x1000 . 1000x1000 con tiled: 138.367584 ms.

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

cs01@slave3:~/Desktop$
```

Figura 3: Tiempo de ejecución de la multiplicación de matrices ($2000 * 2000$) sin utilizar memoria compartida

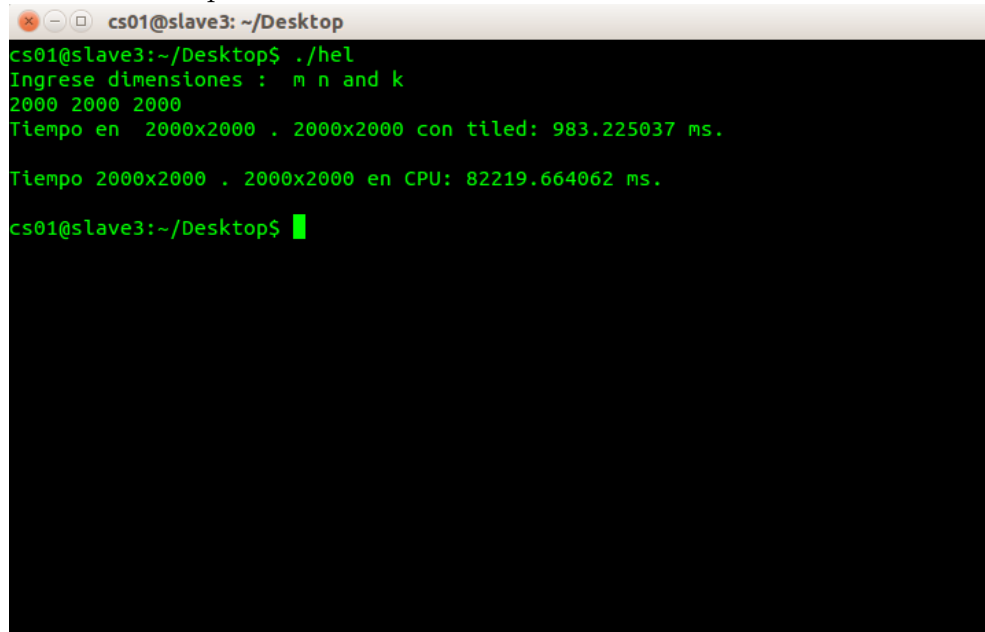


```
cs01@slave3: ~/Desktop
cs01@slave3:~/Desktop$ ./hel
Ingrese dimensiones : m n and k
2000 2000 2000
Tiempo en 2000x2000 . 2000x2000 sin tiled: 2712.588379 ms.

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

cs01@slave3:~/Desktop$
```

Figura 4: 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.