ISSN 1870-4069

Obtención de parámetros de la prueba de la estrella utilizando algoritmos genéticos y cómputo de alto rendimiento

Juan Miguel Pérez-García, Moisés Emmanuel Ramírez-Guzmán, Jorge González-García

Universidad Tecnológica de la Mixteca,
Instituto de Computación,
México

jnxho123d@gmail.com, {merg, jgonzal}@mixteco.utm.mx

Resumen. El uso de cómputo de alto rendimiento para la reducción del tiempo en la obtención de los parámetros para la prueba de la estrella usando algoritmos genéticos es de gran importancia debido a que estos algoritmos tardan un largo periodo de tiempo en dar respuesta si se ejecutan en computadoras convencionales, por lo que se opta en el uso de plataformas como CUDA. Además, las estructuras de datos que utilizan los algoritmos genéticos, se acoplan perfectamente a la estructura de ejecución de CUDA. La implementación del algoritmo, tomando en cuenta lo anterior, se lleva a cabo eficientemente y de esta manera se puede demostrar que la obtención de los parámetros se realiza en menor tiempo. Esta reducción de tiempo permite llevar a cabo una mayor cantidad de pruebas y hacer modificaciones a los parámetros del algoritmo para poder así reducir las aberraciones en el sistema óptico.

Palabras clave: Óptica, prueba de la estrella, algoritmos genéticos, CUDA.

Obtaining Parameters of the Star Test using Genetic Algorithms and High-Performance Computing

Abstract. The use of high-performance computation to reduce the time in obtaining the parameters for the star test using genetic algorithms is of great importance because these algorithms take a long period of time to respond if they are executed in Conventional computers, so you choose to use platforms like CUDA. Furthermore, the data structures that the genetic algorithms use fit perfectly into the CUDA execution structure. The implementation of the algorithm, taking into account the above, is carried out efficiently and in this way it can be shown that obtaining the parameters is performed in less time. This reduction in time allows

a greater number of tests to be carried out and modifications to the algorithm parameters to be able to reduce aberrations in the optical system.

Keywords: Optics, star test, genetic algorithms, CUDA.

1. Introducción

1.1. Prueba de la estrella

En el área de la física y de manera más específica, en la rama de la óptica, la prueba de la estrella es un método de medición de la calidad de una superficie óptica, de entre estas, las más comunes son los espejos cóncavos y los lentes. El mecanismo de esta prueba consta de un arreglo óptico experimental y un algoritmo de optimización. Del arreglo óptico se obtiene una imágen con las aberraciones del sistema y el algoritmo de optimización se encarga obtener los parámetros que generen una nueva imágen con menos aberraciones de tal manera que la calidad de la lente mejore; para llevar a cabo la optimización, se pueden utilizar metodos heurísticos como lo son los algoritmos genéticos.

Arreglo óptico. Este arreglo consiste en colocar una fuente puntual de luz a una distancia muy grande (S_0) y hacer incidir su luz a través del lente (L), observando o grabando el patrón de irradiancia generado a una distancia del foco (S_i) de la superficie, utilizando un dispositivo de carga acoplada (CCD por sus siglas en inglés) para registrar las ondas de luz en una imágen, así como se muestra en la Figura1. De este patrón de irradiancia se obtienen las aberraciones que se deben optimizar, las cuales son errores en el sistema óptico que generan alteraciones en la imágen generada por el CCD.

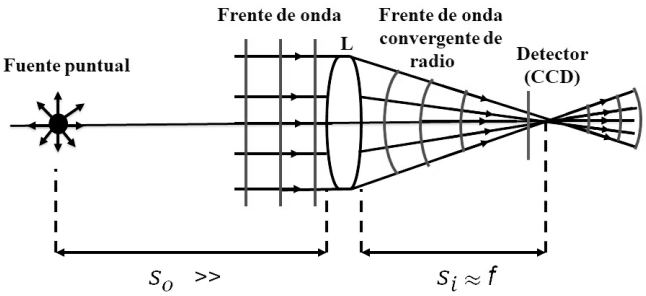

Fig. 1. Arreglo experimental de la prueba de la estrella para la prueba de lentes.

Método de optimización. Para la optimización de las aberraciones se usa el método heurístico de los algoritmos genéticos. Este método está basado en la evolución y la selección natural, que consiste en presentar un conjunto de posibles soluciones, llamado población, donde cada solución es un individuo. Durante varias generaciones se modifica la información de los individuos, simulando los principios del proceso de evolucion, es decir, la reproducción, recombinacion, mutación y la selección natural.

1.2. Algoritmos genéticos

Los Algoritmos Genéticos (AG) son técnicas de optimización y heurísticas de búsqueda, basados en la selección natural y el proceso evolutivo. La meta de estas técnicas, es encontrar la mejor solución posible o grupo de soluciones a un problema, con respecto a uno o más criterios. Un AG permite que una población compuesta por muchos individuos (soluciones) evolucionen bajo ciertas reglas a un estado que maximice su aptitud.

Los principales elementos que componen un AG se listan a continuación [1,2]:

- **Población:** Conjunto de soluciones factibles para un problema, cada una de estas se llama individuo
- **Población inicial:** Generación aleatoria inicial de soluciones factibles al problema dado
- **Selección:** Elección aleatoria y/o sesgada de individuos en base a lo que se considere "mejor"
- **Alteración de los individuos:** La generación de nuevos individuos en la población se puede hacer mediante dos operaciones: Mutación y Cruza
- **Representación:** Forma de representar a cada individuo para el problema tratado. Algunos ejemplos son: bits, enteros, rangos, permutaciones, etc.
- **Función de evaluación, error o aptitud:** Indica la calidad de un individuo en la población

Algunas de las limitaciones que se deben considerar al implementar un AG, son:

- Identificar la función de aptitud.
- Definir la representación para el problema.
- Se puede producir convergencia prematura.
- Se pueden detectar múltiples óptimos locales e inclusive no llegar a un óptimo global.

Antes de implementar un AG, se deben llevar a cabo los siguientes pasos:

1. Analizar la forma de representar las soluciones para el problema.
2. Identificar una función de aptitud.

Dados los puntos anteriores, la Figura 2 muestra gráficamente la implementación del AG general, considerando los siguientes pasos [1]:

1. Generar una población inicial de manera aleatoria con N elementos.
2. Decodificar el cromosoma de cada individuo y calcular su aptitud.
3. Utilizar un método de selección y aparear los cromosomas (realizar la cruza) para generar descendientes.
4. Aplicar las operaciones de mutación a los descendientes.
5. Aplicar un criterio de elitismo para generar la siguiente generación.
6. Si se cumple con algún criterio de parada terminar, en caso opuesto regresar al paso 2.

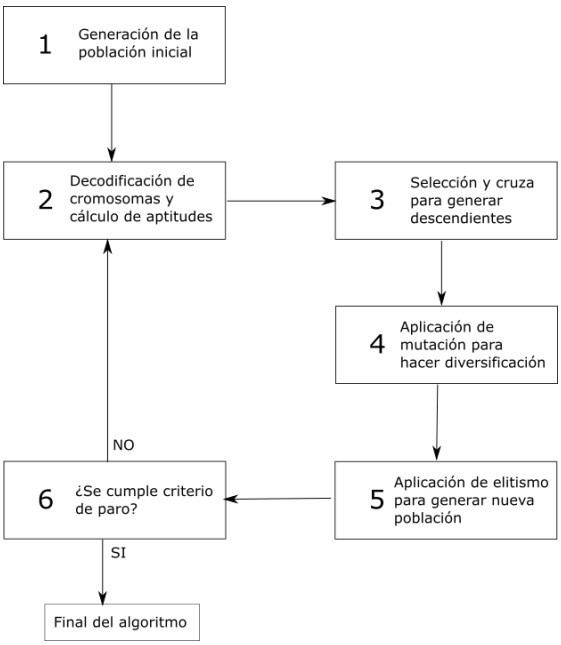

Fig. 2. Diagrama de flujo de un algoritmo genético general.

Como se mencionó anteriormente, los AGs pueden implementarse, de tal manera que se pueda realizar la búsqueda de soluciones de manera concurrida. Es gracias a esta característica, que se puede llevar a cabo la implementación del AG en plataformas de cómputo de alto rendimiento con programación paralela y concurrente, como lo es el caso de CUDA.

En 2019 Jhon Runwei Cheng y Mitsuo Gen, plantean un panorama general de las ventajas y problematicas del diseño de algoritmos genéticos sobre plataformas de uso de GPUs. Esta propuesta plantea representar los datos en el algoritmo de tal manera que los recursos se aprovechen de manera más óptima, ya sea definiendo kernels que esten a nivel de genes o a nivel de individuos, es decir, que un hilo se encargue de evaluar un gen o bien que evalue a un individuo completo.

Por otro lado los problemas que pueden existir en la paralelizacion son, la cantidad de bloques e hilos por bloque se vayan a utilizar para la implementación, así como el problema de la memoria compartida, ya que los dispositivos pueden tener limitación en este aspecto [9].

2. Programación paralela y distribuida

Para acelerar la ejecución de un programas diseñado originalmente de forma secuencial, se debe analizar la forma en que los datos son procesados. De esta forma se pueden detectar múltiples flujos de instrucciones independientes que pueden operar sobre los datos (MIMD). Así, se pueden explotar operaciones repetitivas sobre datos que pueden ser procesados vectorialmente usando un esquema de memoria compartida, enseguida se pueden visualizar tareas de más alto nivel que en este caso pueden ser ejecutadas de forma concurrente sobre procesos independientes aplicando un esquema de procesamiento distribuido. La programación paralela es un modelo para escribir programas que se ejecuten de manera concurrente sobre equipos con varios procesadores. La programación distribuida se ocupa cuando se deberá ejecutar un programa en varios equipos de cómputo que se comunican a través de una red de interconexión [3].

Usualmente, se desea que un programa paralelo pueda ser general y ejecutarse sobre una variedad grande de equipos de cómputo de manera eficiente. La implementación de programas paralelos puede llevarse a cabo usando bibliotecas que pueden invocarse desde lenguajes de programación secuenciales, extensiones a lenguajes o modelos nuevos de ejecución. El modelo de programación paralela usando memoria compartida puede implementarse usando la biblioteca OpenMP. Esta biblioteca contiene directivas que permiten la ejecución de regiones de código en paralelo.

Debido a que el presente trabajo utiliza programación paralela, a continuación se describen algunos términos importantes [4]:

- **Tarea**: es la unidad de trabajo al momento de ejecutarse un programa paralelo. Cada tarea se ejecuta secuencialmente pero existe concurrencia con respecto a otras tareas. Las tareas pueden clasificarse según su granularidad en tareas sencillas y tareas complejas.
- **Proceso**: Son unidades abstractas que realizan las tareas asignadas a los procesadores. Los procesos se comunican y sincronizan para realizar las tareas.
- **Procesador**: es una unidad de procesamiento que ejecuta un proceso.

Para realizar el proceso de paralelización como se muestra en la Figura 3 se deben llevar a cabo 4 tareas principales [4]:

- **Descomposición del problema** en tareas pequeñas. El ideal es que las tareas sean lo mas pequeñas posibles representando cantidades similares de trabajo, y evitando redundancia de proceso de cómputo y/o almacenamiento. La identificación de estas tareas debe poderse escalar con el tamaño del

problema. Los limites de esta descomposición están acotados por la Ley de Amdahl.
- **Mecanismo de asignación** por el cual las tareas se deben distribuir entre los procesos. Se deben considerar aspectos como equilibrio de la carga y si es necesario algún algoritmo de planificación buscando evitar en la medida de lo posible las comunicaciones y los puntos de detención (*deadlocks*) [5].
- **Orquestación** para estructurar la comunicación entre procesos y efectuar la sincronización a través de algunos patrones de comunicación: local/global, estático/dinámico, estructurado/no estructurado o síncrono/asíncrono.
- **Mapeo** es la asignación de procesos o hilos de ejecución a las unidades de procesamiento buscando minimizar el tiempo de ejecución.

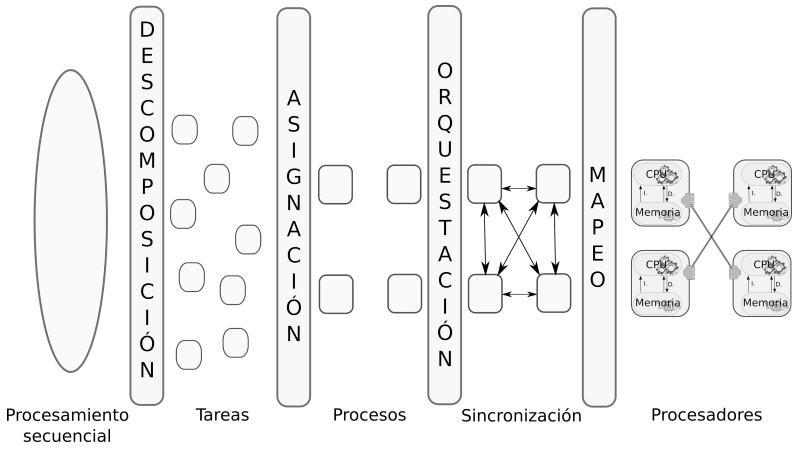

Fig. 3. Proceso de paralelizacion [4].

Al momento de buscar tareas que se pueden ejecutar de manera concurrente, es común detectar paralelismo en datos y tareas [6]:

- El **paralelismo en datos** ocurre cuando existen tareas que aplican la misma operación a diferentes elementos de un conjunto de datos. El ejemplo de la Figura 4 muestra que la operación de suma puede ser realizada de manera independiente, sobre los 100 elementos de los vectores y estas operaciones pueden ejecutarse de manera simultanea en diferentes procesadores.
- El **paralelismo de tareas** o funcional existe cuando se pueden ejecutar tareas de manera independiente sobre diferentes conjuntos de datos. En el ejemplo de la Figura 5, las asignaciones hacia **a** y **b**, así como **m** y **q** pueden ejecutarse de manera concurrente, sin embargo, **m** y **q** deben esperar a las asignaciones de **a** y **b** para poderse ejecutar, de la misma manera que h puede ser ejecutado hasta que **m** y **q** obtengan su valor correspondiente.

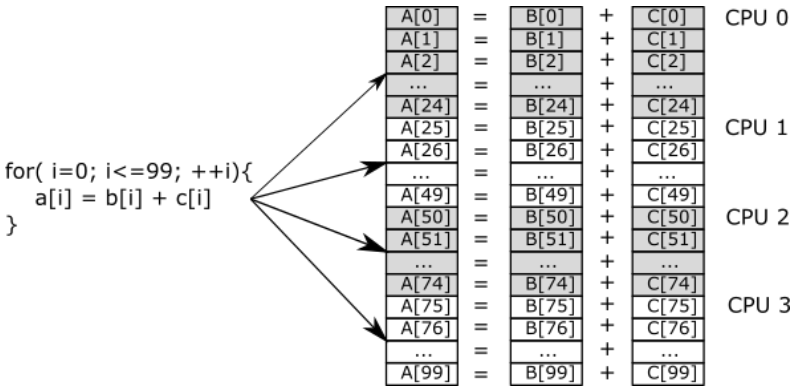

Fig. 4. Ejemplo del proceso de paralelizacion a nivel de datos. Se aprecia que el ciclo de asignaciones es dividido en cuatro bloques independientes, donde cada uno es llevado a cabo por una unidad de procesamiento.

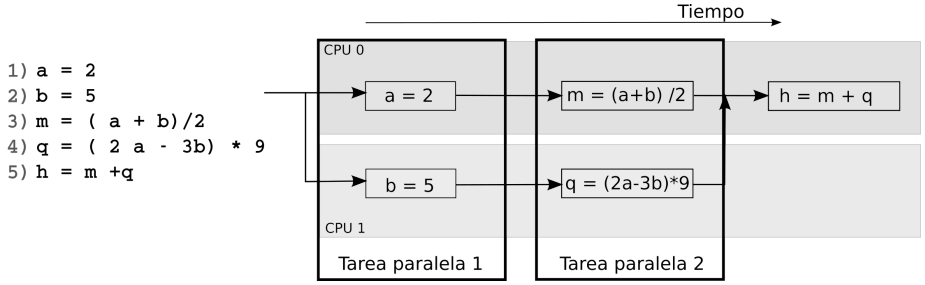

Fig. 5. Ejemplo del proceso de paralelizacion a nivel de tareas. Las primeras dos instrucciones pueden ejecutarse en paralelo. Al terminarse, pueden ejecutarse las siguientes dos también en paralelo, pero la quinta instrucción debe esperar hasta que terminen la cuarta y quinta instrucción.

A partir del año 2003, la industria de los semiconductores para el desarrollo del cómputo paralelo tomó dos caminos diferentes, el primero es la rama de los multiprocesadores, que consiste en mantener la velocidad de un programa secuencial ejecutando partes de este en varios procesadores (CPUs), tal es el caso de la plataforma que ofrece MPI y OpenMP, a través del paso de mensajes dentro de un cluster, que permite enviar parte de los datos y evaluarlos en diferentes computadoras. Por otro lado se encuentra la rama de las Unidades de Procesamiento Gráfico (GPUs), NVIDIA es una de las empresas que desarrolló esta tecnología, la cual consiste principalmente en manejar una gran cantidad de hilos en los GPUs y CUDA es el modelo de programación para la implementación de algoritmos en este modelo de cómputo paralelo.

En pruebas realizadas en 2012 la razón entre los multi-hilos de los GPUs y los multiprocesadores de los CPUs en el cálculo de números de punto flotante fue de casi 10.

Pero esta razón no necesariamente impacta en la velocidad de las aplicaciones ya que no todas las regiones de código pueden aprovechar esta mejora [7]. La diferencia principal entre estos dos modelos de cómputo paralelo, se debe principalmente en la filosofía del diseño de los chips. Por un lado, la arquitectura de un multiprocesador está orientada principalmente a la mejora del rendimiento de programas secuenciales. Mientras que los GPUs están orientados a las tareas con procesos concurrentes independientes. El ancho de banda de memoria es mayor en los GPUs, por lo que pueden mover más datos que en un multiprocesador, y esto permite comenzar a procesar de manera mas inmediata [7].

3. CUDA

CUDA es un modelo de programación paralela, cuyas siglas en inglés significan Arquitectura Unificada de Dispositivos de Cómputo, el cual fue desarrollado para implementar algoritmos que se puedan ejecutar en los GPUs de una tarjeta NVIDIA. CUDA C, es una extinción del lenguaje de programación C, añade palabras reservadas para el uso del cómputo paralelo, que además combina ambos sistemas de cómputo paralelo, GPUs y CPUs. Donde los CPUs son llamados host y con uno o varios CUDA device, que típicamente son los GPUs [7].

En este modelo de programación los algoritmos comúnmente presentan solo partes paralelizadas, es decir, en un segmento del programa, el host se encarga solo de preparar y presentar los datos resultantes, mientras que el procesamiento es llevado a cabo en el CUDA device.

Las estructuras de datos que se utilizan de manera mas eficiente en los GPUs, son las matrices y los vectores, esto se debe a que los GPUs están optimizados para el tratamiento de gráficos, como lo son las imágenes y vídeos. Por este motivo es preferible que para la implementación de algoritmos en esta plataforma se utilicen estas estructuras de datos, para modelar el problema que se desea solucionar. Se debe tomar en cuenta que los datos se representan en arreglos de una sola dimensión, es decir, vectores, por lo que al implementar un algoritmo que haga uso de matrices se debe considerar la forma de accesar a los datos.

3.1. Estructura de un programa en CUDA

La estructura de un programa en CUDA refleja la coexistencia de un host (CPU) y uno o más disposivos CUDA (GPUs) en la computadora. Las funciones y declaraciones de datos son definidos por palabras reservadas para CUDA. Las funciones y declaraciones de datos referentes a CUDA no son aceptados por un compilador de C, por lo que es necesario utilizar el compilador de NVIDIA, el cual es NVCC (NVIDIA C Compiler). Durante la compilación se separa el código del host y se utiliza el compilador estándar de C/C++ del host para su compilación y se ejecuta como un programa típico de C, mientras que NVCC se encarga de compilar el código CUDA, para luego ejecutarlo en los GPUs. En la Figura 6, se puede observar la separación de tareas de un programa en CUDA, cuando éste se compila.

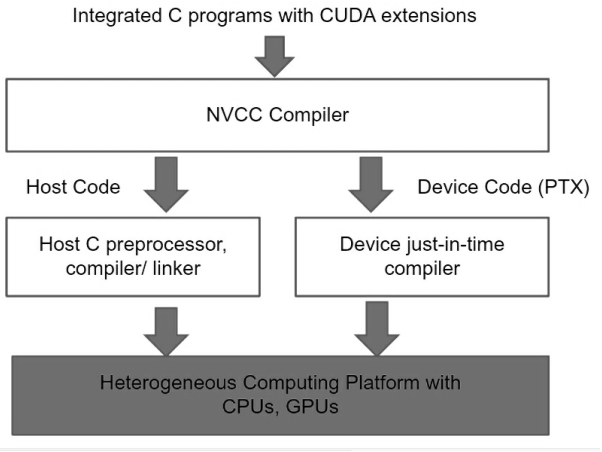

Fig. 6. Proceso de compilación de un programa en CUDA [7].

Las funciones de CUDA que se encargan de paralelizar los datos en los GPUs son los llamados Kernels. Para que un programa se pueda paralelizar en CUDA de manera eficiente es necesario que los datos sean paralelizables, es decir, los cálculos o modificaciones de los datos deben ser independientes y no tener relación con cálculos previos. En la Figura 7 se puede observar el ejemplo mas básico de un programa paralelizado en CUDA, la suma de vectores. Si se realiza la suma elemento a elemento de dos vectores A y B, y se guarda el resultado en un vector C, se puede notar que la suma de los elementos es totalmente independiente y por lo tanto se puede lanzar un hilo para sumar cada par de elementos y así realizar esta tarea de manera más eficiente.

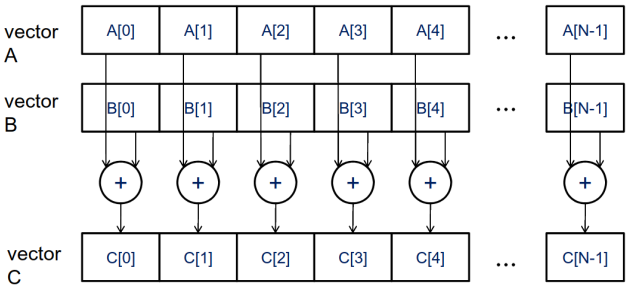

Fig. 7. Suma de dos vectores de forma paralela [7].

3.2. El kernel y los hilos

Como se mencionó anteriormente, los kernel son funciones de CUDA en las cuales se paralelizan los datos del programa. Cuando el host lanza el kernel, CUDA genera una malla de hilos, los cuales están organizados en dos niveles de jerarquía. Cada malla está organizada dentro de un arreglo de bloques de hilos (bloques). Todos los bloques tienen el mismo tamaño y se especifica cuando se lanza el kernel desde el host. El numero de hilos por bloque está disponible en la variable BlockDim. Por cuestiones de rendimiento del hardware, el tamaño de bloque debe ser múltiplo de 32.

Cada hilo en un bloque tiene un único valor dado por threadIdx. De tal manera que sea posible acceder a índices en los datos utilizando la instrucción **i = blockId.x*BlockDim.x + threadIdx.x**. Se puede declarar más de un kernel y se le puede lanzar más de una vez, dentro de un mismo programa [7]. En la Figura 8 se puede observar cómo se ejecutan los hilos al mismo tiempo dentro del código, en un mismo kernel.

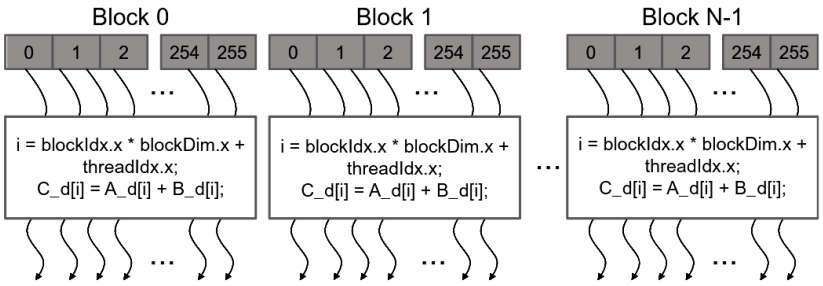

Fig. 8. Hilos de una malla ejecutándose al mismo tiempo en el código de un mismo kernel [7].

El uso de CUDA como herramienta para el desarrollo de investigaciones en la ciencia ha ido en aumento, en los últimos años debido a que esta plataforma ofrece una mejora de rendimiento en el procesamiento de datos, muy por encima que los procesadores más potentes. Como muestra el trabajo realizado por Huang XianLou y Yu ShuangYuan [8], quienes realizan la segmentación de imágenes, que se basa en cortes normalizados y cómputo paralelo con CUDA. Para la implementación de su algoritmo definieron cuatro kernels, (i) calcular la matriz de afinidad, (ii) reducir la matriz simétrica a una forma tri diagonal simétrica, (iii) Resolver el vector característico y (iv) discretizar el vector característico y seleccionar el punto del corte.

Para realizar pruebas, tomaron el Berkley Segmentation Dataset. La máquina que usaron estaba equipada con un CPU intel Core i5 de 2.8GHz, con 4GB de RAM y Sistema Operativo Windows 7.

El GPU usado fue una tarjeta NVIDIA GTX570 con 480 núcleos y 1.5GB de memoria principal, y con CUDA SDK 5.0 para la implementación del algoritmo. Los resultados experimentales muestran que el algoritmo puede segmentar imágenes de tamaño medio (200x160 píxeles) y tener una mejora de rendimiento, 2.3 veces más rápido, en comparación con la implementación orientada al uso de CPUs.

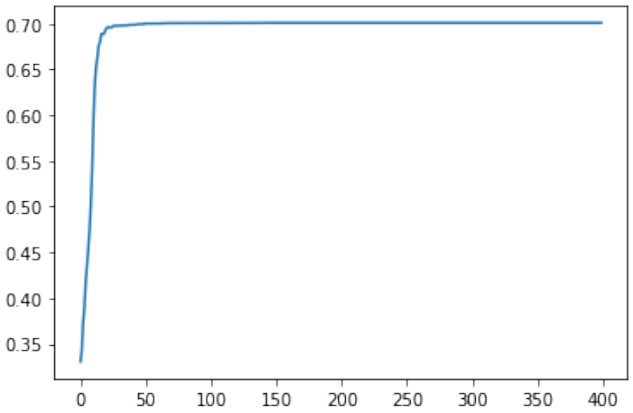

Fig. 9. Comportamiento de las aptitudes que resultan de aplicar el AG a una imágen de 600x500 píxeles, con poblacion de 1000 individuos y 400 genereciones.

4. Metodología

Los algoritmos genéticos pueden representar la la población utilizando matrices, las cuales permiten que se pueda implementar en modelos de programación paralela como CUDA. La metodología para la implementación del algoritmo genético para obtención de los parámetros de la prueba de la estrella se muestra a continuación.

1. Se llevan a cabo los pasos para la implementación de un algoritmo genético.
2. Una vez implementado el algoritmo de manera secuencial, se analizan el algoritmo para obtener las partes del código que se pueden paralelizar.
3. Se implementan los kernels necesarios para cada parte paralelizable del algoritmo.
4. Se realizan pruebas para verificar el rendimiento del algoritmo.

Este algoritmo recibe la información de una imágen con las aberraciones del sistema óptico y los parámetros en los que se llevará a cabo la simulación de la prueba de la estrella, y como salida darán los parámetros que describen la imágen optimizada de las aberraciones de sistema.

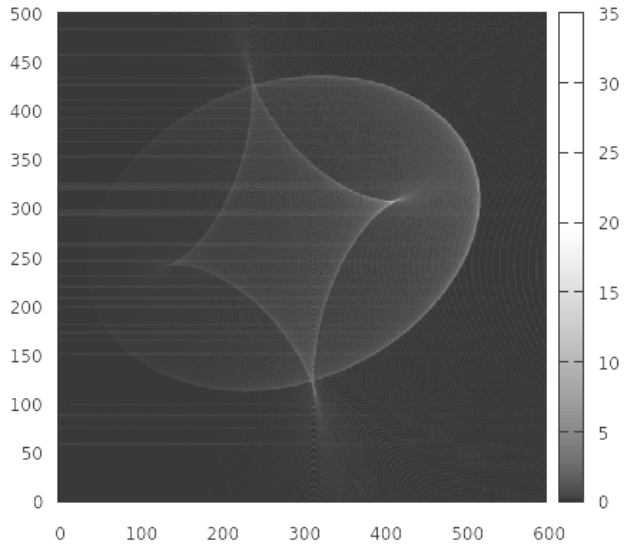

Fig. 10. Imágen original, obtenida del CCD antes de aplicar el AG.

4.1. Representación del problema

En la implementación del AG, cada indiviudo esta compuesto por un vector de 16 elementos, los cuales representan los coeficientes de la ecuacion, que genera la imágen optimizada de las aberraciones del sistema óptico.

La función de aptitud utiliza los polinomios de Zernik, para el coeficiente de correlacion entre un indiviudo y la imágen experimental obtenida del arreglo óptico, donde el coeficiente de correlación representa la aptitud del individuo. Para simular la cruza o reproducción de los individuos, se toman dos individuos y se mezclan a nivel de bits, estableciendo puntos de cruza de manera aleatoria y usandolos puntos para crear rangos, en los cuales la información se intercambia de un indivudo a otro. Esta cruza crea dos individuos nuevos, que contienen la informacion mezclada, de los individuos originales. Para mutar a los descendientes de la población, aleatoriamente se seleccionan algunos individuos y se invierten algunos de sus bits.

5. Resultados

Los resultados presentados a continuación, son un resumen de la ejecución del AG, así como el tiempo de ejecución de cada prueba y la mayor aptitud obtenida. Los parámetros para cada una de las ejecuciones del AG sobre una imágen de 500x500 píxeles son los siguientes:

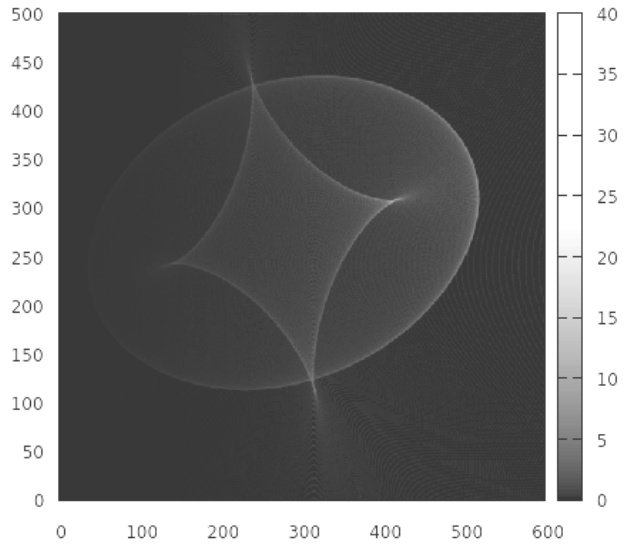

Fig. 11. Imágen de 600x500 píxeles después de aplicar el AG.

- No. de mutaciones: Se elige un número en cada generación de acuerdo al módulo de 6. De acuerdo al resultado del módulo, la cantidad de mutaciones es: 0=5, 1=10, 2=15, 3=20, 4=25, 5=1.
- Puntos de cruza: 100 inicialmente, con una reducción del 10 % a la cantidad por cada 10 generaciones.
- No. de individuos a mutar: comienza con el número total de la población y se reduce 10 % cada 10 generaciones.

La Tabla 1 muestra los resultados del algoritmo aplicado a una imágen de 500x500 píxeles, con una población de 400 individuos y 240 generaciones. El mejor resultado obtenido fue una aptitud de 0.98263 en la ejecución 3 del método de selección simple.

La Tabla **2** muestra los resultados del algoritmo aplicado a una imágen de 600x500 píxeles, con población de 1000 individuos y 400 generaciones. El mejor resultado obtenido, es una aptitud de 0.71741, que corresponde a la primera ejecución usando el método de selección Ruleta.

Como se observa en la figura 9 los AGs convergen muy rápido, pero la respuesta no necesariamente es buena o satisfactoria. La figura 10 muestra la imágen original obtenida del CCD, mientras que la figura 11 muestra el resultado al aplicar el AG, estas imágenes son de la prueba con la ímagen de 600x500 píxeles.

Tabla 1. Resultados de tiempo y aptitud del AG aplicando a una imágen de 500x500 píxeles.

Método	Tiempo(Hrs:min:seg)	Aptitud(por ejecución)
Simple	00:32:23	0.94494
	00:32:04	0.97820
	00:32:25	0.98263
Ruleta	00:31:48	0.59049
	00:32:24	0.96023
	00:32:28	0.83947
Torneo	00:33:11	0.89816
	00:32:30	0.85083
	00:31:51	0.81182
Ranking	00:32:22	0.84102
	00:30:25	0.92106
	00:32:08	0.54901

Tabla 2. Resultados de tiempo y aptitud de la aplicación del AG a una imágen de 600x500 píxeles.

Método	Tiempo(Hrs:min:seg)	Aptitud(por ejecución)
Simple	03:21:03	0.70999
	03:19:15	0.69499
	03:21:28	0.70221
Ruleta	03:18:52	0.71741
	03:21:19	0.61791
	03:20:38	0.67528
torneo	03:19:02	0.69572
	03:21:00	0.70252
	03:19:29	0.61625
Ranking	03:20:20	0.70905
	03:19:04	0.67239
	03:21:04	0.52251

6. Conclusiones

Este trabajo muestra que la implementación de un AG utilizando cómputo de alto rendimiento, puede lograr que el tiempo en el que el algoritmo de obtención de parámetros de la prueba de la estrella obtiene respuestas, sea menor al correspondiente a la implementación en computadoras convencionales. Los AG's al ser un método heurístico, para obtener buenos resultados hay que realizar muchas pruebas y ajustar el algoritmo, algunos ajustes pueden ser la cantidad de mutaciones, número de generaciones y tamaño de la población. Como trabajo futuro se pueden realizar implementaciones que ayuden a mejorar la respuesta del algoritmo, como lo es el cambio al método de optimización, una buena opción puede ser una red neuronal artificial.

Referencias

1. Haupt, R.L., Haupt S.E.: Practical Genetic Algorithms (2nd edition). Wiley-Interscience (2004)

2. Muñoz Duarte, A.: Metaheurısticas. Dykinson (2008)
3. Patterson, D., Hennessy, J.: Computer Architecture: A Quantitative Approach. 4th edition. Morgan Kaufmann, USA (2006)
4. Culler, D.E., Gupta, A., Singh, J. P.: Parallel Computer Architecture: A Hardware/Software Approach. 1st edition. Morgan Kaufmann Publishers, USA (1997)
5. Coulouris, G., Dollimore, J., Kindberg, T., Blair, G.: Distributed Systems: Concepts and Design. 5th edition. Addison-Wesley Publishing Company, USA (2012)
6. Quinn, M.: Parallel Programming in C with MPI and OpenMP. 1st edn. McGraw-Hill (2003)
7. Kirk, D.B., Hwu, Wen-Mei: Programming Massively Parallel Processors, A Hands-on Approach. Morgan Kaufmann (2013)
8. XianLou, H., ShuangYuan, Y.: Image segmentation based on Normalized Cut and CUDA parallel implementation. School of Computer and Information Technology, Bejiing Jiaotong University (2013)
9. Runwei-Cheng, J., Gen, M.: Accelerating genetic algorithms with GPU computing: A selective overview. Computer & Industrial Engineering, Elsevier (2018)