Análisis del desempeño de los algoritmos Redes Neuronales de Impulso (SNN) y Máquinas de Soporte Vectorial (SVM) en programación paralela

Israel Tabarez Paz, Neil Hernández Gress y Miguel González Mendoza

Universidad Autónoma del Estado de México
Atizapán de Zaragoza, México
{itabarezp}@uaemex.mx
http://www.uaem.mx/cuyuaps/vallemexico.html
Tecnológico de Monterrey, Campus Estado de México,
Atizapán de Zaragora, México
{ngress,mgonza}@itesm.mx
http://www.itesm.edu

Resumen En este artículo se analizará el desempeño y la velocidad de entrenamiento de los algoritmos Redes Neuronales de Impulso (SNN) y Máquinas de Soporte Vectorial (SVM). La programación de SNN y SVM se realizan en arquitectura GPU NVIDIA modelo GeForce 9400M. Esto es aplicado a la clasificación de bases de datos. Acerca de la literatura, ningún autor ha codificado SVM QP (Programación Cuadrática) en un GPU, sin embargo existen otros métodos de optimización de SVM utilizando un GPU, tales como SVM light y SMO. En el caso de SNN, éste modelo se ha desarrollado en MatLab, FPGAs o circuitos secuenciales, pero ningún autor lo ha codificado en un GPU. Los resultados nos muestran que no en todos los casos es mejor en un GPU que en CPU, lo cual depende de la complejidad algorítmica y la cantidad de recursos computacionales requeridos. Desafortunadamente, en la actualidad la mayoría del hardware específico para programación paralela tiene memoria muy limitada para grandes bases de datos. Por esta razón, el programador tiene que reducir el grado de paralelización.

Palabras clave: GPU, FPGA, redes neuronales rtificiales, redes neuronales de impulso, máquinas de soporte vectorial.

1. Introducción

Este artículo presenta una investigación acerca del análisis del desempeño de las SVMs en comparando con las SNN para la solución de problemas como clasificación (Herrero Lopez [8], reconociemiento de patrones [29], regresión (Carpenter [19], y construcción de Redes Neuronales Artificiales en un hardware específico, como las FPGAs (Papadonikolakis [7]. Respecto a la programación

paralela [39], existen diferentes formas: nivel bit (microcontrolador), nivel instrucción o dato (GPU), y nivel tarea (como MPI). En la figura 1 podemos ver el principio del cómputo paralelo. Los diversos problemas pueden ser resueltos en lenguajes CUDA, OpenGL, Cg, C, C++ and FORTRAN.

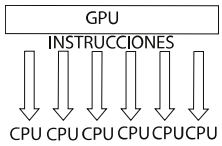

Figura 1. Ejecución de instrucciones en paralelo.

Este artículo está distribuido en las siguientes secciones: la sección 2 contiene estado del arte, la sección 3 consiste en la fase experimentación para comparación de estas metodologías y conclusiones en la sección 4.

2. Estado del arte

Sridhar [1], propone la simulación térmica de circuitos integrados a través de redes neuronales artificiales (RNA) sobre una GPU. El tiempo de aprendizaje es comparado entre una CPU y una GPU para esta aplicación. Asimismo, este autor señala que la principal ventaja de los GPU sobre los CPU es el alto paralelismo y eficiencia con un bajo costo. Sin embargo, nosotros probaremos que sobre las redes neuronales SVM y SNN, el costo computacional es alto y la eficiencia se reduce debido a las pérdida de precisión. También, que aunque existen circuitos integrados para alto paralelismo, es muy difícil trasladar este paralelismo a un software más eficiente.

Prabhu [21], compara la eficiencia del cerebro humano con el enorme poder computacional de los GPU's, aunque éste tiene limitaciones. El autor comenta que la principal función de los GPU's está en aprovechar las RNA. También los GPU's han sido usados para renderizar imgenes de alta calidad en tiempo real, realidad virtual y video juegos. Los GPU's modernos pueden ejecutar tareas en paralelo en grandes cantidades.

De acuerdo con Sergio Herrero et al. [8] el tiempo de entrenamiento de una tarea binaria compuesta de 100,000 puntos con cientos de dimensiones puede tardar horas para la ejecución serial. Sin embargo, es decir, el GPU es más lento para alguna ejecución serial que el CPU; y el GPU es más rápido para ejecuciones en paralelo que los CPU's.

Finalmente, Benjamin Schrauwen [25] propone a contribuye a mejorar el algoritmo spikeprop ajustando los parámetros umbral, retardo y derivada del tiempo. Esto puede ayudarnos a seleccionar las constantes óptimas para nuestros experimentos.

2.1. Programación paralela de las RNA

La taxonomía de paralelización se aproxima a la neurosimulaciones representadas en la figura 2, esto es de acuerdo con Nikola [35].

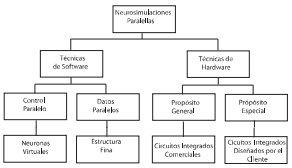

Figura 2. Taxonomía de paralelización para neurosimulaciones.

Las RNA (Redes Neuronales Artificiales) pueden ser paralelizadas en hardware específico como GPU's, FPGA como Thomas [18], circuitos secuenciales, o tarjetas electrónicas específicas [14]. Sin embargo, nos hemos enfocado al modelo GeForce 9400M. En caso de los SVM's, pueden ser algoritmos diseñados tanto en GPU's o CPU's debido al método de optimización requerido para resolver operaciones de matrices repetitivamente. Sin embargo, en el caso de SNN, es una metodología que está orientada más para resolverse en FPGA o circuitos secuenciales, aunque puede ser simulado en un GPU, esto porque es uestá enfocada más a tiempo real; por otro lado su función de activación se puede resolver con aproximaciones matemáticas, de tal manera que si el número de neuronas en la capa de entrada aumenta, entonces también aumenta la complejidad computacional. A pesar de esto, si aplicamos circuitos secuenciales, el umbral puede ser detectado con un detector de nivel de voltaje, que es un circuito muy simple de resolver.

Redes neuronales de impulso (SNN) La arquitectura de las SNN consiste en un conjunto de nodos o neuronas que contienen múltiples retardos como se observa en la figura 3 (Olaf [29]).

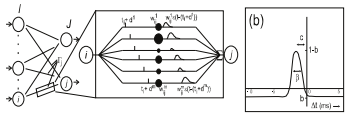

Figura 3. Conectividad de las SNN.

Formalmente, una neurona j, que pertenece a un conjunto Γ_j ('neurona pre–synaptica') Olaf [29], recibe un conjunto de impulsos que son disparados en el tiempo, es decir, $t_i, i\epsilon\Gamma_j$ cuando la terminal alcanza un umbral θ. La dinámica de la variable interna $x_j(t)$ está determinada por la function de respuesta del impulso $\varepsilon(t)$ donde w_{ij} son los pesos de la conxión de la i a la j:

$$x_j(t) = \sum_{i\epsilon\Gamma_j} w_{ij}\varepsilon(t - t_i) \tag{1}$$

El potencial sináptico está definido por la ecuación (2):

$$x_j(t) = \varepsilon(t) = \frac{t}{\tau}e^{1-\frac{t}{\tau}} \tag{2}$$

El retardo d_k de la terminal k está definido por la diferencia entre el tiempo de disparo de la neurona presináptica y el tiempo de subida de la neurona post sináptica. La constante τ es una constante de tiempo que controla el tiempo de subida y el tiempo de caída. Así, múltiples sinapsis por conexión de la neurona que recibe j están descritas por la siguiente ecuación (3):

$$x_j(t) = \sum_{i\epsilon\Gamma_j}\sum_{k=1}^{m} w_{ij}^k y^k(t) \tag{3}$$

Para una terminal de salida, se describe la siguiente ecuación:

$$y_j^k(t) = \varepsilon(t - t_i - d^k) \tag{4}$$

El Modelo de Respuesta Simple asegura la causalidad, por lo que $\eta(s) = 0$ para $s \leq 0$ y usualmente no positivo, se describe por la ecuación (5):

$$\eta(s) = -\vartheta e^{\frac{s}{\tau_r}} H(s) \tag{5}$$

Donde θ es el umbral de la neurona, $H(s)$ es la función escalón unitario τ_r es otra constante de tiempo.

Por lo tanto el potencial de una neurona $u_j(t)$ es:

$$u_j(t) = \sum_{t_j^{(f)}\epsilon F_j} \eta(t - t_j^{(f)}) + \sum_{i\epsilon\Gamma_j}\sum_{t_i^{(g)}} w_{ij}\varepsilon(t - t_i^{(g)} - d_{ij}) \tag{6}$$

Donde

$$F_j = \{t^{(f)}; 1 \leq f \leq n\} = \{t|u_j(t) = \vartheta\} \tag{7}$$

Y, n denota el número de impulsos.

Donde t tiene que ser definido en el GPU como un intervalo de tiempo. En este caso lo bloques contienen 255 hilos, pero en otros modelos de GPU's NVIDIA este valor es de 512 o 1024, lo que depende de las especificaciones técnicas. En nuestro caso disponemos de 60000 hilos distribuidos en 235 bloques con 255 hilos cada uno, aproximadamente. El paralelismo en SNN es implementado en bloques configurados en 3D como sigue:

$$threadx = \frac{SIZE}{NN[num\_cap] * NN[num\_cap + 1]} \tag{8}$$

Donde $threadx$ es el tiempo t; $SIZE$ es el número de hilos por bloque; $NN[num\_cap]$ es el número de neuronas en la capa i; $NN[num\_cap + 1]$ es el número de neuronas en la siguiente capa j. Por otro lado, podemos observar que los retardos no se toman en cuenta al paralelizar los bloques, ya que éstos son aplicados de manera secuencial debido a las limitaciones de memoria; de la tarjeta NVIDIA. Asimismo, si el tiempo excede el número de hilos por bloque entonces necesitamos agregar más bloques, sin embargo las áreas entre bloques son codificadas secuencialmente. Esto consume mucho tiempo de ejecución, el cual afecta al tiempo de aprendizaje de la red neuronal. Esta configuración se puede representar por un cubo como el de la figura 4. Los ejes horizontales representan el peso de los retardos y los valores de la exponencial de tiempo, respectivamente; y el eje vertical representa el número de neuronas de la capa anterior.

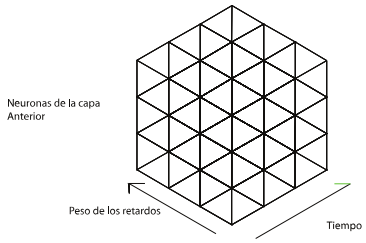

Figura 4. Paralelización de la SNN (3D).

Máquinas de soporte vectorial (SVM) La paralelización de fase de aprendizaje en el caso de SVM QP consiste en la simplificación del tiempo de ejecución de las operaciones matriciales y vectoriales que implica el mismo algoritmo. Estas operaciones ya están resueltas en la página de NVIDIA, en la literatura misma [39], aunque esto depende del método de optimización aplicado, que en nuestro caso es QP (Programación cuadrática) [40]. Debido a la configuración de los bloques e hilos del GPU, esto nos da capacidad computacional para resolver matrices cuadráticas de 16, 22 o 32 datos de entrada respectivamente. Como resultado de esto, el GPU fue configurado en la mayoría de los casos 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 tener más de un kernel combinando código serial con

paralelo y copiando del GPU al CPU varias veces y viceversa, por 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 se muestra a continuación:

$$\text{máximo } \alpha$$

$$q(\boldsymbol{\alpha}) = 0{,}5\boldsymbol{\alpha}^{\boldsymbol{T}} Q\boldsymbol{\alpha} - 1^T\alpha \tag{9}$$

$$\text{sujeto a}$$
$$\boldsymbol{y}^T\boldsymbol{\alpha} = 0$$
$$0 \leq \boldsymbol{\alpha} \leq \boldsymbol{C}$$

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 $y = sign(\sum_{i=1}^{N} y_i \alpha_i K(\boldsymbol{x}, \boldsymbol{x_i}))$. figura 5 representa la arquitectura de SVM.

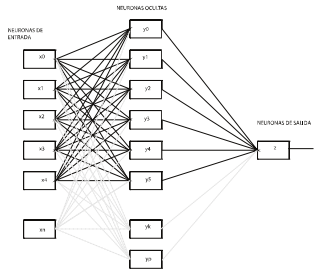

Figura 5. Arquitectura de SVM.

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:

$$BlocksPerRow = \frac{NUMBER\,OF\,DATAS}{THREADS\,PER\,BLOCK} + 1 \tag{10}$$

$$ThreadPerRow = \frac{NUMBER\,OF\,DATAS}{BlocksPerRow}$$

A pesar de eso la tarjeta NVIDIA tiene numero de hilos limitado, por lo tanto esa es la longitud máxima de un vector o número de datos copiados del CPU al GPU divididos entre el número de atributos por dato.

3. Experimentos

3.1. Redes neuronales de impulso (SNN)

En esta sección mostramos el resultado obtenido de la paralelización de SNN que se muestra en el cuadro 1.

Tabla 1. Resultados de algunas bases de datos para SNN.

Datos	Instancias	Atributos	Clases	Iteraciones	Exitos	Tiempo de Aprendizale [ms]
Iris	150	4	3	12	149	52331.625
Cars	1729	6	4	12	1294	1319680.625
Breast Cancer Wisconsin	699	9	2	19	673	253460.0937
Adult	32561	14	2	30	29247	24 hrs aprox.
Heart Disease Cleveland (HDC)	303	13	5	40	221	1069932.75

Sin embargo, observamos que SNN converge exponencialmente hacia la solución, como en la figura 6 (caso XOR).

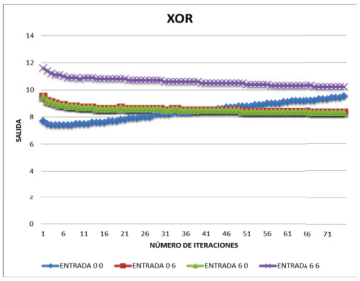

Figura 6. Solución del problema XOR.

En el cuadro 2 se muestra codificación y la solución numéricamente.

3.2. Máquinas de soporte vectorial (SVM)

La metodología SVM es usada para clasificación, regresión, predicción, y densidades de datos. En este caso lo aplicamos a clasificación de datos. Para una

Tabla 2. Caso XOR para SNN.

X1 ms	X2 ms	BIAS ms	SALIDA DESEADA ms
0	0	0	10
0	6	0	8
6	0	0	8
6	6	0	10

red SVM multiclase, aplicamos redes en serie, es decir que la salida de la primer red binaria es la entrada de la siguiente, considerando que se van clasificando por clases binarias. El número de redes en cascada se calcularon:

$$(RedSerial\ SVM) = (Clases) - 1 \tag{11}$$

Los resultados para QP - SVM se muestran en el cuadro 3.

Tabla 3. Resultados de algunas bases de datos para 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

Los resultados de los experimentos, para bases de datos multiclase, el kernel de cada clase tiene que se encontrado ya que el hiperplano por cada clase a considerar cambia de dimensión al clasificar cada clase de la misma base de datos. Por ejemplo, en el caso de la base de datos Cars: la clase 1 de Cars utiliza el kernel "erbf 3", la clase 2 con polinomial de grado 3, la clase 3 con polinomial de grado 6 y la clase 4 con polinomial de grado 8.

De esta manera se llega a los siguientes resultados del cuadro 4 al comparar los tiempos de aprendizaje de ambas metodologías.

Tabla 4. Comparación del tiempo de aprendizaje de SNN vs SVM.

Base de datos	Tiempo de Aprendizaje SNN [ms]	Tiempo de Aprendizaje SVM [ms]
Iris	152331.625	1521128.5
Cars	1319680.625	4148213.125
Breast Cancer Wisconsin (BCW)	253460.09375	326007.0156
Adult	86400000 aprox.	——
Heart Disease Cleveland (HDC)	1069932.75	227423.0906

En la figura 7 se grafica lel cuadro 4 y podemos observar que en una arquitectura GPU SNN es más rápido que SVM. En el caso de la base de datos Adult, no es comparable ya que SVM no es la metodología adecuada en un GPU por el enorme tiempo de aprendizaje que consume. Para bases de datos más pequeñas con menos de 16 instancias, SVM es más rápido que SNN por el uso de un solo bloque con 256 hilos, con lo cual se paraleliza al máximo y dicha metodología converge más rápido que SNN. Debido a que el GPU no provee más de 60000 flops o 245 datos (aprox.) en contraste con la base de datos Adult, ya que esta requiere 1,060,218,721 flops para datos flotantes. Esto nos obliga a reducir la paralelización lo que implica más tiempo de procesamiento que un CPU. La causa es que SVM bajo esas condiciones requiere tomar en cuenta más de un kernel y copiar del CPU al GPU varias veces y viceversa, lo que consume enormes cantidades de tiempo.

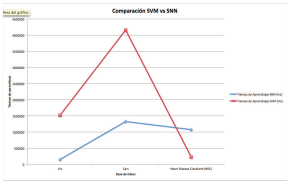

Figura 7. Comparación del Tiempo de Aprendizaje de SNN vs SVM.

Sin embargo, SVM es mayor para bases de datos binaria que SNN, aunque los resultados del cuadro 5 muestran que en un GPU SNN es más preciso que SVM.

En la figura 8 se muestra la gráfica de la comparación de la tabla anterior.

Tabla 5. Comparación de precisión SNN vs SVM.

Base de Datos	Exitos de SVM	Exitos de SNN
Iris	145	149
Cars (SVM light)	1728 (Clase 1) 56 (Clase 2) 4 (Clase 3) 34 (Clase 4)	1294 1294 1294 1294
Breast Cancer Wisconsin	641 (Clase 1) 127 (Clase 2)	673 673
Adult	——	29247
Heart Disease Cleveland (HDC)	147 (Clase 1) 3 (Clase 2) 66 (Clase 3) 25 (Clase 4) 2 (Clase 5)	221 221 221 221 221

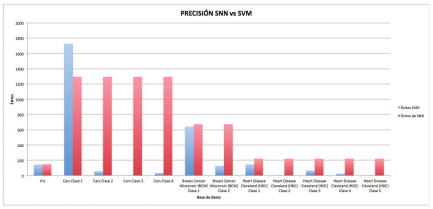

Figura 8. Comparación de precisión SNN vs SVM.

4. Conclusiones y trabajo futuro

Los resultados obtenidos muestran que SNN es más eficiente y veloz que SVM en una arquitectura GPU debido a que éste último requiere enormes cantidades de recursos computacionales. Las SVM convergen más rápido que las SNN pero en un GPU pierde precisión y por lo tanto SVM converge más lento en un GPU que en un CPU. Sin embargo, SNN no consume tantos recursos de memoria como SVM y por lo tanto es más rápida su ejecución a pesar de converger más lento que SVM. También SNN no requiere tanta precisión en la salida porque utiliza ventanas de tiempo para codificar la salida, por ejemplo si la ventana de tiempo va desde 10 a 12 un 10.0 o 11.5 puede representar un 1 lógico.

En el caso de SNN al aplicar algún circuito lógico y secuencial con un circuito de reloj, entonces su precisión puede ser afectada cuando la frecuencia de reloj sea alta. Sin embargo para aplicaciones reales la velocidad de los sensores podrían limitar la detección del umbral.

Como trabajo futuro es necesario implementar SNN en tiempo real con el hardware adecuado para otras aplicaciones. Por otro lado, SNN puede ser extendido para más capas con la finalidad de conocer los efectos sobre el resultado.

Referencias

1. Sridhar, A.; Vincenzi, A.; Ruggiero, M.; Atienza, D.; "Neural Network-Based Thermal Simulation of Integrated Circuits on GPUs,Çomputer–Aided Design of Integrated Circuits and Systems, IEEE Transactions on , vol.31, no.1, pp. 23–36, Jan. 2012.
2. Ahmadi, Arash; Soleimani, Hamid; 'A GPU based simulation of multilayer spiking neural networks,.[El]ectrical Engineering (ICEE), 2011 19th Iranian Conference on , vol., no., pp. 1–5 17–19 May 2011.
3. Lowe, E.W.; Woetzel, N.; Meiler, J. "Poster: GPU–accelerated artificial neural network for QSAR modeling,Çomputational Advances in Bio and Medical Sciences (ICCABS), 2011 IEEE 1st International Conference on , vol., no., pp. 254, 3–5 Feb. 2011.
4. B. Kirk, David; W. Hwu, Wen-mai "Programming Massively Parallel Processors.[Ed]. Morgan Kaufmann, 2010.
5. Qi Li; Salman, R.; Kecman, V.; "An intelligent system for accelerating parallel SVM classification problems on large datasets using GPU,Ïntelligent Systems Design and Applications (ISDA), 2010 10th International Conference on , vol., no., pp. 1131–1135, Nov. 29 2010–Dec. 1 2010.
6. [5] Tsung-Kai Lin; Shao-Yi Chien; "Support Vector Machines on GPU with Sparse Matrix Format,"Machine Learning and Applications (ICMLA), 2010 Ninth International Conference on , vol., no., pp. 313–318, 12–14 Dec. 2010
7. Papadonikolakis, M.; Bouganis, C.:'A novel FPGA–based SVM classifier,"Field–Programmable Technology (FPT), 2010 International Conference on , vol., no., pp. 283–286, 8–10 Dec. 2010
8. Sergio Herrero–Lopez, John R. Williams, Abel Sanchez,: Parallel Multiclass Classification using SVMs on GPUs", November 2010.
9. Izhikevich, E.M.:'Hybrid spiking models", vol. 368, issue 1930, pp. 5061–5070, Nov 2010.
10. Venkittaraman Vivek, Pallipuram Krishnamani: 'ACCELERATION OF SPIKING NEURAL NETWORKS ON SINGLE–GPU AND MULTI–GPU SYSTEMS", May 2010.
11. Bhuiyan, M.A.; Pallipuram, V.K.; Smith, M.C.: "Acceleration of spiking neural networks in emerging multi–core and GPU architectures,"Parallel and Distributed Processing, Workshops and Phd Forum (IPDPSW), 2010 IEEE International Symposium on , vol., no., pp. 1–8, 19–23 April 2010.
12. Scanzio, S.; Cumani, S.; Gemello, R.; Mana, F.; Laface, P.; 'Parallel implementation of artificial neural network training,.[A]coustics Speech and Signal Processing (ICASSP), 2010 IEEE International Conference on , vol., no., pp. 4902–4905, 14–19 March 2010.
13. Yudanov, D.; Shaaban, M.; Melton, R.; Reznik, L.; GPU–Based Simulation of Spiking Neural Networks with Real–Time Performance and High Accuracy, Feb 2010.
14. XIN JIN: Parallel Simulation of Neural Networks on Spinnaker Universal Neuromorphic Hardware, University of Manchester, 2010.

15. Papadonikolakis, M.; Bouganis, C.–S.; Constantinides, G.: 'Performance comparison of GPU and FPGA architectures for the SVM training problem,"Field–Programmable Technology, 2009. FPT 2009. International Conference on , vol., no., pp. 388–391, 9–11 Dec. 2009.
16. Fidjeland, A.K.; Roesch, E.B.; Shanahan, M.P.; Luk, W.; 'NeMo: A Platform for Neural Modelling of Spiking Neurons Using GPUs,.Application-specific Systems, Architectures and Processors, 2009. ASAP 2009. 20th IEEE International Conference on , vol., no., pp. 137–144, 7–9 July 2009.
17. Nageswaran, J.M., Dutt, N.: Krichmar, J.L.; Nicolau, A.; Veidenbaum, A.; 'fficient simulation of large–scale Spiking Neural Networks using CUDA graphics processors,"Neural Networks, 2009. IJCNN 2009. International Joint Conference on , vol., no., pp. 2145–2152, 14–19 June 2009.
18. Thomas, D.B.; Luk, W.: "FPGA Accelerated Simulation of Biologically Plausible Spiking Neural Networks,"Field Programmable Custom Computing Machines, 2009. FCCM '09. 17th IEEE Symposium on , vol., no., pp. 45-52, 5-7 April 2009
19. Carpenter, A.; 'CUSVM: A CUDA IMPLEMENTATION OF SUPPORT VECTOR CLASSIFICATION AND REGRESSION", Jan. 2009.
20. Stewart R.D., Bair W.: 'Spiking neural network simulation: numerical integration with the Parker–Sochacki method", Jan. 2009
21. Prabhu, R.D.; "SOMGPU: An unsupervised pattern classifier on Graphical Processing Unit,.Evolutionary Computation, 2008. CEC 2008. IEEE World Congress on Computational Intelligence. IEEE Congress on , vol., no., pp. 1011–1018, 1–6 June 2008
22. Bryan Catanzaro, Narayanan Sundaram, Kurt Keutzer: Fast Support Vector Machine Training and Classification on Graphics Processors, International Conference on Machine Learning, Helsinki, Finland, 2008.
23. Martínez Z. M., Díaz P. F. J., Díez H. J. F, Antón R. M.: Fuzzy ART Neural Network Parallel Computing on the GPU, Springer (2007).
24. Philipp S., Grbl A., Meier K., Schemmel J.: Interconnecting VLSI Spiking Neural Networks Using Isochronous Connections, Springer (2007).
25. Benjamin Schrauwen, Jan Van Campenhout, Improving SpikeProp: Enhancements to An Error-Backpropagation Rule for Spiking Neural Networks, Ghent University ELIS-PARIS (2006)
26. L. Zhongwen, L. Hongzhi and W. Xincai: Artificial Neural Network Computation on Graphic Process Unit (2005).
27. N.G. Pavlidis, D.K. Tasoulis, V.P. Plagianakos, G. Nikiforidis, M.N. Vrahatis: Spiking Neural Network Training Using Evolutionary Algorithms, IEEE International Joint Conference on, pp. 2190 - 2194, vol. 4 ,december 2005.
28. Seijas Fossi C., Caralli D' Ambrosio A.: Uso de las máquinas de soporte para la estimación del potencial de acción cellular, Revista de Ingeniería UC. Vol. 11, N 1, 56- 61 (2004).
29. Olaf Booij,: Temporal Pattern Classification using Spiking Neural Networks, Intelligent Sensory Information Systems Informatics Institute Faculty of Science Universiteit van Amsterdam (2004).
30. Sander M. B.: Spiking Neural Networks, Universiteit Leiden (2003).
31. Izhikevich, E.M.: 'Simple model of spiking neurons,"Neural Networks, IEEE Transactions on , vol.14, no.6, pp. 1569– 1572, Nov. 2003
32. Sander M. Bohte, Joost N. Kok, Han La Poutre: Error–backpropagation in temporally encoded networks of spiking neurons, Neurocomputing 48, pp. 17 - 37, (2002).

33. Platt C. J.: Sequiential Minimal Optimization: A fast Algorithm for Tarining Support Vector Machines (1998).
34. Osuna E., Freund R., Girosi F.: An Improved Training Algorithm for Support Vector Machines, In Proc. of IEEE NNSP'97.
35. Nikola B. Serbediija: Simulating Artificial Neural Networks on Parallel Architectures (1996).
36. Vapnik V., Cortes C., Support Vector Networks (1995).
37. Fauset Laurene, Fundamentals of Neural Networks, ARCHITECTURE, ALGORITHMS, AND APPLICATIONS, Prentice Halls, 1994.
38. A. L. Hodgkin and A. F. Huxley: A quantitative description of membrane current and its application to conduction and excitation in nerve, J. Physiol. (1952).
39. http://www.nvidia.com
40. http://www.svms.org