Lámpsakos
E-ISSN: 2145-4086
lampsakos@amigo.edu.co
Fundación Universitaria Luis Amigó
Colombia

Insua-Suárez, Ernesto; Fulgueira-Camilo, Marlis; Henry-Fuenteseca, Venus
Paralelización del Algoritmo Expectación-Maximización Utilizando OpenCL
Lámpsakos, núm. 13, enero-junio, 2015, pp. 51-61
Fundación Universitaria Luis Amigó
Medellín, Colombia

Lámpsakos | N°.13 | pp. 51-61 | enero-junio | 2015 | ISSN: 2145-4086 | Medellín–Colombia

Paralelización del Algoritmo Expectación-Maximización Utilizando OpenCL

Maximization of Expectation-Parallelism Algorithm Using OpenCL

Ernesto Insua-Suárez, Ing.
Facultad de Ingeniería Informática
Instituto Superior Politécnico José Antonio Echeverría
La Habana, Cuba
einsuas@ceis.cujae.edu.cu

Marlis Fulgueira-Camilo, Ing.
Facultad de Ingeniería Informática
Instituto Superior Politécnico José Antonio Echeverría
La Habana, Cuba mfulgueira@udio.cujae.edu.cu

Venus Henry-Fuenteseca, Ing.
Facultad de Ingeniería Informática
Instituto Superior Politécnico José Antonio Echeverría
La Habana, Cuba
vhenry@ceis.cujae.edu.cu

(Recibido el 10-09-2014. Aprobado el 11-12-2014)

Resumen. Actualmente, las organizaciones y empresas almacenan grandes volúmenes de datos para lograr sus propósitos. Una de las variantes para obtener información valiosa consiste en el empleo de la minería de datos. Dentro de esta, existen diferentes tareas, una de ellas es el agrupamiento. En esta tarea, los datos se agrupan según sus semejanzas entre sí y sus diferencias con elementos de otros grupos. Dentro de los algoritmos que realizan estos agrupamientos se encuentra expectación-maximización, el cual presenta elevados tiempos de ejecución en la medida que aumenta el tamaño de los datos. En el presente artículo, se discute acerca de la paralelización del algoritmo, utilizando técnicas de programación paralela. El diseño del algoritmo propuesto se basa en el uso de las tarjetas de procesamiento gráfico, GPU. OpenCL, lenguaje empleado para la programación en arquitecturas híbridas, permite aprovechar las arquitecturas de hardware disponibles, con lo que se logra disminuir el tiempo de ejecución de la implementación realizada. La razón principal por la cual es posible mejorar este tiempo se debe a la cantidad de procesos paralelos que se pueden lanzar en hilos de procesamientos independientes. Para el logro de los resultados descritos, se integran conocimientos del campo de la minería de datos y la computación paralela y distribuida. Como parte de esta investigación, se realizó una implementación del algoritmo, utilizando las bibliotecas de OpenCL, para disminuir su tiempo de ejecución. La implementación logra disminuir en un 82% la implementación secuencial. Esto significa que el algoritmo paralelo se ejecuta 5,5 veces más rápido que su correspondiente implementación secuencial.

Palabras claves: Arquitecturas híbridas, computación paralela y distribuida, expectación-maximización, ejecución paralela, OpenCL.

Citación de artículo, estilo IEEE:
E. Insua-Suárez, M. Fulgueira-Camilo, V. Henry-Fuenteseca, "Paralelización del Algoritmo Expectación–Maximización Utilizando OpenCL", *Lámpsakos*, N° 13, pp. 51-61, 2015.
DOI: http://dx.doi.org/10.21501/21454086.1361

Abstract. Nowadays both organizations and companies store big volumes of data to achieve their purposes. One of the variants to obtain valuable information consists on the employment of Data Mining. Inside Data Mining, different tasks exist and one of them is clustering. In this task the data group according to their likenesses among them differ with elements of other groups. One of the algorithms that carry out these clusters is Expectation-Maximization, which presents high times of execution in their data. This article discusses about the parallelization of the mentioned algorithm, using techniques of parallel programming. The design of the proposed algorithm is based on the use of the graphic process unit, GPU. OpenCL, language used for the programming in hybrid architectures, allows to take advantage of the available hardware architectures, which it is possible to diminish the time of execution of the sequential implementation. The reason to improve this time is due to the quantity of parallel processes that can rush in threads of independent prosecutions. For the achievement of the described results, knowledge of the field of Data Mining and Parallel and Distributed Computation are integrated. As part of this investigation, an implementation of the algorithm using the libraries of OpenCL was carried out to diminish the time of execution. The implementation is able to diminish the sequential implementation in 82%, this means that the parallel algorithm is executed 5,5 times quicker that its sequential corresponding implementation.

Keywords: Hybrid architectures, Parallel & Distributed Computation, Maximization of Expectation, Parallel Execution, OpenCL.

1. INTRODUCCIÓN

Con el transcurso del tiempo, las organizaciones han necesitado obtener información procedente de los datos que manejan. Con el creciente volumen de estos datos, procesarlos se convierte en un trabajo complicado. Una de las variantes existentes para realizar este procesamiento es la minería de datos, la cual se puede definir como el proceso de extraer conocimiento útil y comprensible, previamente desconocido, desde grandes cantidades de datos [1].

La minería de datos está compuesta por varios tipos de tareas, una de estas es el agrupamiento. El agrupamiento consiste en obtener grupos a partir de los datos, los cuales se basan en el principio de que la similitud entre los elementos de un grupo debe ser la mayor posible y que la semejanza entre elementos de grupos diferentes debe ser mínima [1].

La presente investigación, se enfoca en la reducción del tiempo de ejecución de la implementación del algoritmo de minería de datos expectación-maximización (EM). Este algoritmo de agrupamiento fue presentado por primera vez por Laird y Rubin Dempster en la revista *J Royal Statistical Society*, en 1977 [2]. Expectación-maximización selecciona el número de grupos a obtener. Se utiliza como parte del preprocesamiento de los datos, para luego aplicar el algoritmo de agrupamiento K-MEANS con la cantidad de grupos obtenida con EM. Dado que el algoritmo es parte de una etapa de preprocesado, se necesita que se ejecute de manera rápida para luego realizar el agrupamiento. Además, con la creciente cantidad de datos a procesar, el factor tiempo viene a jugar un papel decisivo.

Dadas las condiciones anteriores, una de las mejores alternativas de solución la podemos encontrar en la explotación de las características de procesamiento simultáneo que presentan los diferentes dispositivos multinúcleos actuales, los cuales pueden ser procesadores o tarjetas de procesamiento gráfico. Para utilizar la característica de procesamiento simultáneo de estos dispositivos es necesario utilizar una biblioteca que permita la distribución de las tareas entre los diferentes núcleos. La presente investigación muestra los resultados obtenidos luego de utilizar una de estas bibliotecas de procesamiento paralelo en comparación con una implementación secuencial del algoritmo EM, para lo cual se realizaron las implementaciones secuenciales y paralelas del algoritmo. Con la aplicación del procesamiento paralelo, el principal beneficio que se logra es la disminución del tiempo de ejecución de los algoritmos.

La problemática a resolver será el procesamiento de un fichero *.xes* al cual se le desean realizar agrupamientos de sus datos.

Esta investigación representa una segunda etapa de implementación del algoritmo Expectación-Maximización, la primera etapa estuvo constituida por la implementación del algoritmo utilizando las bibliotecas de OpenMP. La actual etapa tiene como principal propósito introducir una nueva biblioteca que permita mejorar el tiempo de ejecución del algoritmo antes mencionado.

2. DESARROLLO DEL ARTÍCULO

2.1. Minería de datos

Dentro de la minería de datos, se pueden distinguir varios tipos de tareas, las cuales se emplean en dependencia de cómo se desea obtener el nuevo conocimiento o qué procesamiento se le desea realizar a los datos. Estas tareas pueden ser de tipo descriptivas o predictivas: las descriptivas sirven para conocer el comportamiento de los datos y las predictivas se utilizan para estimar valores futuros de las variables. Dentro de las principales tareas se pueden encontrar:

A. Predictivas

- Regresión: Consiste en asignarle a una variable un valor real. Solo se utiliza para realizar predicciones de variables numéricas.

- Clasificación: Se clasifica el elemento que se esté analizando, utilizando los atributos propios del elemento. Su principal diferencia con la regresión consiste en que la estimación se puede realizar sobre cualquier tipo de variable.

B. Descriptivas

- Correlación: Consiste en realizar un análisis con el fin de obtener la magnitud de la similitud de una variable con otra.

- Reglas de asociación: Tiene cierta similitud con la correlación, pero su principal objetivo es identificar relaciones no explícitas entre varios atributos: por ejemplo, “Si nombre es igual Ana, entonces sexo es femenino”.

- Agrupamiento: También conocido como *clustering.* Consiste en el proceso de obtener grupos a partir de un conjunto de datos, maximizando la similitud entre los elementos de un grupo y haciéndola mínima con respecto a los elementos de otros grupos [1].

El algoritmo expectación-maximización es un algoritmo de minería de datos que se encuentra dentro de la tarea de agrupamiento.

El agrupamiento es el proceso de examinar una colección de elementos y agruparlos en grupos o “clusters”, de acuerdo a alguna medida de distancia. El objetivo a lograr en las tareas de agrupamiento es que los elementos dentro de un grupo tengan la menor distancia o diferencia posible uno de otro, como se observa en la Figura 1.

2.2. Expectación-Maximización

El algoritmo de expectación-maximización (EM) se ha convertido en una herramienta popular para realizar agrupamientos en grandes volúmenes de datos. Existen dos aplicaciones principales del algoritmo EM. La primera es cuando los datos tienen valores faltantes derivados del proceso de observación y la segunda está dada por la posibilidad de la estimación de patrones. Este algoritmo fue presentado por primera vez por Laird y Rubin Dempster en J Royal Statistical Society en 1977.

Debido a su implementación, el algoritmo EM puede ser visto como un método metaheurístico, garantizando que se encuentre, con cierta probabilidad, un máximo local del registro de los datos. EM brinda una aproximación estadística al problema buscando el número de “cluster” más probable dados los datos. La base de este tipo de agrupamiento se encuentra en un modelo estadístico llamado mezcla de distribuciones, donde cada distribución representa la probabilidad de que un objeto tenga un conjunto particular de pares atributo-valor. El algoritmo EM es un procedimiento iterativo que calcula la máxima probabilidad (MP) estimada en presencia de datos faltantes u ocultos.

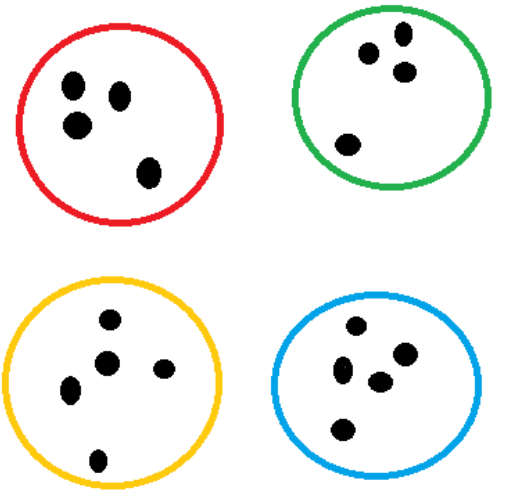

Fig. 1. Agrupamiento de elementos.

Cada iteración del algoritmo EM consiste en dos procesos: el E-paso y el M-paso.

En la expectación, o el E-paso, los datos faltantes son calculados con los datos observados y la estimación actual del modelo de los parámetros. Este archivado se realiza usando la expectativa condicional.

En el M-paso, la función de probabilidad es maximizada bajo la suposición de que los datos faltantes son conocidos. La estimación de los datos faltantes del E- paso se usa en lugar de los datos faltantes reales.

El algoritmo detiene las iteraciones cuando ocurre la convergencia, lográndose esta cuando al calcular la verosimilitud (relación entre elementos), esta sea menor que un valor predefinido [1-3].

2.2.1. Idea general del algoritmo en pseudocódigo

En la Figura 2, se muestra el seudocódigo del algoritmo expectación máxima.

2.2.2. Campos de aplicación

EM ha sido exitosamente aplicado en problemas de aprendizaje de máquina y reconocimiento de patrones en general. Generalmente, se utiliza como algoritmo de clustering (agrupamiento), en donde los puntos a clasificar se asumen como si estos fueran

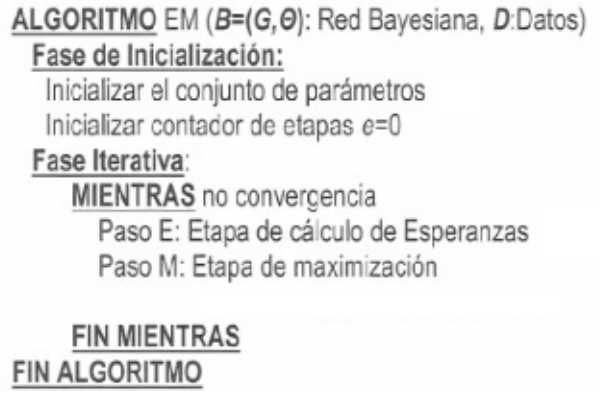

Fig. 2. Seudocódigo del algoritmo expectación-maximización.

derivados de una mezcla de componentes gaussianas o Poisson y el interés es encontrar los parámetros de dicha distribución [1-3].

2.2.3. Escenarios de utilización

a. Visión computacional (para etiquetar regiones de imágenes: tipos de tejidos).

b. Biología computacional (inferencia de árboles filogenéticos).

c. Minería de datos (análisis de bases de datos).

d. Finanzas (manejo de riesgos y composición de portafolios).

2.3. Variantes

Varios métodos se han propuesto para acelerar la lenta convergencia del algoritmo EM, como los que utilizan el declive conjugado y modificaron técnicas de Newton-Raphson.

La expectativa maximización condicional (ECM) sustituye cada M de paso por una secuencia de pasos de maximización condicional (CM) que cada parámetro θ se maximiza individualmente, con reservas en los otros parámetros restantes fijado [4].

Maximización de la expectativa generalizada (GEM), en la cual uno solo busca un aumento de la función objetivo F tanto para el paso de E como para M, según la descripción alternativa [4].

2.4. Solución propuesta para resolver el problema

Una de las alternativas existentes es hacer uso del paralelismo, el cual consiste en la ejecución de manera simultánea, en un instante de tiempo, de sentencias de un lenguaje de programación específico, realizando un mejor aprovechamiento de las arquitecturas que actualmente existen.

Para lograr un nivel de paralelismo lo más eficiente posible, se siguió la metodología de diseño de algoritmos paralelos enunciada por Ian Foster [5]. Esta está constituida por cuatro pasos a seguir en el momento que se diseña un algoritmo para ejecutar de forma paralela. Los cuatro pasos son [5]:

- Particionado: Se dividen los datos o las funciones en dependencia de las características del algoritmo a paralelizar sin tener en cuenta la arquitectura sobre la cual se realizará la ejecución.

- Comunicación: Se realiza un análisis referente a cómo se realizan las comunicaciones entre los datos o funciones en la implementación secuencial.

- Agrupamiento: Se agrupan los datos ya estando particionados, teniendo en cuenta la arquitectura del sistema de cómputo y las características de comunicación analizadas en el paso anterior.

- Mapeo: Se le asignan a cada hilo de ejecución de la arquitectura seleccionada, cada grupo de datos o funciones obtenidos en el paso anterior.

Para lograr cumplir el objetivo de disminuir el tiempo de ejecución del algoritmo EM, se deben seguir una serie de pasos adicionales luego de la implementación secuencial del algoritmo:

- Encontrar las secciones del código que ocupan mayor tiempo al procesador; para esto se utilizan herramientas como Intel Parallel Studio.

- Con las secciones seleccionadas en el paso anterior, realizar un análisis con el fin de descubrir si estas secciones permiten ser paralelizadas. Secciones paralelizables son aquellas

en la que unos datos no dependan de otros; si existiera dependencia, no se podría aplicar paralelismo [5].

- Luego de tener filtradas estas secciones, se procede a realizar las transformaciones correspondientes al proceso de paralelismo que se vaya a utilizar.

Como resultado del seguimiento de los pasos definidos anteriormente, se decidió aplicarle el proceso de transformación al paso E, debido a que es el paso que más procesamiento necesita y el cual permite, además, que el paralelismo sea aplicado.

2.5. Alternativas de implementación paralela

Para cada una de las alternativas analizadas a continuación se necesita realizar un estudio de las transformaciones que es necesario aplicar al lenguaje para hacer uso del paralelismo y un mejor aprovechamiento de las arquitecturas de procesamiento disponibles en función de la alternativa de implementación que se desee utilizar.

OpenMP (Open Multi-Processing) es una biblioteca para implementaciones paralelas basada en el uso de memoria compartida [6, 7].

Requisitos: Se necesita poseer un microprocesador multinúcleo para poder ejecutar las sentencias de manera simultánea.

El paradigma de memoria compartida consiste en que múltiples núcleos de un dispositivo acceden a la misma memoria que, en este caso, es la memoria caché de los microprocesadores disponibles, al contario del paradigma de memoria distribuida, donde se presentan varios núcleos, cada uno con su propia memoria local y que cooperan en conjunto para resolver tareas mediante alguna herramienta para administrar la ejecución de la tarea [8, 9].

CUDA (Dispositivo de Arquitectura Computacional Unificada o **C**ompute **U**nified **D**evice **A**rchitecture) es una biblioteca para implementaciones paralelas creada por la compañía NVIDIA y basada en el uso de GPUs (tarjetas de procesamiento gráfico) de dicha compañía [10, 11].

Requisitos: Se necesita poseer una tarjeta procesadora de gráficos de la marca NVIDIA.

OpenCL (Open Computing Language o lenguaje de computación abierto) consta de una interfaz de programación de aplicaciones y de un lenguaje de programación. OpenCL permite crear aplicaciones con paralelismo a nivel de datos y de tareas que pueden ejecutarse tanto en unidades centrales de procesamiento como en unidades de procesamiento gráfico [12-15].

Un kernel de OpenCL es una porción paralela (muy similar a una función) que es ejecutada en el device (tarjeta de procesamiento gráfico o unidades centrales de procesamiento). Este kernel puede ser ejecutado por muchos hilos en un instante de tiempo [12-14].

Un thread o hilo (en inglés) se ejecuta en el device mediante un arreglo de ellos que ejecutan el mismo código kernel, teniendo cada hilo un identificador (id) para su identificación y control. Un hilo es la unidad básica para el procesamiento paralelo. Los hilos pueden ser de dos tipos: físicos y virtuales [12-14].

Los hilos pueden efectuar comunicaciones entre sí en un momento dado, ya sea para compartir resultados o accesos a memoria, reduciendo así el ancho de banda. Estos hilos dentro de un bloque cooperan mediante la memoria compartida, pudiendo realizarse su sincronización en caso de que se requiera. Cada hilo tiene su propia memoria local. La Figura 3 muestra cómo se encuentra estructurado un device.

La memoria local está localizada en el dispositivo DRAM de la PC donde se ejecuta el algoritmo, por lo cual se debe realizar un acceso mínimo a esta memoria para reducir el tiempo de ejecución. Se debe siempre tratar que entre accesos a la memoria DRAM, se realicen la mayor cantidad de procesamientos posibles en el dispositivo, con el objetivo de minimizar el tiempo de comunicación [13, 15].

Se decide utilizar las bibliotecas de OpenCL para el procesamiento paralelo, pues se puede estar utilizando en una sola implementación CPU o GPU. OpenCL es libre de licencias, por lo cual una aplicación implementada utilizando esta biblioteca se pudiera estar comercializando sin ningún problema.

2.6. Arquitectura y fichero de prueba

El fichero que contiene los datos sobre los que se realizará el agrupamiento es de extensión .xes. Los ficheros .xes son utilizados en la minería de proce-

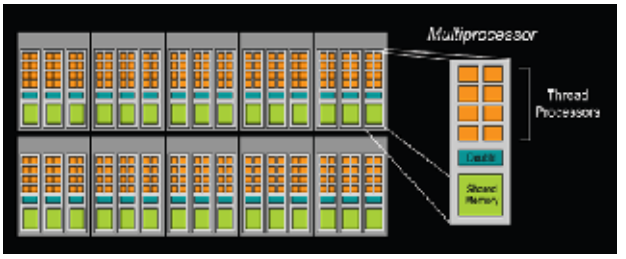

Fig.3. Estructura de un device.

sos para registrar eventos de un proceso específico. La estructura de los ficheros .xes es muy parecida a la que presentan los de extensión .xml.

En este caso, el fichero .xes utilizado se corresponde con un proceso que ocurre dentro de un hospital, donde se desean agrupar los datos en base a valores numéricos.

El agrupamiento se realizará sobre la variable “Age”, que contiene 2071 instancias en el fichero.

Las arquitecturas donde se probará el algoritmo es la siguiente:

Estas arquitecturas de prueba fueron seleccionadas con base en las diferencias en cuanto a la cantidad de núcleos que estas presentan. Además, se toman ya que son unas de las arquitecturas más comunes que se pueden encontrar en el mercado.

3. RESULTADOS

3.1. OpenMP

La 4, 5 y 6 muestran las comparaciones de los resultados obtenidos con las ejecuciones paralela y secuencial, y sus respectivas representaciones gráficas del uso de cada CPU, ejecutándose el algoritmo sobre la tercera arquitectura.

Tabla 1. Arquitecturas empleadas.

Computadora	Núcleos/hilos	Memoria RAM	Frecuencia
Intel i7 860	4/8	8 GB	2.8 GHz
Intel Core 2 Quad Q9300	4/4	4 GB	2.5 GHz
Intel Core 2 Duo E7300	2/2	2 GB	2.66 GHz

Tabla 2. Características de las GPUs empleadas

GPU	Cantidad de núcleos
nVidia GeForce GTX 460	336
nVidia GeForce GTX 550 Ti	192
nVidia GeForce GTX 260	216

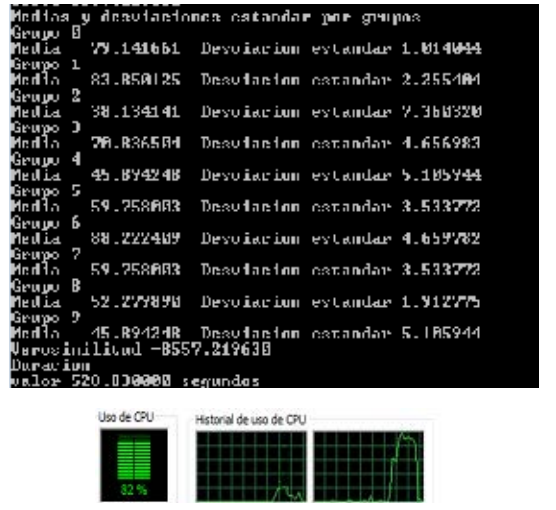

Fig. 4. Resultados de la ejecución secuencial.

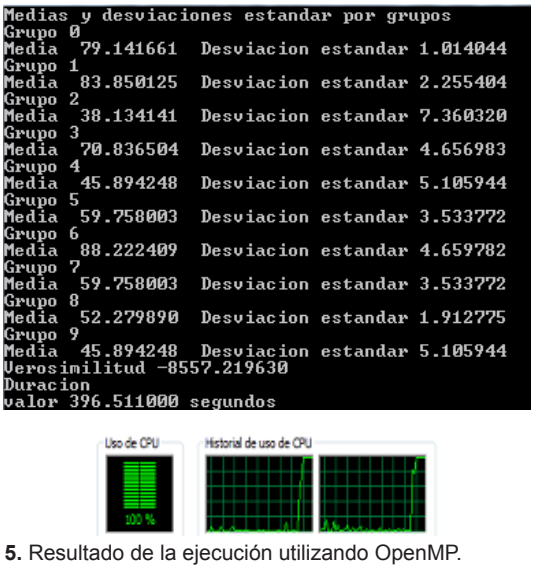

Fig. 5. Resultado de la ejecución utilizando OpenMP.

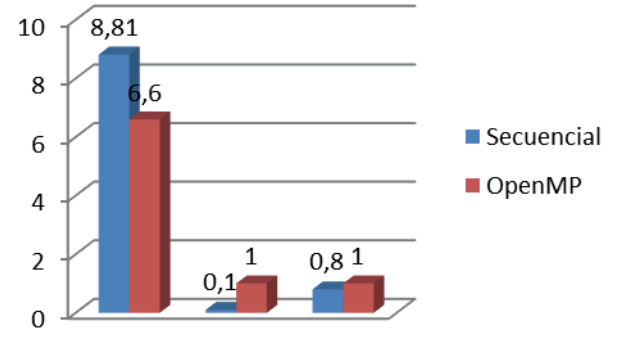

Fig. 6. Comparación de las ejecuciones secuencial y paralela utilizando OpenMP.

- Se obtuvo una disminución del 35% del tiempo de ejecución secuencial, lo cual significa que se está reduciendo considerablemente el tiempo de ejecución con la nueva alternativa de implementación.
- Se obtuvo el mismo patrón de los datos, lo que quiere decir que el algoritmo funciona correctamente con ambas implementaciones.
- Se realizó un aprovechamiento mayor de las capacidades multiprocesamiento del medio de cómputo ya que ambos núcleos fueron aprovechados al máximo de su capacidad y se redujo considerablemente el tiempo ocioso o de espera de los procesadores para la asignación de tareas.
- El algoritmo implementado de forma paralela se ejecuta 1,33 veces más rápido que el secuencial.

3.2. OpenCL

La Figura 8 muestra las comparaciones de los resultados obtenidos con las ejecuciones paralela y secuencial.

Nota: La ejecución del algoritmo utilizando OpenCL se realizó utilizando la GPU porque con este dispositivo se obtienen los mejores resultados, al tener muchas más unidades de procesamiento que un CPU.

```
Medias y desviaciones estandar por grupos
Grupo 0
Media   79.141661  Desviacion estandar 1.014044
Grupo 1
Media   83.850125  Desviacion estandar 2.255404
Grupo 2
Media   38.134040  Desviacion estandar 7.360293
Grupo 3
Media   70.836503  Desviacion estandar 4.656984
Grupo 4
Media   45.894234  Desviacion estandar 5.105981
Grupo 5
Media   59.758005  Desviacion estandar 3.533770
Grupo 6
Media   88.222409  Desviacion estandar 4.659782
Grupo 7
Media   59.758005  Desviacion estandar 3.533770
Grupo 8
Media   52.279891  Desviacion estandar 1.912773
Grupo 9
Media   45.894234  Desviacion estandar 5.105981
Plataforma: NVIDIA CUDA
Nombre dispositivo: GeForce GTX 550 Ti
Verosimilitud -8557.219630
Duracion
valor 84.611000 segundos
```

Fig. 7. Resultado de la ejecución utilizando OpenCL.

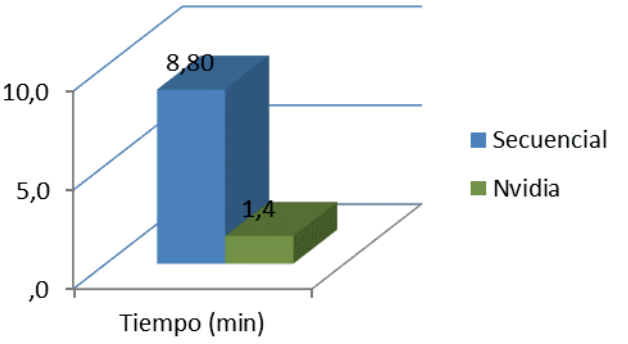

Fig. 8. Comparación de las ejecuciones secuencial y paralela utilizando OpenCL.

La Figura 9 y Figura 10 muestran los diferentes tiempos de ejecución utilizando OpenCL sobre CPU y GPU.

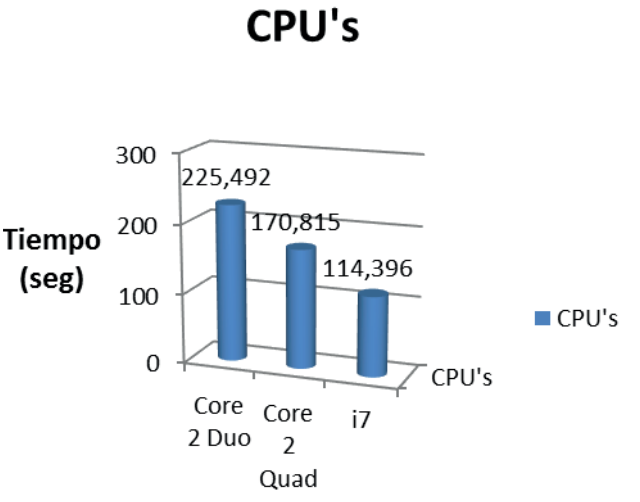

Fig. 9. Comparación entre CPUs OpenCL.

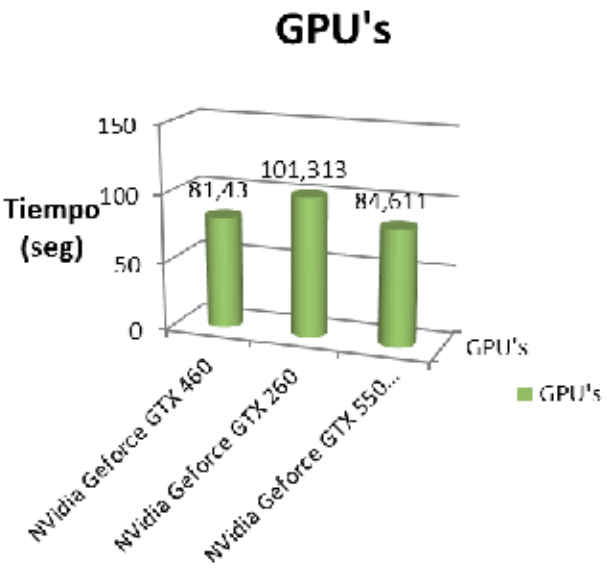

Fig. 10. Comparación entre GPUs OpenCL.

Se obtuvieron los mismos resultados en las 2 ejecuciones, o sea que la ejecución es correcta, partiendo de que la implementación secuencial también lo es.

El tiempo de ejecución de la plataforma NVIDIA fue de 1,4 minutos, que representa solo el 15% del tiempo obtenido por la ejecución secuencial; esto significa que se disminuyó el tiempo de ejecución en un 85%.

Se realizó un aprovechamiento mayor de las capacidades heterogéneas de cómputo disponibles.

El algoritmo implementado de forma paralela se ejecuta 6,3 veces más rápido que el secuencial.

La Tabla 3 muestra un resumen de los valores obtenidos con las diferentes bibliotecas empleadas.

4. CONCLUSIONES

El aprovechamiento de las arquitecturas actuales permite que se realicen ejecuciones de algoritmos en menor tiempo; en especial, aquellos que trabajan con grandes volúmenes de datos, como es el caso de los algoritmos relacionados con la minería de datos. Para lograr un aprovechamiento de estas posibilidades, es necesario realizar estudio de la tecnología y del proceso al cual se le desee realizar la paralelización. Se deben buscar cuáles son las zonas críticas en procesamiento, para así dirigir los esfuerzos hacia esa sección del proceso.

Tabla 3. Resumen de tiempos

Ejecución	Tiempo(min)	% reducción
Secuencial	8,8	0
OpenMP	6,6	35
OpenCL	1,4	85

Como resultado de esta investigación, se cuenta con una implementación del algoritmo expectación-maximización, que disminuye su tiempo de ejecución secuencial y que puede ser usada en procesos que requieran agrupamiento de los datos.

El menor tiempo de ejecución de los CPUs utilizados, lo presenta el Intel Core i7, mientras que el mayor tiempo de ejecución lo presenta el Intel Core 2 Duo. La diferencia de tiempos se debe a la cantidad de hilos de ejecución y frecuencias de reloj que tiene cada uno de los procesadores. El CPU Intel Core i7 es la elección si se desea ejecutar el algoritmo sobre CPU.

El menor tiempo de ejecución de las GPUs utilizadas lo presenta la tarjeta de video nVidia GeForce GTX 460, mientras que el mayor tiempo de ejecución lo presenta la nVidia GeForce GTX 260. La diferencia de tiempos se debe a la cantidad de hilos de ejecución, anchos de banda, frecuencias de reloj y memoria que tiene cada una de las tarjetas gráficas. Debido a que el algoritmo no es 100% paralelizable, la parte secuencial se realiza en el CPU; y cuando se ejecuta sobre la GPU nVidia GeForce GTX 460, la parte secuencial se ejecuta en el CPU Intel Core i7, que es el CPU más rápido. Teniendo en cuenta los aspectos anteriores, la elección, si se desea ejecutar el algoritmo sobre GPU, es nVidia GeForce GTX 550 Ti, ya que la diferencia de tiempo de ejecución entre esta y la 460 es poco significativa y esta arquitectura cuenta con el CPU más lento de los empleados, que es el Intel Core 2 Duo.

Existe una considerable disminución del tiempo de ejecución con respecto a la implementación secuencial, siendo el tiempo paralelo 2 o más veces más rápido que el secuencial.

5. RECOMENDACIONES

Se recomienda realizar una implementación paralela del algoritmo, enfocada en un ambiente distribuido, para utilizar los recursos de más de una computadora y así realizar una mayor disminución de los tiempos de ejecución obtenidos como resultado de la investigación.

6. ANEXO: GLOSARIO

CUDA: Siglas de Compute Unified Device Architecture (Arquitectura de Dispositivos de Cómputo Unificado). Hace referencia tanto a un compilador como a un conjunto de herramientas de desarrollo creadas por NVidia que permiten a los programadores usar una variación del lenguaje de programación C para codificar algoritmos en GPU de NVidia [10, 11, 16].

OpenCL: Open Computing Language, en español, Lenguaje de Computación Abierto. Consta de una interfaz de programación de aplicaciones y de un lenguaje de programación. Juntos permiten crear aplicaciones con paralelismo a nivel de datos y de tareas que pueden ejecutarse tanto en unidades centrales de procesamiento como en unidades de procesamiento gráfico [12-14].

OpenMP: Interfaz de programación de aplicaciones (API) para la programación multiproceso de memoria compartida en múltiples plataformas. Permite añadir concurrencia a los programas escritos en C, C++ y Fortran sobre la base del modelo de ejecución fork-join [6].

Expectación-Maximización: Se usa en estadística para encontrar estimadores de máxima verosimilitud de parámetros en modelos probabilísticos que dependen de variables no observables [2-4, 17].

Minería de datos: Consiste en la extracción no trivial de información que reside de manera implícita en los datos. Dicha información era previamente desconocida y podrá resultar útil para algún proceso [1, 18, 19].

Arquitecturas multinúcleo: Es el diseño conceptual y la estructura operacional fundamental de un sistema de computadora cuya unidad de procesamiento está compuesta por varios núcleos con uno o varios hilos de ejecución [5, 20].

Kernel: Un kernel de OpenCL es una porción paralela (muy similar a una función) que es ejecutada en el device. Los hilos pueden ser de dos tipos: físicos y virtuales [12-14].

Thread: Hilo de ejecución que va a tener cada uno de los núcleos del procesador [12-14].

Device: Tarjeta de procesamiento gráfico [12-14].

REFERENCIAS

[1] J. Hernández-Orallo, J. Ramírez-Quintana, Introducción a la minería de datos, Primera edición, Madrid, Pearson Prentice Hall S.A., 2004, 680p.

[2] A. P. Dempster, D. B. Rubin,"Maximum likelihood from incomplete data via the EM algorithm". Journal of the *Royal Statistical Society*, pp. 39-45, 1977.

[3] Collins, M, *The EM Algorithm,* p. 28, 1997. Disponible en: http://faculty.washington.edu/fxia/courses/LING572/EM_collins97.pdf

[4] S.Purcell, *Maximum Likelihood Estimation* (MLE), 2007, Disponible en: http://statgen.iop.kcl.ac.uk/bgim/mle/sslike_1.html

[5] I. Foster, *Designing and building parallel programs*, Addison-Wesley, 1995. Disponible en: http://www.mcs.anl.gov/~itf/dbpp

[6] OpenMP.org. *The OpenMP API Specification for parallel programming,* 2014, Disponible en: http://www.openmp.org

[7] R. Pas, "*An Overview of OpenMP 3.0*", IWOMP 2009, TU Dresden (Alemania), Disponible en: https://iwomp.zih.tu-dresden.de/downloads/2.Overview_OpenMP.pdf

[8] H. Hoeger, *Introducción a la Computación* Paralela, Centro Nacional de Cálculo Científico Universidad de Los Andes, Mérida (Venezuela)–CeCalCULA, 2011, p. 48.

[9] M. Tosini, *Introducción a las arquitecturas paralelas*, 2011, p. 51. Disponible en: http://bit.ly/1H9pylQ

[10] NVIDIA, High Performance computing with CUDA, Users Group Conference, San Diego, CA (USA), 2009. Disponible en: www.nvidia.com

[11] J.A. Raya-Vaquera, E.X. Martín-Rull, *Aceleración con CUDA de procesos 3D,* 2009. Disponible en: http://bit.ly/1Gg7mRu

[12] Wang, P., OpenCL Optimization, GPU Technology Conference, San José, CA (USA), 2009.

[13] A. Munshi, T. G. Mattson, J. Fung, D. Ginsburg, *OpenCL Programming Guide,* Boston, MA (USA), Pearson Education, 2012.

[14] A. Barak, E. Levy, A. Shiloh, A Package for OpenCL Based Heterogeneous Computing on Clusters with Many GPU Devices, IEEE International Conference on Cluster Computing Workshops and Posters, Heraklion, Isla de Creta, 2010.

[15] B. R. Gaster, L. Howes, D. R. Kaeli, P. Mistry, D. Schaa, "Chapter 13–OpenCL Profiling and Debugging", en *Heterogeneous Computing with Opencl*, Editado por B. R. Gaster, L. Howes, D R. KaeliPerhaad, M.D. Schaa, M. Kaufmann, Boston, 2013, pp. 243-261, ISBN 9780124058941.

[16] *NVIDIA, NVIDIA CUDA Compute Unified Device Architecture,* 2014. Disponible en: http://www.nvidia.com/object/cuda_home_new.html

[17] E.W. Weisstein, *K-Means Clustering Algorithm.* 2014. Disponible en: http://mathworld.wolfram.com/K-MeansClusteringAlgorithm.html

[18] M.I. Ángeles-Larrieta, A.M. Santillán-Gómez "Minería de datos: Concepto, características, estructura y aplicaciones", *Contaduría y Administración,* no. 190, p.p. 79-84, 2004.

[19] Vallejos, S.J. "Minería de datos", *Facultad de Ciencias Exactas, Naturales y Agrimensura,* Argentina: Universidad Nacional del Nordeste, 2006.

[20] P. Graham, *Parallel Computation Notes,* 1996. Disponible en: http://www.cs.umanitoba.ca/~comp4510/notesDIR/COMP4510Models_2up.pdf.