Instituto Tecnológico Autónomo de México
Maestría en Ciencia de Datos
Mayo 2018

Reporte

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

Por

Diego Alejandro Estrada Rivera

165352

1 Introducción

La descomposición SVD (singular value decomposition) de una matriz m x n es una de las labores más demandadas en varias aplicaciones. El método unilateral Jacobi es reconocido por poder computar los valores singulares así como los vectores singulares izquierdos y derechos con gran exactitud, lo cual es importante en varias disciplinas como la física cuántica y la química. Este algoritmo había sido considerado como uno de los más lentos hasta recientemente, cuando varias ideas para la implementación de este algoritmo, como el procesamiento por bloques de la matriz en vez de por elementos, entre otras; han logrado acelerarlo de manera considerable.

Entre las principales ventajas del procesamiento sobre bloques de una matriz, se encuentran el mejor uso de la memoria disponible, como la cache; así como el permitir un nivel más elevado de paralelismo.

2 Algoritmo Unilateral por Bloques de Jacobi

El algoritmo unilateral por bloques de Jacobi empieza por dividir la matriz A en bloques-columnas de la siguiente forma:

$$A = [A_1, A_2, ..., A_r],$$

donde el ancho de A_i es n_i, p $\leq$ i $\leq$ r, para entonces $n_1 + n_2 + ... + n_r = n$. Se recomendada n_1 = n + 2 = .. = n_{r-1}, para entonces tener n = (r - 1) $n_0 + n_r, n_r \leq n_o$. Aquí n_0 puede **seleccionarse según sea el tamaño de la memoria cache disponible**, ya que esta es mucho más rápido de accesar que la memoria principal y como se verá más adelante en la implementación del algoritmo, esta ventaja es **clave** para el velocidad del mismo.

El algoritmo unilateral por bloques de Jacobi puede escribirse como un proceso iterativo de la siguiente manera:

$$A^{(0)} = A, V^{(0)} = I_n,$$

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

Aquí la matriz ortogonal de dimensión n x n, $U^{(k)}$ es el bloque de rotación:

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

Donde los bloques no identificados de la matriz son zeros. El propósito de multiplicar las matrices $A^{(k)}U^{(k)}$ es el de ortogonalizar mutuamente las columnas entre los bloques de columnas i y j de $A^{(k)}$. Los bloques matriz $U_{ii}^{(k)}$ y $U_{jj}^{(k)}$ son cuadrados de orden i y j, respectivamente, mientras que las matrices identidad I son de orden $\sum_{s=1}^{i-1} n_s$, $\sum_{s=1+1}^{j-1} n_s$ y $\sum_{s=j+i}^{r} n_s$, respectivamente. La matriz ortogonal $\hat{U}^{(k)}$:

$$\begin{bmatrix} U_{ii}^{(k)} & U_{ij}^{(k)} \\ U_{ji}^{(k)} & U_{jj}^{(k)} \end{bmatrix}$$

de orden $n_i + n_j$ es llamada **submatriz pivote** de $U^{(k)}$ en la ronda k. En cada ronda k del algoritmo, el par pivote, definido como (i, j), donde i = i(k) y j = j(k), es seleccionado de acuerdo a una estrategia pivote que puede identificarse como una función F : {0,1,...}→ P_r = {(l,m) : $1 \leq l < m \leq r$ }. Si O = {(l_1, m_1), (l_2, m_2), ..., ($l_{N(r)}$, $m_{N(r)}$) } es algun ordenamiento de P_r con N(r) = r(r -1)/2, entonces la estrategia ciclica es definida por:

Si k ≡ r -1 mod N(r) entonces (i(k), j(k)) = (l_s, m_s) para $1 \leq s \leq N(r)$.

Así las primeras N(r) iteraciones del algoritmo son la primer ronda del algoritmo unilateral Jacobi por bloques, cuando esta ronda termina, el par pivote (i, j) se repite para la segunda ronda y así para tantas rondas sean necesarias para que el algoritmo converja.

Podemos ver que en el proceso iterativo, solo la matriz $V^{(k)}$ es calculada iterativamente por actualizaciones ortogonales. Si el proceso termina en la iteración t, por ejemplo, entonces $A^{(t)}$ tiene columnas mutua y altamente ortogonales. Sus normas son los valores singulares de A, y las columnas normalizadas constituyen una matriz de valores singulares izquierdos. Considerando entonces los calculos necasarios en un paso serial del algoritmo unilateral Jacobi por bloques, podemos dividir el algoritmo en 3 partes, **muy importante de distinguir, ya que a cada una se le hará una modificación que nos permitirá acelerar la implementación del algoritmo**:

- Para el par pivote (i,j) dado, se computa la matriz $\hat{A}_{ij}^k$ de productos cruzados simétrica, positiva semidefinitiva. Esta parte consta en el cálculo de productos punto de tamaño m, lo cual es relativamente es rápido.
- Se diagonaliza $\hat{A}_{ij}^k$, es decir, se computa la descomposición en eigenvalores de $\hat{A}_{ij}^k$, que también será rápido si escogemos un bloque con un ancho apropiado para realizar los cálulos en la memoria cache.
- Finalmente, se requiere una actualización de los bloques columnas $A^{(k)}$ y $V^{(k)}$, que será la parte más costosa del algoritmo.

Con todo lo anterior considerado, un paso k de la ejecución estandar requerirá:

$$N_{flop}(k) \approx m(n_i n_j + n_i + n_j) + 8(n_i + n_j)^3 + 2m(n_i + n_j)^2 = 64n_0^3 + (9n_0^2 + 2n_0)n(sim = n = n_0 r)flops$$

2.1 Acelerando el algoritmo

Una forma de mejorar o "acelerar" algoritmo unilateral por bloques de Jacobi tiene la idea de reducir el número de rondas que se realizan para llegar a la convergencia deseada dada una presición. Este acercamiento regularmente requiere realizar un pre procesamiento a la matriz original A.

2.1.1 Factorización QR / LQ

Uno de estos pre procesamientos consiste en aplicar a una matriz A, de orden m x n donde m es mucho mayor a n, una factorización QR en donde el factor R es de dimension n x n, obteniendo una gran ganancia en cuanto a la dimension de la matriz con que se trabaja. Posteriormente se aplica una factorización LQ al factor R, para finalmente aplicar los cálculos de la SVD al factor L. En pocas palabras, teniendo una matriz A, se realiza el siguiente pre procesamiento:

$$A = Q_1 R P^T y R = L Q_2^T$$

aplicando finalmente el cálculo de la SVD al factor L. Es importante que ambas factorizaciones conserven el nivel de exactitud deseado para el proceso Jacobi, de lo contrario las factorizaciones se podrían ver como una especie de cuello de botella para la exactitud. Dado que el factor L es casi diagonal, el número de rondas en el algoritmo Jacobi necesarias para la convergencia dada una presición se reduce en gran medida.

La experiencia con implementaciones en serie demuestra que en general toma solamente $n^2/2$ rondas Jacobi. Otro punto de gran importancia en este pre precesamiento es que los vectores singulares derechos pueden calculse con suficiente exactitud a posteriori a partir del sistema lineal LV = UΣ, lo que significa que se puede descartar la actualización ortogonal de $V^{(k)}$ en el proceso iterativo, lo que nos ahorra muchas operaciones de punto flotante en la parte más costosa del algoritmo.

De esta forma podemos calcular los flops que le siguen al pre procesamiento. Tenemos mn^2 flops por las dos factorizaciones, $n^3/2$ flops por el calculo final de V y aproximadamente $mn^2 + n^3$ flops por las multiplicaciones de matrices requeridas para calcular los vectores singulares derechos e izquierdos de A. Todo junto nos suma $2mn^2 + 1.5n^3$ flops a las necesarias durante las iteraciones de Jacobi, pero haciendo esto logramos reducir en gran medida el número de iteraciones necesarias para la convergencia, así como deshacernos de la actualización ortogonal de $V^{(k)}$ durante el proceso iterativo.

2.1.2 Inicialización

A fin de inicializar los cálculos, aplicamos el siguiente algoritmo a cada bloque columna obtenido del factor L:
1.$for\ i = 1 : r$
2. $\hat{A}_{ii}^{(0)} = L_i^T L_i$;
3. $\hat{A}_{ii}^{(0)} = Q_{ii}^{(0)} \Gamma_i^{(0)} Q_{ii}^{(0)T}$; descomposición espectral
4. $A_i^{(0)} = L_i Q_{ii}^{(0)}$; no se realiza, solamente ilustra la conexión
5.end;

Por lo tanto, este algoritmo inicializa las siguientes tres matrices: $B^{(0)} = [B_1^{(0)}, B_2^{(0)}, ..., B_r^{(0)}] = [L_1, L_2, ..., L_r]$, $Q^{(0)} = diag(Q_{11}^{(0)}, Q_{22}^{(0)}, ..., Q_{rr}^{(0)})$, $\Gamma^{(0)} = diag(\Gamma_1^{(0)}, \Gamma_2^{(0)}, ..., \Gamma_r^{(0)})$. La idea es posteriormente actualizar recursivamente estas tres matrices, labor que se puede hacer de una forma muy eficiente; en vez de actualizar $A_i^{(k)}$ durante el procesos iterativo.

2.1.3 Fast scaled block-orthogonal transformation

Ahora necesitamos encontrar recursiones para el cálculo de $B^{(k)}, Q^{(0)}$ y $\Gamma^{(k)}$ en cada ronda k del proceso Jacobi. **La idea principal aquí es usar matrices pequeñas de orden** $n_i,\ n_j\ o\ n_i x n_j$ **para todas las actualizaciones, que son multiplicaciones de matrices, para que estas puedan llevarse a cabo en la memoria cache**.

Entonces, para la primera parte del algoritmo Jacobi necesitamos calcular la matriz de producto cruzado $\hat{A}_{ij}^{(k)}$ para el par pivote dado (i,j); usando la simetría de $\hat{A}_{ij}^{(k)}$, tenemos que calcular solamente $n_i n_j$ productos punto con columnas de $(B_i^{(k)}, B_j^{(k)})$ más dos multiplicaciones de matrices adicionales, por $Q_{ii}^{(k)T}$ por la derecha y $Q_{jj}^{(k)}$ por la izquierda. Todo junto requiere $n_i n_j (n + n_i + n_j)$ flops.

Enseguida, en la segunda parte, calculamos la descomposición en eigenvalores de $\hat{A}_{ij}^{k}$. Teniendo aquí la matriz de eigenvectores ortogonal $\hat{U}^{(k)}$, se propone calcular su descomposición coseno-seno

$$\hat{U}^{(k)} \equiv \hat{V}^{(k)} \hat{T}^{(k)} \hat{W}^{(k)T}$$

La tercera parte del proceso Jacobi es la multiplicacion de la matriz bloque-columna pivote $(A_i^{(k)}, A_j^{(k)})$ por $\hat{U}^{(k)}$ por la izquierda para obtener la nueva iteración $(A_i^{(k+1)}, A_j^{(k+1)})$. Haciendo uso de los realizado en pasos anteriores y de la descomposición coseno-seno de $\hat{U}^{(k)}$, la siguiente iteración puede factorizarse de tal manera que da lugar inmediatamente a las recursiones para las matrices B y Q:

$$(B_i^{(k+1)}, B_j^{(k+1)}) = (B_i^{(k)}(Q_{ii}^{(k)}V_{ii}^{(k)}), B_j^{(k)}(Q_{jj}^{(k)}V_{jj}^{(k)})\hat{T}^{(k)})$$

$$Q_{ii}^{(k+1)} = W_{ii}^{(k)T}, Q_{jj}^{(k+1)} = W_{jj}^{(k)T}$$

Se puede ver de manera inmediata que el número original de operaciones de punto flotante necesarios para actualizar en una ronda del proceso iterativo es reducido en gran medida usando el algoritmo modificado.

- Primero, se utilizaron únicamente dimensiones pequeñas n_i y n_j para obtener las matrices $(Q_{ii}^{(k)}V_{ii}^{(k)})$ y $(Q_{jj}^{(k)}V_{jj}^{(k)})$.
- Segundo, se actualizan B_i y B_j, que solamente requieren la multiplicación de matrices de la forma XY, donde X es de orden n x n_i, o n x n_j, y Y es cuadrada de orden n x n_j.
- La última actualización de B_i y B_j requiere se multiplique por la matriz $\hat{T}^{(k)}$ por la izquierda, que es equivalente a una simple rotación de columnas de tamaño n.

2.1.4 Criterio para detenerse

Como se menciono anteriormente, una de las consideraciones más importantes del algoritmo unilateral por bloques de Jacobi, es que se puede detener cuando ha convergido lo suficiente según una exactitud deseada. Deterlo muy pronto podría terminar en resultados inexactos, mientras que deterlo muy tarde resultaría en calculos innecsarios, lo cual anula uno de los principales objetivo de muchas de las ideas aquí mencionadas.

Para establecer un criterio confiable, definimos la siguiente notación:

$$\hat{A} = A^T A, D = (diag(\hat{A})^{1/2}, \hat{A}_S = D^{-1}\hat{A}D^{-1}$$

y para k ≥ 1,

$$\hat{A}^{(k)} = \hat{A}^{(k)T}A^{(k)} =$$

$$\hat{A}^{(k)} = \hat{A}^{(k)T}A^{(k)} = \begin{bmatrix} \tilde{A}_{11}^{(k)} & ... & \tilde{A}_{1r}^{(k)} \\ ... & & ... \\ \tilde{A}_{1r}^{(k)T} & ... & \tilde{A}_{rr}^{(k)} \end{bmatrix}$$

$$D_k = diag(\hat{A}^{(k)})^{1/2} = diag(||A^{(k)}e_1||, ..., ||A^{(k)}e_n) = (\Gamma^{(k)})^{1/2},$$

$$\hat{A}_S^{(k)} = D_k^{-1}\hat{A}^{(k)}D_k^{-1} \ matriz\ escalada$$

Para cualquier matriz simétrica X,

$$off(X) = \frac{\sqrt{2}}{2}||X - diag(X)||_F$$

es llamada la "la salida de la forma diagonal" y es igual a la norma Frobenius de su forma triangular superior (o inferior). Se introduce entonces dos medidas para seguir avanzando en el proceso Jacobi:

$$\alpha_k \equiv off(\hat{A}_S^{(k)}) = \frac{\sqrt{2}}{2}||\hat{A}_S^{(k)} - I||_F,$$

$$\omega_k \equiv off(\hat{A}_S^{(k)}) = \sqrt{\sum_{r=1}^{p}\sum_{t=r+1}^{p}||(\tilde{A}_{rt}^{(k)}||_F^2}$$

Cuando el proceso Jacobi ha convergido, todas las columnas de $A^{(k)}$ se han vuelto mas y mas ortogonales entre ellas, de manera que $\hat{A}_S^{(k)}$ tiende a I_n conforme k incrementa. Como α_k es la raíz cuadrada de la suma de las raíces de los cosenos de los angulos entre los pares de las columnas de$A^{(k)}$, cuando α_k se se ha acercado lo suficiente a 0, podemos detener nuestras iteraciones. Como el calculo de α_k involucra n(n-1)/2 productos punto y las columnas normalizadas de $A^{(k)}$ se pueden calcular con un error absoluto tan grande como $n\epsilon$, el criterio de convergencia tiene la forma:

$$\alpha_k \leq n^2\epsilon$$

Cuyo problema es que su actualización es muy costosa para ser observado en cada ronda. Mientras tanto, revisar la convergencia utilizando ω_k es mucho más económico en cada ronda, pero este tiende a perder todas sus cifras significativas y deja de ser confiable.

Sabiendo lo anterior, existen dos posibilidades. Se puede verificar que $\alpha_k \leq n^2\epsilon$ se mantengay detengamos el proceso, o el criterio no se cumple y necesitaremos estimar el número de rondas hasta la convergencia.

2.1.5 Paralelización

Entrando a la parte de paralelización del algoritmo, se hace mención de que se utilizaran MPI (Message Passing Interface) y la librería BLACS para comunicación, y la librería ScaLAPACK para el computo distribuido. El computo serial en cada procesador individual se realizará utilizando la libraría LAPACK.

Iniciando, se distribuye la matriz original en bloques de 2 columnas por procesador, computando entonces la factorización QR de A con pivote de columna (QRFCP), seguido de una factorización LQ (LQF) del factor R. En particular, el QRFCP y la LQF pueden ser implementadas por la rutina de ScalAPACK PDGEQPF y PDGELQF, respectivamente. La segunda etapa del proceso consiste en la inicialización que consiste en la descomposición espectral de p bloques diagonales de la matriz de producto cruzado $\hat{A}^{(0)} = L^T L$. Lo cual significa que cada procesador que almacena 2 bloques de columnas i y j calculara serialmente exactamente 2 $\hat{A}_{ll}^{(0)} = L_l^T L_l$, l = i,j productos cruzados, y después 2 descomposiciones espectrales de 2 matrices $\hat{A}_{ll}^{(0)}$ simétricas, positivo definitivas.

Habiendo diagonalizado los bloques diagonales de la matriz de producto cruzado $\hat{A}^{(0)}$, ensegudia se necesita escoger r pares pivote (i,j) que definan r subtareas, que pueden ser calculadas en paralelo. Esto es, asignar 1 par pivote por procesador, y mover (a lo mucho) dos bloques columnas con bloques con indices iguales a los del par pivote a cada procesador. Es por esto que es importante designar un ordenamiento de los bloques en paralelo adecuado.

3 Conclusiones

Tras haber leído el paper y resumido las ideas principales expuestas en él, puedo retomar varias ideas que siento fueron las más importantes. Estando el algoritmo unilateral de Jacobi casi obsoleto, se retomo con ideas modernas como lo son el computo de bloques en paralelo, dándole nueva utilidad. Una de las principales ideas implementadas aquí tambien fue el uso de la factorización QR y LQ para reducir la dimensionalidad de la matriz trabajada, desafortunadamente, siento que no se le dio la importancia debida a la consideración de que este paso, y consecuentemente muchas de las ideas del algoritmo, solo es provechoso en el caso donde el número de filas, m, es mucho mayor que el número de columnas, n, de la matriz original; esta puede ser una limitación grande para el beneficio obtenido del algoritmo. Otra idea que veo fue muy bien implementada fue el uso del cache y su rapido acceso para acelerar los calculos más abundantes a lo largo de la implementación, esto me recordo la importancia que tienen el conocimiento y la familiaridad con el sistema con el que se trabaja, para poder aprovechar al máximo sus cualidades. Por último, se expuso de manera general como se hace uso de varias librerías para realizar una implementación en paralelo, práctica que ya seguía muy naturalmente tras la división en bloques de la labor enfrentada. Para finalizar, puedo decir que más que una receta sobre como se implementa el algoritmo unilateral por bloques de Jacobi, se obtuvieron muy buenas nociones sobre buenas prácticas e ideas de como optimizar un algoritmo, y realmente adecuarlo de forma que se aprovechen al máximo las capacidades del equipo donde se implementa.