Comparación de un Algoritmo de Bidiagonalización para su Utilización en la Recuperación de Información

Osvaldo Spositto[1], Viviana Ledesma[1], Gastón Procopio[1], Hugo Ryckeboer[1], Victoria Saizar[1] y Alexis Vainberg[1]

[1] Departamento de Ingeniería e Investigaciones Tecnológicas,
Universidad Nacional de La Matanza, Buenos Aires, Argentina.
{spositto; vledesma, gprocopio, hugor, vsaizar, avainberg
}@unlam.edu.ar

Abstract. Este artículo presenta parte del trabajo realizado en el marco de una investigación que pretende optimizar un Sistema de Recuperación de Información, mediante la implementación y evaluación de distintos algoritmos secuenciales y paralelos para resolver eficientemente la Descomposición en Valores Singulares. Tal proceso comienza con llevar la matriz inicial a la forma bidiagonal, lo que puede consumir más del 70% del tiempo total del proceso. Por ello, como trabajo preliminar se han estudiado distintos métodos de bidiagonalización. Este trabajo se relaciona al desarrollo e implementación de un algoritmo de bidiagonalización alternativo para comparar posteriormente su comportamiento en distintas arquitecturas, en particular, las basadas en unidades de procesamiento gráfico, monoprocesadores y multiprocesadores. La experiencia de este estudio concreto ha permitido un análisis de rendimiento al ejecutar el algoritmo en cada implementación, cuando se varía el tamaño de las matrices, identificando problemas mínimos en GPU en cuanto a diferencias en la precisión de datos.

Keywords: Descomposición de Valores Singulares, Bidiagonalización, Sistema de Recuperación de Información.

1 Introducción

La Indexación Semántica Latente (ISL) es un método para la búsqueda de información en documentos a través de la indexación de términos [1], lo cual involucra la aplicación de algoritmos matemáticos especializados a fin de simular el análisis que realizaría una persona. Una técnica ampliamente utilizada a tal fin es la Descomposición en Valores Singulares (DVS), luego la recuperación se realiza a partir de los valores y vectores singulares obtenidos al aplicar dicha técnica [2].

Este trabajo se realiza en el contexto de un proyecto de investigación que tiene por objetivo optimizar la resolución de la DVS, para su posterior inserción en un Sistema de Recuperación de Información (SRI) desarrollado por el mismo equipo. En particular, la mejora se enfoca en implementar algoritmos que permitan resolver la primera fase del proceso de la DVS, conocido como bidiagonalización [3]. Se ha

comprobado que esta fase es la que más tiempo insume, estudios realizados muestran que puede consumir más del 70% o 90% del tiempo total para obtener todos los vectores singulares o solo los valores singulares, respectivamente [4]. Esto hace que el método de bidiagonalización, con un alto nivel de paralelismo de sus operaciones, sea un excelente candidato para la utilización de unidades de procesamiento gráfico (GPU, por sus siglas en inglés) ya que estas brindan la potencia de procesamiento requerida. Son varios los autores que han presentado trabajos en los que proponen la utilización de GPU para la bidiagonalización [5], [6].

En principio se han evaluado distintos métodos de bidiagonalización y, a modo inicial, se implementó un algoritmo genérico, secuencial, que sirvió para evidenciar el funcionamiento interno del proceso. Posteriormente, se ha orientado el estudio hacia algoritmos que puedan ser implementados en plataformas paralelas.

Luego de estudiar distintas variantes, se ha decidido adaptar y desarrollar un algoritmo alternativo propuesto por Barlow, Bosner y Drmač [7] (en adelante, Barlow), para evaluar su implementación en tres arquitecturas diferentes, basadas en CPU monoprocesador, multiprocesador y en GPU. Vale aclarar que se han encontrado trabajos en los que el algoritmo en estudio es probado en MatLab, en CPU secuencial y paralelo, pero nunca ha sido probado en GPU. Para la implementación se utilizó el framework de CUDA con el fin de comparar los tiempos de respuesta resultantes cuando se aplica el algoritmo en matrices de distintos tamaños.

El resto del artículo se organiza de la siguiente manera: la Sección 2 repasa los algoritmos de bidiagonalización en el contexto de los métodos utilizados para la recuperación de información; la Sección 3 describe los algoritmos implementados para este trabajo; la Sección 4 muestra los resultados experimentales obtenidos; y finalmente, la Sección 5 presenta las principales conclusiones e ideas para avanzar en esta investigación.

2 Métodos para la Recuperación de Información

Un SRI necesita componerse, por una parte, de un formalismo que permita representar documentos y consultas y, por otra parte, de una medida de similitud entre un documento y una consulta. En la actualidad conviven una variedad de modelos basados en distintos paradigmas para representar tanto documentos como consultas en SRI y comparar la semejanza de tales representaciones [8]. Entre estos, se destacan los modelos clásicos: el modelo booleano, el modelo vectorial y, el modelo probabilístico. En el modelo vectorial se seleccionan las palabras útiles, que por lo general son todos los términos del documento a excepción de las palabras semánticamente vacías, este proceso se enriquece utilizando técnicas de lematización y etiquetado [9]. El trabajo presentado en este artículo está circunscripto en una variante del método de recuperación vectorial, la ISL.

El método de ISL permite la búsqueda de información en documentos mediante la indexación de sus términos [1]. Involucra la definición de un espacio semántico donde los términos y los documentos altamente relacionados son colocados unos cerca de otros, reflejando los patrones de asociación entre los datos más importantes e

ignorando los menos importantes, es decir los que tienen menor influencia al momento de la recuperación.

La aplicación de la ISL, como se dijo antes, implica la utilización de algoritmos matemáticos especializados, que como resultado simulan el análisis que realizaría una persona. Por otro lado, con el método de ISL se pretende resolver dificultades durante la recuperación causadas por problemas de sinonimia y polisemia (o equivocidad del habla corriente). Por ejemplo, si la búsqueda se realiza a partir de la palabra "estación", la cual tiene múltiples significados (polisemia) una búsqueda literal de la palabra produciría muchos resultados posibles (estación de tren, estación del año, etc.). Si lo que se desea buscar es "estación del año", resultaría de interés que los resultados incluyan palabras distintas, pero con un significado igual o parecido, por ejemplo "temporada", "época" y así por el estilo (sinonimia). Por lo tanto, aplicando la ISL es posible buscar por conceptos o definiciones en contraste a lo que sería una búsqueda literal. Para ello, un primer recurso es trabajar con lexemas y no con palabras, ya que palabras derivadas de una misma raíz comparten buena parte de la carga semántica.

En general, al indexar los términos de los documentos, la matriz de documentos resultante se vuelve muy grande, por tal razón, a fin de acelerar el proceso de recuperación de información, suelen aplicarse técnicas de reducción de la dimensionalidad con el fin de transformar dicha matriz en una de menores dimensiones, pero capaz de reflejar las características de la matriz original al momento de llevar adelante las búsquedas. Para tal propósito se aplica la DVS, una técnica de factorización de matrices que permite descomponer una matriz en varias matrices que presentan las propiedades más significativas de la matriz original [1], [10], [11]. Así, una matriz A de tamaño t × d descompuesta con DVS produce tres matrices, tal como se puede observar en la figura 1.

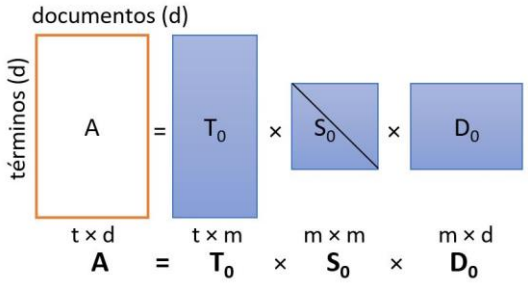

Fig. 1. Reducción de dimensiones en DVS. Fuente: [1].

Las columnas de T_0 y D_0 son ortonormales (ortogonales y de tamaño uno) y son las matrices izquierda y derecha respectivamente, de vectores singulares y, S_0 es una matriz diagonal compuesta de los valores singulares de A. El triple producto indicado da una matriz de t × d de rango m. De todas las matrices de t × d de rango m que aproximen a A, la de menor error, es decir distancia, es una que comparte los mayores m autovalores de A, obtenidos en una descomposición DVS y anula los restantes, comparte sus autovectores. Habiendo autovalores nulos sus correspondientes

componentes en los autovalores no tienen influencia y por lo tanto son recortados a tamaño m.

La ventaja de utilizar estos modelos de orden reducido es que simplifica la comprensión del sistema, reduce el coste computacional en los problemas de simulación, lo cual a su vez implica menor esfuerzo computacional en el diseño de controladores numéricamente más eficientes y se obtienen leyes de control más simples [12]. Esto justifica la importancia y necesidad de buscar modelos matemáticos simplificados que aproximen al máximo el comportamiento del sistema original. El modelo resultante, que tendrá un número menor de estados que el sistema original, se denomina modelo reducido o modelo de orden reducido, mientras que se conoce como reducción del modelo al procedimiento utilizado para conseguirlo.

Existen dos tipos principales de algoritmos que se aplican al cálculo computacional de la DVS de una matriz real, el método unilateral de Jacobi y aquellos que se basan en la bidiagonalización [10]. El número de operaciones para los distintos algoritmos se encuentra en el orden de $O(n^3)$, las diversas propuestas y mejoras que han surgido buscan disminuir operaciones costosas en tiempo. El trabajo de este equipo de investigación se enfoca en los algoritmos basados en bidiagonalización, los cuales aplican transformaciones ortogonales con el fin de obtener una forma bidiagonal para luego conseguir la DVS de la matriz bidiagonal.

2.1 Algoritmos Aplicados para la Bidiagonalización

Tal como se explicó anteriormente, la reducción bidiagonal de una matriz densa general se usa muy frecuentemente como un paso preliminar para el cálculo de la DVS [13]. A partir de la revisión en la literatura se descubrió que existen distintos métodos para la bidiagonalización de una matriz, los enfoques más tradicionales utilizan las transformaciones de Householder por la izquierda y por la derecha de la matriz [10], [14], [15]. Algunos estudios demuestran que dichos métodos presentan dos desventajas: cuando las matrices son de grandes dimensiones requieren tiempos de computación elevados y además repercuten negativamente en los costos de comunicación de una implementación paralela del algoritmo en sistemas de memoria distribuida [16], [17]. De hecho, según Ltaief [13], el número total de operaciones para dicho algoritmo sea $8/3(n^3)$, pudiendo ser n previsible de varios miles.

Pretendiendo dar una solución a tales problemas han surgido diversos trabajos, entre estos se encuentran la propuesta de Ralha [18], mejorada más adelante por Barlow [7], orientada a conseguir un método más sencillo de paralelizar que los métodos tradicionales. En esta propuesta la bidiagonalización es unilateral, es decir, las transformaciones de Householder son aplicadas solamente por el lado derecho de la matriz. Posteriormente, Da Silva Sanches de Campos [17] presenta una mejora al método de Barlow con el objetivo de reducir el número de comunicaciones necesarias para una implementación paralela destinada a sistemas de memoria distribuida.

Las operaciones utilizadas en el proceso hacen que el método de bidiagonalización sea altamente paralelizable [19]. De más está decir, que la correcta ejecución de algoritmos paralelos depende fuertemente de que los tamaños de las matrices se adapten a las capacidades de la máquina donde estos se ejecutan, por lo que, en

matrices de alta dimensionalidad, aparecen problemas como el espacio en la memoria, la correctitud del algoritmo y el incremento en los tiempos de ejecución.

Con lo anterior presente, se han realizado numerosos trabajos que incluyen estudios comparativos en cuanto al rendimiento al bidiagonalizar matrices de distintos tamaños cuando se utilizan distintas implementaciones variando la arquitectura. Entre estos, se han contrastado implementaciones secuenciales y paralelas sobre una arquitectura homogénea basada en CPU [17], se han experimentado algoritmos en mosaico con distinta cantidad de nodos multinúcleo de un sistema de memoria compartida distribuida en paralelo [4], [20]. Otros han buscado aprovechar la capacidad que ofrecen las GPU y experimentaron su uso aplicando algoritmos en arquitecturas tanto homogéneas [5], [3] como también heterogéneas en las que se combinan el uso de CPU con GPU [21].

En este trabajo se decide poner especial interés en uno de los algoritmos alternativos de bidiagonalización, el propuesto por Barlow en [7], dado que está pensado para soportar el paralelismo, lo cual está en consonancia con el objetivo de la investigación en curso, lograr una implementación a partir de una arquitectura basada en GPU.

3 Implementación de Algoritmos de Bidiagonalización

En principio se evaluaron distintos métodos para el cálculo de la bidiagonalización y, a modo inicial, se implementó un algoritmo genérico, secuencial, que sirvió para evidenciar el funcionamiento interno del proceso. Dicho algoritmo, basado en las transformaciones de Householder [4], expresado a continuación en la figura 2 como algoritmo, ha sido desarrollado en el lenguaje C#, parte de este trabajo ha sido presentado en [22]. La idea era que este, aunque secuencial, sirviera de base tanto para comprender el proceso en sí mismo, como también, para tomarlo como referencia en el diseño e implementación del algoritmo de Barlow.

Algoritmo 1: Reducción Bidiagonal vía Reflectores de Householder

```
1  for j = 1 to n do
2      x = A_{j:n,j}
3      u_j=sign(x_1) ||x||_2 e_1+ x
4      u_j= u_j/ ||u_j||_2
5      A_{j:n,j:n} =A_{j:n,j:n} - 2 u_j (u_j^* A_{j:n,j:n} )
6      if j < n then
7          x=A_{j,j+1:n}
8          v_j= sign(x_1) ||x||_2 e_1+ x
9          v_j= v_j / ||v_j||_2
10         A_{j:n,j+1:n} =A_{j:n,j+1:n} - 2 (A_{j:n,j+1:n} v_j) v_j^*
```

Fig. 2. Algoritmo de bidiagonalización basado en Householder. Fuente: [4].

Aunque existía la posibilidad de utilizar la biblioteca LAPACK[1], disponer del código en C# ofrece como ventaja, por una parte, permitir la comprensión de cada etapa interna del proceso, y por otra sienta las bases para que este código posteriormente pueda ser adaptado a diferentes algoritmos de bidiagonalización, e implementarlos en otras arquitecturas paralelas, en particular, aquellas basadas en GPU, a fin de analizar su eficiencia.

El algoritmo de Barlow, objeto de este estudio, y explicado en detalle en [7], consiste en un método para la bidiagonalización de matrices densas en el que las transformaciones de Householder se aplican únicamente por el lado derecho de la matriz. Con esto es posible definir todas las operaciones en términos de las columnas de la matriz a transformar, lo cual permite el desarrollo de implementaciones paralelas de un modo más simplificado en comparación con los métodos tradicionales, por otra parte, se logra reducir las comunicaciones que se necesitan.

El método de Barlow se puede así expresar, en forma de algoritmo, como se muestra en la figura 3, de la siguiente manera:

Algoritmo 2: BarlowBidiagonalización (A, α, β, Q)

1 **for** $r = 1, 2, ..., n-2$ **do**
2 $\quad \alpha_r = ||A(:,r)||_2$
3 $\quad q_r = \dfrac{A(:,r)}{\alpha_r}$
4 $\quad x_r = \mathrm{A}\ (:, r+1:n)^t\ q_r$
5 $\quad H_r$ tal que $H_r^t\ x_r = \beta_r\ e_1$
6 $\quad \mathrm{A}(:, \mathrm{r}+1:\mathrm{n}) = \mathrm{A}\ (:, \mathrm{r}+1:\mathrm{n})\ H_r$
7 $\quad \mathrm{A}(:, \mathrm{r}+1) = \mathrm{A}\ (:, \mathrm{r}+1) - \beta_r\ q_r$
8 **end**
9 $\alpha_{n-1} = ||A(:,n-1)||_2$
10 $q_{n-1} = \dfrac{A(:,n-1)}{\alpha_{n-1}}$
11 $\beta_{n-1} = q_{n-1}^t\ \mathrm{A}\ (:, \mathrm{n})$
12 $\mathrm{A}(:, \mathrm{n}) = \mathrm{A}\ (:, \mathrm{n}) - \beta_{n-1} q_{n-1}$
13 $\alpha_n = ||A(:,n)||_2$
14 $q_n = \dfrac{A(:,n)}{\alpha_n}$

Fig. 3. Algoritmo de bidiagonalización unilateral de Barlow. Fuente: [17].

Como se puede observar, hay un ciclo principal en el que se trabaja la matriz principal por columnas, va desde la columna *0* hasta la antepenúltima columna *(n-2)*. Además de la matriz inicial, hay 3 variables principales que se utilizan a lo largo de todo el algoritmo:

- α: es un vector de n elementos que contiene los valores de la diagonal principal.
- β: es un vector de n-1 elementos que contiene los valores de la diagonal superior.

[1] http://www.netlib.org/lapack/

- q: es una matriz con idénticas dimensiones que la matriz principal, es una matriz de trabajo interno.

En cada iteración se va completando: una posición en el vector α de elementos de la diagonal principal; una posición en el vector β de elementos de la diagonal superior, esto se representa en las líneas 5 y 6 del algoritmo en las que se aplican las reflexiones de Householder; una columna de la matriz q; y además, se modifica la matriz inicial que luego se lee en las iteraciones subsiguientes. Al terminar el ciclo se completan las posiciones restantes de α, β y las columnas que quedan de la matriz q.

Este algoritmo ha sido paralelizado y desarrollado para ser implementado sobre las tres arquitecturas mencionadas previamente, monoprocesador, multiprocesador y GPU, con el fin de comparar el rendimiento en cada una de estas. A continuación se resumen algunos resultados obtenidos.

4 Resultados Experimentales

Las implementaciones han sido desarrolladas utilizando el lenguaje C#, en conjunto con el framework CUDA, versión 6.5.

Las características del equipo utilizado para las pruebas de este experimento son:
- CPU: AMD Ryzen 5 2600 6 nucleos 12 threads a 3.6 GHz
- Memoria: 2 x 8GB DDR4 Crucial Ballistix 2400 Mhz
- GPU: NVIDIA GEFORCE GTX 1050 2GB

En la tabla 1 se presentan los tiempos de ejecución en milisegundos de cada una de las implementaciones realizadas para este estudio. Para las pruebas se utilizaron matrices cuadradas de distintas dimensiones, y tomando como fundamento las conclusiones obtenidas por Da Silva Sanches de Campos en [17], estas contienen valores aleatorios, dado que estos no tienen incidencia en los resultados esperados.

Tabla 1. Tiempos de ejecución del algoritmo en milisegundos para cada arquitectura

Dimensión Matriz	GPU	CPU Monoprocesador	CPU Multiprocesador
10x10	25	1	31
50x50	33	4	47
100x100	50	18	65
500x500	394	2005	1060
1000x1000	1691	16309	6008
2000x2000	10730	155418	34763

De los tiempos obtenidos, como resultado de las pruebas, se observa que cuando las matrices son de menor dimensión es conveniente ejecutar este tipo de algoritmos en CPU monoprocesador. Cuando la matriz comienza a superar las dimensiones, aproximadamente a partir de 200*200 o 300*300, la GPU mejora notoriamente el tiempo de ejecución con respecto a CPU monoprocesador y CPU multiprocesador. Esto se pone en evidencia, por ejemplo, observando los tiempos insumidos para la

matriz de tamaño 2000*2000, donde la GPU logró reducir los tiempos de ejecución en un 93% y 69%, con relación a CPU monoprocesador y multiprocesador respectivamente. En cuanto a la CPU multiprocesador, se puede observar que es constante el tiempo de resolución y este se incrementa lentamente hasta las dimensiones aproximadas entre 200*200 y 300*300.

Debe considerarse que en el proceso monoprocesador se realiza el proceso en serie, tomándose en un principio una columna, se aplica las operaciones matemáticas correspondientes y luego se modifica la matriz general en base al resultado de la columna resultante de dichas operaciones. En cambio, en el proceso multiprocesador las operaciones son distribuidas entre los núcleos disponibles, de esta manera cada thread resuelve una porción de las operaciones correspondientes acelerando de este modo la resolución del algoritmo.

Se ha recurrido a la presentación de un gráfico, que se muestra en la figura 4, en donde pueden apreciarse las evoluciones y las respectivas variaciones en las mediciones de los tiempos de ejecución en la medida que el tamaño de la matriz se incrementa. A medida que la dimensión de la matriz va aumentando, los tiempos entre CPU monoprocesador y GPU se asemejan. A modo resumido, se puede acotar que es ventajoso ejecutar este algoritmo en CPU monoprocesador para matrices de dimensiones inferiores a 300*300, en cambio, para aquellas de mayor dimensión será conveniente una arquitectura basada GPU la cual, como se puede visualizar, mejora notablemente los tiempos de respuesta.

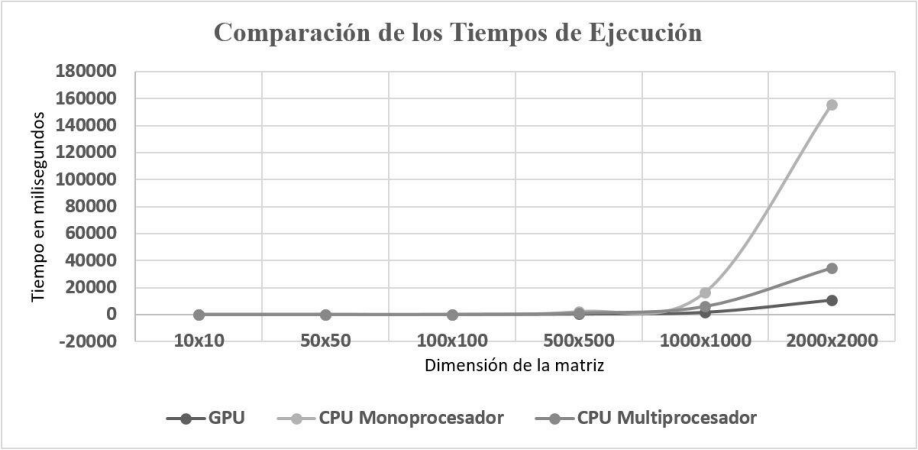

Fig. 4. Gráfico comparativo de los tiempos de ejecución del algoritmo de Barlow en las distintas arquitecturas aplicado a matrices de diferentes dimensiones.

Respecto a la precisión de los datos, al comenzar las mediciones a partir de las matrices de menor dimensión se observó, en algunos resultados, que cuando el algoritmo fue ejecutado en GPU existía una diferencia en el 13º decimal con respecto a las ejecuciones en CPU, tanto en monoprocesador como en multiprocesador, para las cuales los valores obtenidos fueron coincidentes. Sin embargo, a medida que la dimensión de la matriz crece, se detecta que esta diferencia comienza a incrementarse, por ejemplo, en matrices de dimensión 1000*1000 se encontraron diferencias en el 11º decimal. A partir de lo anterior, aunque las variaciones encontradas entre las

distintas implementaciones son ínfimas, teniendo en cuenta que las operaciones son las mismas, habría que investigar si los productos y las sumas en ambos tipos de procesadores son coincidentes, para de este modo evaluar la eficiencia del algoritmo en GPU.

5 Conclusiones

En el presente trabajo se presentó el desarrollo de un algoritmo alternativo que permite resolver el problema de la bidiagonalización de matrices densas y una comparación al implementarlo variando la arquitectura. El algoritmo fue probado adaptándolo a tres arquitecturas distintas: basada en GPU, CPU monoprocesador y multiprocesador. Se realizó un análisis de los tiempos resultantes para cada una de las implementaciones, observando la mejora en el rendimiento al paralelizar el algoritmo en una arquitectura basada en GPU para matrices de dimensiones mayores, cuando las matrices son pequeñas, en el orden de hasta 300*300, es preferible una arquitectura CPU monoprocesador. Es posible afirmar que las discrepancias en la precisión de los datos detectadas durante la ejecución sobre GPU son ínfimas, de todas maneras, sería necesario estudiar con mayor detalle los cálculos de cada procesador, incluso considerando matrices de mayor tamaño, para obtener conclusiones que ayuden a determinar si el comportamiento del algoritmo en dicha implementación se puede considerar exitosa.

En una siguiente etapa se pretende explorar si soluciones híbridas logran una aceleración en los cómputos, asignando en cada parte, CPU o GPU, aquellas tareas en las cuales mejor se desempeñan. Por otro lado, se espera poner a prueba la optimización conseguida en el SRI desarrollado por el equipo con el fin de comprobar el nivel de impacto alcanzado en la productividad del proceso.

Agradecimientos. Se agradece al Departamento de Ingeniería e Investigaciones Tecnológicas de la Universidad Nacional de La Matanza, el presente trabajo se financia en el marco del proyecto PROINCE C225.

Referencias

1. Deerwester, S., Dumais, S., Furnas, G., Landauer, T. & Harshman, R.: Indexing by latent semantic analysis. Journal of the American Society for Information Science. vol. 41(6), pp.391–407. 1990.
2. Mamani Roque, M.: Descomposición en Valores Singulares y Análisis Semántico Latente. Tesis de Maestría. Universidad Politécnica de Valencia, España, 2018.
3. Dong, T., Haidar, A., Tomov, S. & Dongarra, J.: Optimizing the SVD Bidiagonalization Process for a Batch of Small Matrices. Linear Algebra and Its Applications, ELSEVIER, vol. 108, pp. 1008-1018, 2017.
4. Ltaief, H., Luszczek, P., & Dongarra, J.: High performance bidiagonal reduction using tile algorithms on homogeneous multicore architectures. ACM Transactions on Mathematical Software, vol. 39(3), 2013.

5. Lahabar, S. & Narayanan, P.: Singular Value Decomposition on GPU using CUDA. IEEE International Symposium on Parallel & Distributed Processing, pp. 1-10, 2009.
6. Liu, F., Seinstra, F.: GPU-based parallel householder bidiagonalization. HPDC '10 Proceedings of the 19th ACM International Symposium on High Performance Distributed Computing, pp. 288-291, 2010.
7. Barlow, J., Bosner, N., Drmač, Z.: A new stable bidiagonal reduction algorithm. Linear Algebra and Its Applications, ELSEVIER, vol. 397, pp. 35-84, 2005.
8. Tolosa, G. & Bordignon, F.: Introducción a la Recuperación de Información: Conceptos, modelos y algoritmos básicos. Universidad Nacional de Luján, Argentina, 2008. Recuperado el 29/06/2020 de: http://eprints.rclis.org/12243/1/Introduccion-RI-v9f.pdf
9. Jaimes, L. & Riveros, F.: Modelos clásicos de recuperación de la información. Revista Integración. Escuela de Matemáticas. Universidad de Santander, vol. 23(1), pp. 17–26, 2005.
10. J. Demmel, M., Gu, S. Eisenstat, et al.: Computing the Singular Value Descomposition with Hight Relative Accuracy. Linear Algebra and its Application, vol. 299, pp. 21-80, 1999.
11. Berry, M., Dumais, S. & O'Brien, G.: Using Linear Algebra For Intelligent Information Retrieval. Society for Industrial and Applied Mathematics, Review, vol. 37(4), pp. 573-595. Philadelphia, USA, 1995.
12. Fortuna, L., Nunnari, G. & Gallo, A.: Model order reduction techniques with applications in electrical engineering. Springer-Verlag, 1992.
13. Ltaief, H., Kurzak, J. & Dongarra, J.: Parallel Two-Sided Matrix Reduction to Band Bidiagonal Form on Multicore Architectures. IEEE Transactions on Parallel and Distributed Systems, vol. 21(4), pp. 417 – 423, 2010.
14. Golub, G. & Reinsch, C.: Singular Value Decomposition and Least Squares Solutions, Handbook Series Linear Algebra, vol. 14, pp. 403-420, 1970.
15. Chan, T.: An Improved Algorithm for Computing the Singular Value Decomposition. ACM Transactions on Mathematical Software, vol. 8(1), pp. 72-83, 1982.
16. Sangwine, S. & Le Bihan, N.: Quaternion Singular Value Decomposition based on Bidiagonalization to a Real Matrix using Quaternion Householder Transformations. Applied Mathematics and Computation, ELSEVIER, 182(1): 727-738, 2006.
17. Da Silva Sanches de Campos, C.: Algoritmos de Altas Prestaciones para el Cálculo de la Descomposición en Valores Singulares y su Aplicación a la Reducción de Modelos de Sistemas Lineales de Control. Tesis Doctoral. Departamento de Sistemas Informáticos y Computación, Universidad Politécnica de Valencia, España, 2014.
18. Ralha, R.: One-sided reduction to bidiagonal form. Linear Algebra and Its Applications, ELSEVIER, 358(1-3): 219-238, 2003.
19. Guerrero López, D.: Algoritmos Paralelos para la Reducción de Sistemas Lineales de Control Estables. Tesis doctoral. Departamento de Sistemas Informáticos y Computación, Universidad Politécnica de Valencia, España, 2015.
20. Faverge M., Langou, J., Robert, Y. & Dongarra, J.: Bidiagonalization and R-Bidiagonalization: Parallel Tiled Algorithms, Critical Paths and Distributed-Memory Implementation. IEEE Transactions on Parallel and Distributed Processing Symposium, 668 - 677, 2017.
21. Hernández Cortés, J.: Implementación paralela y heterogénea de la transformación de Householder y sus aplicaciones. Tesis de Maestría. Departamento de Computación, Unidad Zacatenco, México, 2017.
22. Spositto, O., Ledesma, V. & Procopio, G.: Aplicación de la Descomposición de Valores Singulares a un Sistema de Recuperación de Información. Revista Digital del Departamento de Ingeniería (ReDDI). Universidad Nacional de La Matanza, vol. 4(2), 2019.