Aceleración de la velocidad de procesamiento a través de la paralelización de algoritmos

Israel Tabarez Paz[1], Neil Hernández Gress[2], Miguel González Mendoza[2]

[1] Universidad Autónoma del Estado de México,
Atizapán de Zaragoza, México

Universidad Politécnica del Valle de Toluca,
Estado de México, México

[2] Tecnológico de Monterrey, Campus Estado de México,
Atizapán de Zaragora, México

itabarezp@uaemex.mx, {ngress, mgonza}@itesm.mx

Resumen. En este artículo se analizarán los aspectos más importantes para la paralelización de algoritmos. Por lo que nuestro interés sobre el tema es debido a sus múltiples campos de aplicación con el propósito principal es acelerar la velocidad de procesamiento para la solución de problemas tengan un alto costo computacional. Su principal funcionalidad es reducir el tiempo de ejecución del programa a traés de la aplicación de técnicas, y bajo ciertas condiciones. Para llevar a cabo esta tarea es necesario diseñar un algoritmo para su programación y elegir una infraestructura de computación paralela. Una de las aplicaciones de los algoritmos paralelos es la optimización de las operaciones matriciales y vectoriales para clasificacin de grandes bases de datos.

Palabras clave: GPU, CUDA, algoritmos paralelos, velocidad de procesamiento, redes neuronales artificiales; máquinas de soporte vectorial.

1. Introducción

El presente trabajo está enfocado al análisis del desempeño de la paralelización de algoritmos para la aceleración de la velocidad de procesamiento, utilizando un CPU Intel Core 2 Duo, de 2.66GHz. La tarjeta gráfica aplicada es una NVIDIA GeForce 9400M. Sus principales problemáticas son: que tan paralelizables son los algoritmos, el tipo de arquitecturas que se pueden utilizar, las ventajas y desventajas de la aceleración del procesamiento. Para analizar estas problemáticas es necesaria la búsqueda o el estudio de diferentes alternativas ya sean métodos, herramientas y aplicaciones que puedan ayudar a resolverlas. El interés de esta investigación es conocer las ventajas y desventajas de la paralelización para sistemas distribuidos con la mayor precisin posible. Este artículo está distribuido en las siguientes secciones: estado del arte en la sección 2, experimentación en la sección 3 y conclusiones en la sección 4.

2. Estado del arte

El paralelismo es una forma en la cual pueden realizarse varios cálculos simultáneamente. Está basado el en principio de dividir problemas grandes para obtener problemas pequeños los cuales posteriormente son solucionados en paralelo.

La programación paralela permite:

- Resolver problemas con gran cantidad de datos.
- Reducción del tiempo de solución.

Pereira [1], describe el paralelismo sobre conjuntos de datos métricos y un algoritmo de búsqueda por similitud basado en una estructura de indexación, utilizando el modelo de filtros. Dicho algoritmo mostró un desempenño eficiente en la praáctica respecto del número de procesadores disponibles, en redes de tamano moderado.

Aracena [3], presenta una optimización del método de detección de puntos SIFT (Scale Invariant Feature Transform), mediante su paralelización empleando una GPU (Unidad de Procesamiento Gráfico). El objetivo es acelerar el tiempo de cómputo, para la detección de puntos característicos. Asimismo, el autor indica que se basa en dos premisas: el balance carga y la distribución de cálculo. Respecto a las pruebas se realizaron en con procesador Core 2 Duo 2.2Ghz, 3GB de RAM y una VGA GeForce 8600GT (32 núcleos) de 512MB. Respecto a los resultados nos indica que se logró un rendimiento de 42.5 milisegundos en promedio. Las pruebas fueron realizadas en diferentes resoluciones e indica que la paralelización de SIFT no muestra pérdidas significativas de puntos claves en comparación con la versión secuencial.

Armenteros [4], estudia diferentes alternativas para la aceleración y muestra su efecto en el análisis de metaheurísticas paralelas, asimismo, ofrece una guía de cómo y cuándo utilizarlas. Finalmente, concluye que el mejor escenario para la aceleración es definir la calidad de las soluciones como condición de parada, porque todas las ejecuciones alcanzan el óptimo y permite una comparación justa entre los métodos.

Uribe [5], presenta la implementación de una estructura genérica adaptada para un sistema multiprocesador con una GPU, mostrando los resultados experimentales en trminos de tiempo y speed-up (aceleración). También, muestra experimentalmente las ventajas comparativas al insertar GPU a una plataforma multicore, así como el análisis del consumo energético.

Hidrobo [6], presenta un Sistema Manejador de Ambientes Reconfigurables para procesamiento paralelo/Distribuido. Asimismo, presenta dos enfoques, en el primero el programa se adapta al sistema y en el segundo sigue el procedimiento inverso. El autor se basa en la teoría de grafos para resolver problemas NP completos. También utiliza Algoritmos Genéticos técnica para encontrar una solución.

Dally [8], implementa la paralelización de redes de comunicación VLSI (integración a muy larga escala) analizando los costos para la realización lo cual

está en función de la arquitectura de la red. Asimismo, analiza el rendimiento de la red.

Duato [9], trabaja sobre la reconfiguración dinámica de la interconexión de la red para reducir la sobrecarga de comunicación. El trabajo se centra en la reconfiguración dinámica, es decir, la topología de red para cambiarla arbitrariamente en tiempo de ejecución. En este trabajo, se exponen en profundidad el algoritmo de reconfiguración y los diferentes conceptos relacionados con ella.

Flynn [10], presenta un modelo jerárquico de las organizaciones de informática en base a un modelo de árbol utilizando recursos de tipo petición / servicio como nodos. Se distinguen dos aspectos del modelo : lógica y física . Las organizaciones generales paralelas y múltiple son examinadas en cuanto a tipo y efectividad. El proceso simplex superpuesto (SISD) está limitado por las dependencias de los datos . La ramificación tiene un efecto particularmente degenerativa. Se analizan los procesadores paralelos [stream –data – stream mltiple – una sola instrucción (SIMD)]. Asimismo, se analiza el rendimiento de un procesador en paralelo, el cual aumenta logarítmicamente. También, los Multiprocesadores (MIMD) son sometidos a saturación basada en el bloqueo general de comunicaciones. De acuerdo con este autor, los modelos de colas simplificado indican que la saturación se desarrolla cuando la fracción de tiempo de la tarea (L/E) se aproxima a 1/n, donde n es el número de procesadores . Al compartir recursos en multiprocesadores se pueden utilizar para evitar otros problemas organizativos clásicos.

Tabla 1. Taxonomía Flynn.

	SIMPLE INTRUCCIÓN	MÚLTIPLES INSTRUCCÍONES
SIMPLES DATOS	SISD	SIMD
MÚLTIPLES DATOS	MISD	MIMD

De acuerdo con de acuerdo con Nikola [12], la taxonomía de paralelización se aproxima a la neurosimulaciones representadas en la figura 1.

Finalmente, es importante resaltar que no se hace un análisis de efectividad en function a la longitud de los datos, tampoco se indica el grado de paralelizacin de los algoritmos. Esto fue una motivación para realizar este trabajo.

2.1. Programación paralela

La programación paralela es el uso de varios procesadores trabajando simultáneamente juntos para resolver una tarea en común.

Existen diferentes formas [7]: nivel bit (microcontrolador), nivel instrucción o dato (GPU), y nivel tarea (como MPI). En la figura 2 podemos ver el principio del cómputo paralelo. Los diversos problemas pueden ser resueltos en lenguajes CUDA, OpenGL, Cg, C, C++ and FORTRAN.

Los principales aspectos que se consideran en la programación paralela son:

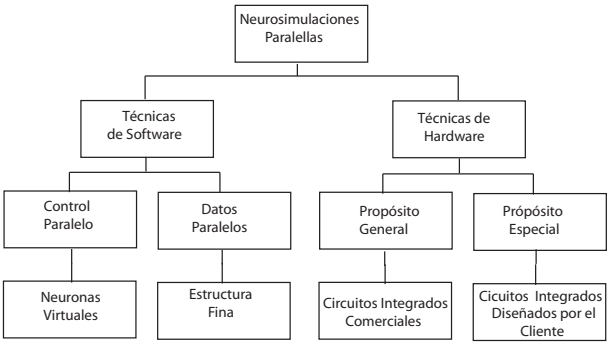

Fig. 1. Taxonomía de paralelización para neurosimulaciones.

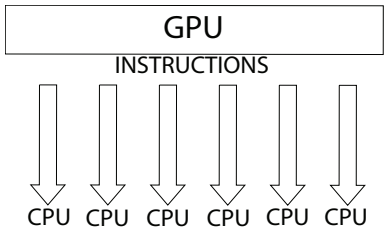

Fig. 2. Ejecución de instrucciones en paralelo.

- Arquitectura.
- Diseño del algoritmos.
- Evaluación de los algoritmos paralelos.

En la actualidad existen algunas opciones de hardware como FPGAs (Field Programmable Gate Array), GPUs (Graphics Processing Unit), a través del diseño de circuitos lógicos y secuenciales, microcontroladores, DSPs (Procesador Digital de Señales), PLCs (Controladores Lógicos Programables). En donde la mayoría de ellos trabajan el procesamiento paralelo a nivel bit, sin embargo los GPUs trabajan dicho procesamiento a nivel instrucción, en donde los diferentes núcleos ejecutan la misma instrucción al mismo tiempo, pero con diferente dato de entrada.

Respecto al diseño de algoritmos, es muy importante considerar que los algoritmos tienen grados de paralelización, lo cual está en función de la cantidad de instrucciones paralelizables. Finalmente, es necesario validar los resultados contrastando con los resultados esperados. En este trabajo, agregamos que la ecuación (1) nos permite calcular el porcentaje de paralelización de un algoritmo.

$$\%\ de\ Paralelizacion\ =\ \frac{Operaciones\ paralelizadas}{Total\ de\ operaciones\ seriales} \times 100 \qquad (1)$$

Aplicaciones de la programación paralela En el diseño de los algoritmos paralelos es importante el enfoque para solucionar problemas grandes para campos de aplicación. Por esta razón se determina que las etapas de diseño de algoritmos paralelos son:

- Particionamiento: Dividir en subproblemas.
- Comunicación: Sincronización de Tareas.
- Agrupación: Avaluación de la eficiencia y costos.
- Asignación de Tareas. Maximizar el uso de los recursos de los procesadores paralelos.

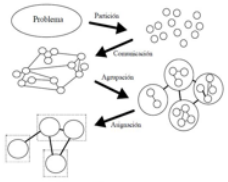

Fig. 3. Etapas de diseño de algoritmos paralelos.

En la figura (3) se ilustra cada etapa del diseño para la paralelización del algoritmo. Por otro lado, existe una gran cantidad de aplicaciones de paralelización de algoritmos como clasificación de grandes bases de datos a través de redes neuronales artificiales (RNA), lo cual nos permite reducir en gran medida la velocidad de procesamiento en un algoritmo que por naturaleza es secuencial y que consume gran cantidad de tiempo de procesamiento, pero con grandes oportunidades de paralelización.

Otras aplicaciones son la predicción a través de redes neuronales artificiales; implementación de algoritmos para tratamiento de imágenes; circuitos de sistemas de control con microcontroladores y procesadores digitales de señal; asimismo en la simulación de imágenes tridimensionales para diseño en ingeniería y simulación de fenómenos físicos, económicos, financieros y sociales, pero sobre todo de videojuegos, en donde se obtiene una gran capacidad de resolución de imágenes y precisión de la dinámica del movimiento, aunado a la gran velocidad de procesamiento. Esto nos permite tener una poderosa herramienta para el desarrollo de la investigación científica para el clculo de arreglos numéricos con dimensiones muy grandes.

Finalmente hay una gran cantidad de aplicaciones sin embargo, la programación paralela es diferente para cada arquitectura. En este caso analizaremos un caso de estudio aplicado a clasificación de bases de datos a través de redes neuronales artificiales de tercera generacin como son las Máquinas de Soporte Vectorial (SVM).

3. Máquinas de soporte vectorial (SVM)

En esta sección se presenta un caso aplicando paralelización de algoritmos que consiste en la simplificación del tiempo de ejecución de las operaciones matriciales y vectoriales del algoritmo SVM QP (Programación Cuadrática para las Máquinas de Soporte Vectorial). La ejecución se realizó sobre la plataforma de Mac OS X Lion 10.7.5, utilizando la tarjeta NVIDIA GeForce 9400M.

El GPU fue configurado con más de un bloque para cubrir todos los elementos de la base de datos. Lo que implicó la reducción de la paralelización a nivel de operaciones vectorial, además, de considerar todos los bloques posibles para las bases de datos con máximo 1400 instancias aproximadamente, y la paralelización óptima para 16 datos, ya que es el tamaño máximo en hilos de un bloque. Por otro lado, a menor paralelismo del código implicó aumentar la cantidad de funciones kernel adems de combinar código serial con paralelo, por lo que se requirió copiar del GPU al CPU varias veces y viceversa, debido a la naturaleza del código. Esto afectó de manera importante a la metodología SVM. En este caso para más de 1400 instancias se consideró a la metodología SVM light. El modelo matemático de SVM QP está descrito como:

$$\begin{gathered} \text{máximo } \alpha \\ q(\boldsymbol{\alpha}) = 0.5\boldsymbol{\alpha^T} Q\boldsymbol{\alpha} - 1^T\alpha \qquad (2) \\ \text{sujeto a} \\ \boldsymbol{y}^T\boldsymbol{\alpha} = 0 \\ 0 \leq \boldsymbol{\alpha} \leq \boldsymbol{C} \end{gathered}$$

La matriz $\boldsymbol{Q}$ está compuesta por $(\boldsymbol{Q})_{ij} = y_i y_j k(x_i, x_j)$, donde:
$i, j = 1, 2, 3, ..., l,\ \boldsymbol{\alpha} = [a^1 ... a^l]^T$
$\mathbf{1} = [1^1 ... 1^l]^T\ \boldsymbol{y} = [y^1 ... y^l]^T$
$\boldsymbol{C} = [C^1 ... C^l]^T$
Finalmente, la salida es
$y = sign(\sum_{i=1}^{N} y_i \alpha_i K(\boldsymbol{x}, \boldsymbol{x_i}))$.
La figura 4 representa la arquitectura de SVM.

En este caso, fue paralelizada la multiplicación matricial para $((Q)_{ij} = y_i y_j k(x_i, x_j))$ en el método QP. El número de hilos fue generalizado como sigue:

$$Hilos\ por\ bloque = \frac{Cantidad\ de\ Datos}{Hilos\ por\ bloque} + 1 \qquad (3)$$

Una desventaja es que algunas veces las tarjetas gráficas tienen un número de hilos por bloque muy limitado, por lo que esa es la longitud máxima de un vector o número de datos copiados del CPU al GPU.

Es así que los resultados de SVM se muestran en la tabla 2. Para los casos mayores a 1400 instancias, se aplicó SVM light, y en los dems casos SVM QP, ya que una los recursos computacionales del GPU no fueron suficientes para hacer los cálculos correspondientes. Finalmente, en dicha tabla observamos los

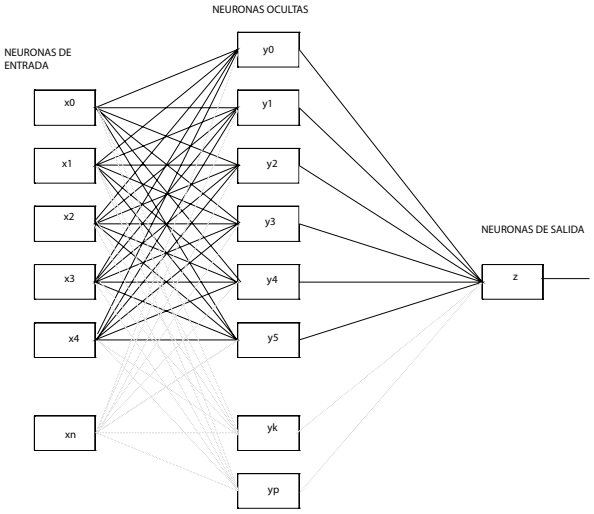

Fig. 4. Arquitectura de SVM.

resultados de paralelizar esta red neuronal SVM, en donde la clasificacin es binaria, como en el caso de la base de datos Cars, la cual consiste de 4 clases, cada una de ellas es clasificada de manera independiente.

Tabla 2. Paralelización del algoritmo SVM.

Datos	Instancias	Atributos	Clases	Iteraciones	Exitos	Tiempo de Aprendizale [ms]
Iris	150	4	3	2758	145	1521128.5
Cars (SVM light)	1728	6	4	———	1728 (Clase 1) 56 (Clase 2) 4 (Clase 3) 34 (Clase 4)	608927.125 1478488.5 1090742.125 970055.375
Breast Cancer Wisconsin	699	9	2	———	641 (Clase 1) 127 (Clase 2)	163790.55625 162216.453125
Adult	32561	14	2	———	———	———
Heart Disease Cleveland (HDC)	303	13	5	———	147 (Clase 1) 3 (Clase 2) 66 (Clase 3) 25 (Clase 4) 2 (Clase 5)	56749.73046 21624.41406 22715.71875 27424.23046 98908.94531

La cantidad mínima de instancias es 150 (Iris), y la cantidad mxima es 1728 (Cars). Respecto al tiempo, el mínimo fue registrado para Heart Disease Cleveland (HDC) para 303 instancias y 5 clases y el máximo para Cars con 4 clases diferentes.

Tabla 3. Resultados en Matlab.

Datos	Instancias	Clase	Tiempo de Aprendizale [s]
Iris	150	clase 1	0.0310
	50	clase 2	0.0000 (Total: 0.0310)
Cars	1728	clase 1	32.642
	1343	clase 2	18.391
	1275	clase 3	17.625 (Total: 68.658)
Breast Cancer Wisconsin	699	Calse 1	3.906
Hepatitis	155	Calse 1	1.516

En la tabla 3 se muestran los resultados aplicando MatLab, en donde observamos que el mínimo tiempo de procesamiento es de 0.0310 s, y el máximo es de 68.658 s. Comparando con los resultados obtenidos en la Tabla 2, el tiempo de procesamiento es menor en el CPU ya que el alto grado de paralelización del algoritmo y la gran cantidad y de parámetros y constantes tuvo como consecuencia realizar varias veces la operación de copiar del GPU al CPU y viceversa lo cual consume gran cantidad de tiempo.

4. Conclusiones y trabajo a futuro

En la paralelización del caso presentado, se siguieron los pasos para lograr paralelizar de manera óptima el algoritmo, sin embargo se concluye que el nivel de paralelización depende en gran medida de la arquitectura del GPU. En este caso se tuvo que reducir el grado de paralelización, es decir, la cantidad de instrucciones a paralelizar debido a la gran limitación de memoria y de ncleos del GPU disponibles. Por otro lado, es indispensable diseñar óptimamente el algoritmo paralelizado de acuerdo con las características técnicas del GPU. Como trabajo a futuro se plantea la posibilidad de experimentar con otras arquitecturas más potentes y con mayores recursos. Asimismo, es necesario calcular con mayor precisión el grado de paralelización.

Referencias

1. Quintão Pereira, F. Magno, Beatriz Perez, N., Marcelo Berón, M.: Algoritmo paralelo basado en el modelo filter - stream y la infraestructura watershed para búsquedas por similitud en espacios métricos.

2. García Alonso J. M., Berrocal Olmeda J. J., Murillo Rodríguez J. M.: ParallelJ: Entorno de desarrollo y simulación de programas.
3. Aracena Pizarro D., Danerí Alvarado N.: Detección de puntos clave mediante SIFT paralelizado en GPU. Revista chilena de ingeniería, 21(3), pp. 438–447 (2013)
4. García Armenteros A., Luque Polo G., Alba Torres E.: Influencia de Métricas Paralelas en el Análisis de Algoritmos Paralelos Metaheurísticos. Serie Científica de la Universidad de las Ciencias Informáticas, 4(4) (2011)
5. Uribe Paredes R., Cazorla D., Arias E., Sánchez J.L.: Un sistema heterogneo Multicore/GPU para acelerar la búsqueda por similitud en estructuras métricas. Revista chilena de ingeniería, 22(1), pp. 26–40 (2014)
6. Hidrobo F. J., Aguilar J.L.: Sistema Manejador de Ambientes Reconfigurables para procesamiento paralelo/Distribuido. Computación y Sistemas, 3(3), pp. 169–181 (2000)
7. NVIDIA CUDA C BEST PRACTICES GUIDE DG –05603–001 v5.0 (May 2012)
8. Dally, W.J.: Análisis de la ejecución de k-Aria n-Cube redes de interconexión. IEEE Transactions on Computadoras, 39(6), pp. 775–785 (1990)
9. Garcia, J.M. Duato, J.: Dynamic reconfiguration of multicomputer networks: limitations and tradeoffs. In: Proceedings of Euromicro Workshop on Parallel and Distributed Processing, pp. 317–323 (1993)
10. Flynn, M.: Some Computer Organizations and Their Effectiveness. IEEE Transactions on Computers, vol.C-21(9), pp. 948–960 (1972)
11. Bauch, A., Braam, R., Maehle, E.: DAMP - Un sistema multiprocesador reconfigurable dinámica con una red de conmutación distribuida. Spriger, vol. 487, pp. 495–504 (1991)
12. Serbediija, N.B.: Simulating Artificial Neural Networks on Parallel Architectures (1996)