Optimización con PSO de un Modelo TSK Mediante Procesadores Multinúcleo y GPGPU

Castro Liera Marco Antonio*, Castro Liera Iliana*, Castro Jesús Antonio*

** Instituto Tecnológico de La Paz, Blvd. Forjadores de Sudcalifornia #4720 CP 23080, La Paz, B.C.S. (Tel: 612-1210424 ext. 116 email: mcastro@itlp.edu.mx).*

Resumen: El presente trabajo reporta la comparación de los resultados de la optimización de los parámetros de un sistema de identificación basado en reglas difusas del tipo Takagi-Sugeno-Kang (TSK), en diferentes plataformas paralelas de bajo costo usando el algoritmo heurístico conocido como Optimización por Enjambre de Partículas (PSO). Los resultados se comparan con los obtenidos anteriormente usando un algoritmo genético (GA) ejecutado en un cluster implementado con PVM.

Palabras claves: GPGPU, PSO, TSK, Fuzzy, PVM, CUDA.

1. INTRODUCCIÓN

El desarrollo de aplicaciones paralelas ya no está restringido al uso de equipos de alto costo, pues casi todas las computadoras personales que se fabrican actualmente están equipadas con procesadores con múltiples núcleos, capaces de ejecutar instrucciones de manera simultánea. Adicionalmente, muchas computadoras incluyen unidades avanzadas de procesamiento gráfico con las que se puede realizar cómputo de propósito general (GPGPU).

Sin embargo, el desarrollo de aplicaciones que exploten esas capacidades de procesamiento paralelo se ha quedado rezagado en comparación al vertiginoso desarrollo del hardware computacional, por lo que la investigación y desarrollo de software que sea capaz de aprovechar esas características se ha convertido en uno de los principales retos en el campo de la computación.

El presente trabajo se centra en el uso del algoritmo de búsqueda heurística PSO para optimizar los parámetros de un modelo difuso TSK para un proceso fermentativo.

El algoritmo se implementó sobre dos alternativas paralelas de bajo costo: una CPU con múltiples núcleos y dos diferentes GPGPU.

1.1. El proceso fermentativo

Para llevar a cabo el proceso de fermentación, se cuenta con cierta cantidad de microorganismos (biomasa) que se encuentran suspendidos en un medio rico en alimento (sustrato).

Anteriormente se ha estudiado que los procesos de este tipo presentan un comportamiento altamente no lineal (Herrera Fernández, Martinez Jiménez et al. 2003) y que corresponden a un sistema dinámico con la estructura general dada en (1).

$$\frac{dx}{dt} = f(X) + bu \tag{1}$$

Donde $\boldsymbol{X}$ es el vector formado por las dos variables de estado $[x,s]$ (concentración de biomasa y sustrato, respectivamente), u es el sustrato de entrada y b es la razón constante de dilución D.

1.2. La estructura del modelo difuso

Nos encontramos ante un modelo con tres variables de entrada, x_t, s_t y s_{in}, que representan, respectivamente, la concentración de biomasa, sustrato y sustrato de entrada en el instante t. El modelo deberá determinar la concentración de biomasa en el siguiente instante x_{t+1}.

ISBN: 978-607-95534-5-6

Las funciones de membresía propuestas para el sistema difuso en las variables de entrada son del tipo campana generalizada, como se muestra en (2).

$$\Phi(x,\alpha,\beta,\gamma)=\frac{1}{1+\left|\frac{x-\gamma}{\alpha}\right|^{2\beta}} \tag{2}$$

Dado que para el caso que se estudia el sustrato de entrada es constante, la base de reglas difusas tiene la estructura mostrada en (3), donde s_{in} se ha consolidado en el término a_{i0}.

$$\text{IF } x_t \text{ IS } \Phi_j \text{ AND } s_t \text{ IS } \Phi_k \text{ THEN}$$
$$x_{t+1} = a_{i0} + a_{i1}x_t + a_{i2}s_t \tag{3}$$

Con j variando desde 1 hasta el número de conjuntos difusos que dividen a x_t (en nuestro caso 3), k variando desde 1 hasta el número de conjuntos difusos que dividen a s_t (en nuestro caso 3) e i variando desde 1 hasta el número de reglas difusas en la base de reglas (en nuestro caso 9).

Por tanto, debemos encontrar los valores óptimos para un total de 18 parámetros en la parte antecedente y 27 coeficientes en la parte consecuente del modelo TSK.

La biomasa predicha se calcula como el promedio ponderado de la salida de las reglas mediante (4).

$$x_{(t+1)}=\frac{\sum_{i=1}^{9} h_i x_{i(t+1)}}{\sum_{i=1}^{9} h_i} \tag{4}$$

Donde h_i representa el grado en que la i-ésima regla se cumple y es calculado mediante el operador T, en nuestro caso el producto, dado en (5).

$$h_i = T(\Phi_j(x_t),\Phi_k(s_t)) \tag{5}$$

1.3. Estrategia de optimización en dos fases

El proceso de optimización fue dividido en dos etapas; la primera busca encontrar un conjunto de coeficientes para la parte consecuente de la base de reglas difusas que minimicen el error total *ET* calculado mediante (6).

$$ET=\sum_{s=1}^{n}\left|xc_s - xm_s\right| \tag{6}$$

Donde n es el número total de muestras de entrenamiento (en nuestro caso se contó con 490 muestras), x_c es la concentración de biomasa calculada y x_m la concentración de biomasa medida.
La segunda fase del proceso de optimización intenta encontrar un conjunto de parámetros para las funciones de membresía que minimicen el error máximo *EM* dado por (7).

$$EM = \max(\left|xc_1 - xm_1\right|,\ldots,\left|xc_n - xm_n\right|) \tag{7}$$

Para llevar a cabo la primera fase de optimización, se eligen parámetros para las funciones de campana generalizada que crean conjuntos difusos que dividen uniformemente el espacio de las variables de entrada, como se muestra en (8).

$$\begin{aligned}
&\alpha_j = 2 \quad \alpha_k = 2\\
&\beta_j = 8 \quad \beta_k = 8\\
&\gamma_j = x_{\min} + \frac{(x_{\max} - x_{\min})(2j-1)}{6}\\
&\gamma_k = s_{\min} + \frac{(s_{\max} - s_{\min})(2k-1)}{6}
\end{aligned} \tag{8}$$

Dichos parámetros fueron utilizados para encontrar un conjunto de 27 coeficientes de funciones de salida que minimicen el error total de salida del modelo resultante calculado mediante (6).

Para la segunda parte del proceso de optimización, se usa el mejor conjunto de coeficientes para las funciones de salida y se trata de encontrar un conjunto de parámetros para las funciones de membresía que minimicen el error máximo de predicción dado por (7).

2. OPTIMIZACIÓN POR ENJAMBRE DE PARTÍCULAS

El métodos de optimización por enjambre de partículas, propuesto en (Eberhart y Kennedy 1995) está basado en el comportamiento de las bandadas de aves y los cardúmenes de peces en los que su movimiento se guía por la búsqueda de alimento o para escapar de los predadores y en los que el grueso de la población sigue al miembro que posee mayor conocimiento del entorno.

Cada partícula en el enjambre almacena en su memoria su posición actual (x_i), su velocidad (v_i) y aptitud (f_i), así como su mejor posición y aptitud históricas.

ISBN: 978-607-95534-5-6

El algoritmo inicia generando aleatoriamente un conjunto de partículas, denominado enjambre. Cada partícula calcula su velocidad usando (9) y su nueva posición usando (10) para cada dimensión *d*; esto se realiza para una cantidad preestablecida de generaciones o hasta alcanzar una condición de parada.

$$v_{id} = wv_{id} + c_1 r_1 (x_{hd} - x_{id}) + c_2 r_2 (x_{gd} - x_{id}) \quad (9)$$

En la ecuación (9) x_h representa la mejor posición histórica de la partícula, x_g representa la mejor posición del enjambre, w es un factor de peso inercial, c_1 la confianza en su propia información y c_2 la confianza en la información social; r_1 y r_2 son valores generados aleatoriamente con una distribución uniforme en el intervalo [0,1]. (Van de Berg, 2001)

$$x_{id(t+1)} = x_{id(t)} + v_{id} \quad (10)$$

Es claro que para la primera parte del proceso de optimización el vector de posiciones $\boldsymbol{X}$ puede sustituirse por el vector de coeficientes de la parte consecuente de cada una de las nueve reglas del modelo difuso, como se muestra en (11).

$$X = [a_{00}, a_{01}, a_{02} \ldots a_{80}, a_{81}, a_{82}] \quad (11)$$

Asimismo para la segunda parte del proceso de optimización, el vector de posiciones, representará los parámetros de las funciones de pertenencia como se muestra en (12).

$$X = [\alpha_{0x}, \beta_{0x}, \gamma_{0x}, \alpha_{0s}, \beta_{0s}, \gamma_{0s} \ldots \alpha_{2x}, \beta_{2x}, \gamma_{2x}, \alpha_{2s}, \beta_{2s}, \gamma_{2s}] \quad (12)$$

3. ARQUITECTURAS PARALELAS

3.1. El procesador Intel Core i7

El procesador Intel Core-i7 modelo 2600 utilizado en el presente trabajo, cuenta con una arquitectura *Sandy Bridge*, tiene un total de cuatro núcleos con tecnología *Hyper-Threading,* por lo cual es capaz de ejecutar de manera concurrente hasta 8 hilos.

La frecuencia de reloj en modo normal de estos procesadores es de 3.4GHz y puede llegar hasta los 3.8GHz en modo turbo, con la tecnología *Turbo-Boost.*

Cuando se desarrollan aplicaciones para esta arquitectura es importante buscar que la cantidad de hilos que el programa genere sea un múltiplo de los 8 que el procesador soporta, para evitar los tiempos muertos en las unidades de procesamiento.

3.2. La arquitectura CUDA

CUDA es una tecnología de GPGPU desarrollada por NVIDIA en 2006 cuando liberó el lenguaje CUDA C que permite el desarrollo de aplicaciones de propósito general en paralelo utilizando los múltiples procesadores incluidos en la tarjeta gráfica (NVIDIA Corp. 2009). La arquitectura CUDA se muestra en la figura 1.

Los elementos de la arquitectura CUDA son (NVIDIA Corp. 2010): los hilos en donde se ejecutan las funciones simultáneamente y tienen su propio espacio de memoria; bloques, formados por un conjunto de hilos e incluyen un espacio de memoria compartida para su comunicación; una malla, que incluye los bloques que se ejecutan en un dispositivo, con distintas zonas de memoria comunes a todos los bloques.

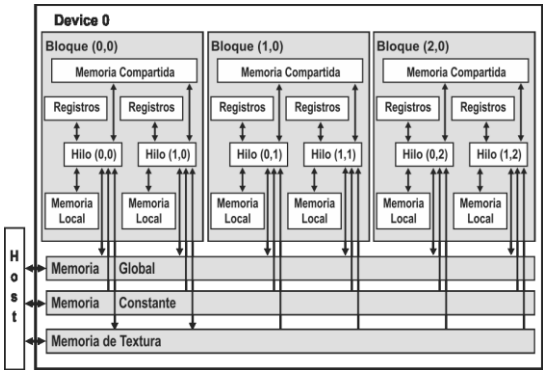

Fig. 1. Arquitectura de CUDA.

La GPU organiza el trabajo en sus multiprocesadores como se muestra en la figura 2 (NVIDIA Corp. 2010).

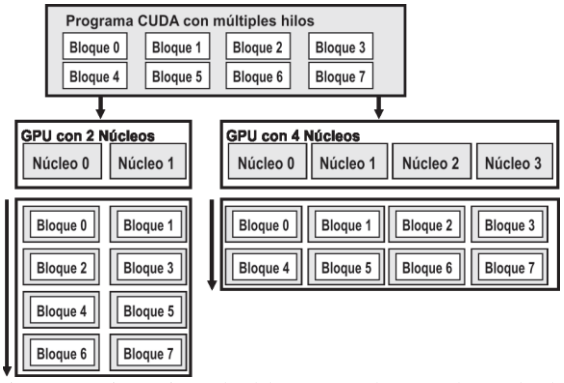

Fig. 2. Asignación de bloques a los núcleos de la GPU

Una función que se ejecuta en la tarjeta gráfica (GPU) se denomina kernel. Cuando se lanza un kernel, es necesario informar a la GPU cuántos bloques (BLK) e hilos (THR) se ejecutarán.

Tanto el modelo GTX-460 como el GTX-670 cuentan con un total de 7 multiprocesadores, por lo que, para aprovechar al máximo la capacidad de procesamiento y evitar tener recursos desperdiciados, es importante que la cantidad de bloques que nuestra aplicación ejecute sea un múltiplo de 7.

4. ESTRATEGIA DE PARALELIZACIÓN

Las estrategias de paralelización de PSO pueden clasificarse en dos grandes grupos: de grano grueso y de grano fino (Li y Wada, 2005). Las estrategias de grano grueso implementan un modelo de islas, donde cada enjambre es ejecutado en un hilo independiente; opcionalmente los hilos pueden intercambiar información sobre la mejor posición encontrada cada cierto número de generaciones. En la estrategia de grano fino cada partícula es procesada en un hilo, por lo que los diferentes hilos deberán actualizar, en todo momento, la posición histórica con mejor aptitud.

En virtud de que para las dos arquitecturas a probar las unidades de proceso son homogéneas, la asignación de carga se efectuó de manera estática y simétrica. Aunque la asignación de carga dinámica en estructuras de cómputo heterogéneas brinda la ventaja de una mayor eficiencia en el aprovechamiento del poder computacional, en el caso que nos compete significaría agregar complejidad al algoritmo sin obtener beneficios de desempeño.

4.1. Implementación de PSO sobre la CPU

Al paralelizar sobre la CPU se utilizó la estrategia de grano grueso, haciendo una dispersión de los resultados con una topología de anillo cada cierto número de iteraciones. Para ello se usaron, en la primera parte del proceso, los siguientes parámetros:

- Número de enjambres 24
- Tamaño del enjambre 25
- Iteraciones 2000
- Periodo de Dispersión 200
- Peso inercial $w = 0.8$
- Coeficientes de confianza $C_1=C_2 = 1.62$.

Y los siguientes para la segunda parte del proceso:

- Número de enjambres 24
- Tamaño del enjambre 25
- Iteraciones 150
- Periodo de Dispersión 15
- Peso inercial $w = 0.8$
- Coeficientes de confianza $C_1=C_2 = 1.62$.

4.2. Implementación de PSO con CUDA

La paralelización sobre CUDA se llevó a cabo usando una mezcla de las estrategias de grano grueso y grano fino. Por una parte, en cada bloque se procesó un enjambre de t partículas y cada partícula dentro del enjambre es procesada por un hilo independiente (figura 3).

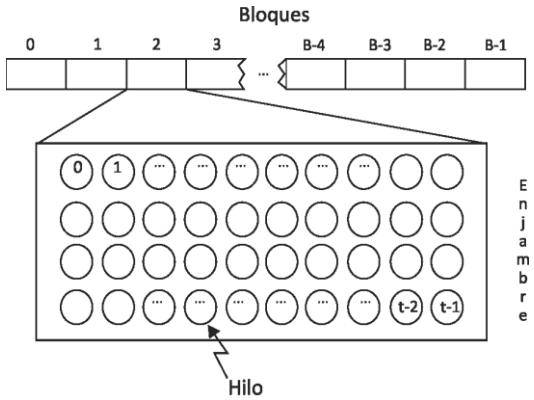

Fig. 3. Asignación de enjambres en la GPU.

Cada cierto número de iteraciones ocurre un proceso de dispersión de la información, que consiste en que cada bloque comunica la información de su mejor posición con una topología de anillo; como se muestra en la figura 4.

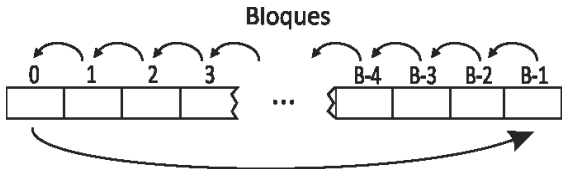

Fig. 4. Dispersión de información.

Para la implementación en CUDA se utilizaron los mismos parámetros que en la CPU, ajustando a 28 el número de enjambres para usar un número de bloques múltiplo de la cantidad de multiprocesadores disponibles en las tarjetas GTX-460 y GTX-670.

5. RESULTADOS

Tras un total de 1,000 pruebas con cada una de las arquitecturas utilizadas, se contrastaron los resultados con los reportados en un trabajo anterior donde se utilizó un algoritmo genético en un cluster de 18 computadoras Pentium IV@1.6GHz en una red fast-ethernet (Castro y Herrera 2010).

ISBN: 978-607-95534-5-6

La ecuación (13) se utiliza para determinar el tamaño de la muestra, cuando se trata de estimar valores medios en poblaciones infinitas.

$$n = \frac{Z_a^2 S^2}{d^2} \tag{13}$$

Donde n es el tamaño de la muestra, S^2 es la varianza estimada de la población y d representa el semi-rango del intervalo de confianza. Usando un valor de Z_a de 2.576, que corresponde a un nivel de certidumbre de 99%, podemos determinar el valor del semi-rango d, usando (14).

$$d = \sqrt{\frac{Z_a^2 S^2}{n}} \tag{14}$$

La tabla I muestra una comparación de la calidad de los resultados obtenidos.

Tabla I. Calidad de la solución

Algoritmo y Arquitectura	**ETP**	***d***	**EMP**	***d***
AG-Cluster	14.81	±0.23	0.086	±0.002
PSO-Core i7	10.13	±0.15	0.191	±0.006
PSO-GXT460	9.51	±0.15	0.220	±0.008
PSO-GTX670	9.56	±0.18	0.353	±0.017

En la tabla I, ETP representa el error total promedio y EMP el error máximo promedio.

El error total promedio de la nueva implementación fue mejor que el obtenido anteriormente usando AG y es prácticamente idéntico para ambas GPGPU.

Para la segunda fase del proceso de optimización no pudieron encontrarse parámetros para PSO que mejoraran la calidad del error máximo que se había obtenido previamente usando AG.

La tabla II muestra un comparativo de los tiempos de ejecución promedio obtenidos.

Tabla II. Tiempos de ejecución

Algoritmo-Arquitectura	**Tiempo(s)**
AG/Cluster	100.00
PSO/Core-i7	32.34
PSO/GXT-460	14.51
PSO/GTX-670	12.37

Puede observarse que tanto para la CPU multi-núcleo como para las GPGPU la implementación de PSO tiene tiempos de ejecución sustancialmente menores que la solución con AG sobre el cluster, siendo el mejor desempeño el obtenido con la tarjeta NVIDIA GTX-670.

6. CONCLUSIONES

El algoritmo propuesto fue capaz de generar soluciones de mejor calidad para la primera etapa del proceso de optimización en un tiempo que llega a ser hasta un 87% menor que la implementación en el cluster.

La segunda fase del proceso de optimización no obtuvo la misma calidad de soluciones que en la versión con AG usando clusters, por lo que se propone una futura investigación sobre la aplicación de una estrategia mixta que use PSO para la primera parte del proceso y AG en la segunda.

El uso de las nuevas plataformas de cómputo paralelo de bajo costo ha probado ser una alternativa excelente para el desarrollo de este tipo de algoritmos de optimización.

REFERENCIAS

Castro Liera, M.A. y F. Herrera, Finding Fuzzy Identification System Parameters Using a New Dynamic Migration Period-Based Distributed Genetic Algorithm, *DYNA,* No. 159, pp 77-83

Herrera Fernández, F., B. Martinez Jiménez, et al. (2003). *Aplicación de las Técnicas de la Inteligencia Artificial en un Proceso Biotecnológico de Reproducción Celular (Embriogénesis Somática)*. Santa Clara, Libre, Universidad Central "Marta Abreu" de Las Villas: 38.

Kennedy, J. y Eberhart, C., (1995), Particle Swarm Optimization In *Proceedings of IEEE International Conference on Neural Networks.* Volumen IV, páginas 1942-1948.

Li, B. y Wada, K. Parallelizing particle swarm optimization *In Proceedings of IEEE Pacific Rim Conference on Communications, Computers and signal Processing* (2005) , 288—291.

NVIDIA Corp., (2009), *CUDA Architecture Introduction & Overview*, NVIDIA Corp., Santa Clara California. Version 1.1.

NVIDIA Corp., (2010), *CUDA C Programming Guide Version 3.2*, NVIDIA Corp., Santa Clara California.

Van de Berg, Frans, (2001) *"An Analysis of Particle Swarm Optimizers",* PhD Dissertation Faculty of Agricultural and Natural Science, Pretoria University, Pretoria South Africa.

Paralelización de un método adaptativo para restauración de imágenes utilizando programación en GPU

Estrella-Ojeda J. *, López-Martínez J. *

**Universidad Autónoma de Yucatán, Facultad de Matemáticas, Unidad Multidisciplinaria Tizimín Tizimín, Yucatán 97700, México*
(Tel: (986)86-32175;e-mail: jose.estrella@correo.uady.mx, jlopezm@uady.mx).

Resumen: Los métodos de restauración son muy populares dentro del área del procesamiento de imágenes. Recientemente, se propuso un método adaptativo de restauración para imágenes degradadas con interferencias aditivas, multiplicativas e impulsivas. Una de las desventajas de este método es su complejidad computacional secuencial, la cual depende del tamaño de la imagen a restaurar y de un método iterativo para la solución de un sistema de ecuaciones lineales asociado al proceso de restauración. En este artículo se presenta la paralelización del método adaptativo utilizando programación en GPU. Los experimentos realizados demuestran que la aplicación del método de restauración en un sistema de Tiempo Real puede ser beneficiada con nuestra propuesta.

Palabras claves: cómputo en GPU, restauración de imágenes, Gradiente Conjugado, análisis de rendimiento.

1. INTRODUCCIÓN

El nacimiento de lo que actualmente es el procesamiento digital de imágenes (PDI) se puede remontar a la disponibilidad de las máquinas y al inicio del programa espacial de los años 60. Fue necesaria la combinación de estos dos desarrollos para poner de relieve el potencial de los conceptos de procesamiento de imágenes digitales (Gonzalez and Woods, 2008). Una de las áreas más populares del PDI es el tópico de restauración de imágenes, debido a la gran cantidad de aplicaciones que se pueden desarrollar (Bovik, 2005). La restauración de imágenes se fundamenta en la eliminación o minimización de degradaciones en imágenes que podrían ser ocasionadas por fallas en los sensores de la cámara con la cual fue obtenida dicha imagen o por agentes externos (Jain, 1988).

Recientemente, se propuso un método ciego adaptativo para restauración de imágenes utilizando la técnica del microescaneo (López-Martínez and Kober, 2012). Dicho método, puede ser aplicado para corregir iluminación no uniforme en las imágenes, para restaurar imágenes que han sido adquiridas utilizando sensores que poseen elementos dañados, y además puede ser empleado para remover un ruido de patrón-fijo característico de los sensores formado por arreglos focales planos. Una de las desventajas de este método es su complejidad computacional secuencial, la cual limita sus aplicaciones en un sistema de Tiempo Real.

El propósito principal del procesamiento en paralelo es efectuar cálculos con mayor velocidad al realizarlos en varios procesadores simultáneamente. Tres han sido los principales factores a la tendencia actual sobre el procesamiento paralelo. El primero, el costo del hardware ha descendido constantemente, segundo, la tecnología de circuitos ha avanzado hasta el punto en que es posible diseñar sistemas complejos que requieren millones de transistores en un solo chip, y tercero, la velocidad en el tiempo del ciclo de instrucción en los procesadores de la arquitectura de Von Neumann ha incrementado acercándose a las limitaciones físicas de cualquier procesador (JáJá, 1992).

Cuando una aplicación es adecuada para ejecutarla paralelamente, una buena opción pueden ser las unidades de procesamiento gráfico (GPU's), ya que éstas pueden ser hasta 100 veces más rápidas que una ejecución secuencial de la misma aplicación (Kirk and Hwu, 2010). Por eso, la programación en las GPU's está revolucionando la simulación científica proporcionando una o dos órdenes de mayor magnitud de rendimiento en la ejecución por cada GPU que se utilice. El beneficio que se tiene actualmente es que estas unidades de