Laboratorio del curso GPGPU 2020
Operaciones DGEMM y DTRSM

1. Introducción

BLAS (Basic Linear Algebra Subprograms) es la especificación de un conjunto de rutinas que conforman los cimientos sobre los cuales se construyen la mayoría de los métodos de Álgebra Lineal Numérica. Dichas rutinas comprenden operaciones básicas sobre vectores y distintos tipos de matrices, y hoy en día existen implementaciones de esta especificación para la mayoría de las plataformas de cómputo modernas.

La especificación está organizada en tres niveles. El Nivel 1 se ocupa de operaciones que involucran vectores y escalares, el Nivel 2 se ocupa de operaciones entre una matriz y un vector, y el Nivel 3 de operaciones entre matrices.

Entre las operaciones del Nivel 3 se encuentran el producto de dos matrices generales (sin ninguna estructura particular) y la resolución de un conjunto de sistemas de ecuaciones lineales triangulares (puede verse como aquél donde el sistema ya se encuentra "escalerizado"). Dichas operaciones se denominan XGEMM y XTRSM respectivamente, donde la X denota la precisión numérica (S para simple precisión y D para doble precisión).

Como se verá a continuación, ambas operaciones presentan un alto grado de paralelismo y están íntimamente relacionadas.

2. Operación DGEMM

DGEMM realiza la siguiente operación matricial:

$$C = \beta C + \alpha A \times B$$

donde α y β son escalares, y A, B y C son matrices, con $A \in \mathbb{R}^{m \times k}$, $B \in \mathbb{R}^{k \times n}$ y $C \in \mathbb{R}^{m \times n}$.

La matriz A se representa mediante un arreglo de tipo `double` con $m \times lda$ elementos. El rectángulo superior izquierdo de de tamaño $m \times k$ del arreglo debe contener la matriz A. El resto no es accedido.

Asimismo, la matriz B se representa mediante un arreglo de tipo `double` con $k \times ldb$ elementos, donde el rectángulo superior izquierdo de tamaño $k \times n$ contiene a B, y la matriz C se representa mediante un arreglo de tipo `double` con $m \times ldc$ elementos, donde el rectángulo superior izquierdo de tamaño $m \times n$ contiene a C.

Naturalmente, es necesario que $lda \geq k$, $ldb \geq n$ y $ldc \geq n$. Estos parámetros son útiles para invocar la operación DGEMM sobre submatrices como se muestra en la Figura 1. Un ejemplo de implementación de la operación DGEMM se muestra en la Figura 2.

Esta operación exhibe un alto nivel de paralelismo, ya que cada elemento $C_{i,j}$ del resultado puede computarse de forma totalmente independiente, como el producto escalar de la fila i de A con la columna j de B.

Para reducir la cantidad de accesos a memoria global, podemos dividir las matrices en *tiles* de tamaño igual al bloque de threads, de forma que cada bloque de threads se encargue de calcular un tile del resultado. Dicho bloque de threads deberá cargar iterativamente un tile de la matriz A y uno de la matriz B en memoria compartida, y multiplicarlos acumulando el resultado parcial en un registro correspondiente a cada thread del bloque, tal como muestra la Figura 3.

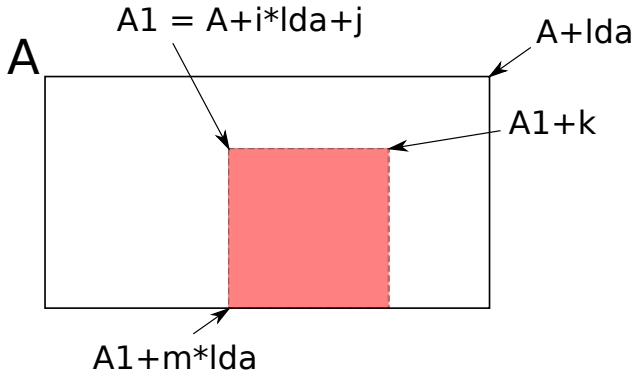

Figura 1: La figura muestra como trabajar con una submatriz de la matriz representada por el arreglo `A`. El recuadro pintado es una submatriz de tamaño $m \times k$, a partir de la posición (i, j) de A. En la memoria, dos elementos consecutivos de una columna de la submatriz, se encuentran separados por `lda` elementos.

```
void dgemm(int m, int n, int p,
           double alpha, double *A, int lda,
           double *B, int ldb,
           double beta, double *C, int ldc)
{
    int i,j,k;

    for(i = 0; i < m; ++i){

        for(j = 0; j < n; ++j)
            C[i*ldc+j] *= beta;

        for(k = 0; k < p; ++k){
            double alpha_a = alpha*A[i*lda+k];
            for(j = 0; j < n; ++j){
                C[i*ldc+j] += alpha_a*B[k*ldb+j];
            }
        }
    }
}
```

Figura 2: Versión serial (ikj) de la operación DGEMM.

Una vez que todos los bloques necesarios hayan sido multiplicados, el registro correspondiente de cada thread tendrá el valor final de $C_{i,j}$, y podrá escribirlo en la memoria global.

- a) Construya una versión sencilla de la operación DGEMM en GPU para tomar de referencia.
  - a-1) Analice la eficiencia del acceso a memoria de esta variante utilizando las herramientas vistas en el curso.
- b) Construya una versión de la operación DGEMM que aproveche la memoria compartida.
  - b-1) Compare el desempeño de las dos variantes para los siguientes tamaños:
    - A (4096×4096) y B (4096×4096)
    - A (4096×512) y B (512×4096)
    - A (512×4096) y B (4096×512)
  - b-2) Analice la eficiencia del acceso a memoria de esta variante utilizando las herramientas vistas en el curso.

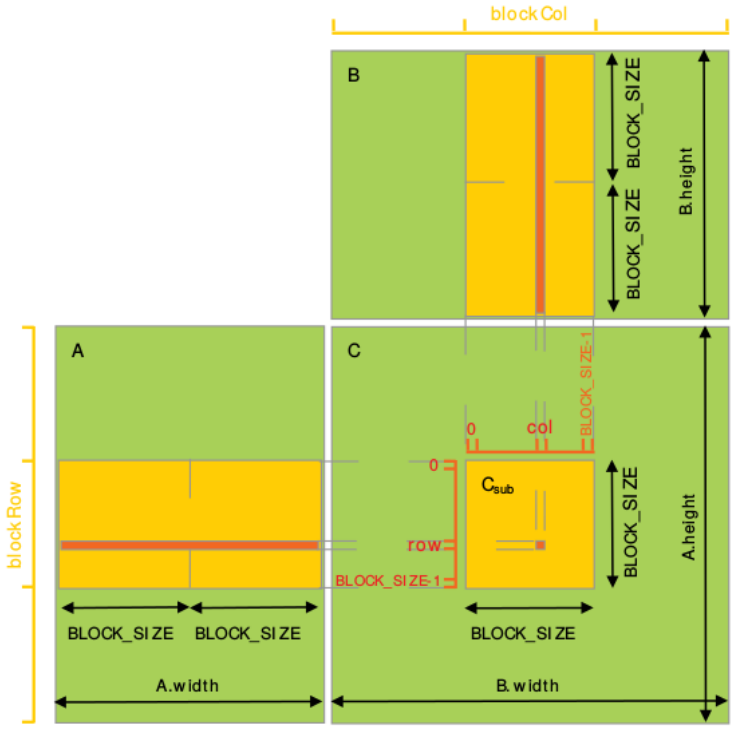

Figura 3: Multiplicación de matrices por bloques

- b-3) Utilizando *nvprof* compare la cantidad de accesos a memoria global con la versión anterior y explique los resultados.
- b-4) Analice el patrón de accesos a memoria compartida, determinando si existen conflictos de bancos.

3. Operación DTRSM

DTRSM resuelve la ecuación matricial

$$A \times X = \alpha B,$$

donde α es un escalar, X y $B \in \mathbb{R}^{m \times n}$, y $A \in \mathbb{R}^{m \times m}$ es una matriz triangular que en nuestro caso será inferior. Esto equivale a resolver n sistemas de ecuaciones de forma $Ax_i = b_i$, donde b_i es una columna de B y x_i es la solución buscada. La operación es *in-place*, es decir, el resultado de la operación se devuelve en la matriz B.

De manera similar al caso de la operación DGEMM, las matrices A y B se representan como arreglos unidimensionales de tipo `double` de $m \times lda$ y $n \times ldb$ elementos, respectivamente. En el caso de A, el triángulo inferior del bloque superior izquierdo de tamaño $m \times m$ del arreglo debe contener a A. El triángulo superior no es referenciado durante la operación.

Para facilitar la implementación de la operación en la GPU dividiremos el proceso, construyéndola incrementalmente.

3.1. Caso B de tamaño $32 \times n$

En esta variante, la matriz B (y el resultado X que se sobrescribe en B) tiene únicamente 32 filas y $n = k * 32$ columnas, donde k es la cantidad de tiles de tamaño 32×32. Esto implica que la matriz triangular inferior A será de tamaño 32×32 como se muestra en la Figura 4.

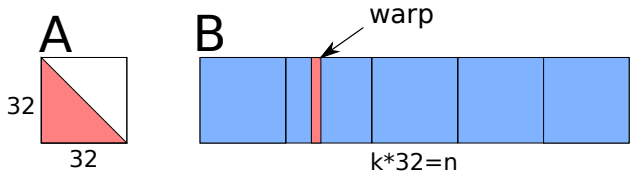

Figura 4: División en tiles de las matrices A y B. Cada bloque de threads debe sobrescribir un tile de B con resultado de la operación. Cada warp dentro del bloque procesa una columna. Cada thread del warp calcula un elemento de la columna.

La estrategia a seguir consistirá en utilizar un warp para procesar cada columna de 32 elementos, es decir, resolver uno de los n sistemas de ecuaciones.

Como todos los warps resolverán un sistema distinto con la misma matriz A, de 32×32, es conveniente almacenar dicha matriz en una memoria rápida.

Cada thread t del warp que procesa la columna i de B deberá realizar la siguiente operación:

$$B_{t,i} = \frac{B_{t,i} - \sum_{k=1}^{t-1} A_{t,k} B_{k,i}}{A_{t,t}}$$

Es importante notar que cada thread lee datos calculados por los threads del warp de índice inferior. Para compartir datos entre hilos del mismo warp, además de la memoria compartida, disponemos de las primitivas de *shuffle* incorporadas a partir de la arquitectura Kepler.

En particular, si `var` es una variable cualquiera, es posible obtener el valor de dicha variable para otro thread t del warp mediante:

```
var_thread_t = __shfl_sync(0xffffffff, var, t );
```

En este caso la máscara `0xffffffff` indica que todos los threads del warp participan del intercambio.

- a) Construya una versión de la operación DTRSM para el caso de tamaño $32 \times n$ que realice las comunicaciones entre threads del mismo warp:
  - a-1) a través de la memoria compartida.
  - a-2) utilizando la primitiva `__shfl_sync`.
- b) Utilizando *nvprof* analice y compare el desempeño de ambas variantes para los tamaños:
  - A (32×32) y B (32×512)
  - A (32×32) y B (32×4096)

3.2. Caso B de tamaño $32k \times n$

En este caso, la matriz triangular A será de tamaño $32k \times 32k$, y podemos dividir conceptualmente la misma en $k \times k$ tiles de 32×32 elementos. Encontraremos dos tipos de tiles: tiles diagonales, los cuales son matrices triangulares de 32×32, y tiles no diagonales, los cuales no poseen estructura triangular.

Para resolver n sistemas de tamaño $32k$, cada bloque de threads procesará 32 columnas de B, recorriendo los tiles de $A_{i,j}$ de A en forma secuencial, de izquierda a derecha y de arriba hacia abajo. Si el tile $A_{i,j}$ es diagonal, la operación a realizar es idéntica al caso anterior. En cambio, si $A_{i,j}$ es no diagonal, la operación a realizar es la actualización del tile B_i mediante una operación DGEMM con tiles de 32×32:

$$B_i = B_i - A_{i,j} B_j$$

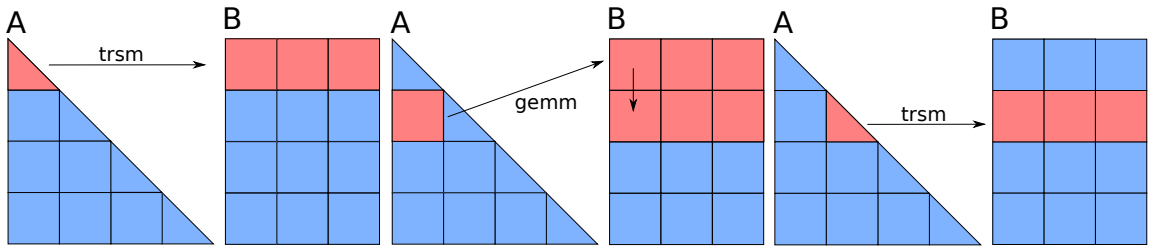

Figura 5: En la imagen de la izquierda, tres bloques resuelven los sistemas que corresponden a los tres tiles de B y el tile de A marcados en rojo. En la imagen central, los mismos tres bloques leen el tile correspondiente de la fila superior de B, lo multiplican por el tile de A marcado en rojo, y se lo restan al tile correspondiente de la segunda fila de tiles de B. En la imagen de la derecha, los tres bloques resuelven los sistemas que corresponden a los tres tiles de B y el tile de A marcados en rojo.

Observar que una operación muy similar es realizada como parte del procedimiento por tiles de la operación DGEMM de la parte anterior.

- a) Construya una versión de la operación DTRSM para el caso de tamaño $32k \times n$ utilizando la mejor de las variantes de la parte anterior.

3.3. Solución recursiva

Una vez que contamos con implementaciones de las operaciones DGEMM y DTRSM, podemos resolver la operación DTRSM de forma más eficiente utilizando un procedimiento recursivo.

Dividiendo la matriz triangular en tiles de 32×32, el paso base de la recursión puede fijarse, o bien como la operación DTRSM para el caso de tamaño $32 \times n$, o como la operación DTRSM de tamaño $32k \times n$, para un k pequeño.

El paso recursivo divide la matriz A en 4 submatrices (y a B de forma coherente con A)

$$\left(\begin{array}{cc|c} A_{11} & 0 & \\ \hline A_{21} & A_{22} & \end{array}\right)\left(\begin{array}{c} B_1 \\ \hline B_2 \end{array}\right)$$

donde A_{11} y A_{22} son matrices triangulares inferiores, y A_{21} es una matriz cuadrada, realizando las siguientes operaciones:

1. $B_1 = A_{11}^{-1}\ B_1$ (DTRSM)
2. $B_2 = B_2 - A_{21}B_1$ (DGEMM)
3. $B_2 = A_{22}^{-1}\ B_2$ (DTRSM)

Un esquema de este procedimiento se muestra en la Figura 6.

Para matrices de gran tamaño, este procedimiento tiene la ventaja de concentrar una gran proporción de su esfuerzo computacional en operaciones DGEMM.

- a) Construya una versión recursiva de la operación DTRSM. La función recursiva puede implementarse en la CPU invocando los kernels correspondientes en cada caso.
- b) Analice y compare el desempeño de esta variante con:
  - b-1) El desempeño de la variante para tamaño $32k \times n$
  - b-2) El desempeño de la función `cublasDtrsm` de la biblioteca CUBLAS.

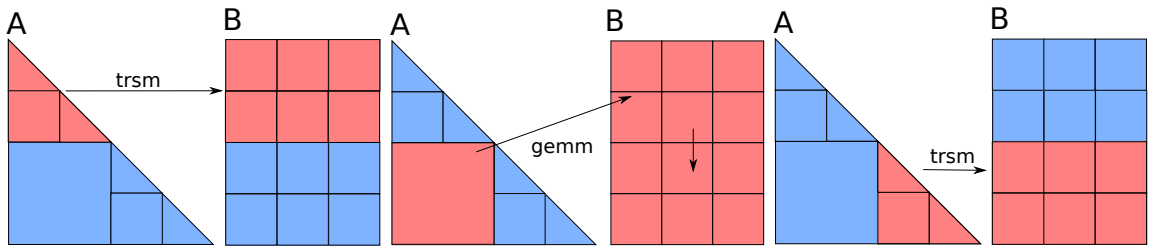

Figura 6: En la imagen de la izquierda, tres bloques resuelven los sistemas que corresponden a los seis tiles de B y los tres tiles de A marcados en rojo. En la imagen central, la submatriz de A marcada en rojo es multiplicada por la submatriz de B representada por los seis tiles superiores, restando el resultado de los seis tiles inferiores de B. En la imagen de la derecha, tres bloques resuelven los sistemas que corresponden a los seis tiles de B y los tres tiles de A marcados en rojo.

Utilice al menos los tamaños:

- A (4096 × 4096) y B (4096 × 512)
- A (4096 × 4096) y B (4096 × 4096)

Entregar

1. Un informe conteniendo una explicación detallada de la solución, incluyendo optimizaciones que haya hecho, así como el análisis de los resultados experimentales pedidos en cada parte.
2. Todos los archivos fuente que compongan la solución, incluyendo archivos de cabecera, makefiles, scripts con casos de prueba, etc.
   - **No entregar proyectos de IDEs como Eclipse o Visual Studio.** Puede utilizarlas como entorno de trabajo pero el código final debe poder compilarse mediante el comando `make` en una consola de linux.
   - El ejecutable debe recibir parámetros por la línea de comandos que permitan ejecutar cada una de las versiones con distintos tamaños de matriz, por ejemplo:

     `./labgpu20 [func] [tam1] [tam2] [tam3]`

     donde:

func=1 DGEMM en memoria global A (tam1×tam2) B (tam2×tam3)
func=2 DGEMM con memoria compartida A (tam1×tam2) B (tam2×tam3)
func=3 DTRSM A (32×32) B (32×tam1)
func=4 DTRSM A (tam1×tam1) B (tam1×tam2)
func=5 DTRSM A (tam1×tam1) B (tam1×tam2)
func=6 DTRSM de la biblioteca CUBLAS A (tam1×tam1) B (tam1×tam2)