Singular Value Decomposition on GPU using CUDA

Mario Vázquez Corte 127252

May 2018

1 Introduction

El paper se enfoca en el método de descomposición por valores singulares de una matriz (SVD) para calcular su pseudoinversa. Esta se usa en problemas muy importantes como OLS y solución de sistemas lineales. Este método no se limita a matrices cuadradas. En general diremos que la matriz A es de $dim(A) = m*n$. El articulo se enfoca en el calculo de dicha descomposición através del método de Golub-Reinsch (**GR**) implementado den GPGPU con ayuda de la libreria de CUBLAS.

Para facilitar la exposición utilizaremos la misma notación que el paper.

2 Algoritmo GR en CUDA

El algoritmo (**GR**) consta de 3 etapas principales y 4 descomposiciones en total que permiten hacer uso del número masivo de núcleos con los que constan los GPGPUs:

1. Bidiagonalización de A
   - $B = Q'AP$
2. Diagonalización de B
   - $\Sigma = X'BY$
3. Completar SVD
   - $U = QX$
   - $V = (PY)'$

2.1 Bidiagonalización de A

La primero se calcula una descomposición de A en QBP'. Esto se logra atraveś del método de proyecciones de Householder que otroga gran estabilidad númerica. Es importante notar que B es una matriz bidiagonal, mientras que las otras dos son de tipo unitario. Este método es computacionalemente pesado, pero de gran estabilidad númerica. Varias librerias utilizan este método, por ejemplo MAGMA o CUBLAS nivel 2. Notar que esta transformación nos da como resultado una propiedad muy importante: $Q' = Q^{-1}$.

2.2 Diagonalización de B

Ahora buscamos diagonalizar la matriz B obtenida en el paso anterior, de tal manera que obtangamos $\Sigma = X'BY$ con X' & Y matrices ortogonales y unitarias. Notar que este paso necesita el calculo de la matrices anteriores

Esta parte del algoritmo tiene partes secuenciales que se realizan en CPU (pues este es más rápido para operaciones secuenciales), y otras partes se realizan en el GPGPU. El calculo de las filas de Y, y el calculo de las columnas de X son dependientes de sus columnas y filas respectivas anteriores, pero los coeficientes de la misma fila o columna no lo son. De ahí la extensión natural al paradigma paralelo. El paper se enfoca en el manejo de memoria compartida de los devices y como esto ayuda a minimizar la latencia de comunicación entre memorias de distinta jérarquia. Discutiremos un poco más de esto en la conclusión.

2.3 Completar SVD

En esta parte se desea obtener $U = QX$ y $V = (PY)'$. Notar que para este paso se necesitan los calculos de los dos pasos enteriores. La multiplicación se lleva acabo con CUBLAS de nivel 3 pues son operaciones de matriz por matriz. Sería conveniente comparar los tiempos de múltiplicación en CPU contra GPU pues se peude tener latencia por comunicación entre device-host y manejo de memoria.

3 Conclusión

Los autores comparar su algoritmo en GPGPU contra el algoritmo de SVD de Matlab y de MKL(intel). Notar que el algoritmo de Matlab es de propietario; y aunque MKL es gratuito para procesadores Intel, las librerias no estan del todo documentadas. Por lo anteriror el algoritmo resulta de gran utilidad para la comunidad, pues es de fácil acceso. El punto problemático es que al igual que MKL, CUBLAS esta bajo la misma licencia.
El método (**GR**) es superior para matrices con dimensiones mayores a $1000 * 1000$, pero inferior para matrices cuadradas o de dimensiones más pequeñas. Como se menciono en el punto 2.3, puede que su algoritmo mejore si se utilizará

la libreria de MKL en el mismo paso, y se realizara multiplicación por bloques através de hilos. Pienso que seria importante explorar esta opción.