Revista Cubana de Ciencias Informáticas
Vol. 10, No. 4, Octubre-Diciembre, 2016
ISSN: 2227-1899 | RNPS: 2301
http://rcci.uci.cu
Pág. 1-14

Tipo de artículo: Artículo original
Temática: Programación paralela y distribuida
Recibido: 26/08/2015 | Aceptado: 28/09/2016

Paralelización del algoritmo Fuzzy Miner con OpenCL

Fuzzy Miner algorithm parallelization with OpenCL

Marlis Fulgueira Camilo [1*], William Reyes Burunate [1]

[1*] Complejo de Investigación de Tecnologías Integradas (CITI). {mfulgueirac, wreyes}@citi.cu

[*] Autor para correspondencia: mfulgueirac@citi.cu

Resumen

La minería de procesos es una disciplina que ofrece un conjunto de algoritmos, que facilitan la obtención de información de registros de eventos. Sin embargo, el procesamiento de grandes cantidades de datos en ocasiones resulta costoso, debido a los tiempos de ejecución dedicados a esta tarea. El presente artículo refleja una implementación del algoritmo de minería de procesos Fuzzy Miner usando el lenguaje C++. El algoritmo implementado es la base de una segunda propuesta realizada en el artículo, en la cual se emplean técnicas de programación paralela, específicamente, OpenCL. El objetivo que se persigue con este estudio es disminuir el tiempo de procesamiento del algoritmo Fuzzy Miner implementado. Las pruebas efectuadas emplean diferentes datos y distintas arquitecturas hardware, logrando disminuir significativamente el tiempo de ejecución al procesar lo datos en paralelo. Los resultados más relevantes se obtienen con los registros de eventos más grandes y las mejores arquitecturas.

Palabras clave: Computación Paralela, Fuzzy Miner, OpenCL.

Abstract

Process mining is a discipline that provides a set of algorithms that facilitate obtaining information from event logs. However, the processing of large amounts of data sometimes costly, due to the execution times dedicated to this task. This article reflects an implementation of the algorithm Fuzzy Miner mining process using the C ++ language. The implemented algorithm is based on a proposal made in the second article, in which parallel programming techniques are used, specifically, OpenCL. The objective pursued with this study is to reduce the processing time of the algorithm implemented Fuzzy Miner. The tests carried out using different data and different hardware platforms, thus

significantly reducing runtime while processing the data in parallel. The most relevant results are obtained with the records of major events and the best architectures.

Keywords: *Miner Fuzzy, OpenCL, Parallel Computing.*

Introducción

La Minería de Procesos es un área de investigación que permite realizar un estudio metódico de un proceso, basado en información y hechos concretos (Drucker and Sierra 2003; Weijters, van Der Aalst et al. 2006; Smart, Maddern et al. 2009; Van Der Aalst 2011). Los algoritmos de minería de procesos deben ser capaces de ofrecer una vista de alto nivel de los procesos analizados. Entre ellos se pueden citar Alpha Mining (van Dongen, De Medeiros et al. 2009; Zeng, He et al. 2011; Aruna Devi. C September – October 2013), Genetic Mining (Ghosh, Biswas et al. 2010; Zeng, He et al. 2011), Heuristic Miner (Zeng, He et al. 2011; .R.J June 2011) y Fuzzy Mining (Zeng, He et al. 2011).

El presente artículo está enfocado específicamente en el algoritmo Fuzzy Miner, el cual permite reducir un registro de eventos (Xia 2010; Zeng, He et al. 2011). Un evento representa una actividad específica, realizada durante un proceso en particular. El evento puede tener varios atributos, como: identificador, fecha de ejecución, originador, entre otros. El uso de este algoritmo con grandes cantidades de datos se identifica como un problema, debido a los largos tiempos de ejecución dedicados a dicha tarea. La disminución del tiempo de ejecución de algoritmos complejos, es tema común en diversos campos de desarrollo en la actualidad. La computación paralela permite realizar simultáneamente un conjunto de cálculos aprovechando para ello los recursos disponibles de las computadoras paralelas, cuyo objetivo fundamental es disminuir el tiempo de ejecución de las aplicaciones respecto a lo que se pudiera lograr con un solo procesador (Parberry 1987; Carriero and Gelernter 1992; Jada 1992; Hariri and Parashar 2004). Diversas son los técnicas dedicados a este propósito, enfocados siempre a lenguajes de programación específicos como OpenMP (Sterling 2002; Bischof 2008; Chapman, Jost et al. 2008; Matloff 2011; Pacheco 2011; OpenMP 2015), CUDA (Luebke 2008; Nickolls, Buck et al. 2008; Sanders and Kandrot 2010; Farber 2011; Cook 2012; Wilt 2013) y OpenCL (Barak, Ben-Nun et al. 2010; Tsuchiyama, Nakamura et al. 2010; Gaster, Howes et al. 2012; Kowalik and Puźniakowski 2012; Munshi, Gaster et al. 2012; Scarpino 2012; AMD 2013; Intel 2013; Tay 2013). La bibliografía consultada (Fulgueira-Camilo, Insúa-Suárez et al. 2016; Fulgueira-Camilo, Insúa-Suárez et al. 2016) muestra avances en la paralelización de algoritmos de minería de procesos. La presente investigación pretende disminuir el tiempo de ejecución del algoritmo Fuzzy Miner incorporando técnicas de programación paralela.

Revista Cubana de Ciencias Informáticas
Vol. 10, No. 4, Octubre-Diciembre, 2016
ISSN: 2227-1899 | RNPS: 2301
http://rcci.uci.cu
Pág. 1-14

Algoritmo Fuzzy Miner

El algoritmo de minería de procesos Fuzzy Miner fue propuesto por Christian W. Günther y Wil M. P. van der Aalst (Günther and Van Der Aalst 2007). El mismo es utilizado en procesos poco estructurados, también conocidos como procesos "espagueti" (ver Figura 1). Dichos procesos por lo general son difíciles de comprender, ya que son visualizados todos los detalles de un proceso sin diferenciar entre importantes o no. La idea general del algoritmo Fuzzy Miner consiste en eliminar los eventos (actividades) menos representativos del proceso, con el objetivo de simplificar y visualizar mejor dicho proceso. Para ello, Fuzzy Miner se basa en dos métricas principales: importancia y correlación (Günther and Van Der Aalst 2007; Xia 2010).

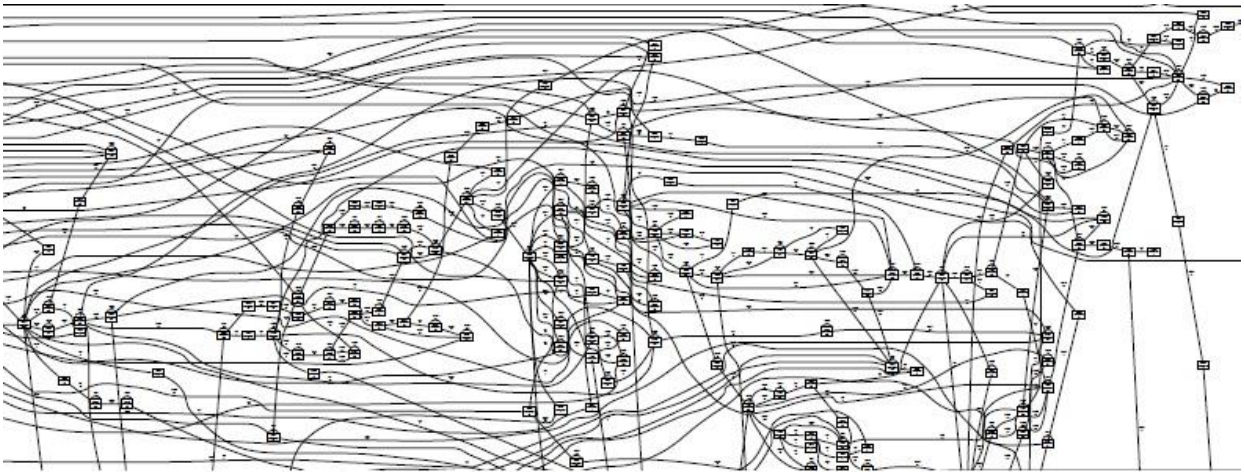


Figura 1. Visualización de un proceso espagueti (Günther and Van Der Aalst 2007).

Las métricas citadas anteriormente están asociados a las actividades presentes en los registros (log) de eventos. La importancia especifica el nivel de interés con que se desea que ocurra un evento o una sucesión de eventos (arista), es decir, la frecuencia de aparición de dicho evento o arista (Günther and Van Der Aalst 2007). Por otra parte, la correlación se determina entre aristas solamente e indica que tan cerca se encuentra un evento de otro (Günther and Van Der Aalst 2007). Dicha correlación se puede establecer de diversas maneras, según el interés particular del usuario final. En su generalidad se usan valores como el tiempo u originador de eventos para determinar una correlación. Por ejemplo, si dos eventos consecutivos se ejecutaron en tiempos cercanos o fueron realizados por un mismo originador se consideran correlacionados.

Basado en los valores de importancia y correlación la reducción del proceso se determina siguiendo estos tres criterios fundamentales (Günther and Van Der Aalst 2007):

1. Nodos con alta importancia persisten en el grafo.
2. Nodos con poca importancia y altamente correlacionados se agrupan en nodos.
3. Nodos con poca importancia y poco correlacionados se eliminan del grafo.

La visualización del algoritmo, por lo general, es guiada por cuatro aspectos fundamentales:

1. *Agregación*: Se forman nodos compuestos, con el objetivo de reducir la información mostrada.
2. *Abstracción*: Los nodos y aristas menos importantes, según su nivel de importancia y correlación no se visualizan.
3. *Énfasis*: La información importante, dígase nodos y aristas, se visualiza de manera exaltada, haciendo uso de colores, contraste y tamaño.
4. *Personalización*: El grafo final de nodos y aristas que se visualiza es cambiante, según el nivel de detalle que se desee visualizar.

Principios de la implementación del Fuzzy Miner

En el artículo se presenta una versión del algoritmo implementado en C++, la cual está dividida en dos fases fundamentales. La primera fase relacionada con la carga de los datos y la segunda fase enfocada a la obtención de cinco valores relacionados con los eventos y las aristas (ver Tabla 1).

Tabla 1. Datos de retorno

Nro	Nombre del Dato
1	Frecuencia Absoluta de eventos
2	Frecuencia Relativa de eventos
3	Frecuencia Absoluta de eventos en Trazas
4	Frecuencia Relativa de eventos en Trazas
5	Promedio de eventos por Traza
6	Frecuencia Absoluta de aristas
7	Frecuencia Relativa de aristas
8	Frecuencia Absoluta de aristas en Trazas
9	Frecuencia Relativa de aristas en Trazas
10	Promedio de aristas por Traza

En la implementación desarrollada, la importancia es representada por un valor decimal entre 0 y 1, basado en la frecuencia de aparición de eventos y aristas. Mientras que para la correlación se erigen los originadores[1], es decir, para dos eventos consecutivos si los originadores son iguales, entonces existe correlación. Por ejemplo, la Figura 2

[1] Originador: es la persona que realiza el evento

Revista Cubana de Ciencias Informáticas
Vol. 10, No. 4, Octubre-Diciembre, 2016
ISSN: 2227-1899 | RNPS: 2301
http://rcci.uci.cu
Pág. 1-14

muestra un pequeño registro de eventos compuesto por dos trazas. Como se puede apreciar existen cuatro eventos distintos A, B, C y D, donde los colores representan el originador del evento.

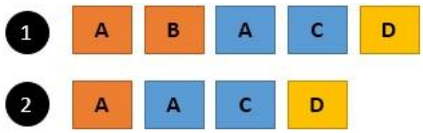


Figura 2. Registro de eventos compuesto por dos trazas.

La

Tabla *2* y la Tabla 3 muestran cómo queda la frecuencia y la importancia para eventos y aristas respectivamente, donde al evento con mayor frecuencia le corresponde la mayor importancia.

Tabla 2. Frecuencia e importancia de eventos.

Eventos	**A**	**B**	**C**	**D**
Frecuencia	4	1	2	2
Importancia	1	0.25	0.50	0.50

Tabla 3. Frecuencia e importancia de aristas.

Aristas	**A->B**	**B->A**	**A->C**	**C->D**	**A->A**
Frecuencia	1	1	2	2	1
Importancia	0.50	0.50	1	1	0.50

El valor de correlación se determina teniendo en cuenta los originadores de los eventos, la

Tabla *4* refleja cómo quedaría dicha correlación y su correspondiente importancia.

Tabla 4. Correlación e importancia de aristas.

Aristas	**A->B**	**B->A**	**A->C**	**C->D**	**A->A**
Correlación	1	0	2	0	0
Importancia	0.50	0	1	0	0

Una vez obtenidos todos los valores mostrados en la Tabla 1, se visualizan todos los datos correspondientes a los eventos y aristas.

Diseño paralelo del algoritmo Fuzzy Miner

En la propuesta de implementación del Fuzzy Miner se identifican dos fases fundamentales, explicadas brevemente en el epígrafe anterior. La primera fase es totalmente secuencial, mientras que en la segunda se identifican regiones que pueden ser paralelizadas. Estas regiones a paralelizar fueron reconocidas después de analizar y ejecutar el algoritmo con diversos registros de eventos. La Figura 3 muestra dicha región y en negrita los valores determinantes para la

Revista Cubana de Ciencias Informáticas
Vol. 10, No. 4, Octubre-Diciembre, 2016
ISSN: 2227-1899 | RNPS: 2301
http://rcci.uci.cu
Pág. 1-14

ejecución eficiente del algoritmo de forma paralela, principalmente las variables que contienen la cantidad de eventos y la cantidad de trazas (*sizeEvents* y *sizeTraces respectivamente)*. Estas dos variables determinan la aceleración de la implementación paralela, es decir, a mayor cantidad de eventos y trazas, el tiempo de ejecución del algoritmo paralelo será mejor respecto al secuencial.

ALGORITMO: Calculate_Fuzzy_Metrics

Requiere: Todos los eventos diferentes que existen en la BD, $pEvents$
Todas las trazas, $pTraces$
1: Sea ***sizeEvents*** el tamaño de $pEvents$
2: Sea ***sizeTraces*** el tamaño de $pTraces$
3: Sea $metricsF$ una estructura formada por:
4: $absoluteFrequency$
5: $absoluteFrequencyInTrace$
6: $average$
7: Sea $metricsFuzzy$ una lista de $metricsF$ para cada evento de $pEvents$
8: **for all** $pEvents_i$ en $pEvents < sizeEvents$ **do**
9: **for all** $pTraces_j$ en $pTraces < sizeTraces$ **do**
10: $metricFuzzy[i].absoluteFrequencyInTrace = Counter\_That\_Event\_In\_That\_Trace(pEvents_i, pTraces_j)$
11: $metricFuzzy[i].absoluteFrequency += metricFuzzy[i].absoluteFrequencyInTrace$
12: $metricFuzzy[i].average = absoluteFrequency/sizeTraces$
13: **end for**
14: **end for**
15: **return** $metricsFuzzy$

Figura 3. Algoritmo identificado para aplicar técnicas de paralelización.

El algoritmo almacena los eventos y las aristas diferentes en dos listas separadas, los cuales son identificados inequívocamente por un número. La paralelización consiste básicamente en obtener los datos 1, 3 y 5 de la Tabla 1 de manera paralela. Siguiendo con la idea de la Figura *2*3 el procedimiento secuencial calcula dichos datos para los eventos A, B, C y D (ver Figura 4). Mientras que de forma paralela se podrían acortar los tiempos de ejecución (ver Figura 5).

Figura 4. Ejecución secuencial.

Revista Cubana de Ciencias Informáticas
Vol. 10, No. 4, Octubre-Diciembre, 2016
ISSN: 2227-1899 | RNPS: 2301
http://rcci.uci.cu
Pág. 1-14

Figura 5. Ejecución paralela.

La implementación paralela del algoritmo es desarrollada utilizando OpenCL. OpenCL fue creado por el Grupo Khronos[2] en el 2008, es un estándar abierto que posibilita el uso de dispositivos CPU y GPU. Provee una API de funciones para interactuar con los dispositivos disponibles en la arquitectura de hardware (Barak, Ben-Nun et al. 2010; Tsuchiyama, Nakamura et al. 2010; Gaster, Howes et al. 2012; Kowalik and Puźniakowski 2012; Munshi, Gaster et al. 2012; Scarpino 2012; AMD 2013; Intel 2013; Tay 2013).

Validación de resultados

Las pruebas realizadas pretenden evaluar el algoritmo paralelo implementado. Para ello se emplean métricas para evaluar algoritmos paralelos como la aceleración (Stallings 2010; Gebali 2011; Pacheco 2011) y la eficiencia paralela (Kumar, Grama et al. 1994; Pacheco 2011; McCool, Reinders et al. 2012). Además, se emplean varios escenarios de prueba los cuales se describen a continuación:

Arquitectura 1: Intel Core 2 Quad y una tarjeta de video AMD Radeon HD 6850.

Arquitectura 2: Intel Core i7 y una tarjeta de video nVidia Geforce GTX 260.

Los detalles de las arquitecturas se muestran en la Tabla 5. Los dispositivos pertenecen a distintos fabricantes, con generaciones y arquitecturas que surgieron entre el 2007 y el 2012.

Tabla 5. Características de las distintas arquitecturas utilizadas

Hardware	Tipo	Elementos de Cómputo	Frecuencia del procesador (MHZ)	RAM	Tamaño RAM (MB)	Frecuencia RAM (MHZ)	Ancho de Bus (bits)	Ancho de banda (GB/s)
Intel Core 2 Quad Q9300	CPU	4	2500	DDR2	4096	800	64	12.4

[2] El Grupo Khronos es un consorcio industrial financiado por sus miembros enfocado a la creación de APIs estándares abiertas y libres que permiten la creación y reproducción multimedia en un amplio abanico de plataformas y dispositivos

Intel Core i7-920	CPU	8	2670	DDR3	6144	1066	64	25.5
nVidia Geforce GTX 260	GPU	216	1240	DDR3	896	1998	448	111.9
AMD Radeon HD 6850	GPU	960	790	GDDR 5	1024	1000	256	128

La Tabla 6 muestra las características de las Bases de Datos empleadas, teniendo como fundamental propiedad, el aumento del número de eventos y trazas de un proceso analizado.

Revista Cubana de Ciencias Informáticas
Vol. 10, No. 4, Octubre-Diciembre, 2016
ISSN: 2227-1899 | RNPS: 2301
http://rcci.uci.cu
Pág. 1-14

Tabla 6. Características de la distintas Bases de Datos utilizadas.

Base de Datos	Nro Eventos	Nro Trazas	Tipo de BD
BD-1	21	124	SQL Server
BD-2	23	2315	Oracle
BD-3	28	675767	SQL Server
BD-4	85184	675767	SQL Server

Los tiempos de ejecución del Fuzzy Miner equivalen al promedio de 5 iteraciones del algoritmo para todas las pruebas realizadas. El sistema operativo empleado es Windows 7 de 64 bits. La Figura 6 muestra los resultados al ejecutar el algoritmo con las 4 bases de datos y sobre la arquitectura 1. Se puede apreciar, que con las primeras tres bases de datos los tiempos de ejecución se comportan relativamente iguales, no siendo así para la mayor de las bases de datos. Los tiempos de ejecución de esta última comienzan a disminuir al ejecutar en paralelo.

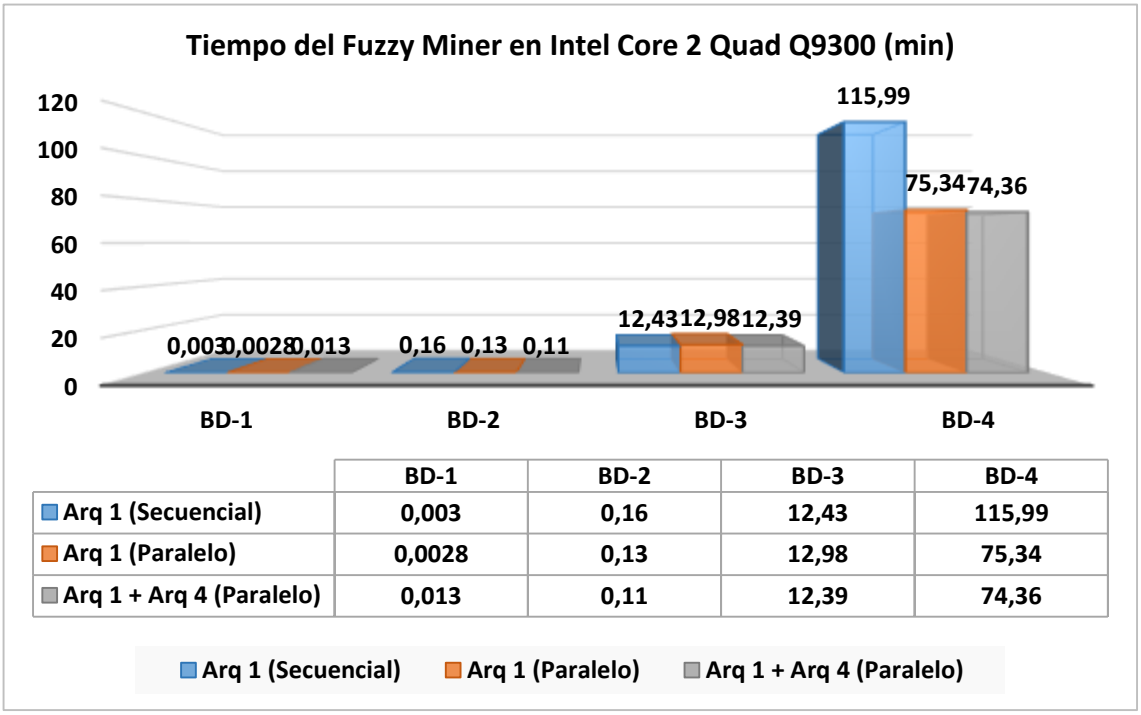


Figura 6. Tiempos de ejecución del Fuzzy Miner con las distintas BD.

La Figura 7 muestra el funcionamiento de dicho procesador, cuando ejecuta la implementación secuencial, específicamente en la porción reflejada en el **¡Error! No se encuentra el origen de la referencia.**. Mientras que la Figura 8 muestra dicho comportamiento pero con la implementación paralela.

Revista Cubana de Ciencias Informáticas
Vol. 10, No. 4, Octubre-Diciembre, 2016
ISSN: 2227-1899 | RNPS: 2301
http://rcci.uci.cu
Pág. 1-14

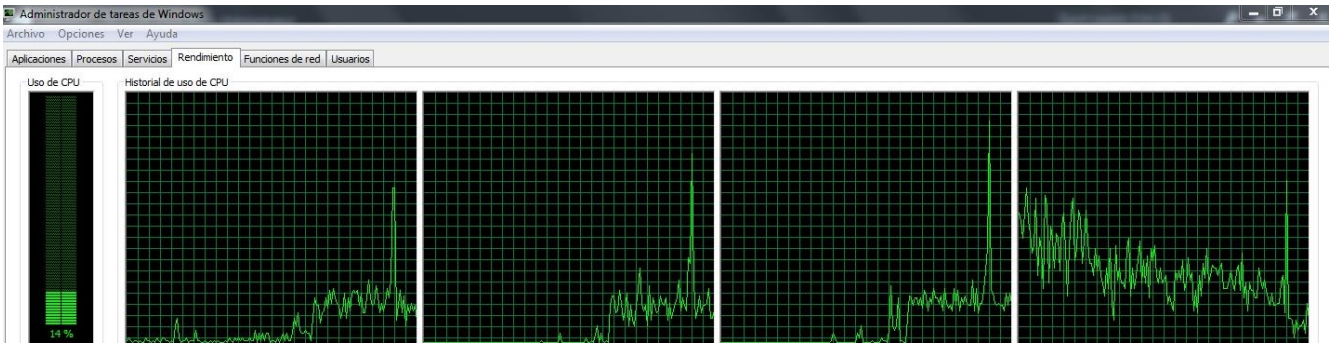


Figura 7. Comportamiento de cuatro hilos de ejecución secuencialmente.

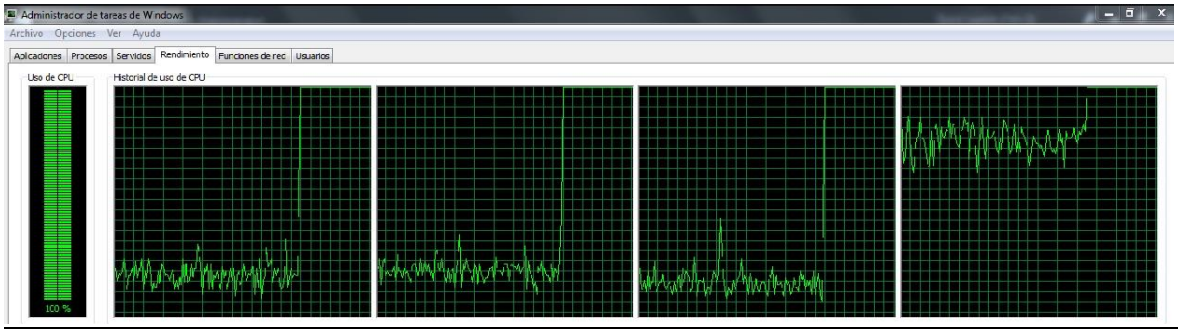
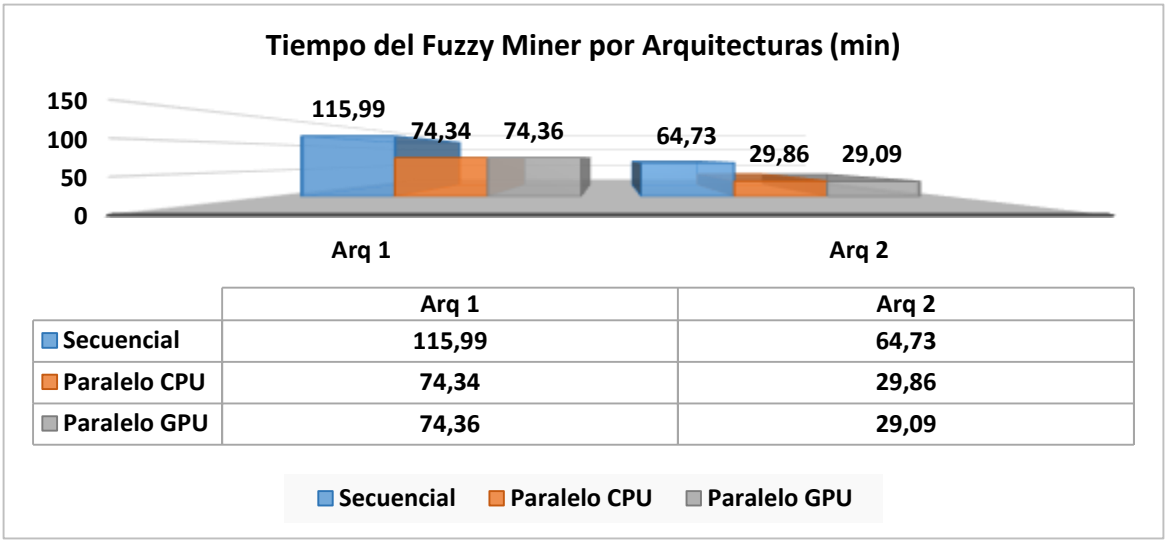


Figura 8. Comportamiento de cuatro hilos de ejecución paralelos.

El análisis anterior es repetido de forma similar en la arquitectura 2, pero solo con la base de datos más grande, ya que esta evidencia los mejores resultados y se realiza una comparación con los resultados obtenidos en la arquitectura 1. La Figura 9 refleja dichos valores junto con el resultado obtenido anteriormente en la Figura 6. Estos valores permiten discernir que la ejecución paralela en CPU y GPU prácticamente mantienen los tiempos de procesamiento muy parecidos. Dicho esto, se determina calcular y analizar las métricas de aceleración y eficiencia enfocándose específicamente en la ejecución secuencial y paralela en CPU.

Revista Cubana de Ciencias Informáticas
Vol. 10, No. 4, Octubre-Diciembre, 2016
ISSN: 2227-1899 | RNPS: 2301
http://rcci.uci.cu
Pág. 1-14



	Arq 1	Arq 2
Secuencial	115,99	64,73
Paralelo CPU	74,34	29,86
Paralelo GPU	74,36	29,09

Figura 9. Tiempo de ejecución del Fuzzy Miner en un procesador Core i7-920.

Como es posible apreciar en las Figura 6 y 9 a mayor cantidad de eventos el algoritmo paralelo es mejor que el secuencial. Teniendo en cuenta la Figura 3, es la cantidad de eventos quien define el tiempo de ejecución de los algoritmos.

Las métricas son aplicadas a los datos de la Figura 9. Según dichos valores la aceleración queda de la siguiente manera:

$$S = \frac{115.99}{75.34} = 1.54 \qquad S = \frac{64.73}{29.86} = 2.17$$

Los resultados permiten apreciar, que la ejecución del algoritmo paralelo alcanza un incremento de velocidad aproximadamente de 1.54 en la arquitectura 1 y 2.17 para la arquitectura 2 respecto a sus implementaciones secuenciales. Teniendo en cuenta estos valores, se observa mediante la **¡Error! No se encuentra el origen de la referencia.**10 un incremento del Speed Up de 0.63 entre estos dos procesadores.

Revista Cubana de Ciencias Informáticas
Vol. 10, No. 4, Octubre-Diciembre, 2016
ISSN: 2227-1899 | RNPS: 2301
http://rcci.uci.cu
Pág. 1-14

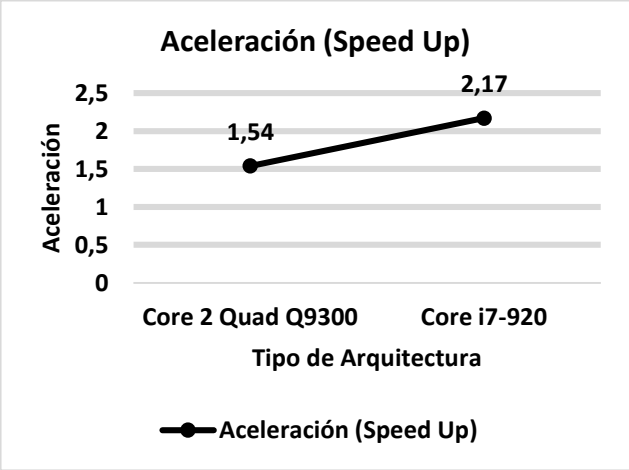


Figura 10. Aceleración de dos arquitecturas.

La eficiencia es calculada y analizada igualmente con las arquitecturas anteriores, enfocándose en los 4 y 8 hilos de ejecución que poseen estas arquitecturas. Los correspondientes resultados se presentan a continuación:

$$E = \frac{1.54}{4} = 0.39 \qquad E = \frac{2.17}{8} = 0.27$$

La Figura 11 muestra una mayor eficiencia por parte de la arquitectura 1, lo que demuestra un mayor uso del procesador al momento de procesar en paralelo el algoritmo Fuzzy Miner.

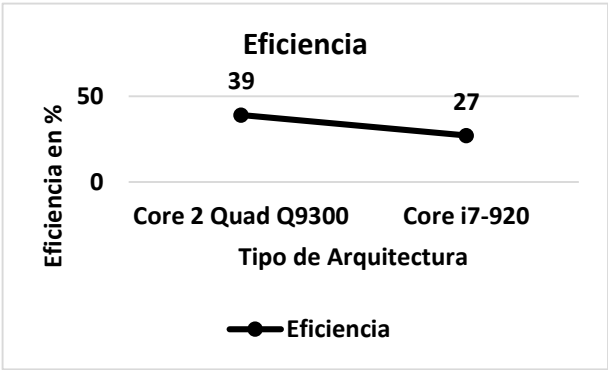


Figura 11. Eficiencia en dos arquitecturas.

Se evidencia que las CPU se usan eficientemente un 39% y 27% del tiempo de ejecución del algoritmo. El algoritmo tiene una componente secuencial que limita la eficiencia que puede lograse en una computadora. Teniendo en cuenta la Figura 8, se evidencia que en la ejecución paralela cuando se llega a la porción paralelizada el algoritmo logra

aprovechar al 100% los hilos de ejecución de la arquitectura, pero existe una parte intrínsecamente secuencial que no puede aprovechar las capacidades de las arquitecturas de pruebas.

Conclusiones

El presente artículo cumple con los objetivos trazados inicialmente, propiciando una implementación paralela que posibilita disminuir el tiempo de la implementación secuencial cuando la cantidad de eventos es igual o mayor que 85184. Los resultados logrados, son de 1.54 y 2.17 de aceleración, para procesadores Intel Core 2 Quad Q9300 e Intel Core i7-920 respectivamente. Esto evidencia nuevamente las fortalezas de los nuevos procesadores y la utilidad de la Computación Paralela para lograr disminuir los tiempos de ejecución de algoritmos complejos. Por otra parte, se identifica como idóneos para este tipo de algoritmos los dispositivos CPU. Se recomienda como trabajo futuro el uso de esta implementación con bases de datos mayores. También es necesario el posterior análisis de la implementación para su ejecución en arquitecturas paralelas y distribuidas.

Referencias

AMD (2013). Accelerated parallel processing: OpenCL programming guide, Advanced Micro Devices, Inc.

ARUNA DEVI. C, D. S. (2013). "Application Of Business Process Mining Using Control Flow Perspective In Manufacturing Unit." (5).

BARAK, A., T. BEN-NUN, et al. (2010). A package for OpenCL based heterogeneous computing on clusters with many GPU devices. Cluster Computing Workshops and Posters (CLUSTER WORKSHOPS), 2010 IEEE International Conference on, IEEE.

BISCHOF, C. (2008). Parallel Computing: Architectures, Algorithms, and Applications, IOS Press.

CARRIERO, N. and D. GELERNTER (1992). How to Write Parallel Programs: A First Course MIT Press.

COOK, S. (2012). CUDA programming: a developer's guide to parallel computing with GPUs, Newnes.

CHAPMAN, B., G. JOST, et al. (2008). Using OpenMP: portable shared memory parallel programming, MIT press.

DRUCKER, P. F. and J. A. SIERRA (2003). El management del futuro, Sudamericana.

FARBER, R. (2011). CUDA application design and development, Elsevier.

FULGUEIRA-CAMILO, M., E. INSÚA-SUÁREZ, et al. (2016). "Aceleración del algoritmo "Alineamiento de trazas" empleando CUDA." Revista Cubana de Ingeniería **7**(1): 27-35.

FULGUEIRA-CAMILO, M., E. INSÚA-SUÁREZ, et al. (2016). "Implementación del Algoritmo Trace Alignment Empleando Técnicas de Programación Paralela." Revista de Ingeniería" Lámpsakos"(15): 11-21.

GASTER, B., L. HOWES, et al. (2012). Heterogeneous Computing with OpenCL: Revised OpenCL 1, Newnes.

GEBALI, F. (2011). Algorithms and Parallel Computing, Wiley.

GHOSH, S., S. BISWAS, et al. (2010). "Mining frequent itemsets using genetic algorithm." arXiv preprint arXiv:1011.0328.

GÜNTHER, C. W. and W. M. Van Der AALST (2007). Fuzzy mining–adaptive process simplification based on multi-perspective metrics. Business Process Management, Springer**:** 328-343.

GÜNTHER, C. W. and W. M. Van Der AALST (2007). Fuzzy mining–adaptive process simplification based on multi-perspective metrics. International Conference on Business Process Management, Springer.

HARIRI, S. and M. PARASHAR (2004). Tools and environments for parallel and distributed computing, John Wiley & Sons.

INTEL (2013). Intel SDK for OpenCL Applications. Optimization Guide, Intel Corporation.

JADA, J. (1992). An introduction to parallel algorithms, Addison Wesley.

KOWALIK, J. and T. PUŹNIAKOWSKI (2012). Using OpenCL: Programming Massively Parallel Computers, IOS Press.

KUMAR, V., A. GRAMA, et al. (1994). Introduction to parallel computing: design and analysis of algorithms, Benjamin/Cummings Publishing Company Redwood City, CA.

LUEBKE, D. (2008). CUDA: Scalable parallel programming for high-performance scientific computing. Biomedical Imaging: From Nano to Macro, 2008. ISBI 2008. 5th IEEE International Symposium on, IEEE.

MATLOFF, N. (2011). "Programming on parallel machines." University of California, Davis.

MCCOOL, M., J. REINDERS, et al. (2012). Structured parallel programming: patterns for efficient computation, Elsevier.

MUNSHI, A., B. GASTER, et al. (2012). OpenCL Programming Guide, Addison-Wesley Professional.

NICKOLLS, J., I. BUCK, et al. (2008). "Scalable parallel programming with CUDA." Queue **6**(2): 40-53.

OPENMP (2015). OpenMP Application Program Interface, OpenMP Architecture Review Board.

PACHECO, P. (2011). An introduction to parallel programming, Elsevier.

PARBERRY, I. (1987). Parallel complexity theory, John Wiley & Son.

SANDERS, J. and E. KANDROT (2010). CUDA by example: an introduction to general-purpose GPU programming, Addison-Wesley.

SCARPINO, M. (2012). Opencl in Action: How to Accelerate Graphics and Computation. NY, USA: Manning.

SMART, P., H. MADDERN, et al. (2009). "Understanding business process management: Implications for theory and practice." British Journal of Management **20**(4): 491-507.

STALLINGS, W. (2010). Computer Organization and Architecture: Designing for Performance, Prentice Hall.

STERLING, T. L. (2002). Beowulf cluster computing with Linux, MIT press.

TAY, R. (2013). OpenCL Parallel Programming Development Cookbook, Packt Publishing.

TSUCHIYAMA, R., T. NAKAMURA, et al. (2010). "The opencl programming book." Fixstars Corporation.

Van Der AALST, W. (2011). Process mining: discovery, conformance and enhancement of business processes, Springer Science & Business Media.

van DONGEN, B. F., A. A. De MEDEIROS, et al. (2009). Process mining: Overview and outlook of petri net discovery algorithms. Transactions on Petri Nets and Other Models of Concurrency II, Springer**:** 225-242.

WEIJTERS, A., W. M. van Der AALST, et al. (2006). "Process mining with the heuristics miner-algorithm." Technische Universiteit Eindhoven, Tech. Rep. WP **166**: 1-34.

WILT, N. (2013). CUDA Handbook: A Comprehensive Guide to GPU Programming, Addison-Wesley Professional.

XIA, J. (2010). "Automatic determination of graph simplification parameter values for fuzzy miner." Eindhoven University of Technology. Netherlands.

ZENG, R., X. HE, et al. (2011). A Method to Build and Analyze Scientific Workflows from Provenance through Process Mining. TaPP.