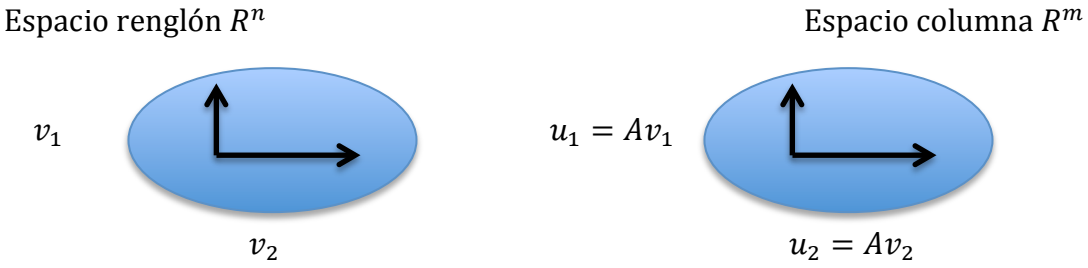
En los últimos tiempos el procesamiento de grandes cantidades de datos ha lanzado un reto sobre la posibilidad de hacer cálculos a gran velocidad. En ese sentido, resulta crucial la velocidad de cómputo de los mismos. En consecuencia, muchos investigadores han buscado formas nuevas de trabajar con matrices densas que incorporan grandes cantidades de información. Concretamente, ante el incremento del potencial de cómputo/procesamiento de las tarjetas gráficas, la investigación se ha centrado sobre el desarrollo de métodos de cómputo en paralelo aprovechando esta característica.

En particular, el trabajo de Lahabar y Narayan, presenta la implementación de la descomposición en valores singulares (SVD) de una matriz densa en GPU usando el modelo CUDA y además se compara con otras técnicas que buscan lo mismo. En particular se contrasta con los resultados de MATLAB e Intel de la librería Math Kernel que usa LAPACK y CPU en su procesamiento. Como se ha visto en clase para ciertos propósitos el uso del GPU puede incrementar la velocidad de cálculo, en este trabajo en particular se observa que puede ser 60 veces más rápido que MATLAB y hasta 8 veces más que INTEL Math Kernel.

¿Qué es una Descomposición en Valores Singulares?

Una transformación lineal, es una matriz que multiplica un vector asociando valores de un conjunto del dominio a otro del condominio. Existe un caso en particular cuando tenemos una base ortogonal en el espacio renglón que queremos trasladar el espacio columna que aplicando una multiplicación por la matriz A como se muestra en la figura.



Sin embargo al multiplicar por la matriz A los vectores v, nada nos garantiza que los vectores u sean ortogonales. Ese es el objetivo de la descomposición en valores singulares de una matriz. Encontrar la matriz A que nos permita "transformar" una base ortogonal en el espacio renglón en otra base ortogonal en el espacio columna. Además no sólo será ortogonal sino también ortonormal. En ese sentido tendremos:

$$A(v_1 \quad v_2 \quad \cdots \quad v_r) = (u_1 \quad u_2 \quad \cdots \quad u_r)\begin{pmatrix} \sigma_1 & 0 & 0 & 0 \\ 0 & \sigma_2 & 0 & 0 \\ 0 & 0 & \ddots & 0 \\ 0 & 0 & 0 & \sigma_r \end{pmatrix}$$

Lo anterior se puede expresar como $AV = U\Sigma$. Podemos despejar A de la ecuación anterior, notando que la inversa de V es la misma que su transpuesta. En consecuencia, tenemos un matriz $A = U\Sigma V^T$. Sin embargo es deseable contar con el lado derecho de la igualdad con puros términos de V ya que es el espacio que conocemos.

$$A^T A = V\Sigma^T U^T \, U\Sigma V^T$$

$$A^T A = V \begin{pmatrix} \sigma_1^2 & 0 & 0 & 0 \\ 0 & \sigma_2^2 & 0 & 0 \\ 0 & 0 & \ddots & 0 \\ 0 & 0 & 0 & \sigma_r^2 \end{pmatrix} V^T$$

Las propiedades de la matriz $A^T A$ son importantes (positiva, simétrica y definida) y del otro lado de la igualdad tendremos los eigenvectores y eigenvalores. Cabe señalar, que la transformación lineal se expresa como una multiplicación de matrices. En ese sentido, como vemos la descomposición en valores singulares representa la multiplicación por 3 tipos de matrices especiales en este caso dos ortogonales y una diagonal.

¿Cuál es la propuesta de los autores para una SVD?

Los autores propone el uso del algoritmo GolubReinsch, que usa bidiagonalización y diagonalización para concretar el objetivo de SVD. Además comparado con otros algoritmos (el Hestenes por ejemplo) el GolubReinsch resulta adecuado ya que además de ser usado por LAPACK se adapta bien a la arquitectura de GPU.

La idea principal para hacer la descomposición es primero reducir la matriz original a una matriz bidiagonal usando transformaciones de householder. Posteriormente la matriz bidiagonal se diagonaliza a través de iteraciones del tipo QR (algoritmo desarrollado por Francis y Kublanovskaya para el cálculo de vectores y valores propios de una matriz).

Bidiagonalización

Lo primero que se debe hacer es una bidiagonalización que descompone la matriz original en:

$$A = QBP^T$$

En ese sentido, mediante el uso de CUBLAS se pretende llegar a ese resultado. El trabajo discute 2 puntos importantes:

1. La interacción CPU/GPU

2. La construcción de bloques para la optimización computacional.

   En el punto 2 los autores señalan claramente que las matrices cuyas dimensiones son múltiplos de 32 son mejores para el cómputo. Lo anterior en línea con nuestro trabajo de reconocimiento de imágenes con redes neuronales en CUDA. También, se hace hincapié en minimizar las tranferencias de CPU a GPU para evitar tener pérdidas en la eficiencia al momento del cómputo.

   Todas las operaciones de esta sección se computan en GPU utilizando por debajo la librería CUBLAS. En ese sentido, en línea con lo anterior como el proceso se hace en data del GPU, no existen transferencias entre el CPU y el GPU.

Diagonalización

Para lograr la diagonalización de la matriz bidiagonal se utilizar el algortimo QR, que descompone la matriz bidiagonal en:

$$\Sigma = X^T BY$$

Donde sigma es la matriz diagonal y B y Y son matrices unitarias y ortogonales.

En esta sección los autores también minimizan las interacciones entre CPU y GPU. Sin embargo, en este paso sí existe comunicación entre CPU y GPU para el cómputo, pero de manera eficiente. Entrando al detalle señalan que la división de renglones en bloques es eficiente en la arquitectura de CUDA ya que cada thread es una operación independiente. Además cada bloque se almacena en memoria compartida que a su vez mantiene la eficiencia en el cómputo.

Completando el SVD

Para completar el proceso se hace multiplicaciones de matrices con CUBLAS:

$$U = QX$$
$$VT = (PY)T$$

Finalmente las matrices ortogonales U y VT puede ser copiadas al CPU.

Resultados

Como se mencionó al principio la idea es hacer los cálculos de manera rápida y eficiente. En ese sentido, en esta sección los autores discuten y comparan las distintas alternativas para ejecutar una SVD.

Para esto se generaron 10 matrices densas aleatoriamente y se corrió la SVD en las distintas alternativas discutidas en un inicio (CUDA, MATLAB e INTEL). Los resultados fueron:

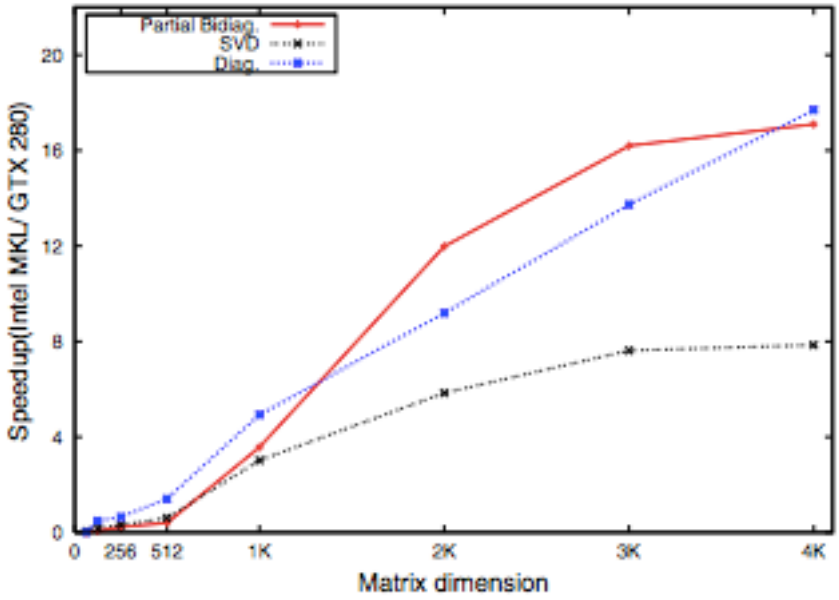


Figure 5. Speedup for square matrices for SVD, Partial Bidiagonalization and Diagonalization on GTX 280 over Intel MKL

Esta figura muestra que tan rápido se hace una SVD de matrices gigantes en NVIDIA(GTX280) comparado con la alternativa de Intel.

Conclusiones

Los autores presentaron una alternativa a los métodos tradicionales para lograr una SVD. En ese sentido, el método propuesto y explicado en este resumen no sólo es más rápido sino que tiene la capacidad de trabajar para matrices más grandes que otras opciones como Intel. Si bien existen errores pequeños en el cálculo(derivado de la precisión de los números), estos no resultan importantes(el error fue menor al .001%). En ese sentido, el procedimiento para SVD aquí descrito puede ser utilizado de manera confiable y eficiente.

Referencias:

Liga al paper:

https://s3.amazonaws.com/academia.edu.documents/30806706/Sheetal09Singular.pdf?AWSAccessKeyId=AKIAIWOWYYGZ2Y53UL3A&Expires=1527795587&Signature=8TSbpC7ykRa7BTH7qaBaGe3beTU%3D&response-content-disposition=inline%3B%20filename%3DSingular_value_decomposition_on_GPU_usin.pdf