ITAM
*Roberto Acevedo 89750*
*MNO-2018*
*Reporte "SVD on GPU using CUDA"*

# Reporte de Lectura: "Singular Value Decomposition on GPU using CUDA by Sheetal Lahabar, P J Narayanan "

## 1. Objetivo

El objetivo de la lectura es mostrar paso por paso el cómputo de la descomposición de valores singulares de una matriz de m x n de la forma:

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

También, propone una metodología de 3 pasos y detallando cada uno desde el punto de vista teórico. Adicionalmente, busca brindarle al lector un contexto de como este problema ha sido abordado por diferentes personas y utilizando. Y finalmente, contrasta los resultados del algoritmo en dos diferentes GPU's y una tercera opción utilizando MATLAB.

## 2. Contexto del Problema

Si bien el artículo hace mención de muchos autores y sus intentos de efectuar la descomposición de valores singulares bajo diferentes ángulos, el que más llamó mi atención fue la de los autores: Ma, Weiwei., Kaye, M., Luke, D. and Doraiswami, R., los cuáles propusieron un algoritmo de que utiliza rataciones por ambos lados del método Jacobi en arquitecturas FPGA. Más adelante Zhang Shu presentó una implementación "Ones sided Jacobi" utilizando CUDA.

## 3. Algoritmo

El autor propone el uso del algoritmo Golub-Reinsch, el cual contempla una bidiagonalización y diagonalización de una matriz, siendo el algoritmo el siguiente:

a) $B \leftarrow Q^T A\,P$ . Que es la bidiagonalización de la matriz A a B utilizando matrices de "Householder" y utilizando funciones CUBLAS en la GPU.
b) $\Sigma \leftarrow X^T B Y$ . Que es la diagonalización de B a $\Sigma$ utilizando factorización QR y guardando los resultados en el CPU, pero realizando el paso en el GPU. Nuevamente se utilizan funciones CUBLAS
c) Se calcula $U = QX$ y $V^T = (PY)^T$ a partir de los valores previamente guardados y se procede a guardarlas en el CPU. De tal manera que se queda en el CPU las matrices $U\,\Sigma\,V^T$. Para el cálculo de $U$ y $V^T$ se utilizan rutinas de multiplicación de matrices de CUBLAS.

## 4. Resultados

Se compararon 4 métodos de computo de SVD con el algoritmo planteado, utilizando diferentes tecnologías. Por un lado se utilizó MATLAB con un Intel MKL y LAPACK, y por otra se permitió "threading" dinámico para mejorar el desempeño. Las otras 2 opciones fueron utilizar una GPU NVIDIA GeForce 8800GTX solamente, y la otra opción una combinación de una NVIDIA GTX 280 en conjunto con una NVIDIA Tesla S1070. Los resultados en los tiempos de computo fueron los siguientes:

| SIZE | SVD MATLAB | SVD MKL | SVD GTX 280 | SVD 8800 | Speedup MKL/280 |
|---|---|---|---|---|---|
| 64 × 64 | 0.01 | 0.003 | 0.054 | 0.048 | 0.05 |
| 128 × 128 | 0.03 | 0.014 | 0.077 | 0.116 | 0.18 |
| 256 × 256 | 0.210 | 0.082 | 0.265 | 0.319 | 0.31 |
| 512 × 512 | 3.19 | 0.584 | 0.958 | 1.129 | 0.61 |
| 1K×1K | 72 | 11.255 | 3.725 | 4.28 | 3.02 |
| 2K×2K | 758.6 | 114.625 | 19.6 | 21.656 | 5.84 |
| 3K×3K | 2940 | 402.7 | 52.8 | 61.31 | 7.62 |
| 4K×4K | 6780 | 898.23 | 114.32 | 133.68 | 7.85 |
| 1K×512 | 5.070 | 2.27 | 1.523 | 3.749 | 1.48 |
| 2K×512 | 10.74 | 12.8 | 3.118 | 4.072 | 4.11 |
| 4K×512 | 34.33 | 54.7 | 8.311 | 12.418 | 6.58 |
| 8K×32 | 24.310 | 17.112 | 3.506 | - | 4.88 |
| 8K×64 | 47.87 | 33.7 | 5.016 | - | 6.72 |
| 8K×256 | 107.57 | 103.8 | 13.96 | - | 7.4 |
| 8K×512 | 137.98 | 215 | 26.33 | - | 8.16 |
| 8K×1K | 254.26 | 417 | 50.364 | - | 8.2 |
| 8K×2K | 1371.9 | 808 | 111.3 | - | 7.25 |

Table 1. Total computation time for SVD (in seconds) for different matrices

Como se puede apreciar, en matrices cuadradas de menor o igual dimensión a 256x256 las opciones de MATLAB y "multithreading" fueron más eficientes. Para matrices de 512x512 todavía fue más rápida la opción de MKL, sin embargo a partir de 1000x1000 el desempeño de las GPU's fue mucho mejor. Mientras que las matrices de mxn la GPU GTX 280 tuvo el mejor desempeño.