Singular Value Decomposition on GPU using CUDA (Sheetal Lahabar & P. J. Narayanan)

Lorena Malpica 124841

31 de mayo de 2018

Abstract

Para diversas aplicaciones computaciones son fundamentales diversos algoritmos de álgebra lineal. Los GPU modernos son adecuados para una gran cantidad de tareas generales y son una alternativa barata para co-procesar tareas de gran magnitud. El artículo presenta una implementación de Descomposición en Valores Singulares (SVD por sus siglas en inglés), para una matriz densa en un GPU utilizando el modelo de programación CUDA. Se implementa el proceso de SVD utilizando pares de pasos de bidiagonalización seguido diagonalización. Esta implementación es más eficiente que implementaciones hechas en un CPU, llegando a ser 60 veces más rápido que la implementación en MATLAB y 8 veces más rapido que una implementación en un equipo Intel Dual Core de 2.88 GHx con NVDIA GTX 280 para matrices grandes.

Introducción

La Descomposición en Valores Singulares es una técnica importante usada para factorizar matrices reales rectangulares o complejas. El uso de esta técnica permite que diversas operaciones matriciales sean más robustas. Algunas aplicaciones son, obtener la pseudoinversa de una matriz, resolver ecuaciones lineales homogéneas, y aplicaciones relacionadas al análisis de componentes principales. También hay muchas aplicaciones en el área de cómputo científico, en procesamiento de señales, control automático, etc.

La descomposición en valores singulares de una matriz A de dimensiones $m \times n$ es cualquier factorización de la forma:

$A = U\Sigma V^T$,

Donde U es una matriz ortogonal de tamaño $m \times m$, V es una matriz ortogonal de tamaño $n \times n$ y Σ es una matriz diagonal de tamaño $m \times n$ con elementos $s_{ij} = 0$ si $i \neq j$ y $s_{ii} \geq 0$ en orden decreciente a lo largo de la diagonal.

El rápido incremento en el desempeño del hardware gráfico y de las GPU, los han vuelto los candidatos ideales para realizar muchas tareas de cómputo intensivo, particularmente tareas de procesamiento de datos en paralelo. Las GPUs actuales contienen unidades de procesamiento completamente programables y que soportan operaciones de punto flotante vectorizadas. Lenguajes de alto nivel han surgido para dar apoyo en la programación de este tipo de tareas en las GPUs. El GPU NVDIA serie 8 con el ambiente de programación CUDA provee el estándar similar a C para programar estos procesadores. Cabe mencionar que el desempeño de las GPUs ha estado creciendo a un ritmo más rápido que el dictado por la Ley de Moore. Recientemente las GPUs se han usado extensivamente para cómputo científico. Sin embargo, muy poco trabajo se ha hecho para resolver problemas como SVD que tiene muchísimas aplicaciones. En el artículo se presenta la implementación de SVD para matrices densas en una GPU utilizando el modelo CUDA. Se ocupa la librería de NVIDIA CUBLAS y kernels de CUDA. Además de acelerar notablemente el procesamiento de la implementación se demuestra la facilidad con la que se puede usar CUDA para programa aplicaciones matemáticas complejas.

Trabajo relacionado

Muchos algoritmos se han desarrollado en las GPUs para cómputo matemático como ordenamiento, cómputo geométrico, multiplicación de matrices y algoritmos de grafos. Ha habido muchos esfuerzos dedicados a optimizar Procesadores Gráficos CUBLAS de nivel 3. También se han hecho trabajos para paralelizar el algoritmo de SVD en arquitecturas como FPGA, Procesadores Cell, GPU, etc. Un ejemplo de una limitación en el desarrollo de estas aplicaciones es la cantidad de memoria compartida disponible. Por otro lado también existen diversas librerías numéricas como ATLAS y la librería de Intel Math Kernel, que se utilizan ampliamente para programar diferentes aplicaciones en la CPU. Éstas están diseñadas para alcanzar una alta precisión así como un gran ancho de banda de memoria y un gran rendimiento computacional en las CPUs.

Algoritmo SVD

La descomposición en valores singulares de una matriz A se puede calcular usando el algoritmo de Golum-Reinsch (Bidiagonalization and Diagonalization) o el método de Hestenes. En el artículo se ocupó el método de Golub-Reinsch por su simplicidad y por ser compacto, también porque mapea bien con la arquitectura del SIMD GPU. El algoritmo de Golub-Reinsch se usa en el paquete LAPACK y es un algoritmo de dos pasos. La matriz primero se reduce a una matriz bidiagonal, a través de una serie de transformaciones. La matriz resultante es a su vez diagonalizada. El algoritmo se describe a continuación.

Algoritmo 1: Descomposición en Valores Singulares (SVD)

1:$B \longleftarrow Q^T AP$ {Bidiagonalización de A a B}

2:$\Sigma \longleftarrow X^T BY$ {Diagonalización B a Σ}

3:$U \longleftarrow QX$

4:$V^T \longleftarrow (PY)^T$ {Cálculo de las matrices ortogonales U y V^T y la descomposición de valores singulares de $A = U\Sigma V^T$}

Bidiagonalización

En este paso la matriz A se descompone como:

$A = QBP^T$

aplicando una serie de transformaciones de Householder. Una transformación de Householder es una transformación lineal del espacio que consiste en una reflexión pura con respecto a un plano.

Diagonalización de la matriz bidiagonal

La matriz bidiagonalizada se puede reducir a una matriz diagonal por medio de aplicaciones iterativas del algoritmo "implicitly shifted QR". La matriz B obtenida en el primer paso se descompone como:

$\Sigma = X^T BY$

Donde Σ es la matriz diagonalizada y X y Y son matrices ortogonales unitarias.

Completar la Descomposición en Valores Singulares

Se realizan dos multiplicaciones de matrices al final para obtener las matrices ortogonales $U = QX$ y $V^T = (PY)^T$. Se usan las rutinas de CUBLAS para rutinas de multiplicación de matrices. Las matrices Q, P^T, X^T, U y V^T se encuentran en el dispositivo. Las matrices ortogonales U y V^T se pueden copiar en este momento al CPU. Los valores singulares son los elementos diagonales $d(i)$ de Σ y se encuentran en el CPU.

Resultados

Se analizó el desempeño del algoritmo tanto en las implementaciones optimizadas para el CPU en MATLAB y en Intel MKL 10.0.4 LAPACK. Para la implementación en Intel MKL se ocupa threading dinámico para mejorar el desempeño. El algoritmo se prueba en un equipo Intel Dual Core de 2.66 GHz con un procesador gráfico NVIDIA GeForce 8800 GTX, un procesador NVDIA Tesla S1070 y NVDIA Tesla S1070. NVDIA TESLA S1070 tiene 4 procesadores Tesla T10 y un total de 16 GB de memoria y puede alcanzar un desempeño teorético máximo de 4TFLOPS. Se ocupó una GPU Tesla S1070 con 240 núcleos y 4 GB de memoria que tiene un poder computacional de 1 TFLOPS. La 8800 GTX tiene 128 procesadores de flujo divididos en 16 multiprocesadores y un total de 768 MB de memoria. La GTX 280 tiene 240 procesadores de flujo divididos en 30 multiprocesadores con un total de 1 GB de memoria. De acuerdo a NVDIA la GTX 280 puede alcanzar un desempeño de 933 GFLOPS y la 8800 GTX puede alcanzar un desempeño de 345.6 GFLOPŜ. Sin embargo la GTX 280 y la 8800 GTX dan un desmpeño de 375 y 120 GFLOPS respectivamente para multiplicación de matrices usando CUBLAS. Se usó la Intel Core 2 Duo CPU E6750
con un procesador de 2.66 GHz que tiene un desempeño de 22.4 GFLOPS.

Se generaron 10 matrices aleatorias densas para cada tamaño, y se ejecutó el algoritmo de SVD en cada matriz 10 veces. Se promedió el resultado para las 10 matrices en cada tamaño.

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

Figure 1: Tiempo total en calcular la Descomposición en Valores Singulares en segundos

Conclusiones

El artículo presentó la implementación completa de una Descomposición en Valores Singulares en GPUs. El algoritmo aprovecha el paralelismo disponible en la arquitectura de la GPU y consigue un gran desempeño computacional. La bidiagonalización de la matriz de logra realizar completamente en la GPU utilizando la

librería optimizada CUBLAS para alcanzar el máximo rendimiento. Para la diagonalización de la matriz se hizo una implementación híbrida que dividía los cálculos entre el CPU y el GPU y también daba un muy buen desempeño. Se pudo calcular la Descomposición en Valores Singulares para matrices muy grandes, hasta del orden de 14K, lo cual es imposible que se realice en el CPU debido a las limitaciones de la memoria. Las GPUs están limitadas a números de precisión simple, aunque esto está cambiando en las nuevas generaciones. El error encontrado debido a esta menor precisión fue de menos del .001% en las matrices que se ocuparon en los experimentos.