Implementación de un Cluster Heterogéneo de GPGPU para Algoritmo Genético

Mauricio Rojas Salazar
División de Estudios de Posgrado e Investigación
Instituto Tecnológico de La Paz
La Paz B.C.S., México
Email: mr.salazar00@gmail.com

Marco Antonio Castro Liera
División de Estudios de Posgrado e Investigación
Instituto Tecnológico de La Paz
La Paz B.C.S., México
Email: mcastro@itlp.edu.mx

Jesús Antonio Castro
División de Estudios de Posgrado e Investigación
Instituto Tecnológico de La Paz
La Paz B.C.S., México
Email: jcastro@itlp.edu.mx

Iliana Castro Liera
División de Estudios de Posgrado e Investigación
Instituto Tecnológico de La Paz
La Paz B.C.S., México
Email: icastro@itlp.edu.mx

***Resumen*—En el presente artículo se describe la implementación de un cluster heterogéneo de Unidades de Procesamiento Gráfico de Propósito General (GPGPU, por sus siglas en inglés) en el cual se ejecutó un algoritmo genético para pruebas de eficacia y eficiencia.**

Para el cluster se implementó OpenMPI y para la programación en la GPU se utilizó OpenCL.

***Index Terms*—OpenCL, OpenMPI, GPGPU, clúster, algoritmo genético, kernel, host.**

I. INTRODUCCÍON

En el computo paralelo existe la tendencia del uso de las Unidades de Procesamiento Gráfico de Propósito General (GPGPU, por sus siglas en inglés).

Mientras los procesadores cuentan con una cantidad muy reducida de ALUs, las tarjetas gráficas poseen una cantidad mucho mayor, permitiendo hacer en una fracción del tiempo lo que a una CPU le tomaría hacer en horas.

Otra práctica que existe es la creación de clusters utilizando equipos con tarjetas graficas del mismo fabricante, arquitectura comúnmente llamada cluster homogéneo que combina las ventajas de un cluster con las ventajas de las GPGPUs.

En el Instituto Tecnológico de La Paz se han realizado varios proyectos de investigación con procesamiento paralelo en GPGPU. Los más relevantes con relación a este trabajo son:

1. La implementación de un AG y un PSO en una GPGPU, utilizando CUDA y una tarjeta Nvidia[1].
2. La implementación de un AG y un PSO en un cluster homogéneo, utilizando CUDA y tarjetas gráficas Nvidia[2].

Actualmente no existe mucho trabajo sobre clusters heterogéneos, de aquí la iniciativa propuesta en el presente artículo.

Para el desarrollo del cluster heterogéneo se implementó OpenMPI para el enlace y la comunicación de los equipos.

En el caso de la programación para la GPGPU se empleó OpenCL. Esta herramienta consta de una interfaz y un lenguaje de programación, brindando un estándar en el desarrollo del código. Las aplicaciones fueron desarrolladas en lenguaje C y, al momento de compilación, se utilizan las librerías de cada fabricante, lo que permite usar el mismo código para tarjetas de diferentes marcas.

II. MARCO TEÓRICO

A. OpenCL

Open Computing Language (OpenCL) es un framework de programación heterogénea que es manejado por el consorcio de tecnología sin fines de lucro Khronos, con la participación de compañías como Intel, ARM, AMD, NVIDIA, QUALCOMM, Apple y muchos otros[3]. Con OpenCL es posible crear aplicaciones que pueden ser ejecutadas en una gran variedad de dispositivos de diferentes fabricantes. Es compatible con una amplia gama de niveles de paralelismo de manera eficiente a sistemas homogéneos o heterogéneos, de uno o varios dispositivos que consten de CPU, GPU y otros tipos de dispositivos. La definición de OpenCL ofrece un lenguaje del lado del dispositivo y una capa de administración del host para los dispositivos en un sistema.

Dentro de las arquitecturas admitidas se incluyen: CPU multinúcleo, procesadores vectoriales (tales como la GPU) y dispositivos paralelos de grano fino (como FPGA). Lo más importante es que la compatibilidad multiplataforma de OpenCL lo convierte en un excelente modelo de programación para que los desarrolladores lo aprendan y utilicen, con la confianza de que continuará estando ampliamente disponible en los próximos años, con alcance y aplicabilidad cada vez mayores[4].

B. OpenMPI

OpenMPI es una implementación de la interfaz de paso de mensajes de código abierto, desarrollada y mantenida por un

consorcio de socios académicos, de investigación y de la industria. Por lo tanto, Open MPI puede combinar la experiencia, las tecnologías y los recursos de toda la comunidad informática de alto rendimiento, para crear la mejor biblioteca de MPI disponible. OpenMPI ofrece ventajas para los proveedores de sistemas y software, desarrolladores de aplicaciones e investigadores de informática[5].

1) MPI: MPI (Message-Passing Interface) es una especificación de la interfaz de biblioteca para el paso de mensajes. Todas las partes de esta definición son significativas. MPI aborda principalmente el modelo de programación paralela de transmisión de mensajes, en el que los datos se trasladan del espacio de direcciones de un proceso a otro mediante operaciones cooperativas en cada proceso. Las extensiones del modelo clásico de paso de mensajes se proporcionan en operaciones colectivas, operaciones de acceso a memoria remota, creación de procesos dinámicos y E/S paralelas. MPI es una especificación, no una implementación. Esta especificación es para una interfaz de biblioteca; MPI no es un lenguaje, y todas las operaciones MPI se expresan como funciones, subrutinas o métodos. El estándar ha sido definido a través de un proceso abierto por una comunidad de vendedores de computo paralelo, informáticos y desarrolladores de aplicaciones[6].

C. Algoritmo Genético

Los algoritmos genéticos (AG) fueron introducidos por Holland (1975) e imitan los principios básicos de la naturaleza formulados por Darwin (1859) y Mendel (1866). Dichos principios básicos son:

1. Existe una población de soluciones. Las propiedades de una solución se evalúan según el fenotipo y los operadores de variación se aplican al genotipo. Algunas de las soluciones se eliminan de la población si el tamaño de la población supera un límite superior.
2. Los operadores de variación crean nuevas soluciones con propiedades similares a las existentes. El operador principal de búsqueda es la recombinación y la mutación sirve como operador de fondo.
3. Los individuos de mejor calidad se seleccionan con mayor frecuencia para la reproducción, mediante un proceso de selección.[7]

III. METODOLOGÍA

A. Desarrollo de un Framework para OpenCL

En OpenCL la mayoría de las funciones tienen varios parámetros, haciéndolas altamente configurables. Esto es debido a que esta herramienta no es única y exclusivamente para las GPUs, sino para muchos otros dispositivos de procesamiento. Por estas dos características del lenguaje es que en este trabajo se desarrolló un framework para facilitar la tarea de programación, disminuyendo en gran medida el cuerpo resultante del programa y el número de parámetros con los cuales se trabaja. Esto se llevó a cabo después de observar que el código suele volverse repetitivo y que muchos parámetros de los métodos suelen permanecer iguales, puesto que son para propósitos específicos.

B. Desarrollo de un RNG para OpenCL

Hasta el momento en que se elaboró este trabajo, OpenCL no cuenta con soporte para la generación de números pseudoaleatorios, por lo cual fue necesario desarrollar un RNG (Random Number Generator), que se logró extrayendo el código fuente referente a la clase RNG de la librería GSL de C (concretamente, el generador que se tomó para trabajar es el basado en mt19937). Se utilizó el código de este generador como base y se ajustó para implementarlo con OpenCL y que pudiera ejecutarse en el lado del kernel.

C. Algoritmo Genético a Implementar

En el algoritmo 1 se puede apreciar, en seudocódigo, el flujo del programa, indicando al final de cada línea en cual equipo se ejecuta la mayor parte (si en la CPU o en la GPU).

Algoritmo 1 Pseudocodigo general del AG

```
1: inicio AG
2: se genera la población inicial (GPU)
3: se ordenan los individuos de mejor a peor aptitud (GPU)
4: para toda i (Generaciones/Periodo migratorio entre nodos) (CPU) hacer
5:     para toda i en (Generaciones/Periodo migratorio entre bloques) (CPU) hacer
6:         LOGICA_AG (GPU)
7:         se ordenan los individuos de mejor a peor aptitud (GPU)
8:         migración copia a los mejores individuos del bloque siguiente en los peores del bloque actual (GPU)
9:     fin para
10:    búsqueda del mejor de cada bloque por reducción binaria (GPU)
11:    búsqueda del mejor individuo de todos los bloques (CPU)
12:    comparte el mejor resultado obtenido con los demás nodos (CPU)
13:    recibe el mejor resultado obtenido en otro nodo y valida si es mejor que el mejor local (CPU)
14: fin para
15: fin AG
```

En el algoritmo 2 se puede apreciar el flujo de la lógica del AG.

Algoritmo 2 Pseudocodigo de la lógica del AG

```
1: inicio LOGICA_AG
2: selección por torneo
3: cruce
4: mutación
5: búsqueda y actualización del mejor del bloque
6: fin LOGICA_AG
```

IV. RESULTADOS

Las funciones de prueba que se utilizaron en la implementación del AG son las propuestas por Suganthan et al [8] para la "Competencia de Computación Evolutiva 2005".

De la batería de 5 funciones unimodales y 7 funciones básicas propuestas, se seleccionaron las siguientes:

- Rastrigin:

$$f(x) = \sum_{i=1}^{D}(x_i^2 - 10\cos(2\pi x_i) + 10) \quad (1)$$

D: dimensiones

Propiedades:

- Multimodal, Desplazable, Separable, Escalable.
- Gran cantidad de óptimos locales.
- $X \in [-5, 5]^D$

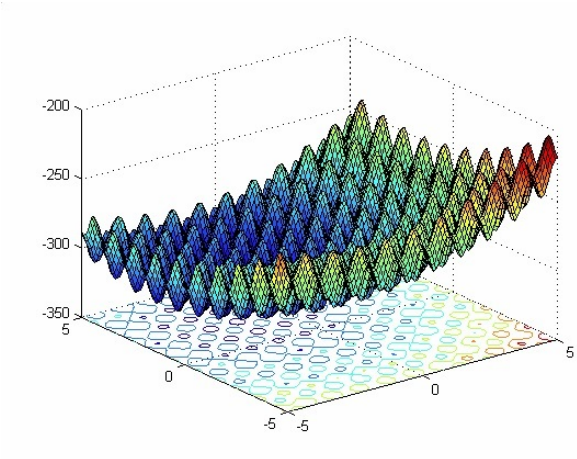

Fig. 1: Función de Rastrigín.

- Ackley:

$$f(x) = -20\exp(-0.2\sqrt{\frac{1}{D}\sum_{i=1}^{D}x_i^2}) - \exp(\frac{1}{D}\sum_{i=1}^{D}\cos(2\pi x_i)) + 20 + e \quad (2)$$

D: dimensiones

Propiedades:

- Multimodal, Rotable, Desplazable, No-Separable, Escalable.
- Optimo global en frontera.
- Si se inicializa la población cerca de la frontera, el problema se resuelve fácilmente.
- $X \in [-32, 32]^D$

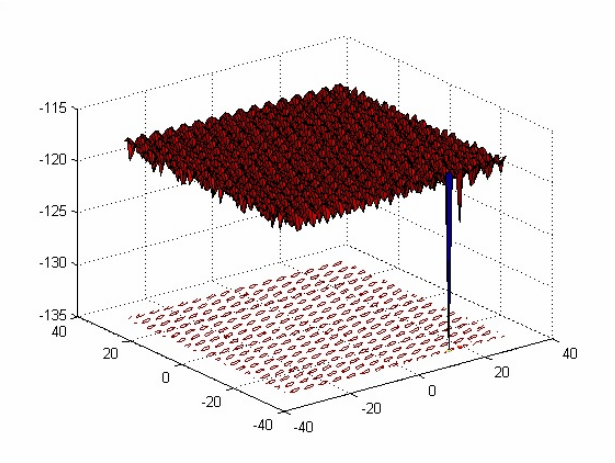

Fig. 2: Función de Ackley.

Equipo 1	
GPU	AMD Radeon R9 270X con 4GB de memoria global, 1280 núcleos y una frecuencia de reloj de 1050 MHz
Procesador	AMD FX(tm)-8320 Eight-Core Processor @ 3.50GHz x 4
RAM	8 GB a 1600 MHz
Librería	OpenCL 1.2
SO	Linux Mint 17.3 Cinnamon 64-bit

Tabla 1. Especificaciones del equipo 1

Equipo 2	
GPU	GeForce GT 730 con 2GB de memoria global, 96 núcleos y una frecuencia de reloj de 700 MHz
Procesador	Intel core i7-4790 CPU @ 3.60 GHz x 4
RAM	8 GB a 1600 MHz
Librería	OpenCL 1.1
SO	Linux Mint 17.3 Cinnamon 64-bit

Tabla 2. Especificaciones del equipo 2

En la optimización de las funciones propuestas, para espacios de búsqueda de 30 dimensiones, se utilizaron 14 bloques con 64 individuos, un periodo migratorio del 10% del total de generaciones, probabilidad de cruce de 0.16, probabilidad de mutación de 0.7, tamaño del torneo de 2 y se obtuvieron los siguientes resultados para 10 muestras:

Generaciones	Aciertos	Tiempo(s)
400	9	8.165438

Tabla 3. Resultados función Rastrigin

Generaciones	Aciertos	Tiempo(s)
400	10	7.794402

Tabla 4. Resultados función Ackley

V. CONCLUSIONES

Es posible implementar aplicaciones de cómputo paralelo por medio de un cluster heterogéneo integrado por tarjetas de gráficos de diferentes fabricantes, utilizando OpenCL como lenguaje de programación y OpenMPI como herramienta de comunicación entre los nodos del cluster. El punto clave reside en que el software propietario se enlaza en tiempo de ejecución, por medio de las librerías proporcionadas por el fabricante de cada tarjeta de gráficos (GPU). Estos resultados reviven la idea que dio origen a los clusters de computadoras de bajo costo: utilizar equipos en desuso, pero ahora incluyendo en el proceso las tarjetas de gráficos que estos equipos contienen.

REFERENCIAS

[1] I. Castro Liera, "Paralelización de Algoritmos de Optimización Basados en Poblaciones Usando GPGPU," p. 112, 2011.
[2] J. I. Mercado Bareño, "Implementación de un Cluster de GPGPU para Algoritmos de Optimización," no. 612, 2015.
[3] R. Tay, *OpenCL Parallel Programming Development Cookbook*. 2013.
[4] B. Gaster, L. Howes, D. R. Kaeli, P. Mistry, and D. Schaa, *Heterogeneous computing with OpenCL, 2nd Edition*. 2011.
[5] M. P. I. Open, "Open source high performance computing," 2012.
[6] E. Lusk, S. Huss, B. Saphir, and M. Snir, "MPI: A message-passing interface standard," 2009.
[7] F. Rothlauf, *Design of Modern Heuristics - Principles and Application*. 2011.
[8] P. N. Suganthan, N. Hansen, J. J. Liang, K. Deb, Y. P. Chen, A. Auger, and S. Tiwari, "Problem Definitions and Evaluation Criteria for the CEC 2006 Special Session on Constrained Real-Parameter Optimization," *KanGAL*, no. May, pp. 251–256, 2005.