Parallel One-Sided Block Jacobi SVD Algorithm: I. Analysis and Design

30 de mayo de 2018

Reporte

Rafael Larrazolo

La descomposición de valores singulares SVD es una de las descomposiciones o aplicaciones de álgebra lineal más ocupada por su diversidad de aplicaciones. En la literatura existen diversos algoritmos capaces de calcular dicha descomposición. El método de Jacobi es conocido por su habilidd de calcular los valores singulares y los vectores asociados con relativa precisión. Sin embargo, este método pertence a los algoritmos más lentos por lo que su uso va en decremento. Es por esto que se buscan alternativas para lograr, de cierto modo, acelerar su implementación y aprovechar su gran precisión tratando de hacerlo más veloz.

Los métodos de Jacobi *one-sided* para el cómputo de la descomposición SVD de una matriz rectangular son en general más eficientes que su contraparte *two-sided* principalmente debido por la disminución por la mitad del número de multiplicaciones necesarias para actualizar los vectores singulares de la izquierda y derecha. De igual modo, los métodos *one-sided* resultan ser al menos igual de precisos que los *two-sided*, y para el cálculo de una descomposición de Cholesky, resulta que estos métodos son muy precisos para la obtención de eigenvalores.

Este reporte trabaja con la generalización del algortimo de Jacobi *one-sided* a un caso por bloque. Se menciona que el trabajo con bloques conlleva a un mejor uso de memoria, además el cálculo por bloques permite un grado mayor de paralelismo. A grandes rasgos la idea del artículo consiste en describir en cómo acelerar el algoritmo serial de *One-Sided Block-Jacobi Algorithm*, lo cual consiste de manera general en preprocesar una matrix dada antes de su descomposición SVD calculando su descomposición QR con "pivoteo" de columnas; la segunda idea consiste en utilizar la descomposición CS (coseno-seno) de ciertos bloques de matrices ortogonales. Con ambas ideas se obtienen resultados eficientes en el uso de memoria cache y para reducir el número de barridos necesarios para la convergencia del algortimo dada una precisión definida.

One-Sided Block-Jacobi Algorithm (OSBJA) es apto para la descomposición SVD para una matrix compleja A de orden mxn, con $m \geq n$, donde

$$A = [A_1, A_2, \ldots, A_r]$$

.

OSBJA puede ser escrito de con el siguiente proceso iterativo:

$$A^{(0)} = A, \quad V^{(0)} = I_n,$$
$$A^{(k+1)} = A^{(k)}U^{(k)}, \quad V^{(k+1)} = V^{(k)}U^{(k)}, \quad k \geq 0.$$

Donde la matriz ortogonal $U^{(k)}$ es la conocida como *block-rotation* de la forma:

$$U^{(k)} = \begin{pmatrix} I & & & \\ & U_{ii}^{(k)} & & U_{ij}^{(k)} \\ & & I & \\ & U_{ji}^{(k)} & & U_{jj}^{(k)} \\ & & & & I \end{pmatrix},$$

Las estrategias más comunes con *row-cyclic* y *column-cyclic* donde los ordenamientos están dados por renglón y columna respectivamente en concordancia con la matriz triangular superior A. Las primeras $N(r)$ iteraciones constituyen en el primer *sweep* de OSBJA. Cuando el primer barrido es completado, los pares pivote (i, j) son repetidos para el segundo y así sucesivamente hasta la convergencia del algoritmo. Personalmente considero que el punto de convergencia es un tema relevante ya que buscamos que ésta se dé de manera no problematica y/o rápida. En el artículo mencionan que el cálculo de los productos punto de la primera parte son más rápidos que la multiplicación de matrices de la tercera, mientras que que la segunda parte que corresponde a la descomposición de los valores propios de la $\hat{A}^{(k)}$ puede ser realizada de manera rápida eligiendo un bloque lo suficientemente delgado para realizar los cálculos en memoria cache.

En el sentido estricto de la convergencia de OSBJA, si la matriz $A^T A$ converge a la matriz diagonal Σ^2 entonces $A^{(k)}$ se acerca al conjunto de matrices ortonormales cuyas columnas los vectores singulares por la izquirda de A. Por lo anterior la norma Euclideana de las columnas de $A^{(k)}$ convergen a los valores singulares de A. Del mismo modo la matriz $V^{(k)}$ es ortogonal por construcción y sus columnas se acercan a los vectores singulares por la derecha de A. En la práctica se deben realizar permutaciones apropiadas para recuperar los valores singulares. Este método siempre converge tanto de manera serial como con ciclos.

Como comentaba anteriormente, el punto importante del algoritmo es encontrar qué tan óptimo o eficaz es en cuestión de la velocidad de convergencia. En el artículo mencionan que el primer acercamiento es tratavio de la matriz original.

En el artículo se comenta que un preprocesamiento eficiente para los métodos de Jacobi, es realizar la factorización QR de A, seguido de la descomposción LQ del factor R; posteriormente el método Jacobi se aplica al factor L final. Esto disminuye de manera crucial el número de pasos de actualizaciones ortogonales de la matriz $V^{(k)}$ de los vectores singulares.

El segundo paso de preprocesamiento inicializa tres matrices que después son iteradas durante el proceso Jacobi. Esto hace que todas las columnas dentro de los bloques columna sean mutuamente ortogonales. Por tanto, al $i-$*ésimo* paso las columnas necesitarán ser ortogonalizadas sólo entre los dos bloques columnas y no dentro de ellos.

Un punto importante de este procedimiento es saber cuándo detener las iteraciones; si se detiene antes podría darse el caso que la convergencia no sea muy precisa, y si se detiene muy tarde se realizarían barridos innecesarios usando más tiempo de lo que realmente era necesario. En el paper precisan que para el método con un bloque es más difícil elegir un criterio de paro más confiable que para el método two-sided block.

Ahora bien, en términos del cómputo en paralelo me pareció interesante que cada etapa de la implementación haya sido pensada buscando la manera más óptima en lugar de hacer la implementación de manera más directa. Para ejemplificar lo anterior, por ejemplo se menciona que se hacen uso de la librería LAPACK y MPI, y que para mantener la comunicación entre p rocesadores, la manera natural de proceder es usar dos bloques columnas por procesador. Inicialmente se tiene una matriz A distribuida con dos bloques columna por procesador. En este punto se realiza la factorización QR de A con pivoteo de columnas seguido de manera opcional la factorización LR del factor R previo. Otra parte del preprocesamiento consiste en la descomposición espectral de p bloques diagonales de la matriz producto $\hat{A}^{(0)} = L^T L$. Esto último implica que cada procesador que almacena dos bloques columna i, j ejecuta de manera secuencial exactamente dos productos cruz y posteriormente dos descomposiciones espectrales de dos matrices simétricas definido positivas.

En términos generales, uno de los aspectos más relevantes del algoritmo comentado son el preprocesamiento de la matriz original usando "pivoteo" de columnas, trabajar con bloques de matrices en lugar de elementos

de las mismas , la inicialización y recursión de bloques matrices y la descomposicón CS.