Acelerando algoritmos de deconvolución para imágenes de microscopía usando GPUs

José M. Cecilia[1], Francisco J. Moreno[1], José M. García[1]

[1]Grupo de Arquitectura y Computación Paralela
Dpto. Ingeniería y Tecnología de Computadores
Universidad de Murcia
Campus de Espinardo, 30100 Murcia, Spain
Email: {chema, fran.moreno, jmgarcia}@ditec.um.es

Resumen

Los algoritmos de deconvolución son típicamente usados para eliminar distorsiones en imágenes de microscopía. Muchos algoritmos han sido propuestos, pero los que producen un mejor resultado son iterativos. Estos algoritmos son intensivos en cómputo y altamente paralelizables en arquitecturas de memoria compartida. En este artículo, analizamos el proceso de deconvolución y los factores influyentes en el mismo, comprobamos la viabilidad de tres algoritmos de deconvolución aplicados sobre imágenes tomadas con microscopios de fluorescencia y confocal, y finalmente, analizamos el rendimiento de estos algoritmos sobre una arquitectura masivamente paralela como es la GPU.

1 Introducción

En microscopía siempre ha existido la necesidad de eliminar las distorsiones ópticas producidas en las imágenes obtenidas por microscopios. El método usado para eliminar dichas distorsiones se denomina proceso de deconvolución de imágenes. La deconvolución es un proceso que se aplica a señales para eliminar una distorsión inherente a la obtención de dichas señales. Esta distorsión normalmente se puede calcular por el proceso de convolución (proceso inverso al de deconvolución). Este proceso es usado comúnmente en sismología [14], astronomía [13], etc.

Si la distorsión se produce de forma lineal e invariante, la señal generada por el microscopio es el resultado de la convolución de la señal real y la función que define la deformación (*Point-Spread Function* PSF[1]).

La ecuación 1 expresa el proceso de convolución

$$o = i \otimes h \tag{1}$$

donde i es la señal real, $\otimes$ es la operación de convolución y h es el PSF.

[1]Point Spread Function, es la función que define la imagen de un objeto luminoso puntual tal y como se registraría en el dispositivo, se puede conseguir mediante procedimientos de calibrado o calcularse teóricamente.

La inversa de dicha operación, para obtener de nuevo la señal real i se conoce como deconvolución [10].

Por el teorema de la convolución[2], la ecuación 1 se puede reescribir como la ecuación 2

$$O = I \cdot H \tag{2}$$

donde O, I, y H son las transformadas de fourier de o, i y h respectivamente, y el operador $\cdot$ representa la operación de producto punto de números complejos. Por tanto el proceso de deconvolución queda descrito en la ecuación 3:

$$i = \mathcal{F}^{-1}(O/H) \tag{3}$$

donde $\mathcal{F}^{-1}$ es la Transformada de Fourier inversa [6].

Existen numerosos algoritmos de deconvolución, que podemos clasificar como iterativos y no iterativos. Entre los métodos no iterativos destacamos el filtro de *Wiener*, que es un método de una pasada que reduce el impacto de los coeficientes pequeños en la Transformada de Fourier. Los algoritmos de *Jansson-van Cittert* y *Maximum Likelihood Estimation* se califican dentro de los métodos iterativos para minimizar la diferencia entre la imagen distorsionada original o, y la convolución de la imagen verdadera i con el PSF.

Los algoritmos iterativos normalmente requieren entre 10 y 20 iteraciones para producir una solución precisa realizando el cómputo en todos los píxeles de la imagen. En un procesador de propósito general, este cálculo puede ser muy costoso computacionalmente. Sin embargo, estos algoritmos son muy adecuados para ser paralelizados en plataformas de memoria

[2]En matemática, el teorema de convolución establece que bajo determinadas circunstancias, la Transformada de Fourier de una convolución es el producto punto de las transformadas. En otras palabras, la convolución en un dominio (por ejemplo, el dominio temporal) es equivalente al producto punto (o interno) en el otro dominio (es decir, el dominio espectral).

compartida, y especialmente en las unidades de procesamiento gráfico.

Las unidades de procesamiento gráfico (*Graphics Processing Units* o GPUs) han estado en los últimos años en el punto de mira del paralelismo a nivel de chip. Las nuevas generaciones de GPUs, como la Tesla C1060 de Nvidia, incorporan hasta 240 procesadores escalares, rozando el Tera FLOP de rendimiento teórico.

Debido a este gran poder computacional, las GPUs han alcanzado un gran interés para la comunidad científica, que ha encontrado un chip masivamente paralelo a un precio mucho más competitivo que los grandes *clusters* de computación. Sin embargo, la programación en estas tarjetas gráficas estaba inicialmente restringida al uso del API gráfico (OpenGL[16] o DirectX[2]), siendo este muy inflexible y tedioso para desarrollar códigos científicos.

Con la aparición de CUDA (*Compute Unified Device Architecture*) [8] de Nvidia, se consiguió un entorno mucho más accesible para la programación de propósito general sobre la GPU [5]. Esta tendencia ha sido seguida por la mayoría de compañías fabricantes de tarjetas gráficas, como ATI/AMD [17] que ha desarrollado su tecnología *Stream Computing*, o el nuevo chip gráfico de Intel, *Larrabe*, que proporciona desarrollo flexible de aplicaciones de propósito general.

Finalmente, la aparición de OpenCL (*Open Computing language*) [18] como estándar para el desarrollo de aplicaciones en chips de múltiples cores, consolida a la GPU como un chip masivamente paralelo de altas prestaciones para aplicaciones de propósito general. De este modo, se pretende establecer unos procesos estandarizados en el desarrollo de aplicaciones paralelas, independientemente de la máquina que lo ejecute en última instancia.

En este artículo, analizamos el rendimiento de tres algoritmos de deconvolución incluidos en la librería de deconvolución de imágenes `Clarity` descrita en [10]. Además, adaptamos esta librería para trabajar con imágenes biomédicas reales tomadas con un microscopio de fluorescencia en el Hospital Universitario Virgen de la Arrixaca (HUVA) de Murcia, y tomadas con un microscopio confocal del departamento de análisis de imágenes de la Universidad de Murcia. Describimos brevemente la arquitectura de la GPU usada para nuestras pruebas, así como el modelo de programación CUDA de Nvidia. Evaluamos la calidad de las imágenes producidas tras aplicar el proceso de deconvolución. Finalmente, mostramos que la GPU es competitiva para estos algoritmos y plasmamos el trabajo a desarrollar en el futuro

2 Antecedentes

Varios paquetes de software comercial incluyen algoritmos paralelos de deconvolución de imágenes para microscopios, entre ellos AutoQuant de MediaCybernetics (Bethesda,EEUU), Huygens de *Scientific Volume Imaging* (Hilversum, Holanda), DeconvLIVE de *Intelligent Imaging Innovations* (Denver, EEUU), y Nis-Elemts de Nikon (N.Y, EEUU). En estas aplicaciones comerciales no hemos encontrado ningún estudio de rendimiento. En cambio, [10] muestra un análisis completo de rendimiento, y plantea la tarjeta gráfica como alternativa de altas prestaciones para la paralelización de estos algoritmos aunque por contra, no realiza un estudio de la calidad de las imágenes obtenidas por sus algoritmos.

3 Algoritmos de reconstrucción de imágenes mediante deconvolución

Hay varias aproximaciones que hacen uso de una inversión lineal de la operación de convolución explicada anteriormente [10]. En esta sección explicamos tres aproximaciones: el filtro de *Wiener*, el método de *Jansen-van Cittert*, y la estimación *Maximum Likelihood Estimation*.

3.1 Filtro de *Wiener*

El filtro de *Wiener* es un filtro lineal que minimiza de forma óptima el error cuadrático medio (mean-squared error, MSE) entre la imagen filtrada y la obtenida del microscopio. Se asume la siguiente ecuación:

$$o = i \otimes h + n \tag{4}$$

donde n representa un ruido que cumple una distribución Gaussiana de probabilidad sumado al resultado de la convolución. Se puede construir un filtro $\hat{h}$ que aplicado a o obtenga una estimación de la señal de entrada $\hat{i}$, como se muestra en la fórmula 5.

$$\hat{i} = o \otimes \hat{h} \tag{5}$$

En el dominio de la frecuencia se puede escribir como sigue:

$$\hat{H} = \frac{H^*}{|H|^2 + (P_n/P_i)} \tag{6}$$

donde * es el operador de complejo conjugado, y P_n y P_i son las estimaciones del espectro de potencia de n y de i respectivamente. Aunque en la práctica el término P_n/P_i se sustituye por una constante en el rango $[0.001, 0.1]$.

El filtro de *Wiener* es rápido de calcular pero tiene muchas limitaciones en la práctica. Es el estimador óptimo para imágenes estacionarias con

ruido que sigue una distribución de probabilidad Gaussiana [10], lo que hace que no sea apropiado para microscopios de fluorescencia. Teóricamente, el ruido producido por un microscopio de fluorescencias sigue una distribución de Poisson. Además, las imágenes obtenidas de microscopios de fluorescencia no son estacionarias, lo que hace que se incumpla una de las condiciones que hacen que el filtro de *Wiener* sea óptimo. Por último, el filtro de *Wiener* no es capaz de restaurar señales en frecuencias mayores que la del PSF. De todos modos, este filtro es muy rápido y se puede usar para obtener una inicialización con menos ruido para los demás algoritmos [10].

3.2 Filtro de *Jansen-van Cittert*

Este algoritmo corresponde a la familia de técnicas iterativas de deconvolución. Los métodos iterativos mejoran la estimación de la señal requerida mediante la suma de diferencias de pesos entre la imagen original y la convolución de la estimación con el PSF.

Los valores estimados de la señal inicial i no pueden ser negativos porque representan la cantidad de fotones contados por un sensor.

El proceso de reconstrucción de *Jansen-van Cittert* consiste en repetir los siguientes pasos:

1. $o^k = i^k \otimes h$
2. $i^{k+1} = i^k + \gamma[o - o^k]$
3. $i^{k+1} = max(i^{k+1}, 0)$

La idea es partir de la imagen tomada por el microscopio (o). Cada factor de corrección es determinado como la diferencia entre o y la convolución de la $k - esima$ señal inicial estimada y el PSF, o^k. Si o^k es más borroso que o, la imagen se perfilará restándole las intensidades correspondientes a la diferencia. El parámetro γ se inicializa a $1-(o^k - A)^2/A^2$ donde A es el valor de intensidad máximo de o dividido por 2; γ restringe las intensidades al rango $[0, 2A]$ [10].

El algoritmo puede ser ejecutado un número determinado de iteraciones o hasta que se cumpla un criterio de convergencia [10]. Experimentos empíricos con imágenes reales muestran que son necesarias entre 20-25 iteraciones para obtener una imagen restaurada aceptable por los expertos.

Este algoritmo tiende a enfatizar tanto el contraste como el ruido en las imágenes. Normalmente se suele aplicar un filtro de suavizado, como por ejemplo *Wiener*, antes de empezar el algoritmo, e incluso suavizados intermedios a los resultados cada varias iteraciones.

3.3 Maximum Likelihood Estimation

El algoritmo de deconvolución *Maximum Likelihood Estimation* (MLE), está diseñado para restaurar la imagen maximizando la probabilidad que la imagen generada se parezca a la original. Los algoritmos derivan del modelo de formación de imágenes que incluyen ruido siguiendo una distribución de probabilidad dada.

En el caso de los microscopios de fluorescencias, la fuente de ruido predominante proviene de la naturaleza estocástica de la emisión de fotones, y por tanto sigue una distribución de Poisson

Este método es iterativo, al igual que el *Jansen-van Cittert*, por lo que en cada una de las iteraciones se producen los siguientes pasos:

1. $a = i^k \otimes h$
2. $b = o/a$
3. $i^{k+1} = ci^k(b \otimes h)$

donde c es una constante de normalización. La multiplicación y división son punto a punto [4] [1].

Este método de deconvolución no requiere un paso de proyección para evitar valores negativos, ya que sólo se producen multiplicaciones y divisiones de números positivos. El ruido en las imágenes resultantes puede crecer con este algoritmo, pero se puede reducir con suavizados Gaussianos en la imagen inicial, o acabando el algoritmo antes de llegar a converger. Este algoritmo suele producir mejores resultados que el *Jansen-van Cittert*, pero a cambio requiere que en cada iteración se produzcan dos convoluciones en lugar de una sola, implicando una elevación del coste computacional.

4 Análisis de las imágenes tomadas con microscopio confocal

En este apartado vamos a analizar los resultados de los algoritmos de deconvolución previamente presentados aplicados sobre imágenes tomadas por un microscopio confocal utilizado por el centro de análisis de imagen de la Universidad de Murcia (SACE). Para ello, comparamos los resultados de estos algoritmos con los resultados obtenidos por el software Huygens Deconvolution.

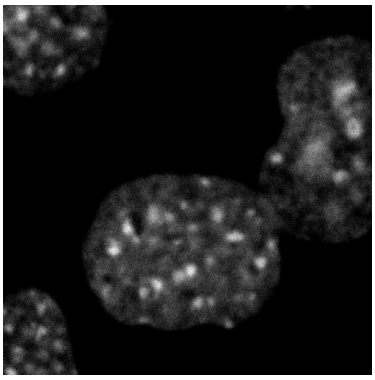
La figura 1(b) muestra el resultado obtenido por el programa *Huygens Deconvolution* [19] aplicado a la imagen original tomada por el microscopio confocal (mostrado en la figura 1(a)). Este programa limpia la imagen de la mayoría del ruido general que rodea los objetos, marcando mejor las partes más densas.

La muestra tomada procede de un núcleo celular marcado con DAPI. El DAPI es un marcador fluorescente que se une de forma específica al ADN bicatenario. Es por esto que su uso está muy extendido en microscopía de fluorescencia como tinción de los núcleos celulares, obteniéndose un contraste muy interesante cuando se marcan otras estructuras celulares en rojo o verde, ya que la emisión del DAPI es de color azul.

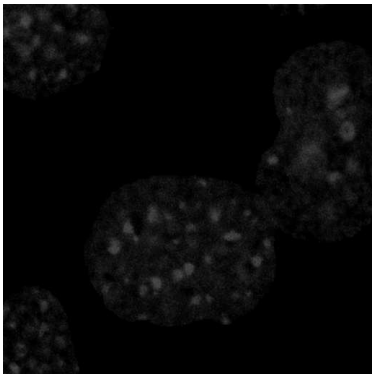
La deconvolución de un núcleo celular teñido con DAPI es importante tanto para evitar la posible

saturación de la imagen como para evitar el *blurring* que podría estar dándonos falsas localizaciones en caso de estar estudiando estructuras celulares perinucleares o cercanas al núcleo celular.

Las imágenes mostradas están en blanco y negro ya que todo el tratamiento de imágenes se hace en una escala de grises de 8 bits. Para ello separamos cada canal de color de la imagen original, tratando las imágenes en escala de grises. Posteriormente, las imágenes pueden volver a unirse en caso de ser necesario.



(a) Imagen original tomada por el microscopio confocal



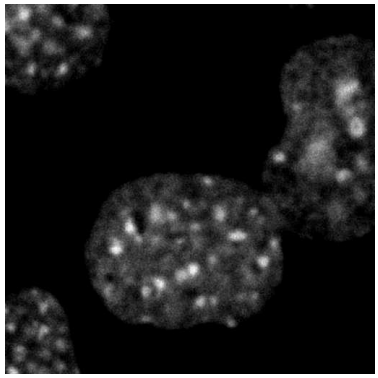
(b) Deconvolución del programa Huygens deconvolution

Figura 1. Resultado de aplicar Huygens sobre la imagen original

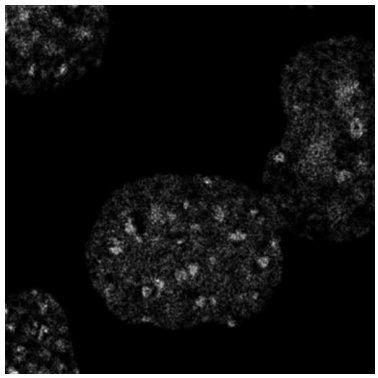
La figura 2 muestra los efectos de cada uno de los algoritmos de deconvolución sobre la imagen tomada por el microscopio confocal (figura 1(a)).

La figura 2(a) muestra el efecto de aplicar el algoritmo de *Wiener* sobre la imagen original usando un PSF teórico que sigue una distribución de probabilidad gaussiana. Como se puede ver, al ser un algoritmo de una sola pasada, la limpieza que se produce es muy liviana.

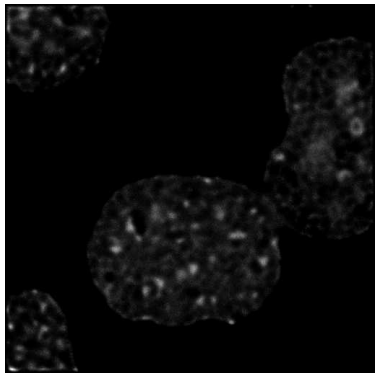
En cambio, el algoritmo de *Jansen-van Cittert* se engloba dentro de los métodos de deconvolución iterativa. Este método realiza el número de pasadas necesarias hasta producir una limpieza profunda en la imagen, o alcanzar un número de iteraciones determinadas. La imagen 2(b) muestra el resultado de aplicar el algoritmo de *Jansen-van Cittert*. En esta



(a) Deconvolución del algoritmo de Wiener



(b) Deconvolución del algoritmo de Jansen-van Cittert



(c) Deconvolución del algoritmo Maximum Likelihood Estimation

Figura 2. Resultado de aplicar los distintos algoritmos de deconvolución en microscopía confocal

ocasión, apreciamos una limpieza más profunda de la imagen original.

A la imagen mostrada en la figura 2(b), realizamos una pequeña convolución usando un PSF gaussiano mucho más ligero que el usado para el algoritmo de *Jansen-van Cittert*. Esto es debido a que al aplicar únicamente este método, los puntos de alta densidad aparecen sobresaturados, aplicándoles una pequeña convolución, se suavizan estos puntos y la imagen tiene el aspecto deseado.

El método *Maximum Likelihood* también forma parte del conjunto de algoritmos iterativos. La imagen 2(c) muestra el resultado de aplicar el algoritmo *Maximum Likelihood Estimation*. Este método obtiene un resultado donde se elimina gran parte del ruido. Además consigue una buena señalización del objeto, marcando muy bien los contornos.

Al igual que en el caso de *Jansen-van Cittert* podemos realizar más pasadas del algoritmo para obtener una mayor profundidad en la eliminación de ruido. También es necesaria la aplicación de una pequeña convolución al final, para suavizar el resultado.

4.1 Evaluación de la validez de las imágenes de microscopía confocal

Después de explicar todos los métodos utilizados para hacer el proceso de deconvolución, y las características que cada uno de ellos presentan, analizamos la calidad de las imágenes obtenidas por estos procesos. Para ello, comprobamos la validez de las imágenes comparándolas con las obtenidas por el software *Huygens Proffessional* [19]. Para realizar esta comparativa utilizamos la métrica *Peak Signal to Noise Ratio* (PSNR) [15].

Peak Signal to Noise Ratio (PSNR) es un término utilizado en ingeniería para definir la relación entre la máxima energía posible de una señal y el ruido que afecta a su representación real. Debido a que muchas señales tienen un gran rango dinámico, el PSNR se expresa generalmente en escala logarítmica, utilizando como unidad el decibelio. Los valores típicos que adopta este parámetro están entre 30 y 50 dB [15].

La fórmula 7 muestra la métrica PSNR

$$PSNR = 10 \cdot \log_{10}\left(\frac{MAX_I^2}{MSE}\right) = 20 \cdot \log_{10}\left(\frac{MAX_I}{\sqrt{MSE}}\right) \quad (7)$$

donde MAX_I es el valor máximo que puede tomar un píxel en la imagen. En nuestro caso, al ser imágenes en blanco y negro con 8 bits de profundidad, el valor máximo será 255, que representa al blanco. MSE (*Mean Squared Error*) mide la media del error entre dos imágenes, es decir, la cantidad en la que el estimador difiere de la expresión estimada. MSE viene definido por la ecuación 8.

$$MSE = \frac{1}{mn}\sum_{i=0}^{m-1}\sum_{j=0}^{n-1}||I(i,j) - K(i,j)||^2 \quad (8)$$

donde m y n son los valores del ancho y alto de una imagen, $I(i,j)$ es el pixel de la imagen original que se encuentra en la fila i, columna j, y $K(i,j)$ el pixel de la imagen generada por el algoritmo que se encuentra en la fila i, columna j.

El objetivo de utilizar una métrica como PSNR es cuantificar la bondad de nuestras imágenes en comparación con las imágenes producidas por un software comercial. Con un valor de PSNR de dos imágenes entre 25 dB y 30 dB podemos considerar que ambas imágenes tienen un parecido aceptable [15].

La tabla 1 muestra los resultados de aplicar la métrica PSNR a las imágenes obtenidas por los distintos algoritmos descritos anteriormente, comparadas con las imágenes obtenidas por el programa *Huygens Deconvolution*. Estos resultados miden el nivel de ruido que hay entre dos imágenes deconvolucionadas. Por tanto, los niveles de ruido de las diversas imágenes son aceptables, ya que todas ellas están en el rango de (25 dB, 30dB). En conclusión, nuestro resultado tiene unos niveles aceptables de ruido en comparación con el de la aplicación *Huygens Deconvolution*.

Tabla 1. Niveles de ruido en las imágenes

Prueba	PSNR
Wiener	28,628991 dB
Jansen-van Cittert	28,962507 dB
Maximum Likelihood	28,548608 dB

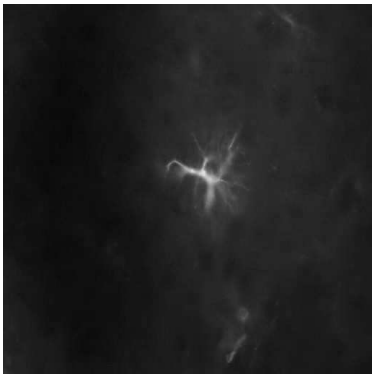
Desafortunadamente, y como podemos ver en las imágenes mostradas en la figura 2, la métrica PSNR consigue un resultado parcial de validez, mostrándose necesario además, realizar una encuesta visual por expertos para evaluar la calidad de los resultados obtenidos.

5 Efectos de los Algoritmos de deconvolución en microscopía de fluorescencias

En este apartado analizamos los resultados de los algoritmos de deconvolución previamente presentados aplicados sobre imágenes tomadas esta vez con microscopio de fluorescencia. El microscopio de fluorescencia utilizado en las pruebas pertenece al departamento de hematología del Hospital Universitario Virgen de la Arrixaca (HUVA). La limpieza de las imágenes tomadas por este microscopio se realiza con el software comercial Nis de Nikon.

En este caso, las imágenes que estudiamos son distintas a las del capítulo anterior. El ruido producido por las imágenes tomadas con un

microscopio de fluorescencia sigue una distribución de probabilidad de Poisson. Sin embargo, el PSF que tenemos disponible para nuestras pruebas sigue una distribución Gausiana, por este motivo los resultados no son tan buenos como los obtenidos por el programa del microscopio, ya que éste posee toda la información necesaria para generar el PSF óptimo para ese microscopio en concreto.



(a) Imagen Original tomada por el microscopio de fluorescencia

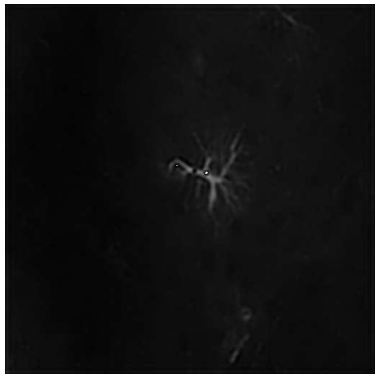
(b) Deconvolución del programa NIS de Nikon

Figura 3. Imagen original y deconvolucionada por software comercial del microscopio

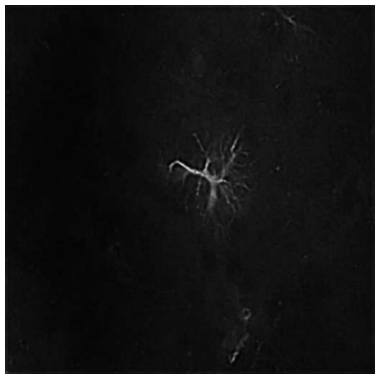
La figura 3(b) muestra el resultado obtenido por el programa del microscopio que tienen en el HUVA aplicado a la imagen de la figura 3(a).

Las imágenes tratadas muestran astrocitos marcados con GFAP (*Glial fibrillary acidic protein*), una proteina cuya función principal es proteger la rígida organización de la estructura tridimensional interna de los astrocitos del cerebro. Este marcaje es de color verde, y sirve para definir la morfología de la célula neuronal. Además de este marcaje, se tinta de rojo el núcleo humano de la célula. La intención es insertar en un cerebro de ratón una serie de células madre del cordón umbilical humano. Con este proceso se puede estudiar, una vez limpios los dos canales (el rojo y el verde) y hecha la reconstrucción 3D de la célula, si se ha conseguido que las células madre usadas en el cerebro del ratón se adapten y se conviertan en parte del sistema nervioso.

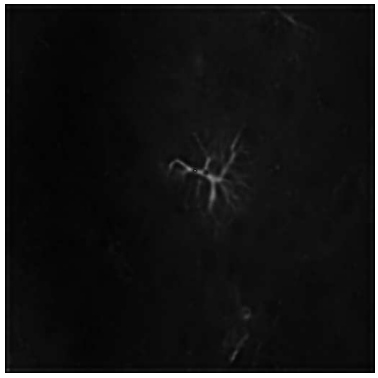
El método usado por el programa NIS de Nikon en el HUVA nos sirve de referencia para el análisis



(a) Deconvolución del algoritmo de Wiener



(b) Deconvolución del algoritmo de Jansen-van Citter



(c) Deconvolución del algoritmo de Maximum Likelihood Estimation

Figura 4. Resultado de aplicar los distintos algoritmos de deconvolución en microscopía de fluorescencia

de la calidad de las imágenes obtenidas por los algoritmos objetos de estudio. En la figura 3(b), el software comercial consigue eliminar la mayoría del ruido de la imagen, quedando marcada únicamente la célula central, parte de la muestra objeto de estudio. Además, el software elimina gran cantidad del contorno de la célula, siendo este contorno brillo emitido por los fluorocromos, y por tanto objeto de eliminación de la imagen.

La figura 4 muestra los efectos de los diferentes algoritmos de deconvolución anteriormente citados aplicados sobre la imagen original tomada por el microscopio de fluorescencias, mostrada en la figura 3(a).

El algoritmo de *Wiener* está diseñado para tratar imágenes que tengan un ruido con una distribución de probabilidad Gaussiana. En cambio, el ruido producido por los microscopios de fluorescencias en la toma de imágenes sigue una distribución de Poisson. Por tanto, este algoritmo no produce un resultado adecuado por dos motivos principales: no es adecuado por su naturaleza, y además realiza una única pasada. Sin embargo, la imagen resultante (figura 4(a)) elimina gran cantidad de ruido del entorno, y consigue delimitar bastante bien la figura. No obstante, quedan restos de brillo alrededor de la figura que pueden ser eliminados con procesos de mayor envergadura.

La figura 4(b) muestra el resultado de tratar la imagen con el algoritmo de *Jansen-van Cittert*. Con este proceso se ha conseguido una mayor definición de la célula, eliminando mayor cantidad de brillo y consiguiendo una mejor visualización de la célula, ya que se logra aumentar el contraste de ésta con respecto al fondo, y marcar mucho mejor su contorno.

La imagen 4(c) muestra el resultado de aplicar el algoritmo *Maximum Likelihood Estimation*. El resultado es similar al del obtenido por el método de *Jansen-van Citter*, sólo que definiendo algo mejor la célula.

5.1 Evaluación de la validez de las imágenes de microscopía de fluorescencias

Tabla 2. Niveles de ruido en las imágenes

Prueba	PSNR
Wiener	28,525321 dB
Jansen-van Cittert	27,962582 dB
Maximum Likelihood	28,228509 dB

La tabla 2 muestra los resultados de aplicar la métrica PSNR a las imágenes obtenidas por los distintos algoritmos descritos anteriormente, comparadas con las imágenes obtenidas por el programa NIS Nikon. El nivel de ruido entre las dos imágenes deconvolucionadas está dentro del rango aceptado (25 dB y 30 dB) [15], por tanto, nuestro resultado está en un rango de ruido aceptable en comparación con la el de la aplicación NIS Nikon.

6 Unidad de procesamiento gráfico (GPU)

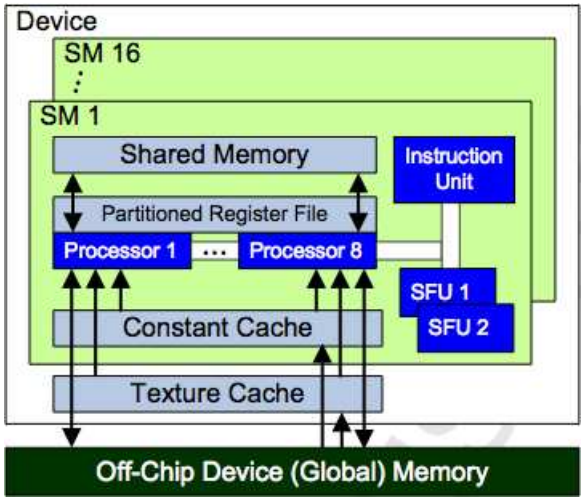


Figura 5. Arquitectura G80 de Nvidia

La figura 5 muestra la arquitectura G80 de Nvidia. En esta arquitectura destacamos [8]:

- *Streamming Multiprocessor* (SM). La GPU presenta 16 SM. Cada SM contiene 8 *Stream Processors* (SPs), cada uno de ellos funcionando a una velocidad de 1.35GHz, y 768 registros de 32 bits, para un total de 8192 registros. Además, cada SM tiene dos unidades funcionales especiales (SFUs) que ejecutan operaciones complejas de punto flotante (seno, coseno, raíces,...), una memoria local, llamada *Shared Memory*, de 16KB de tamaño con muy baja latencia de acceso y un gran ancho de banda, similar a una caché de primer nivel [12].
- *Constant Cache*. Memoria optimizada para el uso de constantes para acceder a ellas de forma rápida.
- *Texture Cache*. Similar a la *Constant Cache* pero para proporcionar un acceso rápido a texturas.
- *Device o Global Memory*. Memoria global de la tarjeta gráfica situada fuera del chip (ver figura 5) y con una latencia entre 400-600 ciclos [8].
- *Shared Memory*. Memoria compartida por todos los hilos asignados a un mismo bloque. Tiene una baja latencia si se accede siguiendo un patrón determinado [8].

La tabla 3 resume los distintos tipos de memoria existentes en la arquitectura G80 de Nvidia, así como sus tamaños y latencias de acceso.

Los SMs tienen una arquitectura SIMT (*Simple Instruction, Multiple Thread*) [12]. Todo el manejo de hilos, desde la creación, hasta la planificación y la sincronización, se hace en hardware por el SM sin apenas sobrecarga, consiguiendo manejar eficientemente un gran número de hilos.

Tabla 3. Características de las memorias incluidas en Tesla C870

Memoria	Lugar	Tamaño	Latencia	Sólo lectura
Global	Off-chip	1,5GB	400-600 ciclos	no
Shared	On-chip	16KB por SM	≃ registros	no
Constant	On-chip cache	64KB	≃ registros	si
Texture	On-chip cache	1,5GB	Más de 100 ciclos	si
Local	Off-chip	1,5GB	400-600 ciclos	no

Los programas paralelos diseñados para la GPU se escriben en lenguaje de programación C/C++, usando además una serie de extensiones (denominadas CUDA) que definen el manejo del programa en la GPU. La GPU se trata como un coprocesador de la CPU capaz de ejecutar funciones denominadas *kernels*. Es tarea del programador diferenciar el trabajo que realizará la CPU y cual la GPU. La CPU envía y recibe datos de la GPU a través de llamadas al API de CUDA, y además es encargada de monitorizar la ejecución del *kernel* [8].

Los hilos se organizan en una jerarquía de tres dimensiones. En el nivel superior, cada *kernel* crea un *grid* que consiste en varios bloques de hilos (véase figura 6). El máximo número de hilos en cada bloque es de 512. Cada bloque es asignado a un SM en particular durante su ejecución. Los hilos dentro de un mismo bloque pueden compartir datos a través de la *Shared Memory* y pueden sincronizarse haciendo uso de la directiva **__syncthreads**. La sincronización entre hilos de diferentes bloques sólo puede hacerse de forma segura mediante la finalización del kernel. Por último, los hilos en un bloque se organizan en *warps* de 32 hilos [11].

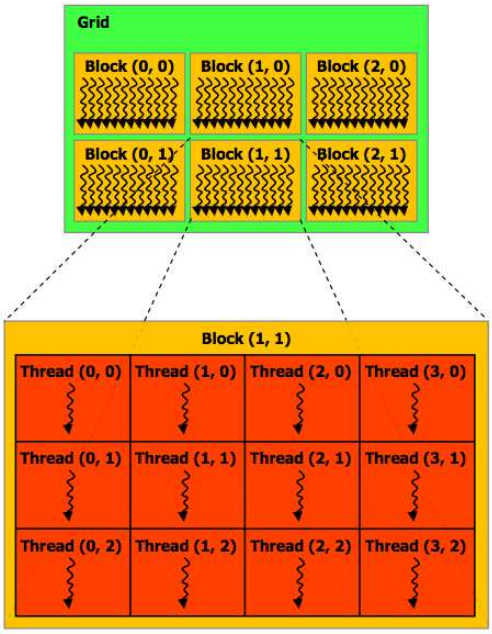


Figura 6. Organización de Threads, Blocks y Grids

La tabla 4 muestra valores de algunos de los parámetros de la Tesla C870.

Tabla 4. Parámetros de la Tesla C870

Parámetro	Valor Máximo
Hilos por SM	768 hilos
Bloques de hilos por SM	8 bloques
Registros de 32 bits por SM	8192 registros
Shared Memory por SM	16384 bytes
Hilos por bloque de hilos	512 Hilos
Warps por SM	24 Warps

La tarjeta gráfica está conectada a la unidad de procesamiento (CPU) a través del bus PCI Express, con un ancho de banda de 4GB/s en ambas direcciones [7].

7 Evaluación de rendimiento

Los algoritmos de deconvolución poseen un elevado coste computacional, debido a la cantidad ingente de operaciones que deben realizar sobre cada uno de los píxeles de la imagen. Además, en los algoritmos de deconvolución que constan de varias pasadas, el coste computacional crece exponencialmente.

Sin embargo, estos algoritmos son altamente paralelizables, ya que las operaciones a realizar sobre cada uno de los píxeles de la imagen es independiente del resto, evitando necesidades de sincronización y/o comunicación entre procesos.

Para analizar el rendimiento de nuestros algoritmos, hemos usado tres entornos *hardware* distintos: un entorno secuencial, un entorno CMP (*Chip Multi Processor*) y el entorno GPU.

La configuración del computador multi-núcleo utilizado para nuestras pruebas consta de: Dos procesadores Intel Xeon E540, de 4 núcleos a 2,33GHz cada uno, una caché L1 de 32KB privada en cada núcleo, 2 bancos de caché L2 de 6144KB compartida. La frecuencia del bus de datos es de 1333MHz y 8GB de memoria RAM DDR3.

La Unidad de Procesamiento Gráfico (GPU) donde realizamos nuestros experimentos es la nVidia Tesla C870, con 1,5GB de memoria GDDR3 dedicadas, 76,8GB/s de ancho de banda de memoria. Está conectada a una máquina con un procesador Intel Core 2 Duo E6850 a 3,00GHz con 3GB de memoria RAM DDR3.

La implementación de la librería *Clarity* requiere la librería fftw3 [3] para funcionar correctamente, esta fue compilada con las opciones *-enable-single -enable-threads* y *-with-pic* para permitir que las transformadas de Fourier pudieran ser optimizadas mediante varios hilos de ejecución.

Para las implementaciones mediante CUDA se ha hecho uso del SDK2.0 [9] y la versión 2.1 del compilador nvcc.

Tabla 5. Tiempo de ejecución en segundos de los algoritmos de deconvolución

Prueba	**1 Núcleo**	**8 Núcleos (SpeedUp)**	**CUDA (SpeedUp)**
Wiener	8,5276 s	3,5958 s (2,37)	2,1937 s (3,88)
Jansen-van Cittert	58,3712 s	22,2503 s (2,62)	9,8040 s (5,95)
Maximum Likelihood	103,6818 s	35,5835 s (2,91)	17,8091 s (5,82)

Para todas las pruebas se ha usado una pila de 64 imágenes de 512x512 píxeles. El PSF para las pruebas ha sido una pila de 32 imágenes de 64x64 píxeles con una distribución Gaussiana.

La tabla 5 muestra los tiempos necesarios para ejecutar los distintos algoritmos estudiados, además el factor de *speedup* conseguido por las versiones de 8 núcleos y CUDA en comparación con la versión secuencial. La versión en OpenMP para 8 núcleos de los tres algoritmos consigue una mejora del orden de tres veces superior respecto a la versión secuencial. La versión CUDA consigue en los casos de *Jansen-van Cittert* y *Maximum Likelihood* un rendimiento de hasta seis veces mejor que en la versión secuencial. Estos algoritmos iterativos obtienen una aceleración mayor que el caso de del algoritmo de Wiener debido a que son computacionalmente más intensivos.

Respecto a los tiempos de los programas comerciales usados por los expertos, en el centro de análisis de imágenes de la Universidad de Murcia utilizan el programa *Huygens Deconvolution*. Este programa tardó para la pila de imágenes tomadas con microscopio confocal mostradas durante este trabajo, y siendo ejecutado en un ordenador de sobremesa con un procesador Intel Pentium IV a 2.4 GHz y 1 GB de memoria RAM, del orden de quince minutos.

En cuanto al programa programa NIS de Nikon usado en el Hospital Universitario Virgen de la Arrixaca, ejecutó las imágenes mostradas en este trabajo de microscopía de fluorescencias sobre un procesador Intel Dual-Core E5200 a 2.5 Ghz tardó del orden de cuatro minutos. Estos tiempos son aproximados, ya que no se podía acceder al código para insertar marcas de tiempo.

La tabla 6 muestra un análisis desglosado del tiempo empleado en cada una de las partes del código de Clarity. En la versión de GPU incluimos los tiempos de transferencia de datos a la tarjeta gráfica.

Como muestra la tabla 6, la mayor parte del tiempo de ejecución de la aplicación se dedica a las transformadas de Fourier, tanto la directa (FFT R2C) como la inversa (FFT C2R). En las versiones de CPU se consume aproximadamente un 45% del tiempo en este proceso, mientras que en las versiones de GPU se dedica el 70% en *Wiener* y más del 90% en los otros dos algoritmos.

Otro aspecto importante a tener en cuenta es el tiempo dedicado a las primitivas de los propios algoritmos. En la versión de GPU el tiempo dedicado a ellas es muy pequeño, casi despreciable, comparado con las versiones de CPU. Esto deja ver que estos algoritmos son bastante paralelizables, y la arquitectura de las tarjetas gráficas es capaz de aprovechar esta característica para obtener un buen rendimiento.

Tabla 6. Desglose tiempo de ejecución de los tres algoritmos en segundos

Parte del código	**1 Núcleo**	**8 Núcleos**	**CUDA**
Wiener			
Total	8,5 s	3,5 s	2,1 s
FFT R2C	5,5 s (65%)	1,3 s (37%)	0.9 s (45%)
FFT C2R	1,6 s (19%)	0.4 s (13%)	0.5 s (25%)
Kernel	1,3 s (16%)	1,8 s (50%)	0.009 s (0,5%)
Copiar a GPU	-	-	0.2 s (13%)
Copy desde GPU	-	-	0.3 s (17%)
Jansen-van Cittert			
Total	58,3 s	22,2 s	9,8 s
FFT R2C	24,9 s (43%)	3,4 s (15%)	4,4 s (45%)
FFT C2R	18,5 s (32%)	3,4 s (16%)	4,4 s (45%)
Modulate	5,5 s (9%)	6,1 s (28%)	0,1 s (1%)
Update Kernel	9,3 s (16%)	9,1 s (41%)	0,2 s (2%)
Copiar a GPU	-	-	0,3 s (4%)
Copiar desde GPU	-	-	0,1 s (2%)
Maximum Likelihood			
Total	103,6 s	35,5 s	17,8 s
FFT R2C	37,2 s (36%)	7,9 s (22%)	7,8 s (44%)
FFT C2R	35,7 s (34%)	7,2 s (20%)	9 s (51%)
Modulate	9,4 s (9%)	6,1 s (17%)	0,1 s (1%)
Multiply	5,4 s (5%)	3,8 s (11%)	0,02 s (0,1%)
Division	6,9 s (7%)	5,2 s (15%)	0,1 s (0,6%)
Scale	4,5 s (4%)	2,6 s (7%)	0,067 s (0,3%)
Reduce	4,3 s (4%)	2,5 s (7%)	0,1 s (0,8%)
Copiar a GPU	-	-	0,2 s (1%)
Copiar desde GPU	-	-	0, 1 s (1%)

8 Conclusion y Trabajo Futuro

En este trabajo adaptamos la librería de deconvolución de imágenes `Clarity` para limpiar imágenes biomédicas de microscopía. Además, analizamos las imágenes producidas por los algoritmos de deconvolución incluidos en la librería, y analizamos la eficiencia de estos algoritmos sobre la GPU, demostrando que obtiene un rendimiento considerable, y que la paralelización de estos algoritmos es productiva en esta plataforma.

Por otro lado, describimos el proceso de deconvolución de imágenes de microscopía; analizamos los

parámetros existentes en dicho proceso, destacando entre ellos la importancia de contar con un correcto PSF para obtener un resultado óptimo de limpieza de las imágenes, describimos los distintos tipos de algoritmos de deconvolución empleados, destacando las cualidades de cada uno de ellos, y comprobamos la influencia del tipo de microscopio en las imágenes y en su posterior procesamiento.

En este trabajo obtenemos un resultado en la calidad de las imágenes aceptable, pero necesitamos perfeccionar estos resultados dependiendo del tipo de microscopio utilizado. Para ello, realizaremos un estudio de la obtención y manejo del PSF para hacer que el proceso esté mejor calibrado, y el resultado sea más cercano al óptimo.

Además, validaremos los resultados realizando test psico-visuales de expertos hasta que consigamos una calidad óptima en las imágenes deconvolucionadas.

Una vez analizado el comportamiento de los algoritmos existentes en la librería `Clarity` implementaremos nuevos algoritmos de deconvolución de imágenes adaptados a las necesidades concretas de nuestras imágenes. Entre estos algoritmos destacamos el *I-Divergence Deconvolution* que, debido a sus características, se adecúa mejor a las imágenes obtenidas por los tipos de microscopios con los que hemos trabajado.

Debido a la gran importancia que presenta la eficiencia de estos algoritmos, analizaremos arquitecturas de altas prestaciones que se adecuen a la naturaleza de estos algoritmos, implementaremos kernel eficientes en clusters de GPUs, así como buscaremos alternativas, como OpenCL, que nos ayuden a portar estos algoritmos a otras plataformas multinúcleo.

Finalmente, desarrollaremos una interfaz web para facilitar el trabajo a los usuarios finales de estos algoritmos, permitiendo un fácil acceso a las imágenes, los parámetros de la deconvolución, distintos tipos de efectos, análisis de las imágenes para detectar patrones y aspectos determinantes a la hora de elegir que algoritmo usar, etc.

Agradecimientos

Agradecemos la ayuda del proyecto de la Fundación Séneca (Agencia Regional de Ciencia y Tecnología, Región de Murcia) con la subvención 00001/CS/2007, al Ministerio de Educación y Ciencia (MEC), y la comisión Europea FEDER con la subvención CSD2006-00046.

También agradecemos su colaboración al departamento de hematología del Hospital Universitario Virgen de la Arrixaca, y en concreto al Dr. Blanquer Blanquer por su estrecha colaboración en este proyecto, así como al departamento de análisis de imágenes de la Universidad de Murcia, y en particular a la Dr. Fara Sáez Belmonte.

Finalmente, nos gustaría agradecer a los revisores anónimos por sus detallados y constructivos comentarios que han ayudado a mejorar la calidad de este artículo.

Referencias

[1] Lucy L. B. An iterative technique for the rectification of observed distributions, Astronomical Journal, 1974.

[2] Bradley Bargen, Peter Donnelly. Inside DirectX, Microsoft Programming Series, 1998.

[3] M. Frigo, S. Johnson. The design and implementation of FFTW3, Proceedings of the IEEE 93 vol 2, 2005.

[4] Richardson W. H. Bayesian-based iterative method of image restoration, Journal of Optical Society of America, vol 62.

[5] David Luebke, Mark Harris, Jens Krüger, Tim Purcell, Naga Govindaraju, Ian Buck, Cliff Woolley, Aaron Lefohn. GPGPU: general purpose computation on graphics hardware, International Conference on Computer Graphics and Interactive Techniques, 2004.

[6] James McNally, Tatiana Karpova, John Cooper, José-Angel Conchello. Three-dimensional imaging by deconvolution microscopy, Journal of Methods, 1999.

[7] John Nickolls, Ian Buck, Michael Garland, Kevin Skadron. Scalable Parallel Programming, ACM Queue, 2008.

[8] NVIDIA. CUDA Programming Guide, 2008.

[9] NVIDIA. CUDA SDK Code Samples, 2009.

[10] Cory W. Quammen, David Feng, Russel M. Taylor II. Performance of 3D Deconvolution Algorithms on Multi-Core and Many-Core Architectures, Technical report, University of North Carolina at Chapel Hill Department of Computer Science, 2009.

[11] Shane Ryoo, Christopher I. Rodrigues, Sam S. Stone, John A. Stratton, Sain-Zee Ueng, Sara S. Baghsorkhi, Wen-mei W. Hwu. Program Optimization Carving for GPU Computing, Journal of Parallel and Distributed Computing, 2008.

[12] Nadathur Satish, Mark Harris, Michael Garland. Designing Efficient Sorting Algorithms for Manycore GPUs, IEEE International Parallel and Distributed Processing Symposium, 2009.

[13] J. L. Starck, E. Pantin, F. Murtagh. Deconvolution in Astronomy: A Review, 2002.

[14] Manuel T. Silva, Enders A. Robinson. Deconvolution of Geophysical Time Series in the Exploration for Oil and Natural Gas, Elsevier Scientific Publishers, 1979.

[15] Zhou Wang, Alan C. Bovik, Hamid R. Sheikh, Eero P. Simoncelli. Image quality assessment: From error visibility to structural similarity, IEEE Transactions on Image Processing, 2004.

[16] M. Woo, J. Neider, T. Davis, D. Shreiner. OpenGL programming guide, Addison-Wesley, 1999.

[17] ATI. `http://ati.amd.com/`.

[18] OpenCL. `http://www.khronos.org/opencl/`.

[19] Huygens Proffesional. `http://www.svi.nl/products/professional/`.