Procesamiento de múltiples flujos de datos con Growing Neural Gas sobre Multi-GPU

S. Orts-Escolano, J. García-Rodríguez, V. Morell-Giménez

Resumen— **Los gases neuronales crecientes (GNG) han sido utilizados con éxito tanto para clustering y reconocimiento de patrones como para la representación topológica de conjuntos de datos. En trabajos anteriores se presentó la aceleración de la red neuronal GNG usando Unidades de Procesamiento Gráfico (GPUs) para paralelizar los cálculos del proceso de aprendizaje. En este trabajo proponemos la utilización de múltiples GPUs para paralelizar el aprendizaje de distintas redes neuronales que actúan sobre distintos espacios de entrada de forma simultánea. La arquitectura propuesta para llevar a cabo la paralelización combina la utilización de procesadores multinúcleo y múltiples GPUs conectadas a un mismo sistema. Varios experimentos son presentados con el objetivo de demostrar la validez del sistema para procesar en paralelo distintos flujos de datos.**

Palabras clave— **Gases Neuronales Crecientes, GPGPU, CUDA, multi-core CPU, multi-GPU.**

I. Introducción

En los últimos años la democratización de la computación de alto rendimiento ha sido posible gracias a la utilización de GPUs para cómputo de propósito general (GPGPU) y esto además ha atraído a un gran número de investigadores. Implementaciones relacionadas con redes neuronales [1-3], visión por computador y procesamiento de imagen [4-10] tienen un particular interés en este trabajo.

El incremento producido en los últimos años en la cantidad de datos que manejan aplicaciones de todo tipo requiere una mayor capacidad de cómputo que permita aprovechar el paralelismo inherente en los datos manejados. La GPU está dotada de una cantidad masiva de procesadores, encaja perfectamente en estas tareas y provee de la capacidad de cómputo necesaria.

El comportamiento intrínsecamente paralelo de las redes neuronales artificiales permiten su implementación sobre GPUs acelerando el proceso de aprendizaje. Extendiendo nuestro trabajo inicial [11] que adaptaba e implementaba el algoritmo de aprendizaje los Gases Neuronales Crecientes [12] sobre GPUs, proponemos el tratamiento en paralelo de múltiples espacios de entrada utilizando una arquitectura compuesta por procesadores multinúcleo y múltiples GPUs.

Las GPUs actuales poseen un elevado número de procesadores que pueden ser utilizados para cómputo de propósito general. La GPU esta específicamente diseñada para resolver problemas que tienen un elevado coste computacional y que por lo tanto requiere el procesamiento de una gran cantidad de información en paralelo.

S. Orts-Escolano, J. García-Rodríguez y V. Morell-Giménez. Departamento de Tecnología Informática y Computación, Universidad de Alicante, Ap 99. 03080, Alicante, Spain (Tel: +34 965903400; fax: +34965909643; e-mail: {sorts jgarcia,vmorell}@dtic.ua.es.

Sin embargo, la implementación de algoritmos sobre la GPU requiere que se lleve a cabo un proceso de rediseño del algoritmo de forma que este sea adaptado en la medida de lo posible a la arquitectura de la GPU para explotar todo su paralelismo computacional. Además la programación de estos dispositivos tiene una serie de restricciones: necesidad de una elevada cantidad de cómputo con el objetivo de ocultar las latencias producidas por accesos a memoria, gestión y sincronización de diferentes hilos ejecutándose de forma simultánea, el uso correcto de la jerarquía de memorias y otras consideraciones. Previamente, estos dispositivos ya se han aplicado con éxito para resolver en la GPU problemas que tradicionalmente eran resueltos en la CPU [13-16].

La implementación GPU propuesta en este trabajo esta basada en la arquitectura CUDA de NVIDIA [17], soportada por la mayoría de tarjetas gráficas de NVIDIA.

El nivel de paralelismo de nuestra implementación original de la GNG sobre GPUs ha sido extendido combinando un procesador multinúcleo y multiples GPUs, de forma que cada hilo de ejecución en el procesador multinúcleo se encarga de llevar a cabo el aprendizaje de distintos flujos de datos o espacios de entrada con diferentes redes neuronales, de forma simultánea sobre múltiples GPUs, acelerando el proceso de forma considerable respecto a la versión secuencial en un procesador con un único núcleo.

El resto del artículo se organiza de la siguiente forma: en la sección 2 se describe la arquitectura de la GPU y su escalado utilizando múltiples GPUs. En la sección 3 se describe brevemente el algoritmo de aprendizaje de la red GNG así como la implementación paralela del algoritmo de aprendizaje y su extensión sobre múltiples GPUs. La sección 4 muestra los experimentos llevados a cabo, y la utilización de esta implementación para el procesamiento de múltiples flujos de datos en paralelo, seguido de las conclusiones y trabajos futuros.

II. Arquitectura GPU/multi-GPU

Una arquitectura GPU compatible con CUDA se encuentra organizada en un conjunto de multiprocesadores como se puede observar en la figura 1 [18]. Estos multiprocesadores llamados Streaming Multiprocessors (SMs) son altamente paralelizables a nivel de ejecución de hilos. Sin embargo, el número de multiprocesadores varía dependiendo de la generación de la GPU utilizada. A su vez, cada SM esta formado por una serie de Streaming Processors (SPs) que comparten la lógica de control y una memoria cache. Cada uno de estos SPs puede manejar en paralelo una gran cantidad de hilos.

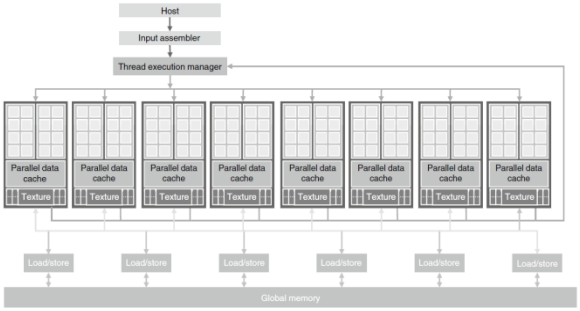

Fig. 1. Arquitectura CUDA.

La arquitectura CUDA refleja el modelo de ejecución SIMT: una única instrucción sobre múltiples hilos. Estos hilos son ejecutados simultáneamente procesando en paralelo una gran cantidad de información. Cada uno de ellos ejecuta una copia del *kernel* (pieza de código que es ejecutada) en la GPU y utiliza índices locales para identificar sobre que porción de información procesar. Los hilos a su vez son agrupados en bloques para ser ejecutados. Cada uno de estos bloques se aloja sobre un único SM permitiendo la ejecución de varios bloques en un mismo SM. El número de bloques que son ejecutados depende de los recursos disponibles así como de las características de la arquitectura. CUDA cuenta con un planificador de ejecución encargado de coordinar de forma automática la ejecución de los distintos bloques de hilos sobre los recursos hardware, SMs.

Dentro de cada uno de estos bloques, los hilos son agrupados a su vez en conjuntos de 32 hilos, de forma que estos conjuntos se ejecutan de forma totalmente paralela gracias a la segmentación del cauce de ejecución de las GPUs.

La arquitectura CUDA presenta diferentes niveles de memoria con el fin de obtener el máximo rendimiento: memoria constante, de texturas, global, compartida por los SMs y registros locales representan los tipos más importantes de la arquitectura. La memoria compartida juega un papel crucial al utilizarse como una cache a nivel de ejecución en cada SMs. Las memorias de texturas y constantes también son ampliamente utilizadas para evitar las altas latencias que supone el acceso a memoria global.

CUDA ofrece la posibilidad de escalar la potencia de cómputo utilizando múltiples GPUs en un mismo sistema. Además, a partir de la versión 4.0 de su driver la programación de estos dispositivos se ha facilitado mucho de cara a la gestión y sincronización de los dispositivos. Cada dispositivo compatible con CUDA se maneja como un contexto dentro del sistema que puede ser además manejado por distintos hilos de cómputo de nuestra CPU. Otra consideración importante es la capacidad para transferir datos entre las distintas memorias de las GPUs utilizando un hilo de la CPU. Por otro lado la utilización de múltiples GPUs para resolver ciertos problemas presenta la restricción de tener que dividir el dominio del problema para aprovechar todos los dispositivos de cómputo de forma eficiente.

En los últimos años un gran número de aplicaciones han utilizado las GPUs para acelerar el procesamiento de algoritmos de redes neuronales artificiales [19-24], aplicados, por ejemplo, sobre varios problemas de visión artificial tales como la representación y seguimiento de objetos en escenas [25], seguimiento y modelado de caras [26] o estimación de la pose [27].

III. Implementación GNG sobre múltiples GPUs

Del modelo Neural Gas [28] y Growing Cell Structures [29], Fritzke desarrolló el modelo Growing Neural Gas [12], que no presenta una topología inicial predefinida entre las neuronas que componen su estructura, siendo estas añadidas a la red de forma incremental (figura 2).

El algoritmo de aprendizaje de la red GNG tiene un alto coste computacional, por lo que se propone su aceleración utilizando GPUs y por lo tanto aprovechando la arquitectura masivamente paralela que nos ofrecen estos dispositivos, así como su paralelismo a nivel de instrucción. Las GPUs son hardware especializado para tareas computacionalmente costosas y que posean un alto nivel de paralelismo de datos. Las GPUs dedican un número más elevado de transistores al procesamiento de datos que a la unidad de control a diferencia de las unidades de procesamiento convencionales. Para la implementación de la GNG sobre GPUs hemos utilizado la arquitectura y conjunto de herramientas de programación proporcionado por NVIDIA para explotar al máximo el paralelismo de su hardware.

A. *Algoritmo GNG*

GNG es un algoritmo de clustering incremental no supervisado que dada una distribución de entrada en R^d crea incrementalmente un grafo, o red de nodos, donde cada nodo en el grafo tiene una posición en R^d. El modelo puede ser utilizado para la cuantización de vectores encontrando así las diferentes agrupaciones presentes en una determinada distribución. Estos vectores significativos son representados por los vectores referencia (posición) de los nodos. Puede ser también utilizado para localizar estructuras topológicas que reflejan cierta similitud respecto a la estructura del espacio de entrada. El aprendizaje de la red GNG es un algoritmo dinámico. En el sentido de que si la distribución de entrada cambia ligeramente en el tiempo, la red tiene la capacidad para adaptarse moviendo los nodos hacia la nueva posición del espacio de entrada.

Empezando con dos nodos, el algoritmo crea un grafo en el cual los nodos son vecinos si están conectados por una arista. La información de vecindad es mantenida a lo largo de la ejecución por medio de una regla de aprendizaje competitivo Hebbiano (CHL).

El grafo generado por CHL crea una “triangulación inducida de Delaunay” que es un sub-grafo de la triangulación de Delaunay correspondiente al conjunto de nodos. La triangulación inducida de Delaunay preserva de forma óptima la topología de entrada [30]. CHL es un componente esencial del algoritmo de aprendizaje de la GNG ya que es utilizado para gestionar la adaptación local de los nodos que conforman la red y la inclusión de nuevos nodos a esta.

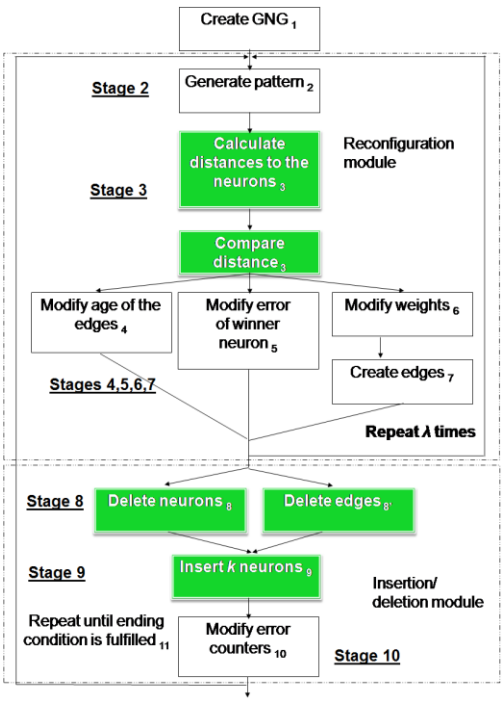

Fig. 2. Etapas del algoritmo Growing Neural Gas

La red se especifica como:

– Un conjunto N de nodos (neuronas). Cada neurona $c \in N$ tiene su vector de pesos asociado $\omega_c \in R^d$. El vector de pesos puede ser considerado como la posición en el espacio de entrada de sus correspondientes neuronas.

– Un conjunto de aristas (conexiones) entre pares de neuronas. Estas conexiones no tienen pesos y su propósito es definir una estructura topológica. Un esquema basado en la antigüedad de las aristas es utilizado para eliminar aquellas conexiones que han dejado de ser validas debido al movimiento de las neuronas en el proceso de adaptación.

Se utilizan parámetros constantes en el tiempo. Además, no es necesario definir el número de nodos utilizados a priori, ya que los nodos son añadidos de forma incremental durante la ejecución del algoritmo. La inserción de nodos se detiene cuando se alcanza un criterio de rendimiento definido por el usuario o alternativamente el tamaño máximo de la red es alcanzado.

El proceso de adaptación de la red a los vectores del espacio de entrada se produce en el paso 6. La inserción de conexiones (paso 4) entre la neurona ganadora y la segunda más cercana a la muestra del espacio de entrada establece la relación topológica entre las neuronas (figura 2).

La eliminación de conexiones (paso 8) elimina las aristas que han dejado de pertenecer a la topología. Esto se realiza eliminando las conexiones entre neuronas que no están ya cerca o que tienen otras neuronas más cercanas, de forma que la edad de las aristas entre estas neuronas excede un umbral.

La acumulación del error (paso 5) sirve para identificar aquellas áreas del espacio de entrada donde es necesario incrementar el número de neuronas para mejorar la representación.

B. Implementación GPU

Para llevar a cabo la aceleración del algoritmo de aprendizaje de la GNG sobre GPUs utilizando CUDA ha sido necesario rediseñar el algoritmo original de forma que encaje dentro de la arquitectura de la GPU. Muchas de las operaciones realizadas por el algoritmo son paralelizables debido a que estos cálculos se realizan sobre todas las neuronas de la red de forma simultánea. Es posible llevar a cabo estas operaciones en paralelo ya que no existe dependencia a nivel operacional entre las neuronas. Sin embargo existe dependencia de cálculo entre distintos ajustes de la red neuronal, lo cual afecta al final de cada iteración, forzando la sincronización de varias operaciones en paralelo. La figura 2 muestra, en verde, los distintos pasos del algoritmo de la GNG que han sido acelerados implementando kernels sobre la GPU.

1) Cálculo paralelo de las distancias de las neuronas al espacio de entrada

La primera etapa del algoritmo que ha sido acelerada es el cálculo de la distancia euclídea entre cada una de las neuronas y el patrón aleatorio del espacio de entrada que se genera en cada iteración. Estos cálculos son ejecutados en paralelo computando la distancia de cada neurona en un hilo distinto dentro de la GPU, por lo que se lanzan tantos hilos como neuronas forman la red neuronal. Es posible calcular más de una distancia por hilo, pero solo es eficiente para vectores de tamaño muy elevado, del orden de cientos de miles.

2) Reducción paralela

La segunda tarea paralelizada dentro del algoritmo de aprendizaje es la búsqueda de la neurona ganadora: aquella neurona con menor distancia respecto al patrón del espacio de entrada, y a su vez la segunda neurona más cercana. Para la búsqueda de estos dos mínimos, hemos adaptado la técnica de reducción paralela [31] para obtener, dividiendo el problema en subproblemas, no solo el mínimo, sino los dos mínimos de un vector de neuronas. Esta técnica acelera operaciones tales como la búsqueda del máximo, mínimo, suma, en general operaciones unarias sobre conjuntos de datos. Para nuestro trabajo hemos modificado el algoritmo original, de forma que con una única reducción obtenemos los dos valores mínimos de un conjunto de datos, en el trabajo nos referimos a este método como 2MinParallel Reduction. La reducción paralela puede ser descrita como un árbol binario donde en log2 (n) pasos ejecutados en paralelo aplicando una operación sobre los pares de elementos se obtiene el resultado deseado. De esta forma se reduce el coste computacional de n a $log_2(n)$ pasos.

3) Otras optimizaciones

Para acelerar el resto de pasos hemos seguido la misma estrategia usada durante la primera fase, donde cada hilo es responsable en diferentes casos de llevar a cabo operaciones sobre una neurona: comprobación de las edades de las aristas entre neuronas, eliminación de aristas que superan un determinado umbral, actualización de errores locales a cada neurona, etcétera.

En la etapa de la búsqueda de la neurona con mayor error se ha seguido la misma estrategia que la utilizada en la búsqueda de la neurona ganadora, pero en este caso la reducción busca solo la neurona con error máximo entre el conjunto, aplicando también la operación de reducción en paralelo.

Además se han seguido una serie de buenas prácticas de programación sobre la arquitectura CUDA con el fin de obtener todavía mayor rendimiento. Entre estas, cabe destacar la utilización de memoria constante para almacenar los parámetros constantes de la red neuronal: ε_1, ε_2, α, β, α_{max}. Almacenando estos parámetros en esta memoria se consigue un acceso con menor latencia que trabajando con estos valores en memoria global.

Nuestra implementación de la GNG sobre la arquitectura CUDA esta también limitada por el ancho de banda disponible en el acceso a la memoria. Aunque en la sección de experimentos se mostrarán los anchos de banda de cada dispositivo utilizado, estos son teóricos y alcanzables solo bajo situaciones idealizadas. La utilización de patrones de acceso a memoria como la utilización de la memoria compartida como una cache a nivel de multiprocesador, la utilización de los registros de forma adecuada y un acceso coalescente a memoria por parte de los hilos nos permiten aproximar ese ancho de banda teórico.

En cuanto a la ocupación de espacio en memoria, el almacenamiento de la estructura de la red neuronal, así como el manejo de distintas redes en memoria GPU no supone un problema. Por ejemplo, una red neuronal GNG compuesta por 20.000 neuronas y las estructuras auxiliares requeridas para el aprendizaje requieren solo 17 megabytes de memoria. Sin embargo, las transferencias de datos entre memorias CPU y GPU suponen un cuello de botella para obtener aceleración, ya que al existir dependencia de cálculos en la inserción de nuevas neuronas, es necesario copiar la red neuronal de vuelta a la CPU para procesar estos cálculos de forma secuencial. Las primeras versiones de la red sobre GPU no obtenían rendimiento respecto a la versión secuencial debido al elevado tiempo de transferencia que suponía la cantidad total de movimientos de datos entre memorias CPU y GPU. Estas latencias ocurren debido al ancho de banda limitado que nos ofrece el bus PCI-Express. Después de varias consideraciones sobre el flujo de ejecución, se dedujo que el tiempo de ejecución sobre un único hilo en la GPU era más eficiente que copiar entre memorias toda la información de la red neuronal, sobretodo para redes de tamaño considerable.

C. Adaptación de la red neuronal

Hemos llevado a cabo varios experimentos donde se muestra como la versión acelerada de la GNG no es solo capaz de llevar a cabo el aprendizaje inicial de forma más rápida, sino que además permite llevar a cabo sobre la red neuronal más ajustes por segundo que la versión CPU. Por ejemplo después del aprendizaje de una red compuesta por 20.000 neuronas hemos llevado a cabo 17 ajustes por segundo utilizando la GPU mientras que la CPU solo obtiene 2.8 ajustes por segundo (figura 3). Esto significa que la implementación GPU es también capaz de obtener una mejor representación topológica del espacio de entrada bajo restricciones temporales.

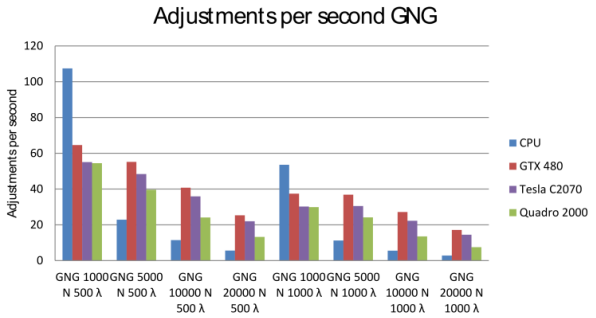

Fig. 3. Número de ajustes por segundo que manejan distintas GPUs y CPU. Versión acelerada GNG sobre 1 GPU.

IV. PROCESAMIENTO DE MÚLTIPLES ESPACIOS DE ENTRADA USANDO MULTI-GPUS

Después de acelerar el algoritmo de aprendizaje de la GNG utilizando la GPU como dispositivo de cómputo y aplicarlo a un flujo simple de datos. Proponemos abstraer y escalar esta capacidad de cómputo utilizando múltiples entradas de datos y múltiples GPUs conectadas a un mismo sistema. Todas las GPUs y los procesamientos llevados a cabo en ellas se coordinan mediante la utilización de varios hilos de procesamiento también paralelizados sobre un procesador múltinúcleo. Proponemos la utilización de múltiples GPUs conectadas a un mismo sistema para acelerar el procesamiento de múltiples flujos de entrada de información, grandes conjuntos de datos. En nuestro caso lo aplicamos al procesamiento de múltiples flujos de imágenes de video sobre las que obtenemos la estructura topológica de las mismas, acelerando todavía más el procesamiento respecto a la versión secuencial ejecutado sobre una CPU convencional. En la sección de experimentos se observará la capacidad de escalar la aceleración de un problema que presenta la utilización de múltiples GPUs, así como sus restricciones.

Esta nueva implementación incluye la utilización de procesadores multinúcleo y la utilización de varias GPUs conectadas al mismo sistema (figura 4). A través de una gestión eficiente de los hilos en la CPU y asignando diferentes flujos de datos sobre las distintas GPUs, es posible llevar a cabo en paralelo el aprendizaje de varias redes neuronales a su vez paralelizadas sobre una arquitectura GPU como dispositivo de cómputo. Este sistema propuesto permite al usuario escalar el sistema de forma sencilla utilizando una CPU con un número mayor de núcleos y añadiendo más GPUs para acelerar todavía más el procesamiento de un número mayor de flujos de información. Para llevar a cabo la implementación y gestión de los distintos hilos en la CPU se ha utilizado el estándar POSIX pthreads.

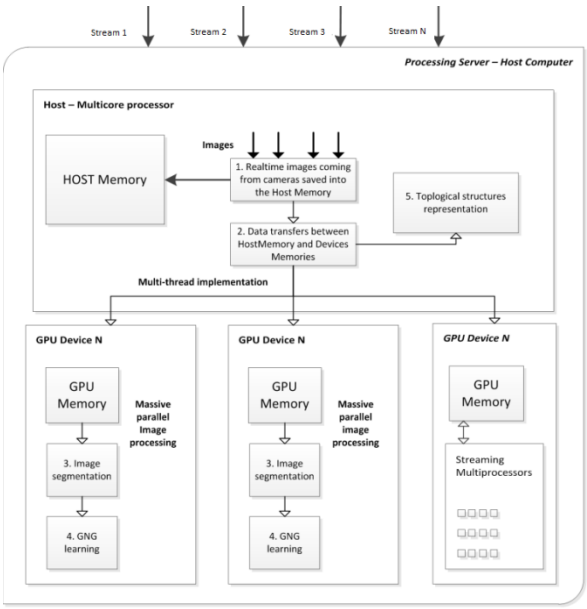

Fig. 4. Implementación Multi-GPU propuesta

A. *Experimentos*

En esta sección se presentan los resultados obtenidos de los experimentos llevados a cabo utilizando la GNG para aprender las estructuras topológicas de imágenes 2D previamente procesadas. Los múltiples flujos de información son simulados utilizando tantas imágenes en paralelo como número de cámaras sería capaz de procesar en paralelo nuestro sistema.

En la figura 5 se muestran los resultados de los experimentos para diferente número de flujos de datos. En este experimento se compara el tiempo requerido para obtener la representación topológica del espacio de entrada con la GNG utilizando redes de distinto tamaño: 5000 y 20000 neuronas respectivamente. Además las redes son entrenadas utilizando 500 y 1000 patrones de entrada por iteración. Todos los experimentos fueron llevados a cabo sobre un computador dotado de un procesador multinúcleo y dos GPUs conectadas al sistema central mediante dos puertos PCI-express a 16x.

TABLA I

ESPECIFICACIONES TÉCNICAS DE LAS GPUS UTILIZADAS Y DEL PROCESADOR MULTINÚCLEO

Device	SMs	Cores	Memory	Bandwith
GTX 480	15	480	1.5 GB	177.4 GB/s
Tesla C2050	6	448	6 GB	144 GB/s
Quadro 2000	4	192	1 GB	41,6 GB/s
Processor	**#Cores**	**#Threads**	**Clock**	**Memory**
Intel i3 540	2	4	3.06 Ghz	8 GB

Se puede apreciar también como la implementación secuencial requiere mas tiempo conforme el número de flujos de datos aumenta. También se puede apreciar como la complejidad temporal de la versión secuencial crece más rápido que la versión multiprocesador/multi-GPU. Por lo tanto se demuestra como estas aplicaciones serían inabordables sobre CPUs convencionales y la versión propuesta permite el manejo de varios flujos sin degradar el rendimiento de forma significante. La aceleración obtenida para un número elevado de flujos es sobre 2.5 veces más rápida que la versión secuencial

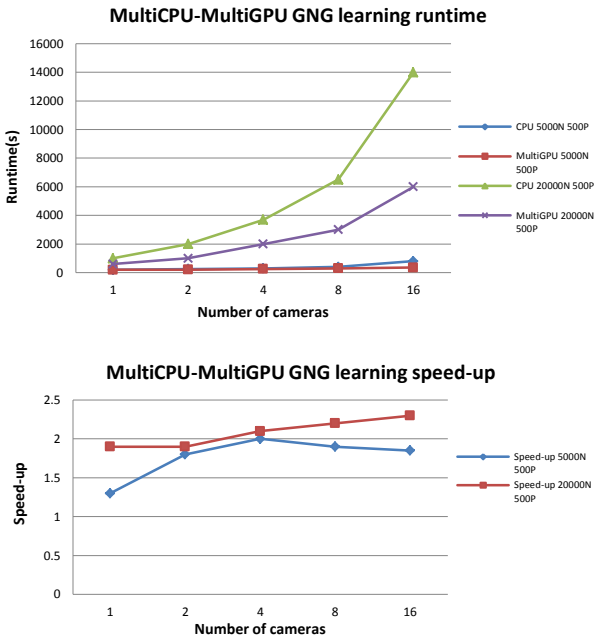

Fig. 5. Tiempo de aprendizaje de la red GNG con distintos parámetros utilizando la versión propuesta (Multi-GPU) y la versión secuencial.

En la figura 6 se demuestra como la implementación propuesta es capaz de procesar en paralelo hasta 16 flujos de datos diferentes y de llevar a cabo 3 ajustes por segundo en cada uno de ellos, de forma que la estructura topológica ofrecida por la red GNG se adapte ante cambios en el espacio de entrada y sea capaz de hacerlo bajo restricciones temporales. También se puede apreciar como la versión CPU obtiene un número de adaptaciones significativamente menor comparado con la versión propuesta, obteniendo un número de ajustes entre 5 y 10 veces mayor.

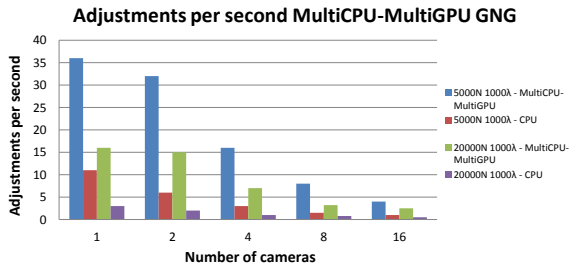

Figura. 6. Número de ajustes por segundo para diferentes parámetros de aprendizaje de la red GNG y para distinto número de flujos de datos utilizando la versión propuesta y la versión secuencial.

V. CONCLUSIONES

En este trabajo se ha propuesto una implementación acelerada del aprendizaje de la red neuronal GNG bajo restricciones temporales utilizando una arquitectura procesador multinúcleo combinada con múltiples GPUs. La implementación propuesta es capaz de aprender de forma paralela distintas redes neuronales sobre múltiples fuentes de datos.

Nuestra principal aportación con este trabajo es la modificación y aceleración del algoritmo original de la GNG utilizando procesadores multinúcleo y múltiples GPU.

A través de la implementación de la aplicación de procesamiento de diferentes flujos de imágenes de forma paralela ha quedado demostrada la capacidad del sistema para aprender la topología de las imágenes bajo restricciones temporales y adaptarse ante cambios en el espacio de entrada también bajo restricciones temporales. Como se demuestra en los experimentos de la implementación GPU el tiempo de ejecución de la versión secuencial de la GNG crece de forma proporcional al número de neuronas que conforman la red, mientras que la versión paralelizada incrementa el número de neuronas y su aprendizaje sin degradarse el rendimiento significativamente. De esta forma se obtiene una aceleración considerable respecto a la versión secuencial.

En los experimentos de la implementación utilizando múltiples GPUs se demuestra como la versión paralela puede manejar hasta 16 flujos de datos, de forma que cada red neuronal que aprende la topología de cada una de las imágenes es capaz de realizar hasta 3 ajustes por segundo, mientras que la versión CPU solo es capaz de manejar un único flujo de información bajo las mismas restricciones temporales.

Como trabajos futuros proponemos la utilización de algoritmos acelerados de generación de patrones aleatorios en la GPU, así como la utilización de la GPU para acelerar otras etapas de nuestra aplicación.

AGRADECIMIENTOS

El presente trabajo ha sido financiado parcialmente mediante el proyecto GRE09-16 de la Universidad de Alicante y el proyecto GV/2011/034 concedido por la Generalitat Valenciana. Los experimentos se han podido llevar a cabo gracias a la generosa donación de hardware por parte de la empresa de GPUs NVIDIA.

REFERENCIAS

[1] J. Moorkanikara Nageswaran, N. Dutt, J.L Krichmar, A. Nicolau, A. Veidenbaum. Efficient Simulation of Large-Scale Spiking Neural Networks Using CUDA Graphics Processors. In proc. IJCNN 2009.

[2] F. Nasse, C. Thurau, and G.A. Fink. Face Detection Using GPU-Based Convolutional Neural Networks. CAIP 2009, LNCS 5702, pp. 83–90, 2009.

[3] R. Uetz and S. Behnke. Large-scale Object Recognition with CUDA-accelerated Hierarchical Neural Networks. In Proceedings of the 1st IEEE International Conference on Intelligent Computing and Intelligent Systems 2009

[4] S. Che, M. Boyer, J. Meng, D. Tarjan, J. W. Sheaffer, K. Skadron , A Performance Study of General-Purpose Applications on Graphics Processors Using CUDA. Journal of Parallel and Distributed Computing, Vol. 68(10), 2008.

[5] H Jang, A. Park, K. Jung, "Neural Network Implementation Using CUDA and OpenMP," dicta, pp.155-161, 2008 Digital Image Computing: Techniques and Applications, 2008.

[6] J. Kim, M. Hwangbo, T. Kanade. Realtime Affine-photometric KLT Feature Tracker on GPU in CUDA Framework.Workshop on Embedded Computer Vision, ICCV 2009.

[7] S. Oh, K. Jung. View-Point Insensitive Human Pose Recognition using Neural Network and CUDA. World Academy of Science, Engineering and Technology 60, 2009.

[8] M. Schwarz and M. Stamminger. Fast GPU-based Adaptive Tessellation with CUDA, EUROGRPHICS Vol. 28(2), 2009.

[9] V. Simek and R. Asn. GPU Acceleration of 2D-DWT Image Compression in MATLAB with CUDA. In Computer Modeling and Simulation, 2008. EMS'08. Second UKSIM European Symposium on, pages 274–277, 2008.

[10] S.S. Stone, J.P. Haldar, S.C. Tsao, W. W. Hwu, Z. Liang, B.P. Sutton. Accelerating Advanced MRI Reconstructions on GPUs. in Proceedings of the 5th International Conference on Computing Frontiers, 2008

[11] J. Garcia-Rodríguez, A. Angelopoulou, V. Morell, S. Orts, A. Psarrou, J. M. Garcia-Chamizo, Fast image representation with gpu-based growing neural gas, in: IWANN (2), 2011, pp. 58–65.

[12] B. Fritzke, A Growing Neural Gas Network Learns Topologies, In Advances in Neural Information Processing Systems 7, (1995) G. Tesauro, D.S. Touretzky y T.K. Leen (eds.), MIT Press, Cambridge, Mass.

[13] W.-m. W. Hwu, GPU Computing Gems Emerald Edition, 1st Edition, Morgan Kaufmann Publishers Inc., San Francisco, CA, USA, 2011.

[14] J. Garcia-Rodriguez, A. Angelopoulou, J. Garcia-Chamizo, A. Psarrou, S. Orts-Escolano, V. Morell-Gimenez, Fast autonomous growing neural gas, in: Neural Networks (IJCNN), The 2011 International Joint Conference on, 2011, pp. 725 –732.

[15] J. Nickolls, W. J. Dally, The gpu computing era, IEEE Micro 30 (2010) 56– 69.

[16] N. Satish, M. Harris, M. Garland, Designing efficient sorting algorithms for manycore gpus, in: Proceedings of the 2009 IEEE International Symposium on Parallel&Distributed Processing, IPDPS '09, 2009, pp. 1–10.

[17] CUDA Programming Guide, version 4.0, (2011). http://developer.download.nvidia.com/compute/DevZone/docs/html/OpenCL/doc/OpenCL_Programming_Guide.pdf

[18] D. B. Kirk, W.-m. W. Hwu, Programming Massively Parallel Processors: A Hands-on Approach, 1st Edition, Morgan Kaufmann Publishers Inc., San Francisco, CA, USA, 2010.

[19] H. Jang, A. Park, K. Jung, Neural network implementation using cuda and openmp, in: Proc. DICTA '08.Digital Image Computing: Techniques and Applications, 2008, pp. 155–161.

[20] K. Oh, GPU implementation of neural networks, Pattern Recognition 37 (6) (2004) 1311–1314.

[21] J. M. Nageswaran, N. Dutt, J. L. Krichmar, A. Nicolau, A. Veidenbaum, Efficient simulation of arge-scale spiking neural networks using cuda graphics processors, in: Proc. Int. Joint Conf. Neural Networks IJCNN 2009, 2009, pp. 2145–2152.

[22] C.-F. Juang, T.-C. Chen, W.-Y. Cheng, Speedup of implementing fuzzy neural networks with high-dimensional inputs through parallel processing on graphic processing units, IEEE T. Fuzzy Systems 19 (4) (2011) 717–728.

[23] J. G. Rodríguez, A. Angelopoulou, V. Morell, S. Orts, A. Psarrou, J. M. G. Chamizo, Fast image representation with gpu-based growing neural gas, in: IWANN (2), 2011, pp. 58–65.

[24] J. Igarashi, O. Shouno, T. Fukai, H. Tsujino, 2011 special issue: Real-time simulation of a spiking neural network model of the basal ganglia circuitry using general purpose computing on graphics processing units, Neural Netw. 24 (2011) 950–960.

[25] R. Uetz, S. Behnke, Large-scale object recognition with cuda-accelerated hierarchical neural networks, in: Proc. IEEE Int. Conf. Intelligent Computing and Intelligent Systems ICIS 2009, Vol. 1, 2009, pp. 536–541.

[26] F. Nasse, C. Thurau, G. Fink, Face detection using gpu-based convolutional neural networks, in: X. Jiang, N. Petkov (Eds.), Computer Analysis of Images and Patterns, Vol. 5702 of LNCS, Springer, 2009, pp. 83–90.

[27] S. Oh, K. Jung, View-point insensitive human pose recognition using neural network and cuda, World Academy of Science 60.

[28] T. M. Martinetz, S. G. Berkovich, K. J. Schulten, 'neural-gas' network for vector quantization and its application to time-series prediction 4 (4) (1993) 558–569.

[29] B. Fritzke, Growing cell structures - a self-organizing network for unsupervised and supervised learning, Neural Networks 7 (1993) 1441–1460.

[30] T. Martinetz, Competitive Hebbian Learning Rule Forms Perfectly Topology Preserving Maps, in: S. Gielen, B. Kappen (Eds.), Proc. ICANN'93, Int. Conf. on Artificial Neural Networks, Springer, London, UK, 1993, pp. 427– 434.

[31] M. Harris, Optimizing parallel reduction in cuda, NVIDIA Dev. Technology.