Asociación Argentina
de Mecánica Computacional

Mecánica Computacional Vol XXIX, págs. 3053-3061 (artículo completo)
Eduardo Dvorkin, Marcela Goldschmit, Mario Storti (Eds.)
Buenos Aires, Argentina, 15-18 Noviembre 2010

RESOLUCIÓN DE SISTEMAS TRIANGULARES EN TARJETAS GRÁFICAS (GPU)

Pablo Ezzatti [a], **Enrique S. Quintana-Ortí** [b] **y Alfredo Remón** [b]

[a] *Centro de Cálculo–Instituto de Computación, Universidad de la República, 11.300–Montevideo, Uruguay,* pezzatti@fing.edu.uy

[b] *Depto. de Ingeniería y Ciencia de Computadores, Universidad Jaume I, 12.071–Castellón, Spain,* {quintana,remon}@icc.uji.es

Palabras Clave: sistemas lineales triangulares, TRSM, GPUs

Resumen. Este trabajo presenta el uso del poder de cómputo ofrecido por las GPUs (tarjetas gráficas) para resolver sistemas lineales triangulares. En concreto se han implementado las variantes RUN y RLN de la especificación de la rutina TRSM definida en BLAS; no obstante las mismas técnicas son fácilmente aplicables al resto de variantes. Las distintas versiones de las variantes fueron desarrolladas utilizando la metodología formal propuesta en el proyecto FLAME. Dicha metodología permite derivar en forma directa las distintas variantes algorítmicas para la resolución de un problema expresado formalmente. Los experimentos realizados sobre una computadora con dos procesadores Intel Xeon QuadCore y una GPU NVIDIA Tesla C1060, demuestran el alto desempeño de las rutinas propuestas, mejorando muy notablemente el alcanzado por la implementación de NVIDIA sobre GPUs incluida en la biblioteca CUBLAS.

1. INTRODUCCIÓN

Las operaciones de álgebra lineal numérica son básicas en la resolución de numerosos problemas de ingeniería, como por ejemplo el tratamiento de señales, el control óptimo o la simulación de sistemas. Dada su importancia, numerosas investigaciones han tenido como objetivo principal el desarrollo de implementaciones de alto desempeño para las operaciones de álgebra lineal más comunes, propiciando la aparición de diferentes bibliotecas. Son destacables, entre otras, las bibliotecas BLAS (Lawson et al., 1979; Dongarra et al., 1988, 1990) y LAPACK (Anderson et al., 1999) para arquitecturas paralelas de memoria compartida, y ScaLAPACK (Blackford et al., 1997) y PaLAPACK (van de Geijn et al., 1996) para arquitecturas paralelas de memoria distribuida.

El uso de arquitecturas de cómputo de alto desempeño (supercomputadoras, clusters, etc.) aunque reporta importantes reducciones en el tiempo de cómputo, es en muchas ocasiones inviable por su alto costo. Esta situación ha motivado el estudio del uso de hardware secundario, y en particular de los procesadores gráficos (GPUs), que potencialmente ofrecen una importante capacidad de cómputo y tienen un costo asociado reducido. Desde el comienzo de la década el uso de las GPUs para la resolución de problemas de propósito general (GPGPU) ha ido en aumento, pero es desde los comienzos del año 2007, con el lanzamiento por parte de nVidia de CUDA (NVIDIA, 2008), que realmente tuvo un crecimiento exponencial. La arquitectura CUDA permite acceder a las GPUs como verdaderos multiprocesadores de memoria compartida, facilitando el uso de las mismas para la resolución de diversas problemáticas. CUDA incluye una implementación de la especificación BLAS sobre GPUs denominada CuBLAS. Adicionalmente, en el último tiempo varios trabajos han mostrado las bondades que ofrecen las GPUs accedidas mediante CUDA para acelerar problemas de álgebra lineal (Barrachina et al., 2008; Volkov y Demmel, 2008; Benner et al., 2009).

Una de las operaciones básicas más importantes del álgebra lineal es la resolución de sistemas lineales triangulares. Esta operación es básica para otras operaciones más complejas como, por ejemplo, la resolución de sistemas lineales, el cálculo de la factorización LU o el cálculo de los valores propios de una matriz.

En base a lo expresado anteriormente, en este trabajo se estudia la aceleración de la resolución de sistemas lineales triangulares utilizando GPUs. Para el desarrollo de las diferentes variantes, se ha utilizado la metodología para la obtención sistemática de algoritmos propuesta en el proyecto FLAME (http://www.cs.utexas.edu/flame/).

Diversos experimentos realizados sobre una computadora con dos procesadores Intel Xeon QuadCore y una GPU nVidia Tesla C1060, demuestran el alto desempeño de las rutinas propuestas, mejorando muy notablemente el alcanzado por la implementación de nVidia sobre GPUs incluida en la biblioteca CuBLAS.

El resto del documento se estructura de la siguiente manera. En la Sección 2 se presenta el estudio de los diferentes enfoques para desarrollar la rutina `TRSM` accediendo a las matrices por bloques. Posteriormente, en la Sección 3, se describen las distintas implementaciones sobre GPU desarrolladas. Los experimentos llevados a cabo para evaluar y validar las propuestas se presentan en la Sección 4. Por último, se resumen las conclusiones principales del trabajo y se detallan las líneas de trabajo futuro.

2. LA FUNCIÓN `TRSM`

BLAS establece la especificación de la función `TRSM` para resolver sistemas lineales triangulares. En el caso de trabajar con números reales en simple precisión la función se denom-

ina STRSM (DTRSM para reales en doble precisión, CTRSM para números complejos, ...), en (Lawson et al., 1979) se puede encontrar una descripción de la nomenclatura utilizada en BLAS para nombrar las funciones.

La función TRSM resuelve una de las siguientes ecuaciones $op(A) \times X = \alpha \times B$ ó $X \times op(A) = \alpha \times B$, donde α es un escalar, X y $B \in \mathbb{R}^{m \times n}$, y $A \in \mathbb{R}^{m \times m}$ es una matriz triangular. La matriz A puede asumirse con unos en la diagonal o no, puede ser triangular superior o inferior y el operador $op(A)$ puede ser A ó su traspuesta.

En lo que resta de la sección se detallan las distintas variantes de la función, dependiendo de los valores de los parámetros y se explican las distintas opciones para el de acceso a bloque a las matrices en la función TRSM.

2.1. Variantes de TRSM

Este trabajo se centra en matrices con números reales en simple precisión por lo cual de aquí en adelante se utilizaran como base de estudio las características particulares de la función STRSM, sin embargo muchos de los conceptos presentados en el trabajo son directamente aplicables a doble precisión (e incluso a números complejos o hermitianos).

En la Figura 1 se presenta el cabezal de la rutina STRSM de BLAS.

```
SUBROUTINE STRSM(SIDE,UPLO,TRANSA,DIAG,M,N,ALPHA,A,LDA,B,LDB)
```

Figura 1: Cabezal de la rutina STRSM de BLAS.

Donde el primer parámetro, SIDE, define si se resuelve el sistema $op(A) * X = alpha * B$ o bien $X * op(A) = alpha * B$; el parámetro UPLO epecifica si A es una matriz triangular superior o inferior; TRANSA define si $op(A)$ es A o bien su traspuesta; DIAG determina si A tiene unos en la diagonal principal; M y N son el número de filas y columnas de B respectivamente. Al finalizar, la matriz B es reemplazada por la matriz resultado, X.

La combinación de los tres primeros parámetros de la función implica ocho diferentes variantes de la rutina, dependiendo de los valores utilizados en ellos: LUN, LUT, LLN, LLT, RUN, RUT, RLN y RLT.

2.2. STRSM a bloques

En las arquitecturas actuales, los accesos a memoria representan las operaciones más costosas. El número de accesos a memoria y la forma en que éstos se realizan, son claves en la obtención de software de alto desempeño. Es bien conocida la estrategia de procesamiento por bloques para las operaciones matriciales, en particular este tipo de estrategias son la base de las bibliotecas BLAS y LAPACK. El acceso por bloques a los elementos de matriz posibilita una reutilización de los datos almacenados en la memoria Caché, lo que permite reducir el número de accesos a memoria principal y una óptima utilización de la jerarquía de memoria.

En la función STRSM, se pueden especificar hasta cuatro estrategias de acceso a los elementos por bloques. Por ejemplo, en el caso de la variante RLN de la rutina, que permite resolver el sistema $XA = B$ donde $X \in \mathbb{R}^{m \times n}$, $A \in \mathbb{R}^{n \times n}$ y $B \in \mathbb{R}^{m \times n}$, una forma equivalente de escribir

el sistema lineal anterior es la que se presenta en la Ecuación 1.

$$\left(\begin{array}{c|c} X_0 & X_1 \end{array} \right) \left(\begin{array}{c|c} A_{00} & \\ \hline A_{10} & A_{11} \end{array} \right) = \left(\begin{array}{c|c} B_0 & B_1 \end{array} \right) \quad (1)$$

En base a la nueva notación del sistema, es fácil deducir que se puede resolver el sistema $X_0 A_{00} = B_0$ (un sistema también triangular), luego con la solución de X_0 actualizar el sistema de la Ecuación 1 y posteriormente resolver el nuevo sistema triangular para encontrar el otro bloque de incógnitas. Notar que las ideas presentadas anteriormente se pueden extender para dividir la dimensión de la matriz en una cantidad k de partes, quedando el algoritmo expresado como la resolución de distintos subproblemas de tamaño $tam_{blk} = n/k$. En la Figura 2 se presenta el desarrollo completo para una versión de STRSM a bloques utilizando la notación de FLAME (fla, Accedido 2010).

La primera opción a bloques la denominamos Versión 1. Además, de la versión presentada anteriormente, se puede procesar el sistema en el orden inverso en el particionamiento de la matriz A (Versión 2). Adicionalmente, se pueden derivar dos versiones más que se basan en dividir la matriz B (y X) en dos submatrices $[B_0; B_1]$ y resolver primero el sistema $X_0 A = B_0$ y luego el sistema $X_1 A = B_1$ (Versión 3) o en el orden contrario dando lugar a la Versión 4.

Las derivaciones para la variante RUN son análogas, permitiendo generar también cuatro versiones (Versión 1, 2, 3 y 4).

Algorithm: $[C] := \text{xTRSM}_{\text{BLK}}(C, A)$

Partition $X \rightarrow \left(\begin{array}{c} C_T \\ \hline C_B \end{array} \right)$, $A \rightarrow \left(\begin{array}{c|c} A_{TL} & \\ \hline A_{BL} & A_{BR} \end{array} \right)$

where C_T has 0 rows, A_{TL} is 0×0

while $m(C_T) < m(C)$ **do**

Determine block size b

Repartition

$$\left(\begin{array}{c} C_T \\ \hline C_B \end{array} \right) \rightarrow \left(\begin{array}{c} C_0 \\ \hline C_1 \\ \hline C_2 \end{array} \right), \left(\begin{array}{c|c} A_{TL} & \\ \hline A_{BL} & A_{BR} \end{array} \right) \rightarrow \left(\begin{array}{c|c|c} A_{00} & & \\ \hline A_{10} & A_{11} & \\ \hline A_{20} & A_{21} & A_{22} \end{array} \right)$$

where C_1 has b_r rows; A_{11} is $b_r \times b_r$;
with $b_r := \min(b, m(C_B))$

$$\begin{array}{lcl} C_1 & := & A_{11}^{-1} \cdot C_1 \\ C_2 & := & C_2 - A_{21} \cdot C_1 \end{array}$$

Continue with

$$\left(\begin{array}{c} C_T \\ \hline C_B \end{array} \right) \leftarrow \left(\begin{array}{c} C_0 \\ \hline C_1 \\ \hline C_2 \end{array} \right), \left(\begin{array}{c|c} A_{TL} & \\ \hline A_{BL} & A_{BR} \end{array} \right) \leftarrow \left(\begin{array}{c|c|c} A_{00} & & \\ \hline A_{10} & A_{11} & \\ \hline A_{20} & A_{21} & A_{22} \end{array} \right)$$

endwhile

Figura 2: TRSM a bloques.

En Gunnels et al. (2001) y Bientinesi et al. (2005) se puede profundizar sobre la derivación de algoritmos de álgebra lineal a bloques y en particular las distintas versiones de la función STRSM.

3. IMPLEMENTACIÓN

En este trabajo se abordaron únicamente las variantes `RUN` y `RLN` de la rutina `STRSM`, ya que con estas dos variantes se puede resolver el sistema $X \times A = B$ utilizando la factorización LU de la matriz $A = L \times U$, objetivo original del trabajo. Sin embargo, el proceso para derivar las otras variantes de la rutina es análogo.

Se desarrollaron las cuatro versiones descritas para cada una de las dos variantes, adoptando la metodología de desarrollo propuesta en el proyecto FLAME. Además, siguiendo las recomendaciones del trabajo Barrachina et al. (2008) las rutinas desarrolladas emplean estrategias de padding, aumentando las dimensiones de las matrices A y B al múltiplo de 32 más cercano.

La metodología de uso de las GPUs para el procesamiento de las rutinas se describe a continuación.

1. Al comienzo de los algoritmos, se transfieren las matrices del espacio de memoria de la CPU al de la GPU.

2. La recursión implícita en los algoritmos se transforma en una iteración que es conducida por la CPU, mientras que las operaciones matriciales, al igual que la resolución de los bloques finales (paso base de la recursión), se realizan en GPU utilizando las rutinas proporcionadas por CUBLAS.

3. Por último, se devuelve el resultado al espacio de memoria de la CPU.

Todas las versiones utilizan el esqueleto básico descrito anteriormente, diferenciándose en la estrategia de división de las matrices a bloque y el orden en que las operaciones que se realizan (paso 2).

4. EXPERIMENTACIÓN

Para la evaluación experimental de la propuesta se generaron matrices en forma aleatoria. En un estudio inicial se estableció el mejor tamaño de bloque para cada versión y para cada tamaño de matriz empleado, para ello se realizarón ejecuiones con diferentes tamaños de bloque (16, 32, 64, 128, 192, 256, 320, 384, 448 y 512). Una vez establecido el mejor tamaño de bloque para cada versión, se comparan las mejores versiones para cada variante con la implementación de referencia de NVIDIA.

4.1. Plataforma de experimentación

La plataforma utilizada para llevar adelante los experimentos consiste en un computador con dos procesadores Intel Xeon QuadCore conectado a una tarjeta gráfica NVIDIA Tesla C1060, en la Tabla 1 se detallan las características de la plataforma de cómputo.

Tanto para las rutinas desarrolladas como para la implementación de referencia, se emplea la versión 2.1 de CUBLAS. En todos los casos el hardware fue utilizado en forma dedicada.

4.2. Experimentación

En primera instancia se estudió para cada versión desarrollada en GPU y para cada tamaño de matriz evaluando el tamaño de bloque óptimo. Luego, se compararon las distintas versiones para matrices con dimensiones que varían entre 1024 y 8192. En este sentido, en la Figura 3 se presenta el desempeño alcanzado (en GFlops) por cada implementación (únicamente se

Procesador	#cores	Frecuencia (GHz)	L2 caché (MB)	Memoria (GB)
Intel Xeon QuadCore E5405	8	2.3	12	8
NVIDIA TESLA C1060	240	1.3	–	4

Tabla 1: Hardware utilizado en los experimentos.

muestran los resultados para el tamaño de bloque óptimo) de las variantes RUN y RLN de STRSM evaluadas. Para el cálculo de los GFlops, únicamente se computa el tiempo de ejecución de las funciones en GPU, excluyendo el tiempo de transferencia.

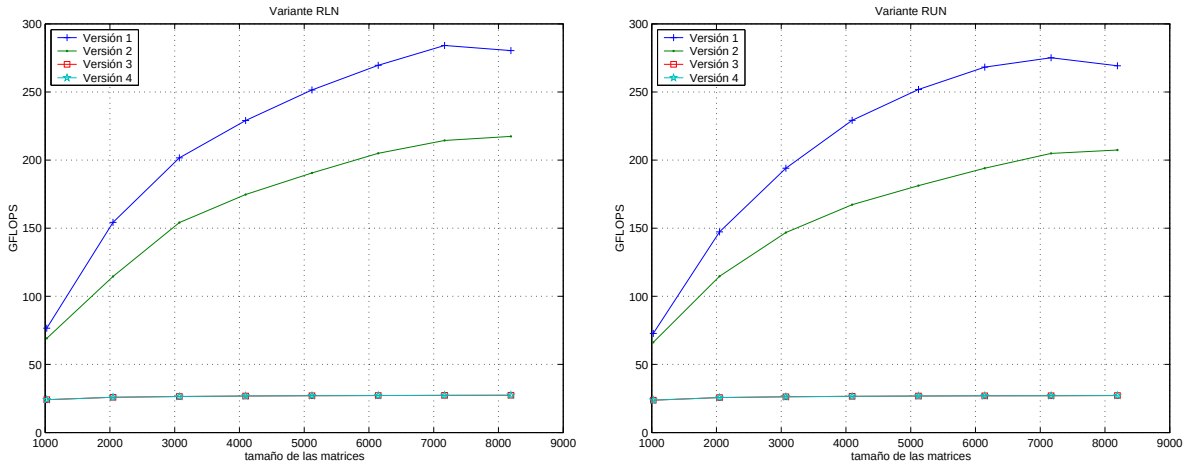

Figura 3: GFlops alcanzados por las distintas versiones implementadas de las variantes RLN y RUN de STRSM.

En base a los resultados obtenidos, se puede deducir que la Versión 1 de la variante RUN es notoriamente la que obtiene mejores resultados, siendo la misma situación para la variante RLN. Las versiones 3 y 4 para ambas variantes presentaron malos desempeños, una hipótesis para explicar esta situación es que la arquitectura de las GPUs privilegia el cómputo matricial regular (matrices completas), mientras que estas versiones justamente dividen la matriz con patrón regular de datos presente en el cómputo. Sin embargo, estas versiones parecen buenas opciones para efectuar paralelismo de "memoria distribuida" que en el contexto de GPUs se mapea a las técnicas de uso de múltiples GPUs.

Una vez identificados los tamaños de bloque óptimos y las mejores rutinas implementadas, se compararon los desempeños de las rutinas desarrolladas con el alcanzado por la implementación de referencia en el área, la ofrecida por NVIDIA para GPUs incluida en la biblioteca CUBLAS. Es así que en la Figura 4 se muestran los GFlops alcanzados por las dos variantes (RUN y RLN) de la propuesta y de $CUBLAS\_STRSM$ para diferentes tamaños de matrices. Para el cálculo de los GFLOPS alcanzados se incluyen los tiempos de transferencia de las matrices como tiempo de cómputo. Los experimentos se realizaron utilizando matrices con dimensiones múltiplo de 32, para evitar que el efecto del padding aumentara las diferencias en el desempeño entre las rutinas.

Al observar las gráficas se pueden apreciar los beneficios de utilizar las estrategias de acceso a bloque para acelerar operaciones de álgebra lineal en GPUs, ya que las rutinas propuestas

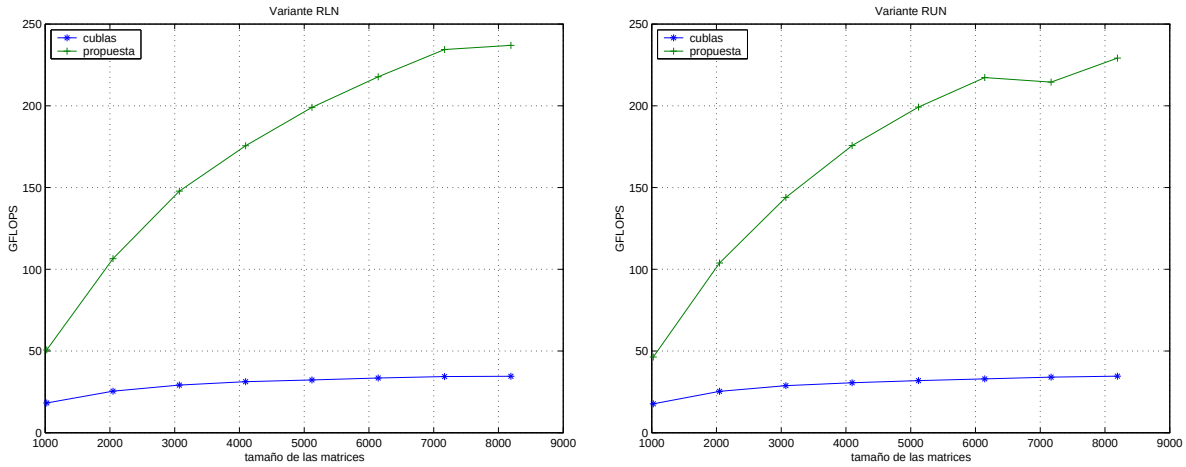

Figura 4: GFlops alcanzados por $CuBLAS\_STRSM$ y por las rutinas propuestas.

alcanzan desempeños claramente superiores a los conseguidos por la versión de CUBLAS, este hecho se resume en la Tabla 2, donde se muestran los coeficientes de aceleración de las versiones propuestas para los distintos tamaños de matrices evaluadas, al compararse con la implementación de NVIDIA. Los coeficientes de aceleración se presentan teniendo en cuenta y no los tiempos de transferencia de las matrices.

	aceleración c/transferencia		aceleración s/transferencia	
	RLN	RUN	RLN	RUN
1024	2.79	2.62	3.66	3.61
2048	4.19	4.11	5.57	5.41
3072	5.07	5.00	6.54	6.36
4096	5.21	5.74	7.03	7.13
5120	6.17	6.26	7.52	7.64
6144	6.51	6.60	7.84	7.93
7168	6.82	6.31	8.03	7.89
8192	6.86	6.62	7.94	7.60

Tabla 2: Factor de aceleración de la rutina propuesta.

5. CONCLUSIONES Y TRABAJO FUTURO

En este trabajo se estudió el uso de GPUs para acelerar la resolución de operaciones de álgebra lineal numérica, en particular se desarrollaron las variantes RUN y RLN de la rutina STRSM de BLAS utilizando el poder de cómputo de las GPU. La biblioteca CUBLAS de NVIDIA ofrece una implementación eficiente de dicha rutina. Sin embargo, en los experimentos realizados sobre una GPU Tesla C1060, la implementación de CUBLAS alcanzó los 40 GFlops (incluyendo los tiempos de transferencia), mientras que la versión propuesta logró superar los 240 GFLOPS, mostrando un factor de aceleración de hasta $6{,}86\times$. Mientras que en el caso de no incluir el tiempo de transferencia para el cálculo de los GFLOPS el factor asciende hasta $8\times$.

En base a los resultados obtenidos se puede destacar el beneficio del uso de GPUs para acelerar operaciones matriciales. Además, se puede apreciar la importancia en el empleo de

técnicas de acceso por bloques, que en la función `STRSM`, permite transformar el algoritmo para aumentar el uso de operaciones sobre matrices completas (multiplicación de matrices) y disminuir el uso de operaciones con acceso irregular a los datos.

Distintas líneas de trabajo se están abordando o se pretende abordar en el futuro cercano.

- En primer término resulta interesante desarrollar todas las variantes de `STRSM`.
- Parece necesario la automatización en la elección del tamaño de bloque óptimo de las rutinas.
- Otro aspecto no cubierto en el trabajo es el uso de números en doble precisión, únicamente se evaluaron números en simple precisión. En el hardware utilizado la degradación en el desempeño al pasar a trabajar en doble se encuentra en el entorno de $8\times$ a $10\times$. Por esta razón, es necesario evaluar el desempeño con números en doble precisión, estudiar estrategias de refinamiento iterativo y evaluar el comportamiento de los algoritmos en hardware de última generación, particularmente bajo la arquitectura Fermi que anuncia abatir las penalizaciones a un orden de $2\times$.
- Por último, es de interés extender el estudio a contexto donde la plataforma de ejecución disponga de múltiples GPUs, esto permitiría acelerar los cálculos y al mismo tiempo alcanzar una escalabilidad en las dimensiones de los problemas tratables.

AGRADECIMIENTOS

Los autores agradecen a Francisco Igual Peña por su colaboración en el uso de las herramientas de FLAME, así como sus valiosas suguerencias sobre los distintos algoritmos.

REFERENCIAS

FLAME website, Accedido 2010. `www.cs.utexas.edu/flame/` .

Anderson E., Bai Z., Bischof C., Demmel J., Dongarra J., DuCroz J., Greenbaum A., Hammarling S., McKenney A., y Sorensen D. *LAPACK Users' Guide*. SIAM, Philadelphia, PA, third edición, 1999.

Barrachina S., Castillo M., Igual F.D., Mayo R., y Quintana-Ortí E.S. Solving dense linear systems on graphics processors. In *Euro-Par '08: Proceedings of the 14th international Euro-Par conference on Parallel Processing*, páginas 739–748. Springer-Verlag, Berlin, Heidelberg, 2008. ISBN 978-3-540-85450-0. doi:http://dx.doi.org/10.1007/978-3-540-85451-7_79.

Benner P., Ezzatti P., Quintana E.S., y Remón A. Using hybrid cpu-gpu platforms to accelerate the computation of the matrix sign function. In *Lecture Notes in Computer Science, 7th Int. Workshop on Algorithms, Models and Tools for Parallel Computing on Heterogeneous Networks – HeteroPar'09*. 2009.

Bientinesi P., Gunnels J.A., Myers M.E., Quintana-Ortí E.S., y van de Geijn R.A. The science of deriving dense linear algebra algorithms. *ACM Trans. Math. Soft.*, 31(1):1–26, 2005.

Blackford L., Choi J., Cleary A., D'Azevedo E., Demmel J., Dhillon I., Dongarra J., Hammarling S., Henry G., Petitet A., Stanley K., Walker D., y Whaley R. *ScaLAPACK Users' Guide*. SIAM, Philadelphia, PA, 1997.

Dongarra J.J., Croz J.D., Hammarling S., y Duff I. A set of level 3 basic linear algebra subprograms. *ACM Trans. Math. Soft.*, 16(1):1–17, 1990.

Dongarra J.J., Croz J.D., Hammarling S., y Hanson R.J. An extended set of FORTRAN basic linear algebra subprograms. *ACM Trans. Math. Soft.*, 14(1):1–17, 1988.

Gunnels J.A., Gustavson F.G., Henry G.M., y van de Geijn R.A. FLAME: Formal linear algebra methods environment. *ACM Trans. Math. Soft.*, 27(4):422–455, 2001.

Lawson C.L., Hanson R.J., Kincaid D.R., y Krogh F.T. Basic linear algebra subprograms for Fortran usage. *ACM Trans. Math. Soft.*, 5(3):308–323, 1979.

NVIDIA. *NVIDIA CUDA Programming Guide 2.1*. 2008.

van de Geijn R., Alpatov P., Baker G., Chtchelkanova A., Eaton J., Edwards C., Guddati M., Gunnels J., Guyer S., Klimkowski K., Lin C., Morrow G., Nagel P., Overfelt J., y Pal M. Parallel linear algebra package (PLAPACK): Release r0.1 (beta) users' guide, 1996.

Volkov V. y Demmel J. LU, QR and Cholesky factorizations using vector capabilities of GPUs. Informe Técnico UCB/EECS-2008-49, EECS Department, University of California, Berkeley, 2008.