Revista Cubana de Ciencias Informáticas
Vol. 10, No. 3, julio-septiembre, 2016
ISSN: 2227-1899 | RNPS: 2301
http://rcci.uci.cu
Pág. 42-56

Tipo de artículo: Artículo original
Temática: Reconocimiento de patrones
Recibido: 25/03/2016 | Aceptado: 30/06/2016

Una implementación de la meta-heurística "Optimización en Mallas Variables" en la arquitectura CUDA

An implementation for the meta-heuristic "Variable Mesh Optimization" on CUDA architecture

Ernesto Díaz López [1*], Adrian Martínez Prieto [1], Daniel Gálvez Lio [1,2]

[1] Departamento de Computación, Universidad Central "Marta Abreu de Las Villas", Carretera a Camajuaní Km 5 ½ Santa Clara, Villa Clara, Cuba. C.P: 54830. {ediaz, adrianmartinez}@uclv.cu
[2] Universidad Metropolitana del Ecuador (UMET), La Coruña N26-95 y San Ignacio, Quito, Ecuador.
dgalvez@uclv.edu.cu, d.galvez@umet.edu.ec

[*] Autor para correspondencia: ediaz@uclv.edu.cu

Resumen

En el campo de la optimización, tanto académico como industrial los problemas que se presentan a diario son a menudo complicados y NP-difícil. La solución computacional de los mismos no puede llevarse a cabo de una manera exacta en un plazo de tiempo razonable, y el uso de recursos de hardware de cómputo para esta solución aumenta considerablemente. Para lidiar con tal asunto, el diseño de los métodos de solución debe estar basado en el uso conjunto de los enfoques avanzados de la optimización combinatoria y los métodos de paralelismo a gran escala. En las últimas décadas, en la rama de la Inteligencia Artificial las meta-heurísticas basadas en poblaciones representan una poderosa herramienta para la solución de problemas complejos. El objetivo de este trabajo es exponer los resultados obtenidos a partir de la aplicación de técnicas paralelas utilizando la arquitectura CUDA a la meta-heurística "Optimización en Mallas Variables", con el propósito de lograr una mejora en el costo temporal. Las etapas más costosas del algoritmo en el proceso de expansión y contracción de la malla para la obtención de la nueva población inicial son implementadas en paralelo. Los resultados obtenidos fueron validados utilizando una versión secuencial de la meta-heurística. Los tiempos de ejecución de ambos algoritmos se comparan, mostrándose mejoras sustanciales en la versión paralela con el uso de CUDA.

Palabras clave: Meta-heurística, Optimización en Mallas Variables, Programación Paralela, CUDA.

Abstract

Revista Cubana de Ciencias Informáticas
Vol. 10, No. 3, julio-septiembre, 2016
ISSN: 2227-1899 | RNPS: 2301
http://rcci.uci.cu
Pág. 42-56

In the field of optimization, both academic and industrial problems that occur daily are often complicated and NP-hard. The computational solution thereof can't be performed in an accurate manner within a reasonable time, and use of computer hardware resources for this solution increases significantly. To address this issue, the design of the solution should be based on the joint use of advanced combinatorial optimization approaches and methods of large-scale parallelismIn recent decades, in the field of artificial intelligence, population based meta-heuristics represent a powerful tool for solving complex problems. The objective of this work is to expose the results obtained from the application of parallel techniques using the CUDA architecture to the meta-heuristic "Variable Mesh Optimization", with the purpose of achieving an improvement in the temporary cost. The most expensive steps of the algorithm in the process of expansion and contraction of the mesh to obtain a new initial population are implemented in parallel. The results were validated using a sequential version of the meta-heuristics. The runtimes of both algorithms are compared, showing substantial improvements in the parallel version using CUDA.

Keywords: *Meta-heuristic, Variable Mesh Optimization, Parallel Programming, CUDA.*

Introducción

En la sociedad actual los problemas de optimización están presentes prácticamente en todos los lugares. Muchos de estos problemas poseen una solución con un alto grado de complejidad o se hace muy difícil encontrar una solución exacta, por lo que se suele recurrir a métodos de solución aproximados. Las meta-heurísticas denotan un tipo general de métodos de aproximación sobre la base de las heurísticas, las cuales están basadas en estrategias para diversificar el esfuerzo de búsqueda y evitar caer en extremos locales. Durante las últimas décadas han demostrado un rendimiento extraordinario en la solución a problemas de optimización discreta y continua (Cáceres, 2009; Luong, 2011).

Teniendo en cuenta la complejidad computacional de los problemas, no debe sorprendernos que la computación paralela sea objetivo de investigaciones para mejorar el rendimiento de las meta-heurísticas, tanto en el caso discreto como continuo (Luong, 2011).

Actualmente, el uso de las Unidades de Procesamiento Gráfico (*Graphics Processing Unit*; GPU) ha aumentado extraordinariamente, en principio por el acelerado desarrollo que ha alcanzado la industria de los video juegos, arrastrando consigo la necesidad de desarrollar nuevas GPU capaces de aumentar sus prestaciones y los recursos disponibles que den soporte a las nuevas creaciones de esta industria. Estas unidades están disponibles en muchos equipos: computadoras portátiles, máquinas de escritorio, e incluso hasta en los teléfonos móviles. Pero no solo eso, estas unidades, además de ser utilizadas en aplicaciones gráficas y de video, se han extendido a otros dominios de

aplicación, como el procesamiento paralelo en la computación científica gracias a la aparición de la Arquitectura Unificada de Dispositivos de Cómputo (*Compute Unified Device Architecture*; CUDA), el nuevo conjunto de herramientas de desarrollo de propósito general de NVIDIA sobre la GPU, también conocido este concepto como Computación de Propósito General sobre Unidades de Procesamiento Gráfico (*General Purpose Graphics Processing Unit*; GPGPU) (González, 2014; Owens et al., 2007).

La programación en CUDA se ha convertido en un nuevo paradigma de programación paralela, aunque se mantienen los principios generales, incluye características especiales como nuevos tipos de memoria y un modelo de ejecución de hilos diferente. CUDA intenta explotar las ventajas de la GPU frente a la Unidad Central de Procesamiento (*Central Processing Unit*; CPU) de propósito general utilizando el paralelismo que ofrecen sus múltiples núcleos, permitiendo la ejecución de un gran número de hilos de manera simultánea, procesando la misma secuencia de instrucciones de forma secuencial por cada hilo ejecutado de manera paralela (Corporation, 2014).

En algunas áreas como la computación numérica, se está presenciando una proliferación de bibliotecas de software como CUBLAS para el trabajo con Álgebra Lineal Básica en la GPU, sin embargo, en otras áreas como la optimización combinatoria, especialmente en meta-heurística, la utilización de la GPU no crece al mismo paso (Luong, 2011). Debido a la importancia de esta área dentro de la investigación científica y en el campo de la Inteligencia Artificial se ha generado un interés cada vez mayor por el desarrollo de la misma.

Por ese motivo, se desarrolló una implementación paralela utilizando técnicas paralelas sobre la arquitectura CUDA de la meta-heurística poblacional Optimización en Mallas Variables (VMO), desarrollada en el Centro de Estudios de Informática de la Universidad Central ¨Marta Abreu¨ de Las Villas.

En su implementación se hizo uso del conjunto de herramientas que ofrece NVIDIA, integrado con el lenguaje de programación C++. Luego se realizó un análisis estadístico de la versión paralela respecto a su versión secuencial, verificándose una mejora significativa en el costo computacional de la misma.

En el presente artículo, después de una breve introducción sobre la computación paralela y su uso en los problemas de optimización, se exponen de manera general conceptos esenciales de meta-heurística, algunos temas relacionados con las meta-heurísticas paralelas y su vinculación con la GPU, así como una breve caracterización de la meta-heurística poblacional VMO. Finalmente se presentan los resultados obtenidos del análisis de las comparaciones entre la versión secuencial y paralela del algoritmo, seguido de las conclusiones y referencias bibliográficas utilizadas.

Revista Cubana de Ciencias Informáticas
Vol. 10, No. 3, julio-septiembre, 2016
ISSN: 2227-1899 | RNPS: 2301
http://rcci.uci.cu
Pág. 42-56

Materiales y métodos

Meta-heurística y GPU

En muchas ocasiones nos enfrentamos a problemas en los que encontrar la mejor solución trae consigo un alto grado de complejidad computacional o en algunos casos no se cuenta con un algoritmo para encontrar esta solución. Para resolver este tipo de problemas se pueden emplear los métodos de búsqueda heurísticos los cuales son capaces de encontrar una aproximación a una posible solución del problema, estos métodos están orientados a reducir la cantidad de búsqueda requerida para encontrar una solución. De lo que se desprende la siguiente definición (Hooker, 1995):

Un método heurístico es un procedimiento para resolver un problema complejo de optimización mediante una aproximación intuitiva, en la que la estructura del problema se utiliza de forma inteligente para obtener una buena solución de manera eficiente.

En los últimos años se ha presenciado un aumento en el desarrollo de procedimientos heurísticos para resolver problemas de optimización. Este hecho queda claramente reflejado en la creación de revistas especializadas para la difusión de este tipo de procedimientos; como es el caso de la revista "*Journal of Heuristics*"[1] editada por primera vez en el año 1995 (Cáceres, 2009).

Con el desarrollo de la ciencia a través de los años se han desarrollado un conjunto de métodos bajo el nombre de meta-heurístico con el propósito de obtener mejores resultados que los alcanzados por los heurísticos tradicionales (Osman & Kelly, 1996).

Los procedimientos meta-heurísticos son una clase de métodos aproximados los cuales han sido diseñados para resolver problemas complejos de optimización. Los meta-heurísticos proporcionan un marco general para crear nuevos algoritmos híbridos combinando diferentes conceptos derivados de la inteligencia artificial, la evolución biológica y los mecanismos estadísticos (Glover & Kochenberger, 2003).

Debido al gran desarrollo que han alcanzado los procedimientos meta-heurísticos, se han realizado diversas propuestas de clasificación de los mismos, dentro de las que se encuentran las siguientes (Blum & Roli, 2003):

- Meta-heurísticas de trayectoria simple: se utiliza el término de trayectoria simple porque el proceso de búsqueda que desarrollan estos métodos se caracteriza por una trayectoria en el espacio de soluciones; es decir, partiendo de una solución inicial, son capaces de generar un camino o trayectoria en el espacio de búsqueda a través de operaciones de movimiento. Algunas de estas meta-heurísticas son las siguientes: la Búsqueda Tabú (*Taboo Search*) (Baykasoglu,

[1] url: http://www.springer.com/math/applications/journal/10732

Revista Cubana de Ciencias Informáticas
Vol. 10, No. 3, julio-septiembre, 2016
ISSN: 2227-1899 | RNPS: 2301
http://rcci.uci.cu
Pág. 42-56

Owen, & Gindy, 1999; Klinowski, 2015; Taillard, 1991), Recocido Simulado (*Simulated Annealing*) (Davis, 1987; Goffe, Ferrier, & Rogers, 1994; Szu & Hartley, 1987), Búsqueda de Vecindades Variables (*Variable Neighborhood Search*) (P Hansen & Mladenović, 1997; Pierre Hansen & Mladenović, 1999; Mladenović & Hansen, 1997), Búsqueda Local Guiada (*Guided Local Search*) (Voudouris & Tsang, 1995, 2015), Búsqueda Local Iterativa (*Iterated Local Search*) (Kramer, 2014; Lourenço, Martin, & Stützle, 2010; Stützle, 2006), entre otras.

- Meta-heurísticas poblacionales: Las meta-heurísticas basadas en población o meta-heurísticas poblacionales, son aquellas que emplean un conjunto de soluciones (población) en cada iteración del algoritmo, en lugar de utilizar una única solución como las meta-heurísticas del grupo anterior. Estas proporcionan de forma intrínseca un mecanismo de exploración paralelo del espacio de soluciones, y su eficacia depende en gran medida de cómo se manipule dicha población. Dentro de esta clasificación se destacan los Algoritmos Evolutivos (*Evolutionary Algorithms*; EA) (B, 1996) y los algoritmos basados en Inteligencia Colectiva (*Swarm Intelligence*; SI) (Blum, 2015; Engelbrecht, 2005; Parsopoulos & Vrahatis, 2002).

En la **Figura 1** se muestra una taxonomía de las principales meta-heurísticas.

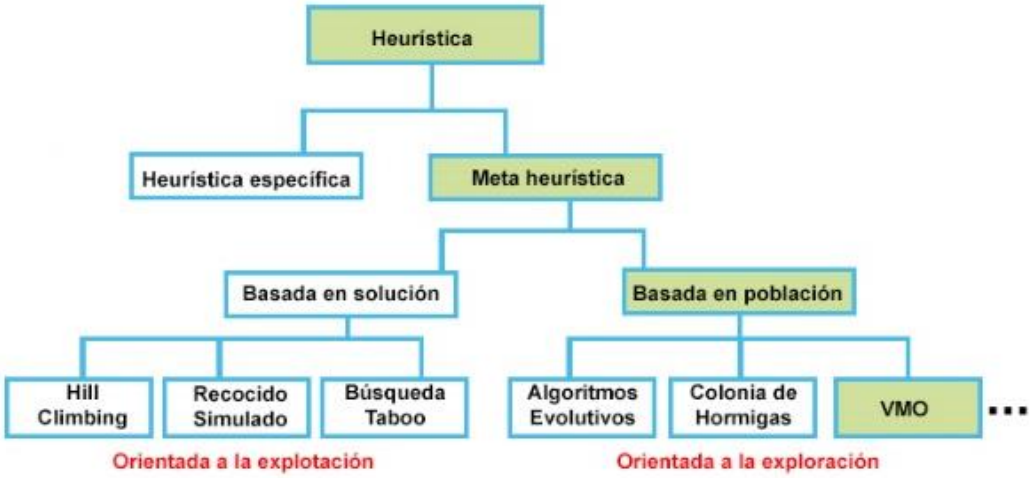

Figura 1. Taxonomía de los métodos heurísticos

Meta-heurísticas paralelas

El uso de meta-heurísticas paralelas permite reducir la complejidad computacional del proceso de búsqueda considerablemente, en particular, cuando la función objetivo y las restricciones relacionadas con el problema de optimización necesitan grandes recursos de cómputo y el tamaño del espacio de búsqueda es enorme.

El paralelismo aparece naturalmente cuando en una población de soluciones (o un vecindario), cada una de las soluciones que pertenecen a la misma es una unidad independiente. Debido a esto, el rendimiento de las meta-heurísticas mejora significativamente cuando se ejecuta en paralelo (Luong, 2011).

La computación paralela y distribuida puede ser usada en el diseño e implementación de meta-heurísticas por las razones siguientes:

- Acelerar la búsqueda: uno de los objetivos principales de paralelizar una meta-heurística es reducir el tiempo de búsqueda. Esto es un aspecto crucial para algunas clases de problemas donde tenemos requisitos en los tiempos de búsqueda.
- Mejorar la robustez: una meta-heurística paralela podría ser más robusta en correspondencia con solucionar problemas de optimización diferentes y ejemplos diferentes de un problema en particular de una manera eficaz. La robustez también puede ser medida en relación con la sensibilidad de la meta-heurística hacia sus parámetros.
- Mejorar la calidad de las soluciones obtenidas: algunos modelos de meta-heurísticas paralelas permiten mejorar la calidad de la búsqueda. Ciertamente, intercambiar información entre meta-heurísticas cooperativas modificará su comportamiento en términos de búsqueda en el ámbito asociado al problema. El objetivo principal de una cooperación paralela entre las meta-heurísticas es mejorar la calidad de las soluciones.

Solucionar problemas a gran escala: las meta-heurísticas paralelas permiten solucionar ejemplos complicados de problemas de optimización a gran escala. Un ejemplo de ello es solucionar más modelos matemáticos exactos relacionados con problemas de optimización diferentes.

Meta-heurística y GPU

Durante años, el uso de procesadores gráficos se ha dedicado a aplicaciones gráficas. Conducido por la demanda de gráficos 3D de alta definición sobre computadoras personales, la GPU ha evolucionado hacia un alto paralelismo, un ambiente multi-hilos y muchos procesadores. Realmente, esta arquitectura posee un alto poder de procesamiento computacional y un gran ancho de banda de memoria comparado con la CPU tradicional. En la **Figura 2** tomada de (Corporation, 2014) se muestran las arquitecturas de la CPU y la GPU respectivamente.

Como se puede apreciar la CPU no tiene un gran número de Unidades Aritmético Lógicas (ALU), pero si posee una gran caché y una unidad de control importante. Como resultado, la CPU está especializada para manejar múltiples y diferentes tareas en paralelo que requieren muchos datos. Así, los datos son almacenados dentro de la caché para acelerar sus accesos. La unidad de control manejará el flujo de instrucciones para maximizar la ocupación en las ALU, y optimizar la gestión de la caché.

Revista Cubana de Ciencias Informáticas
Vol. 10, No. 3, julio-septiembre, 2016
ISSN: 2227-1899 | RNPS: 2301
http://rcci.uci.cu
Pág. 42-56

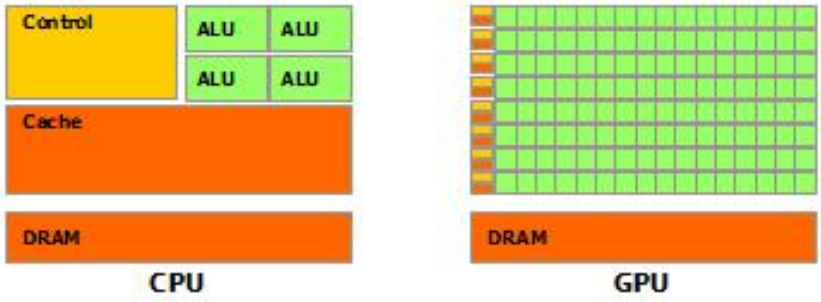

Figura 2. Distribución de las unidades de procesamiento en la CPU y GPU

Contrario a esto, se puede observar como en la GPU se tiene un gran número de ALU, con un caché limitado y pocas unidades de control. Esto permite que la GPU se especialice más en calcular de forma masiva y paralela un conjunto de elementos pequeños e independientes, mientras se tiene un gran flujo de datos a procesar. Debido a que más transistores están dedicados al procesamiento de datos, en vez de al almacenamiento de los mismos y al control del flujo de instrucciones (Corporation, 2014).

La optimización combinatoria paralela sobre GPU requiere un esfuerzo enorme en el diseño. Algunas de las consideraciones más importantes son: la distribución eficiente del procesamiento de datos entre CPU y GPU, la sincronización de los hilos, la optimización de la transferencia de datos entre las diferentes memorias, las restricciones de capacidad de estas memorias, entre otras. Tales asuntos deben ser tenidos en cuenta para el diseño de modelos meta-heurísticos paralelos con el objetivo de solucionar problemas de optimización a gran escala sobre arquitecturas de GPU (Luong, 2011).

En general, para las arquitecturas distribuidas, el rendimiento global en meta-heurísticas se ve limitado por los retrasos de comunicación y el rendimiento alcanzado en los accesos a la memoria en las arquitecturas GPU. Efectivamente, cuando se realiza un estudio de las soluciones paralelas, se puede apreciar que el principal obstáculo en las arquitecturas distribuidas es la eficiencia de comunicación.

Caracterización de la meta-heurística VMO para espacios continuos

La Optimización Basada en Mallas Variables (*Variable Mesh Optimization*; VMO) es una meta-heurística poblacional con características evolutivas donde un conjunto de nodos que representan soluciones potenciales a un problema de optimización, forman una malla (población) que se expande y contrae dinámicamente, desplazándose por el espacio de búsqueda. Esta meta-heurística consta esencialmente de cuatro etapas, en las tres primeras se realiza un

proceso de expansión en cada ciclo, donde se generan nuevos nodos en dirección a los extremos locales (nodos de la malla con mejor calidad en distintas vecindades) y el extremo global (nodo obtenido de mejor calidad en todo el proceso desarrollado), así como a partir de los nodos fronteras de la malla (nodos de mayor y menor norma). Luego en la última etapa se realiza un proceso de contracción de la malla, donde los mejores nodos resultantes en cada iteración son seleccionados como malla inicial para la iteración siguiente. La formulación general de esta meta-heurística abarca tanto los problemas de optimización continuos como los discretos (Cáceres, 2009).

La esencia del método VMO es crear una malla de puntos en el espacio m dimensional, sobre la cual se realiza el proceso de optimización de una función FO (x1, x2,…, xm); la cual se mueve mediante un proceso de expansión hacia otras regiones del espacio de búsqueda. Esta malla se hace más "fina" en aquellas zonas que parecen ser más promisorias. Se dice que es una malla dinámica en el sentido que la malla cambia su tamaño (cantidad de nodos) y configuración durante el proceso de búsqueda. Los nodos se representan como vectores de la forma n (x1, x2,…, xm). El proceso de generación de la nueva población en cada iteración comprende los siguientes pasos:

1. Generación de la malla inicial.
2. Generación de nodos en dirección a los extremos locales (nl).
3. Generación de nodos en dirección al extremo global (ng).
4. Generación de nodos a partir de las fronteras de la malla (nf).

El método incluye los siguientes parámetros de configuración:

- Cantidad de nodos de la malla inicial (Ni).
- Cantidad máxima de nodos de la malla en cada ciclo (N), donde $3 \cdot Ni < N$.
- Tamaño de la vecindad (k).
- Condición de parada (M), generalmente se establece una cantidad de evaluaciones de la FO.

En (Bello, Puris, & Falc, 2008; Cáceres, 2009; Navarro, Bello, & Abraham, 2013; Puris, Bello, & Molina, 2012) se presenta una descripción más detallada de cada una de las etapas de la meta-heurística VMO.

A continuación se presenta la estructura general del algoritmo VMO (ver **Figura 3**), como se puede apreciar la condición de parada está descrita por un número máximo de evaluaciones de la función objetivo.

Revista Cubana de Ciencias Informáticas
Vol. 10, No. 3, julio-septiembre, 2016
ISSN: 2227-1899 | RNPS: 2301
http://rcci.uci.cu
Pág. 42-56

1. Generación de la malla inicial (*Ni*) de forma aleatoria. 2. Evaluar los nodos de la malla inicial, seleccionar el mejor (*ng*). 3. **Repetir:** 4. **Para cada** nodo en la malla inicial del ciclo **hacer** 5. Encontrar sus (*k*) nodos más cercanos. 6. Determinar el mejor de los vecinos (*ne*). 7. **Si** (*ne*) es mejor que el nodo actual, **entonces** 8. Generar nuevo nodo usando ecuación (2.1). 9. **Fin Si** 10. **Fin Para**	11. **Para cada** nodo en la malla inicial del ciclo **hacer** 12. Generar un nuevo nodo usando la ecuación (2.2). 13. **Fin Para** 14. Seleccionar los nodos más externos de la malla. 15. Generar nuevo nodo usando la ecuación (2.3). 16. **Mientras** los *N* nodos de la malla en el ciclo actual **no se hayan generado** 17. Generar nuevo nodo aleatoriamente. 18. Construir la malla inicial actual de manera elitista. 19. Ordenar los nodos de la malla inicial según su calidad. 20. Aplicar operador de limpieza adaptativo. 21. **Hasta** *M* evaluaciones

Figura 3. Algoritmo general VMO

Resultados y discusión

Para desarrollar la versión paralela de la meta-heurística VMO se utiliza un diseño cooperativo entre la CPU y la GPU, siguiendo la misma metodología de la versión secuencial, contando esta nueva versión con las mismas etapas. Se implementa en paralelo la generación aleatoria de la malla inicial, para lo que se utiliza la librería cuRAND incluida en el conjunto de herramientas de CUDA, también fueron paralelizadas las generaciones de las nuevas soluciones en dirección a los extremos locales, extremo global y fronteras de la malla, para el proceso de contracción de la malla se construye la matriz de distancia de las soluciones en la GPU.

En la realización de este trabajo se seleccionaron 15 funciones de referencia para problemas de optimización a gran escala, este banco de pruebas está disponible en la organización de una competición en asociación con la IEEE CEC 2015 Sesión Especial sobre Optimización Global a Gran Escala (2015 *IEEE CEC Special Session on Large Scale Global Optimization*) (Li, Xiaodong; Tang, Ke; Yang, Zhenyu; Molina, 2015). Esta competición permite que los participantes ejecuten sus propios algoritmos sobre estas 15 funciones de referencia. El código fuente de estas funciones está disponible en tres lenguajes de programación: Java, C++ y MatLab, en el caso específico de este trabajo se utilizó el código en C++ (Li, Tang, Omidvar, Yang, & Qin, 2013).

Revista Cubana de Ciencias Informáticas
Vol. 10, No. 3, julio-septiembre, 2016
ISSN: 2227-1899 | RNPS: 2301
http://rcci.uci.cu
Pág. 42-56

Análisis estadístico de los resultados

La implementación de la meta-heurística VMO fue realizada en el lenguaje de programación C++ utilizando el IDE de desarrollo Netbeans 7.4 en el sistema operativo Debian 7.3, sobre un microprocesador Intel Core 2 Quad a 2.4 GHz con 3 Gb de memoria RAM DDR2 y una tarjeta de video Geforce GT 630, 192 núcleos, 2Gb DDR3.

En los experimentos realizados sobre los algoritmos implementados se seleccionaron los siguientes parámetros de configuración:

- Problemas: 15 problemas de minimización.
- Dimensión: 1000.
- Cantidad de nodos iniciales (Ni): 12.
- Cantidad total de nodos (N): 42.
- Tamaño de la vecindad (k): 3.
- Condición de parada (M): 1.2x106.
- Cantidad de corridas por problema: 15.

Primeramente se realiza el análisis descriptivo de los datos a ambas versiones para observar de manera general el comportamiento de las soluciones encontradas, obteniéndose los resultados mostrados en Tabla 1 y Tabla 2.

Tabla 1: Evaluación de la FO (Secuencial).

	f_1	f_2	f_3	f_4	f_5	f_6	f_7	f_8
Mejor	2,30E+11	5,88E+04	21,60	1,80E+13	3,91E+07	1,07E+06	8,86E+14	1,15E+18
Mediana	4,06E+11	1,15E+05	21,6727	7,44E+13	9,13E+07	1,08E+06	4,07E+16	5,81E+18
Peor	5,22E+11	1,43E+05	21,75	5,62E+14	2,08E+08	1,08E+06	7,07E+17	1,73E+19
Media	4,07E+11	1,03E+05	21,6661	1,45E+14	1,03E+08	1,08E+06	1,74E+17	6,21E+18
Des. Est.	6,55E+10	3,05E+04	0,04958	1,60E+14	4,91E+07	3,47E+03	2,45E+17	4,86E+18

	f_9	f_{10}	f_{11}	f_{12}	f_{13}	f_{14}	f_{15}	-
Mejor	2,96E+09	9,45E+07	9,22E+12	3,12E+12	1,04E+15	5,76E+15	3,87E+17	
Mediana	6,06E+09	9,76E+07	1,59E+19	8,36E+12	6,52E+18	9,14E+18	3,47E+18	
Peor	2,05E+10	9,86E+07	1,28E+23	9,61E+12	6,98E+20	7,94E+19	1,72E+19	
Media	8,62E+09	9,72E+07	8,68E+21	8,14E+12	1,20E+20	1,91E+19	4,61E+18	
Des. Est.	5,33E+09	1,29E+06	3,30E+22	1,69E+12	2,10E+20	2,42E+19	4,76E+18	

Tabla 2: Evaluación de la FO (paralela).

	f_1	f_2	f_3	f_4	f_5	f_6	f_7	f_8
Mejor	2,40E+11	5,24E+04	21,60	8,56E+12	3,73E+07	1,06E+06	5,12E+14	5,94E+17
Mediana	3,36E+11	9,43E+04	21,6567	6,63E+13	8,01E+07	1,08E+06	3,05E+16	4,31E+18
Peor	4,64E+11	1,21E+05	21,75	8,62E+14	1,21E+08	1,08E+06	7,23E+19	1,58E+19
Media	3,61E+11	9,39E+04	21,6628	1,51E+14	7,90E+07	1,07E+06	1,11E+19	5,29E+18

Revista Cubana de Ciencias Informáticas
Vol. 10, No. 3, julio-septiembre, 2016
ISSN: 2227-1899 | RNPS: 2301
http://rcci.uci.cu
Pág. 42-56

Des. Est.	6,96E+10	2,50E+04	0,04575	2,48E+14	2,45E+07	7,24E+03	2,45E+19	4,49E+18

	f_9	f_{10}	f_{11}	f_{12}	f_{13}	f_{14}	f_{15}	-
Mejor	2,78E+09	9,50E+07	3,00E+17	1,80E+12	6,78E+16	5,49E+16	8,05E+15	
Mediana	6,69E+09	9,74E+07	3,08E+19	5,36E+12	5,94E+17	2,25E+19	1,98E+18	
Peor	1,79E+10	9,90E+07	1,75E+22	9,84E+12	2,01E+20	1,35E+23	1,12E+19	
Media	7,34E+09	9,73E+07	1,41E+21	5,57E+12	1,67E+19	9,07E+21	2,45E+18	
Des. Est.	3,60E+09	1,30E+06	4,49E+21	2,46E+12	5,13E+19	3,48E+22	2,94E+18	

Para comparar algoritmos sobre múltiples conjuntos de datos se utilizan diversas técnicas estadísticas con el objetivo de determinar la existencia o no de diferencias significativas entre ellos. En este trabajo se usarán las pruebas no paramétricas, debido a que se desea establecer una comparación ente dos muestras relacionadas, se utilizará el test de Wilcoxon.

Como se muestra en la Tabla 3 y Tabla 4 los valores de significación son mayores a 0.05, aceptándose la hipótesis nula de que la mediana de las diferencias entre las soluciones de ambas versiones es igual a cero, en otras palabras la hipótesis de igualdad es aceptada, garantizándose que no existen diferencias significativas en el comportamiento entre los dos algoritmos.

Tabla 3. Resultados del Test de Wilcoxon (1)

Rangos				
		N	Rango promedio	Suma de rangos
P Eval. FO - S Eval. FO	Rangos negativos	129 [a]	109,84	14169,00
	Rangos positivos	95 [b]	116,12	11031,00
	Empates	1 [c]		
	Total	225		
a. P Eval. FO < S Eval. FO				
b. P Eval. FO > S Eval. FO				
c. P Eval. FO = S Eval. FO				

Tabla 4: Resultados del Test de Wilcoxon (2)

Estadísticos de contraste [a]	
	P Eval FO - S Eval FO
Z	-1,616 [b]
Sig. asintót. (bilateral)	,106
a. Prueba de los rangos con signo de Wilcoxon	
b. Basado en los rangos positivos.	

Análisis temporal de los algoritmos

Se realizó el análisis descriptivo de los tiempos de ejecución a ambas versiones para observar de manera general las ventajas de la versión paralela sobre la versión secuencial, éste tiempo está expresado en milisegundos, dado que en la implementación de ambos algoritmos no se realizaron cambios en la forma de evaluar las funciones, en este análisis

Revista Cubana de Ciencias Informáticas
Vol. 10, No. 3, julio-septiembre, 2016
ISSN: 2227-1899 | RNPS: 2301
http://rcci.uci.cu
Pág. 42-56

se tomó la diferencia entre el tiempo total de ejecución y el tiempo de evaluación de la función objetivo, obteniéndose los resultados mostrados en Tabla 5 y Tabla 6.

Tabla 5. Tiempo de ejecución VMO (Secuencial)

	f_1	f_2	f_3	f_4	f_5	f_6	f_7	f_8
Mejor	317,55	358,95	261,45	277,16	288,48	270,74	286,73	277,86
Peor	664,88	622,59	1084,75	287,78	670,22	295,31	418,78	550,02
Media	363,50	436,77	567,35	282,796	362,61	278,97	305,19	301,69

	f_9	f_{10}	f_{11}	f_{12}	f_{13}	f_{14}	f_{15}	-
Mejor	289,16	273,81	288,60	350,22	266,52	261,82	307,86	
Peor	830,65	1108,30	744,33	428,17	640,02	521,39	531,14	
Media	397,45	454,28	331,88	388,29	310,25	284,88	358,21	

Tabla 6. Tiempo de ejecución VMO (paralela)

	f_1	f_2	f_3	f_4	f_5	f_6	f_7	f_8
Mejor	113,51	119,79	114,90	111,91	113,22	113,51	102,94	113,46
Peor	119,56	126,19	119,32	116,01	123,92	124,38	108,06	118,39
Media	116,95	122,53	116,32	114,26	117,28	117,91	105,98	115,37

	f_9	f_{10}	f_{11}	f_{12}	f_{13}	f_{14}	f_{15}	-
Mejor	111,57	111,53	111,33	106,75	109,28	108,64	108,53	
Peor	118,87	118,15	115,64	112,55	115,64	112,38	119,81	
Media	114,26	113,79	113,74	110,76	111,10	111,28	114,97	

Como se puede apreciar existe menor variabilidad en los tiempos de ejecución del algoritmo paralelo que en su versión secuencial, observándose mejoras significativas en el mismo.

Para calcular la aceleración que representa los tiempos de ejecución del algoritmo paralelo con respecto al secuencial se utilizó la siguiente ecuación:

$$Aceleración = \frac{Tiempo\ de\ ejecución\ Secuencial}{Tiempo\ de\ ejecución\ Paralelo}$$

Al utilizar la anterior ecuación se obtuvo en promedio una aceleración de 3.1577, representando el tiempo de ejecución paralelo un 35,02 % del tiempo de ejecución secuencial.

Conclusiones

Como resultado de este trabajo se implementó una versión paralela de la meta-heurística poblacional VMO utilizando las ventajas de paralelización que ofrece la arquitectura CUDA.

Haciendo uso del alto grado de paralelismo que posee la obtención de nuevas soluciones en la meta-heurística VMO, ejecutándose sobre la arquitectura CUDA se logró obtener una disminución aproximada de un 35,02 % en el tiempo de procesamiento de la versión secuencial a la paralela, con una aceleración promedio de 3,1577.

A partir de las recientes características incorporadas por CUDA, la paralelización de la evaluación de la función objetivo y un mejor manejo de memoria en el algoritmo serán líneas futuras de trabajo.

Referencias

B, T. (1996). *Evolutionary algorithms in theory and practice: evolution strategies, evolutionary programming, genetic algorithms*. Oxford University Press.

BAYKASOGLU, A., OWEN, S., & GINDY, N. (1999). A taboo search based approach to find the pareto optimal set in multiple objective optimization. *Engineering Optimization*, *31*(6), 731–748. http://doi.org/10.1080/03052159908941394

BELLO, R., PURIS, A., & FALC, R. (2008). Feature Selection through Dynamic Mesh Optimization.

BLUM, C. (2015). *Swarm Intelligence in Optimization*. http://doi.org/10.1007/978-3-540-74089-6

BLUM, C., & ROLI, A. (2003). Metaheuristics in combinatorial optimization: Overview and conceptual comparison. *ACM COMPUTING SURVEYS*, *35*, 268–308.

CÁCERES, A. Y. P. (2009). *Desarrollo de meta-heurísticas poblacionales para la solución de problemas complejos*. *Departamento de Ciencia de la Computación*. Universidad Central “Marta Abreu” de Las Villas, Santa Clara.

CORPORATION, N. (2014). *CUDA C Programming Guide*. Retrieved from www.nvidia.com

DAVIS, L. (1987). Genetic algorithms and simulated annealing.

ENGELBRECHT, A. P. (2005). *Fundamentals of computational swarm intelligence* (Vol. 1). Wiley Chichester.

GLOVER, F., & KOCHENBERGER, G. A. (2003). *Handbook of metaheuristics*. Kluwer Academic

Publishers.

GOFFE, W. L., FERRIER, G. D., & ROGERS, J. (1994). Global optimization of statistical functions with simulated annealing. *Journal of Econometrics*, *60*(1), 65–99. http://doi.org/http://dx.doi.org/10.1016/0304-4076(94)90038-8

GONZÁLEZ, O. E. L. (2014). *Aplicación de técnicas paralelas utilizando CUDA, al proceso de simulación de poblaciones en secuencias genéticas.* Universidad Central "Marta Abreu" de Las Villas, Santa Clara.

HANSEN, P., & MLADENOVIĆ, N. (1997). Variable neighborhood search for the p-median. *Location Science*, *5*(4), 207–226. http://doi.org/http://dx.doi.org/10.1016/S0966-8349(98)00030-8

HANSEN, P., & MLADENOVIĆ, N. (1999). An Introduction to Variable Neighborhood Search. In S. Voß, S. Martello, I. Osman, & C. Roucairol (Eds.), *Meta-Heuristics SE - 30* (pp. 433–458). Springer US. http://doi.org/10.1007/978-1-4615-5775-3_30

HOOKER, J. N. (1995). Testing Heuristics: We Have It All Wrong. *Journal of Heuristics*, *1*, 33–42.

KLINOWSKI, J. (2015). Taboo search : An approach to the multiple minima problem for continuous functions, (November).

KRAMER, O. (2014). Iterated Local Search. In *A Brief Introduction to Continuous Evolutionary Optimization SE - 5* (pp. 45–54). Springer International Publishing. http://doi.org/10.1007/978-3-319-03422-5_5

LI, X., TANG, K., OMIDVAR, M. N., YANG, Z., & QIN, K. (2013). Benchmark Functions for the CEC'2013 Special Session and Competition on Large-Scale Global Optimization.

LI, XIAODONG; TANG, KE; YANG, ZHENYU; MOLINA, D. (2015). Benchmark Functions for the CEC'2015 Special Session and Competition on Large Scale Global Optimization. Retrieved from http://goanna.cs.rmit.edu.au/~xiaodong/cec13-lsgo/competition/

LOURENÇO, H., MARTIN, O., & STÜTZLE, T. (2010). Iterated Local Search: Framework and Applications. In M. Gendreau & J.-Y. Potvin (Eds.), *Handbook of Metaheuristics SE - 12* (Vol. 146, pp. 363–397). Springer US. http://doi.org/10.1007/978-1-4419-1665-5_12

LUONG, T. VAN. (2011). Parallel Metaheuristics on GPU. PhdThesis

MLADENOVIĆ, N., & HANSEN, P. (1997). Variable neighborhood search. *Computers & Operations Research*, *24*(11), 1097–1100. http://doi.org/http://dx.doi.org/10.1016/S0305-0548(97)00031-2

NAVARRO, R., BELLO, R., & ABRAHAM, A. (2013). Niche-Clearing-based Variable Mesh Optimization for Multimodal Problems.

OSMAN, I. H., & KELLY, J. P. (1996). *Meta-heuristics: theory and applications*. Kluwer Academic.

OWENS, J. D., LUEBKE, D., GOVINDARAJU, N., HARRIS, M., KRÜGER, J., LEFOHN, A. E., & PURCELL, T. J. (2007). A Survey of General-Purpose Computation on Graphics Hardware. *Computer Graphics Forum*, *26*(1), 80–113. http://doi.org/10.1111/j.1467-8659.2007.01012.x

PARSOPOULOS, K. E., & VRAHATIS, M. N. (2002). Recent approaches to global optimization problems through Particle Swarm Optimization. *Natural Computing*, *1*, 235–306.

PURIS, A., BELLO, R., & MOLINA, D. (2012). Variable mesh optimization for continuous optimization problems, 511–525. http://doi.org/10.1007/s00500-011-0753-9

STÜTZLE, T. (2006). Iterated local search for the quadratic assignment problem. *European Journal of Operational Research*, *174*(3), 1519–1539. http://doi.org/http://dx.doi.org/10.1016/j.ejor.2005.01.066

SZU, H., & HARTLEY, R. (1987). Fast simulated annealing. *Physics Letters A*, *122*(3), 157–162. http://doi.org/http://dx.doi.org/10.1016/0375-9601(87)90796-1

TAILLARD, E. (1991). Robust taboo search for the quadratic assignment problem. *Parallel Computing*, *17*(4–5), 443–455. http://doi.org/http://dx.doi.org/10.1016/S0167-8191(05)80147-4

VOUDOURIS, C., & TSANG, E. (1995). Guided Local Search, (August).

VOUDOURIS, C., & TSANG, E. (2015). Partial Constraint Satisfaction Problems and Guided Local Search, (DECEMBER 2000).