Instituto Tecnológico Autónomo de México
Enero-Mayo 2018
Métodos Numéricos y Optimización
Resúmen sobre lectura '*Singular Value Decomposition on GPU using CUDA*'
Prof. Erick Palacios Moreno

Alumno: Federico Riveroll
Clave Unica: 105898

30 de mayo de 2018

Introducción

Los algoritmos de álgebra lineal son indispensables y fundamentales en muchísimas aplicaciones computacionales desde que existen las computadoras. En el documento que leímos se hizo énfasis en la presentación de la implementación de descomposición en valores singulares (SVD) de matrices densas, específicamente en un GPU usando el modelo de programación CUDA.

La descomposición en valores singulares presentada en el documento fue realizada usando una metodología muy particular llamada la 'bidiaginalización' seguida de la diagonalización tradicional que permite realizar mejor la paralelización.

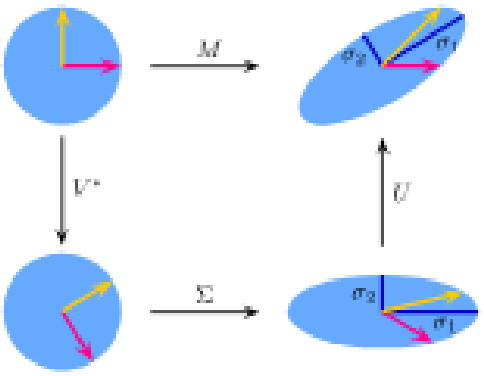
Específicamente, la descomposición en valores singulares es una técnica que se usa para factorizar matrices rectangulares reales o complejas, también para computarizar la pseudoinversa de una matriz, resolver ecuaciones lineales homogéneas, resolver minimización por mínimos cuadrados totales y también es ampliamente utilizado para la realización de PCA (Análisis de componentes principales).

La representación de la descomposición en valores singulares se puede ver como a continuación:

$$\mathbf{M} = \mathbf{U\Sigma V}(t)$$

Donde:
M es una matriz de mxn
U es una matriz ortogonal de mxm
V es una matriz ortogonal de nxn
Σ es una matriz diagonal acompletada por ceros en las dimensiones sobrantes

La interpretación intuitiva es que una matriz o transformación lineal (**M**) se puede representar de manera equivalente como una rotación (**V**(t)) más una redimensión (**Σ**) más una rotación de los ejes canónicos (**U**).



$$M = U \cdot \Sigma \cdot V^*$$

Por otra parte, el rápido incremento en rendimiento de hardware ha hecho GPU un candidato estable para muchas tareas paralelas con datos. Especialmente la serie de NVIDIA (8-series), con interfaces en lenguajes de alto nivel como C. El rendimiento de GPU ha crecido más rápido que la ley de Moore.

En el documento se presentó una implementación de una descomposición en valores singulares para matrices densas en el GPU usando el modelo CUDA que fué capaz de computarizar la descomposición en valores singulares simplificando la programación de CUDA para librerías matemáticas complejas.

Cabe señalar que varios algoritmos se habían realizado previamente en GPUs para computarizaciones como ordenamiento de arreglos, computarizaciones geométricas, multiplicación de matrices, grafos, etc.

Algoritmo SVD (descomposición en valores singulares)

El artículo indica que para resolver la descomposición en valores singulares, los autores utilizaron el algoritmo de Golub-Reinsch ya que es simple y compacto y mapea bien con la arquitectura SIMD GPU.

Algorithm 1 Singular Value Decomposition

1: $B \leftarrow Q^T AP$ {Bidiagonalization of A to B}
2: $\Sigma \leftarrow X^T BY$ {Diagonalization of B to Σ}
3: $U \leftarrow QX$
4: $V^T \leftarrow (PY)^T$ {Compute orthogonal matrices U and V^T and SVD of $A = U\Sigma V^T$}

Aquí entra un concepto interesante; La bidiagonalización, la cual se lleva a cabo cuando se diagonaliza con las primera y segunda columnas simultáneamente de la siguiente figura:

A partír de la descomposición:

A = QBT(t)

Se obtiene la siguiente descomposición:

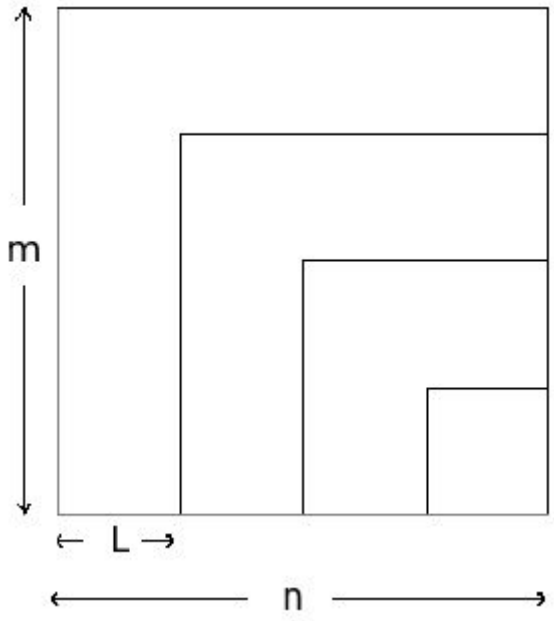
B = Q(t) AP
Mediante la siguiente figura:

$$\begin{aligned} \hat{A}_1 &= (I - \sigma_{1,1}\mathbf{u}^{(1)}\mathbf{u}^{(1)^T})A(I - \sigma_{2,1}\mathbf{v}^{(1)}\mathbf{v}^{(1)^T}) \quad (3) \\ &= H_1 A G_1 = \begin{bmatrix} \alpha_1 & \beta_1 & 0 & \dots & 0 \\ 0 & x & x & \dots & x \\ \vdots & \vdots & & & \vdots \\ 0 & x & \dots & & x \end{bmatrix}. \end{aligned}$$

$\hat{A}_1$ has zeros below the diagonal and to the right of the superdiagonal of the first row and $A(1,1)$ is updated to α_1 and $A(1,2)$ is updated to β_1. This is the first column-row elimination.

Se obtiene mediante eliminación iterativa la diagonal principal de la matriz A y la diagonal que esta junto a ésta.

De esta forma se obtienen Q y P. A continuación se utiliza una tecnica de dividir en “L” la matriz después de que las l filas se hayan diagonalizado, como se ve en la siguiente figura:



Bidiagonalización en el GPU

El algoritmo visto anteriormente se puede ejecutar usando las funciones de CUDA BLAS, ya que tiene un alto desempeño específicamente en operaciones de matriz - vector y matriz - matriz, también el esquema por bloques para la bidiagonalización se pude calcular de forma eficiente debido al gran desempeño que tiene en el producto matiz - vector y matriz - matriz.
También se discute el tema de cuál es la mejor estrategia para pasar información entre el CPU y el GPU ya que la bidiagonalización de la matriz se hace en la GPU y la memoria utilizada es menor. El pseudocódigo del algoritmo de bidiagonalización se puede ver en la siguiente figura:

Algorithm 2 Bidiagonalization algorithm

Require: $m \geq n$

1: $kMax \leftarrow \frac{n}{L}$ {L is the block size}
2: **for** $i = 1$ *to* $kMax$ **do**
3: $t \leftarrow L(i-1)+1$
4: Compute $\hat{\mathbf{u}}^{(t)}$, $\alpha_{1,t}, \sigma_{1,t}, \hat{\mathbf{k}}^{(t)}$
5: Eliminate $A(t:m,t)$ and update $Q(1:m,t)$
6: Compute new $A(t,t+1:n)$
7: Compute $\hat{\mathbf{v}}^{(t)}$, $\alpha_{2,t}, \sigma_{2,t}, \hat{\mathbf{l}}^{(t)}$
8: Eliminate $A(t,t+1:n)$ and update $P^T(t,1:n)$
9: Compute $\hat{\mathbf{w}}^{(t)}$, $\hat{\mathbf{z}}^{(t)}$ and store the vectors
10: **for** $k = 2$ *to* L **do**
11: $t \leftarrow L(i-1)+k$
12: Compute new $A(t:m,t)$ using $k-1$ update vectors
13: Compute $\hat{\mathbf{u}}^{(t)}$, $\alpha_{1,t}, \sigma_{1,t}, \hat{\mathbf{k}}^{(t)}$
14: Eliminate $A(t:m,t)$ and update $Q(1:m,t)$
15: Compute new $A(t,t+1:n)$
16: Compute $\hat{\mathbf{v}}^{(t)}$, $\alpha_{2,t}, \sigma_{2,t}, \hat{\mathbf{l}}^{(t)}$
17: Eliminate $A(t,t+1:n)$ and update $P^T(t,1:n)$
18: Compute $\hat{\mathbf{w}}^{(t)}$, $\hat{\mathbf{z}}^{(t)}$ and store the vectors
19: **end for**
20: Update $A(iL+1:m, iL+1:n)$, $Q(1:m, iL+1:m)$ and $P^T(iL+1:n, 1:n)$
21: **end for**

El autor hace énfasis en que las transferencias de CPU a GPU deben ser mínimas y que la bidiagonalización se hace en un sitio y una vez que se hace, sólamente la diagonal y la superdiagonal se copian a la CPU para dicha diagonalización.

Diagonalización en el GPU

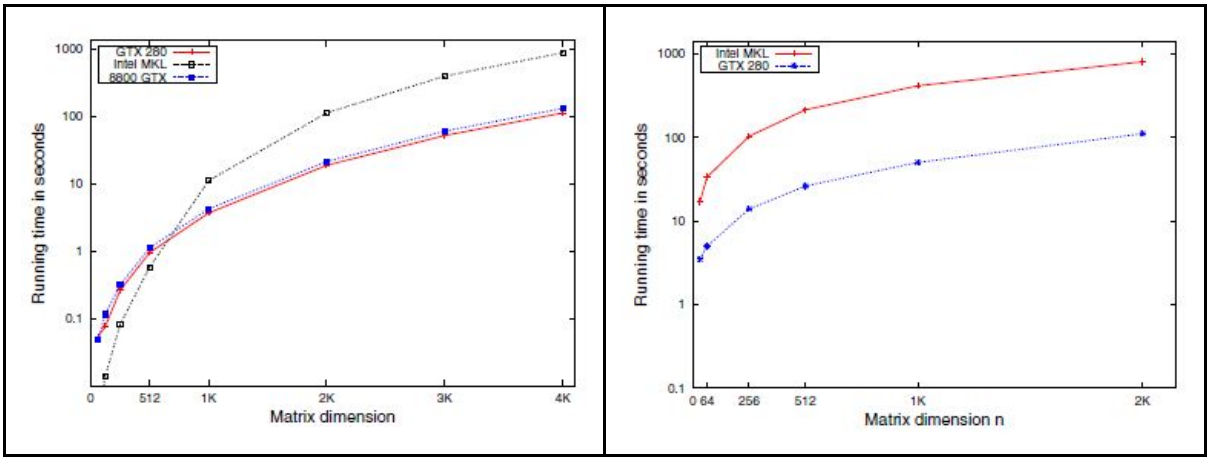
El algoritmo de diagonalización, lo que hace es dividir en bloques una matriz y operar sobre cada elemento de cada renglón haciendose eficientemente con la arquitectura de CUDA ya que cada hilo es independiente de los demás, los datos que se usan en memoria compartida y las actualizaciones suceden en un ciclo que va modificando dos renglones simultáneamente, de tal modo que el primer renglón (de los dos seleccionados) es el único que se regresa, el otro se mantiene en memoria ya que será el “primer renglón” de la siguiente iteración.

De esta forma se efectúan los cálculos de los dos últimos algoritmos descritos en el artículo en la NVIDIA (los paralelos).

Finalmente en esta sección se efectuan un par de ejemplos de multiplicaciones matriz - matriz usando CUBLAS para obtener los valores singulares que vendría siendo la diagonal de la matriz Σ

Resultados

Los autores realizaron experimentos para comparar la velocidad y rendimiento de varias implementaciones optimizadas de descomposición en valores singulares, usando MATLAB y LAPACK (Intel MKL) probado con 10 matrices diferentes y en varias computadoras. Los resultados son los siguientes:



Conclusiones

En el artículo visto se presentó la implementación de la completa descomposición de valores singulares en GPUs. El algoritmo enseñado explota la paralelización en la arquitectura GPU y logra un rendimiento computacional sobresaliente con respecto a sus 'competidores'. La bidiagonalización de una matriz se realiza completamente en el GPU usando la librería CUBLAS para lograr la máxima capacidad posible.

Se utilizó una implementación híbrida para la diagonalización que divide los procesos computacionales entre el CPU y el GPU dando resultados excelentes en 'performance'.

Por el momento los GPUs son limitados a números de precisión singular, pero eso está cambiando con las nuevas generaciones. El error dada la precisión menor fue muy pequeño en las matrices con las que se experimentó. No obstante, el enfoque utilizando CUDA y las librerías utilizadas pueden ser usadas para resolver muchas otras tareas tanto gráficas como no gráficas.

La paralelización en todos los niveles es un problema que vale la pena resolver, optimizar y mejorar constantemente. Los procesos asíncronos con el poder de cómputo actual tienen que hacer frente a la muy creciente oferta de “datos grandes”.