Mirtha Ayala Macías

Singular Value Decomposition on GPU using CUDA

By Sheetal Lahabar, P J Narayanan

SVD es un método de factorización de una matriz. Las matrices que usan SVD son más robustas a errores numéricos. SVD de una matriz mxn es una factorización de la forma $A=U\sum V^T$ donde U es una matriz ortogonal m x m, V es una matriz ortogonal v x v y $\sum$ es una matriz diagonal m x n con los elementos distintos a la diagonal igual a cero y mayores a cero los que están en la diagonal.

Las mejoras en el desempeño del hardware han hecho de GPU un candidato fuerte para desempeñar tareas de cómputo intensivo. Nuevos lenguajes de alto nivel NVIDIA's 8series GPU with CUDA provee a C como el lenguaje de interface para los procesadores programables. CTM es otra interface para programar GPU's.

GPU's han sido utilizados recientemente para hacer cómputo científico aunque poco para resolver problemas como SVD. Se mencionan algunos esfuerzos realizados para la implementación de SVD algunos más eficientes que otros.

En el paper se mencionan diferentes trabajos que has buscado permitir implementar SVD optimizando diferentes aspectos (ej. matrices grandes, pequeñas, métodos, etc.) con el uso de diferentes lenguajes y librerías en conjunto con el poder computacional de GPU's.

Se muestran diferentes métodos para implementar SVD de una matriz y menciona las ventajas y desventajas de cada uno de ellos:

Golub-Reinsch se usa en el paquete de LAPACK. Es un algoritmo de dos pasos: la primera parte consiste en reducir la matriz a una bidiagonal usando una serie de transformaciones. Posteriormente, la matriz bidiagonal se diagonaliza a través de iteraciones QR.

En la tabla abajo se describe el algoritmo SVD para una matriz A:

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 $A = U\Sigma V^T$}

El artículo detalla el algoritmo de Bidiagonalización y su implementación en la GPU con CUDA BLAS que proporciona funciones de alto rendimiento matriz-vector, matriz-multiplicación y norma de cálculo. Posteriormente se describe el algoritmo de Diagonalización: la matriz bidiagonal se puede reducir a una matriz diagonal mediante la aplicación iterativa del algoritmo QR modificado. La matriz B obtenida en el primer paso se descompone como $\Sigma=XTBY$ donde Σ es la matriz diagonal X y Y las matrices unitarias ortogonales.

Posteriormente se describe el algoritmo de diagonalización en GPU y su implementación: los elementos diagonales y superdiagonales de B se copian en la CPU. Aplicando rotaciones Givens en B y el cálculo de los vectores de coeficientes se realiza de forma secuencial en la CPU, ya que solo requiere acceso a los elementos diagonales y superdiagonales.

Finalmente se realizan dos multiplicaciones matriz-matriz al final para calcular las matrices ortogonales U = QX y V T = (PY) T Usando rutinas de multiplicación de matrices CUBLAS. Las matrices Q, PT, XT, Y T, U y V T están en el dispositivo. Las matrices ortogonales U y V T se pueden copiar a la CPU. d (i) 'contiene los valores singulares, es decir, elementos diagonales de Σ y está en la CPU.

En la sección de resultados, se presenta un análisis del rendimiento del algoritmo con la implementación optimizada de CPU de SVD en MATLAB y Intel MKL 10.0.4 LAPACK. Probaron el algoritmo en una PC Intel Dual Core 2.66GHz y un procesador de gráficos NVIDIA GeForce 8800 GTX, un procesador NVIDIA GTX 280 y NVIDIA Tesla S1070. NVIDIA Tesla S1070 tiene 4 procesadores Tesla T10 GPU y una memoria total de 16 GB y puede alcanzar un máximo teórico de 4 rendimiento TFLOPS. Usaron una GPU de Tesla S1070 con 240 núcleos y 4 GB de memoria con una potencia de procesamiento de 1 TFLOPS. El 8800 GTX tiene 128 procesadores de flujo

Generamos 10 matrices densas al azar de números de precisión individuales para cada tamaño. El algoritmo SVD se ejecutó para cada matriz 10 veces. Para evitar una muestra particularmente buena o mala, promediaron las matrices aleatorias para cada tamaño. El promedio no varió mucho si se usaron 10 o más matrices. La Tabla 1 proporciona el tiempo medio general en segundos en GPU, MATLAB e Intel MKL. El cómputo SVD incluye el tiempo necesario para la bidiagonalización, la diagonalización y el cálculo de las matrices ortogonales. Logramos una aceleración de 3.04 a 8.2 sobre la implementación de Intel MKL y de 3.32 a 59.3 sobre la implementación de MATLAB para varias matrices cuadradas y rectangulares en GTX 280. La CPU aún supera a la implementación de GPU para matrices pequeñas. Para matrices cuadradas grandes, la aceleración aumenta con el tamaño de la matriz. La Figura 3 muestra el tiempo para calcular la SVD de matrices cuadradas y la Figura 4 muestra el tiempo para calcular la SVD de matrices rectangulares con una dimensión inicial m = 8K.

Conclusiones

En este documento, presentamos la implementación de la descomposición del valor singular completo en GPU de productos básicos. El algoritmo explota el paralelismo en la arquitectura de GPU y logra un alto rendimiento de computación en ellos.

La bidiagonalización de una matriz se realiza completamente en la GPU utilizando la biblioteca optimizada CUBLAS para obtener el máximo rendimiento. Usamos una implementación híbrida para la diagonalización de la matriz que divide los cálculos entre la CPU y la GPU, dando buenos resultados de rendimiento. Podríamos calcular la SVD de matrices muy grandes hasta la orden 14K que es imposible en la CPU debido a limitaciones de memoria. Las GPU están limitadas a números de precisión simple, aunque eso está cambiando con las nuevas generaciones. El error debido a la menor precisión fue menor al 0.001% en las matrices aleatorias con las que experimentamos. Nuestro enfoque de usar CUDA y las bibliotecas de software disponibles con él se puede utilizar para resolver muchos otros gráficos y tareas no gráficas.