Solución de Múltiples Sistemas Lineales en GPUs

J. M. Molero[1], E. M. Garzón[2], I. García[3], E.S. Quintana-Ortí[4] y A. Plaza[5]

Resumen— **Este trabajo se centra en el cálculo, de forma concurrente, de múltiples sistemas lineales definidos por matrices densas de una dimensión media. Se considera una solución basada en la factorización de Cholesky y su implementación sobre plataformas GPUs. Este tipo de problema algebraico forma parte de distintos modelos de procesamiento de imagen, y además exhibe distintos niveles de paralelismo que pueden ser explotados muy eficientemente por las GPUs. En este trabajo, como aplicación del problema, nos centramos en la detección de anomalías sobre imágenes hiperespectrales, lo que supone una importante tarea en la explotación de este tipo de imágenes obtenidas en la observación de la Tierra. Concretamente, consideramos la versión local del ampliamente usado algoritmo RX (Reed-Xiaoli), denotado como LRX, en el que una de sus etapas más costosas consiste en la solución, para cada píxel de la imagen, de un sistema lineal cuyas dimensiones coinciden con el número de bandas de la imagen. En este contexto se describe y evalúa la solución de múltiples de sistemas lineales en GPUs, y se comprueba el factor de aceleración obtenido con la implementación propuesta en este trabajo.**

Palabras clave— **Factorización de Cholesky, Computación GPU, Detección de anomalías, Imágenes hiperespectrales , Algoritmo RX.**

I. Introducción

La factorización de Cholesky es un método algebraico ampliamente usado para la resolución numérica de sistemas lineales de ecuaciones cuya matriz es simétrica y definida positiva (SPD) [1, 2]. Varias implementaciones por bloques de este método están disponible en gran variedad de librerías algebraicas de uso tan extendido como LAPACK[1], MKL[2] y recientemente MAGMA[3] que incluye una implementación híbrida CPU-GPU de esta factorización. Cabe destacar que todas estas implementaciones están diseñadas para acelerar el cálculo de una única factorización aplicada a matrices de grandes dimensiones [3].

Por otra parte, existen aplicaciones que requieren un esquema de cálculo diferente, donde es necesario resolver muchos sistemas lineales independientes de dimensión pequeña o media. Las operaciones de álgebra lineal relacionadas con este esquema particular, se identifican como *batched* en el contexto de la computación en GPU [4]. Dos niveles de paralelismo pueden ser explotados en este tipo de soluciones: (1) el paralelismo de la solución estándar de cada sistema que está limitado por las dependencias de datos y el tamaño del problema; y (2) la concurrencia intrínseca a la solución de múltiples sistemas lineales. Existen algunas referencias que analizan la solución múltiple de sistemas en GPUs [5]. Cabe destacar que las plataformas GPUs permiten explotar eficazmente ambos tipos de paralelismo si las dimensiones de los sistemas no son excesivas. De esta forma, si las dimensiones de las matrices son pequeñas cada sistema podrá ser resuelto por un hilo de la GPU. En cambio, si la dimensión es media será necesario que los hilos de un bloque colaboren en la solución de cada sistema; y si la dimensión es grande entonces será necesario que todos los hilos colaboren en la solución de un único sistema y por tanto el segundo nivel de paralelismo mencionado no podrá ser explotado en la GPU.

En este trabajo, se analiza una versión CUDA de la solución de múltiples sistemas lineales de dimensión media basada en la factorización de Cholesky (en lo sucesivo, `CuBCholS`) para GPUs de NVIDIA [6, 7]. Con este propósito, como caso de estudio nos centramos en una aplicación de procesado de imágenes hiperespectrales. Este campo se caracteriza por trabajar con imágenes de teledetección compuestas por cientos de canales diferentes (correspondientes a diferentes longitudes de onda) para una área en la superficie de la Tierra. Estos datos suponen enormes posibilidades desde el punto de vista de la observación de la Tierra, pero su procesamiento presenta altos requerimientos computacionales. Para hacer frente a estos problemas con una alta carga computacional, se hace necesario el uso y el aprovechamiento eficiente de arquitecturas de alto rendimiento, tales como las actuales plataformas de procesamiento gráfico o GPUs.

En la próxima sección se analizan los aspectos más importantes de la implementación de `CuBCholS` en GPUs. A continuación, la Sección III se centra en el algoritmo RX local para la detección de anomalías en imágenes hiperespectrales, usado como un prototipo de aplicación de la librería que se propone en este trabajo. La Sección IV analiza y describe los principales detalles de la implementación y evaluación experimental con una tarjeta GPU NVIDIA GeForce GTX 680, resumiendo la evaluación del rendimiento alcanzado por `CuBCholS` en el contexto de la aplicación descrito en la sección previa. Para finalizar, la Sección V destaca las principales conclusiones y

[1]Dpto. de Informática y CEIA3, Univ. Almería, e-mail: `jmp384@ual.es`
[2]Dpto. de Informática y CEIA3, Univ. Almería, e-mail: `gmartin@ual.es`
[3]Dpto. de Arquitectura de Computadores, Univ. Málaga, e-mail: `igarciaf@uma.es`
[4]Dpto. de Ingeniería y Ciencia de los Computadores, Univ. Jaime I, e-mail: `quintana@icc.uji.es`
[5]Laboratorio de Computación Hiperespectral, Univ. Extremadura, e-mail: `aplaza@unex.es`
[1]http://www.netlib.org/lapack/
[2]http://software.intel.com/en-us/articles/intel-math-kernel-library-documentation
[3]http://icl.cs.utk.edu/magma/

trabajos futuros.

II. Solución de múltiples sistemas en GPUs basado en la factorización de Cholesky

Dada una matriz $A \in \mathbb{R}^{n \times n}$ simétrica y definida positiva, la factorización de Cholesky calcula la descomposición $A = LL^T$, donde $L \in \mathbb{R}^{n \times n}$ es triangular inferior, por tanto permite expresar la solución del sistema lineal $Ax = b$ en la solución de dos sistemas triangulares [1].

El siguiente algoritmo calcula la factorización de Cholesky [8]:

```
for (i=1; i≤n; i++) {
   for (sum=A[i][j], j=i; j≤n; j++) {
      for (k=i-1; k≥1; k- -)
         sum -= A[i][k]*A[j][k];
      if (i = j) {
         if (sum ≤ 0.0) error;
         p[i]=sqrt(sum);
      } else A[j][i]=sum/p[i];
   }
}
```

Una vez que la matriz ha sido factorizada, la matriz triangular L calculada se almacena en dos estructuras, la triangular de A y los elementos diagonales en el vector p. Con estos elementos, para completar la solución es necesario resolver sucesivamente dos sistemas triangulares. Se pueden resolver ambos sistemas por sustitución de la siguiente manera:

```
for (i=1; i≤n; i++) {
   for (sum=b[i],k=i-1; k≥1; k- -)
      sum-=A[i][k]*x[k];
   x[i]=sum/p[i];
}
for (i=n; i≥1; i- -) {
   for (sum=x[i],k=i+1; k≤n; k++)
      sum-=A[k][i]*x[k];
   x[i]=sum/p[i];
   }
```

En este trabajo, nuestro interés se centra en escenarios en los que es necesario resolver cientos o miles de sistemas lineales densos (SPD) de dimensión media (concretamente, $50 \leq n \leq 500$) utilizando la factorización de Cholesky. Por lo tanto, este tipo de problemas tiene una alta carga computacional, por lo que se hace necesaria la explotación de plataformas de alto rendimiento.

En este trabajo, se propone una implementación *batched* para resolver estos sistemas de forma simultánea en una GPU, por lo que se propone crear una nueva rutina denominada (`CuBCholS`). Por esto, se desarrolla este algoritmo que aprovecha los recursos de computación de la GPU para realizar el cálculo concurrentemente de cientos de sistemas lineales independientes.

La implementación de la rutina `CuBCholS` en GPU explota dos niveles de paralelismo: (1) internamente, cada sistema se puede descomponer en una serie de tareas con dependencias entre ellas, y (2) externamente, el cálculo de los sistemas lineales se puede realizar de forma independiente.

De acuerdo con este esquema, en nuestra implementación GPU de la resolución de sistemas, cada bloque CUDA calcula una factorización de Cholesky y resuelve el sistema correspondiente. De esta forma, cada bloque de CUDA tiene un tamaño igual al de las matrices de los sistemas, con lo que cada hilo realizará las operaciones correspondientes a un elemento de la columna de dichos sistemas, repitiendo el proceso para cada fila de los mismos con el objetivo de satisfacer las dependencias de datos existentes.

Cabe destacar que en este contexto, con el objetivo de optimizar la intensidad aritmética del algoritmo [5], se ha expresado la solución de los sistemas integrando la factorización de Cholesky con la solución del primer sistema triangular, de esta forma, se puede explotar de manera eficiente el uso de la memoria compartida de la GPU. Para terminar, se resuelve el segundo de los sistemas obteniendo como resultado el vector x con la solución.

A continuación, se analiza un caso de estudio en el campo del procesado de imágenes hiperespectrales ya que una de las fases que consume más recursos computacionales corresponde a la solución múltiple de sistemas de ecuaciones. Es en este contexto donde se motiva y evalúa nuestra rutina `CuBCholS`.

III. RX Local basado en `CuBCholS`

Una imagen hiperespectral es obtenida por un dispositivo de teledetección o sensor hiperespectral y sus píxeles se caracterizan por incluir información de un conjunto de bandas o longitudes de onda. A esta información se le denomina *la firma espectral de cada píxel*, y caracteriza de forma única al elemento asociado a ese píxel. Nuestro interés se centra en la detección de anomalías en este tipo de imágenes, lo que trata la identificación, sin conocimiento *a priori* sobre la imagen, de un conjunto reducido de píxeles cuya firma espectral es anómala cuando se compara con la de los píxeles que le rodean. Este tipo de procesado es importante, por ejemplo para la detección de incendios o puntos termales [9, 10]. La imágenes hiperespectrales están caracterizadas por las dimensiones espaciales en número de píxeles ($l \times c$ donde l representa el número de líneas de la imagen y c el número de píxeles que componen cada línea) y el número de bandas de la firma espectral (b).

El algoritmo RX, desarrollado por Reed y Xiaoli, es un detector de anomalías ampliamente usado que ha demostrado ser eficaz en imágenes hiperespectrales [11, 14]. El algoritmo RX compara la firma de cada píxel con la del resto de píxeles de la imagen mediante el cálculo de la distancia de Mahalanobis [12]. De esta forma, para cada píxel se asocia un valor, $\delta(\mathbf{x})$, que representa una valoración de la discrepancia entre la firma espectral del píxel y de las

firmas del resto de píxeles de la imagen.

En este trabajo consideramos la versión local del algoritmo RX denominada RX local o LRX, descrita en [13,15,16] que es especialmente adecuada para la localización de anomalías de pequeñas dimensiones o sub-píxel. El algoritmo LRX define una ventana de tamaño $\kappa \times \kappa$ centrada en cada píxel $\mathbf{x}$, y evalúa la distancia de Mahalanobis para el píxel central (píxel bajo estudio) con el resto de la ventana mediante la siguiente expresión:

$$\delta_{\kappa}^{LRX}(\mathbf{x}) = \mathbf{x}^T \mathbf{R}_{\kappa\times\kappa}(\mathbf{x})^{-1}\mathbf{x}, \qquad (1)$$

donde $\mathbf{R}$ es la matriz de correlación y $\mathbf{x}$ es la firma espectral del píxel (es decir un vector de dimensión igual al número de bandas de la imagen, b).

Teniendo en cuenta esta descripción, el algoritmo LRX se puede considerar como un proceso iterativo de tres etapas aplicado a cada píxel de la imagen ($\mathbf{x}$):

1. Cálculo de las matrices de correlación $\mathbf{R}_{\kappa\times\kappa}(\mathbf{x})$ [18].
2. Computación del vector intermedio $y(\mathbf{x}) = \mathbf{R}^{-1}\mathbf{x}$, con el objetivo de evitar el cálculo de la inversa de forma explícita. Esta etapa es expresada en términos de la solución de múltiples sistemas de ecuaciones lineales, ya que para cada píxel se debe resolver un sistema de dimensión $b \times b$. Las matrices de correlación $\mathbf{R}_{\kappa\times\kappa}(\mathbf{x})$ son simétricas y definidas positivas, por tanto la solución basada en la descomposición de Cholesky es la más ventajosa en términos de coste computacional [1,8,17].
3. Cálculo del filtro de salida $\delta(\mathbf{x}) = \mathbf{x}^T\mathbf{y}$ que asocia un valor positivo a cada píxel cuya magnitud es proporcional a la probabilidad de que el píxel forme parte de una anomalía en la imagen. Esta fase consiste en calcular un producto escalar de vectores de dimensión b para cada píxel.

Las etapas (1) y (2) requieren recursos computacionales muy elevados, ya que por cada píxel de la imagen se necesita reservar espacio en memoria para almacenar una matriz densa de dimensión b y computar cada matriz de correlación, más la solución del sistema correspondiente. Teniendo en cuenta las dimensiones típicas de las imágenes hiperespectrales que se procesan actualmente ($64 \lesssim c \lesssim 4096$), la arquitectura de las modernas GPUs ofrecen suficientes recursos para procesar concurrentemente el algoritmo LRX para los píxeles de una o varias líneas de la imagen. El kernel GPU de la primera etapa se estructura de forma que los hilos de cada bloque colaboran para calcular la matriz $\mathbf{R}_{\kappa\times\kappa}(\mathbf{x})$, de esta manera, se activan tantos bloques como píxeles se procesan concurrentemente. Una vez que se obtienen las correspondientes matrices de correlación de cada píxel, se ejecuta el kernel `CuBCholS` para resolver los múltiples sistemas con un esquema similar, es decir, cada bloque de hilos resuelve cada sistema, activando de nuevo el mismo número de bloques. Cabe destacar que en esta aplicación de la rutina $\mathbf{R}_{\kappa\times\kappa}(\mathbf{x})$, no es necesario incluir procesos de comunicación GPU-CPU que tanto penalizan el rendimiento de este tipo de plataformas, ya que las matrices se calculan en la GPU y se almacenan en su memoria global.

IV. Evaluación Experimental

La evaluación de `CuBCholS` se ha llevado a cabo en una plataforma equipada con un procesador multicore Intel Xeon E5640 (2.67 GHz) con 12 GB de memoria RAM. La GPU que se conecta al sistema es una tarjeta NVIDIA GeForce GTX 680 (con la arquitectura GK104 también conocida como *Kepler*) [19], con 8 multiprocesadores y 192 núcleos por multiprocesador (dispone de un total de 1536 cores), frecuencia de reloj 1.06 GHz, y 2 GB de memoria global.

Para evaluar el rendimiento de `CuBCholS`, hemos considerado una imagen real adquirida por el sensor AVIRIS, operado por el *Jet Propulsion Laboratory* de NASA. La imagen fue tomada el 16 de septiembre de 2001, justo 5 días después de los ataques terroristas sobre el *World Trade Center*. La imagen original está descrita en [20]. Consideramos unas dimensiones espaciales de 512×512 píxeles y 224 bandas, por tanto, de acuerdo con la notación que hemos introducido tenemos $l = 512$, $c = 512$ y $b = 224$. Además, a partir de esta imagen también hemos construido otros tests variando el número de bandas y el número de columnas con el objetivo de evaluar nuestra rutina `CuBCholS`.

La Tabla I muestra los resultados obtenidos al evaluar la implementación de nuestra rutina `CuBCholS` ejecutada en la GPU, frente a una implementación en secuencial de la misma función usando un core de la CPU. Para esto y como estudio preliminar, partiendo de la imagen descrita anteriormente, la rutina ha sido evaluada para el procesado de una línea de la imagen, no solo teniendo en cuenta todas las columnas y todas las bandas de la imagen, sino que también han sido consideradas versiones reducidas de las imágenes test con distintos valores de columnas (c, número de sistemas a resolver) y de bandas (b, dimensión de los sistemas).

Aunque los tiempos de ejecución son pequeños, los resultados de la Tabla I nos muestran que se ha reducido el tiempo de cómputo con un factor de aceleración que va desde 10 a 13,9. Téngase en cuenta que las imágenes reales están compuestas por cientos o miles de líneas (l), y por tanto el tiempo de procesamiento para la segunda etapa de LRX es de varios órdenes de magnitud superior al mostrado en la Tabla I. Por tanto, `CuBCholS` supone una mejora significativa para el algoritmo LRX ya que resuelve la etapa más costosa dentro del mismo, lo que demuestra los beneficios del uso de la rutina `CuBCholS` sobre una GPU, y justifica su desarrollo e implementación.

V. Conclusiones

Este trabajo describe y estudia una nueva implementación de solución de múltiples sistemas lineales de dimensiones medias, cuya solución se basa en la factorización de Cholesky. Esta implementación, denominada `CuBCholS`, es especialmente apropiada

TABLA I

Tiempo de cómputo (en milisegundos) para las etapas más importantes de LRX, en el procesado de una línea de la imagen hiperespectral real WTC, usando diferente número de bandas (b) y de columnas (c).

columnas	bandas	`CuBCholS`(GPU)	Solución Cholesky (CPU)	SpeedUp
512	128	36,5	508,3	13,9 ×
	160	66,1	685,2	10,3 ×
	192	121,9	1501,4	12,3 ×
	224	155,6	2030,2	13,1 ×
256	128	18,2	254,4	13,9 ×
	160	32,4	342,8	10,5 ×
	192	61,2	751,3	12,3 ×
	224	84,6	1015,2	12,1 ×
128	128	10,8	127,9	11,8 ×
	160	16,1	171,4	10,1 ×
	192	31,4	375,6	11,9 ×
	224	42,9	507,7	11,8 ×

para resolver cientos de sistemas lineales definidos positivos de tamaño medio. En nuestra implementación, asignamos cada una de las factorizaciones de Cholesky junto con su correspondiente resolución del sistema lineal, a un bloque CUDA de la GPU. `CuBCholS` es aplicada y evaluada en el contexto de la detección de anomalías sobre una imagen hiperespectral real, basada en el algoritmo LRX. Los resultados preliminares muestran la ventaja de usar esta rutina en términos de rendimiento, incluso con imágenes de reducidas dimensiones. En este caso, hemos desarrollado una implementación a medida, que cumple con los requisitos específicos que requiere el procesado de imágenes hiperespectrales, que ha sido el objeto de este trabajo. Como línea de trabajo futura, se pretende generalizar esta rutina para que sea válida para otras aplicaciones que muestren como requisitos la solución de múltiples sistemas independientes. En esta línea, se pretende flexibilizar la rutina para que se pueda adaptar a distintos tamaños de problemas, de acuerdo con la carga computacional de la aplicación y los recursos computacionales disponibles en la plataforma GPU especifica.

Acknowledgements

Este trabajo ha sido financiado por el Ministerio de Economía y Competitividad (TIN2008-01117, TIN2011-23283, TIN2012-37483-C03-01 and AYA2011-29334-C02-02), Junta de Andalucía (P10-TIC-6002 y P011-TIC7176), Junta de Extremadura (PRI09A110 y GR10035), CAPAP-H4 (TIN2011-15734-E) y por los fondos FEDER.

Referencias

[1] G. Golub and C. V. Loan, *Matrix Computations Third Edition*, The Johns Hopkins University Press, 1996.

[2] A. Quarteroni, R. Sacco and F. Saleri, *Numerical Mathematics*. Springer, ISBN 0-387-98959-5, 2000.

[3] *Matrix Algebra on GPU and Multicore Architectures*, http://icl.cs.utk.edu/magma

[4] NVIDIA Cublas manual, *http://docs.nvidia.com/cuda/cublas/index.html*

[5] M. J. Anderson, D. Sheffield and K. Keutzer, *A Predictive Model for Solving Small Linear Algebra Problems in GPU Registers*, IEEE 26th International Parallel and Distributed Processing Symposium (IPDPS). (2012) p 2–13.

[6] NVIDIA CUDA C Programming Guide, *http://developer.nvidia.com/category/cuda-zone*, October, 2012.

[7] R. Farber, *CUDA, Application Design and Development*, Morgan Kaufman, 2010.

[8] W. H. Press, S. A. Teukolsky, W. T. Vetterling, and B. P. Flannery, *Numerical recipes in C (2nd ed.): the art of scientific computing*, Cambridge University Press, 1992.

[9] G. Shaw and D. Manolakis, "Signal processing for hyperspectral image exploitation," *IEEE Signal Processing Magazine*, vol. 19, pp. 12–16, 2002.

[10] A. Plaza and C.-I. Chang, *High performance computing in remote sensing*, Boca Raton: CRC Press, 2007.

[11] C.-I. Chang, *Hyperspectral Imaging: Techniques for Spectral Detection and Classification*, Norwell, MA: Kluwer. (2003).

[12] J. Richards and X. Jia, *Remote Sensing Digital Image Analysis: An Introduction.* Springer, 2006.

[13] J. M. Molero, E. M. Garzón, I. García and A. Plaza, *Analysis and Optimizations of Global and Local Versions of the RX Algorithm for Anomaly Detection in Hyperspectral Data*, IEEE JSTARS. vol. 6, no. 2, pp. 801-8014, April 2013.

[14] I. S. Reed and X. Yu, *Adaptive multiple-band cfar detection of an optical pattern with unknown spectral distribution.*, IEEE Trans. Acoustics Speech and Signal Processing. 138 (1990) 1760–1770.

[15] Y. P. Taitano, B. A. Geier, and K. W. B. Jr, *A locally adaptable iterative RX detector*, EURASIP Journal on Advances in Signal Processing. **10** (2010).

[16] S. Matteoli, M. Diani, and G. Corsini, *A tutorial overview of anomaly detection in hyperspectral images*, IEEE Aerospace and Electronic Systems Magazine. 25 (2010) 5–28.

[17] J. J. Dongarra, I. S. Duff, D. C. Sorensen, and H. V. D. Vorst, *Solving Linear Systems on Vector and Shared Memory Computers.* Society for Industrial and Applied Mathematics Philadelphia, 1990.

[18] J. M. Molero, E. M. Garzón, I. García, E. S. Quintana and A. Plaza, *Accelerating the KRX Algorithm for Anomaly Detection in Hyperspectral Data on GPUs*, Proc. of the CMMSE 2012.

[19] NVIDIA GeForce GTX 680 Whitepaper, *http://www.geforce.com/Active/en_US/en_US/pdf/GeForce-GTX-680-Whitepaper-FINAL.pdf* (2012).

[20] R. Green, *et al.*, "Imaging spectroscopy and the airborne visible/infrared imaging spectrometer (AVIRIS)," *Remote Sensing of Environment*, vol. 65, no. 3, pp. 227–248, 1998.