Vol. VII, No. 1, enero - abril, 2016, pp. 27 - 35

Ingeniería Informática

Aceleración del algoritmo “Alineamiento de trazas” empleando CUDA

Marlis Fulgueira Camilo
correo electrónico: mfulgueirac@citi.cu
Complejo de Investigaciones Técnológicas Integradas (CITI), La Habana, Cuba

Ernesto Insua Suárez
correo electrónico: einsuas@citi.cu
Complejo de Investigaciones Técnológicas Integradas (CITI), La Habana, Cuba

Humberto Díaz Pando
correo electrónico: hdiaz@citi.cu
Complejo de Investigaciones Técnológicas Integradas (CITI), La Habana, Cuba

Artículo Original

Resumen
Actualmente, los procesos de negocios que se ejecutan en las empresas generan grandes volúmenes de trazas. Dichas trazas son almacenadas en registros de eventos para su posterior análisis. El empleo de herramientas que permiten extraer conocimiento útil de la información registrada posibilita conocer qué sucede exactamente en una empresa y la existencia o no de anomalías del proceso ejecutado. El algoritmo “Alineamiento de Trazas” permite identificar el comportamiento común o más probable del proceso ejecutado, la ocurrencia de desviaciones y los contextos en que una o varias actividades son ejecutadas. Los experimentos realizados demuestran que el tiempo de ejecución depende de la cantidad de trazas que se desean alinear. El artículo que se presenta introduce técnicas de programación paralela, CUDA, con el objetivo de disminuir el tiempo de ejecución del algoritmo. Las características principales del algoritmo, así como otras implementaciones paralelas son analizadas con el fin de unificar las técnicas que puedan lograr los mejores tiempos. El algoritmo Parallel TA propuesto, disminuye aproximadamente once veces, respecto a la implementación secuencial.

Palabras claves: CUDA, “Alineamiento de trazas”, procesos de negocio

Recibido: 16 de noviembre del 2015 Aprobado: 27 de enero del 2016

INTRODUCCIÓN

La informatización de los procesos organizacionales ha fomentado el crecimiento de la información almacenada en las bases de datos. Como consecuencia de ello, auditar una empresa y conocer sus fortalezas o debilidades en cualquier proceso de negocio ejecutado, resulta ser una tarea muy engorrosa. La Minería de Procesos (MP) es una disciplina que provee un conjunto de herramientas que permiten entender y mejorar los procesos que se llevan a cabo [1]. La MP hace énfasis en tres áreas fundamentales: Descubrimiento de Procesos, Chequeo de Conformidad y Mejoramiento de Modelos. La técnica de descubrimiento permite determinar, partiendo de un registro de eventos, un modelo de procesos que caracterice el comportamiento presente en el registro analizado [1, 2]. El “Alineamiento de Trazas” es un algoritmo para descubrir el proceso que se basa en la alineación de las trazas. El algoritmo tiene como antecedente la alineación de secuencias biológicas [3]. Sigue los mismos principios que la bioinformática, pero su objetivo es alinear un conjunto de trazas. Al alinear un conjunto de trazas, los principales aspectos que demoran la ejecución del algoritmo, estuvieron influenciados por la cantidad de trazas y la marcada diferencia, respecto al tamaño, de cada una de ellas. La bibliografía consultada demuestra

avances importantes referentes a la paralelización de algoritmos para alinear secuencias biológicas [4-8]. El objetivo de la investigación es realizar un diseño paralelo del algoritmo que permita reducir el tiempo de ejecución del algoritmo secuencial.

El artículo se divide en cinco secciones, una primera donde se caracteriza el algoritmo “Alineamiento de Trazas” y una segunda enfocada a los principales resultados obtenidos con algoritmos paralelos similares a este. En la sección 3 se realiza el diseño e implementación paralela del algoritmo. Al finalizar, en la cuarta y quinta secciones, se realiza una serie de experimentos que permiten comprobar el objetivo de investigación y obtener las conclusiones del trabajo, a partir de los resultados expuestos.

MATERIALES Y MÉTODOS

El algoritmo “Alineamiento de Trazas” trata de posicionar un conjunto de trazas de forma tal que los eventos ocurridos en una traza coincidan en gran medida con los eventos de las restantes trazas. El el algoritmo cuenta con cinco pasos fundamentales [3]. En el primer paso se leen las trazas de las Bases de Datos y se seleccionan aquellas que son diferentes. Los pasos dos y tres se realizan con el objetivo de definir el orden en que alinean las trazas. El paso cuatro es donde se realiza el alineamiento de las trazas, la cual se detalla en la siguiente sección.

Descripción del algoritmo “Alineamiento de Trazas”

El algoritmo requiere antes de comenzar a alinear, la construcción de un árbol guía que indique cuáles son las trazas más parecidas. Para determinar estas trazas se construye una matriz, la cual recibe el nombre de *matriz de distancia*. La siguiente formulación explica el proceso seguido:

Sea S una matriz de orden n, donde n representa la cantidad de trazas a alinear según la distancia de Levenshtein. T_i y T_j representan dos trazas a las cuales se le calcula la distancia. La ecuación 1 muestra el procedimiento descrito.

$$S(i,j) = Levenshtein(T_i, T_j) \quad (1)$$

La métrica empleada en este trabajo está determinada por la distancia de Levenshtein [9]. La cual permite identificar cuáles son las trazas más cercanas entre sí. Los valores obtenidos en la matriz son la base para la construcción del árbol guía.

Los pares de trazas más cercanos son los primeros en alinearse. La alineación de pares de trazas se basa en el algoritmo Needleman-Wunsch [9] y este en la programación dinámica para encontrar el alineamiento óptimo entre dos secuencias. La idea fundamental es construir un alineamiento óptimo basado en las soluciones previas de las subsecuencias más pequeñas. La descripción siguiente explica cómo alinear pares de trazas.

Sean T_1 y T_2 dos trazas, se construye una matriz F (matriz de alineamiento), con fila i y columna j tal que $F(i,j)$ es la puntuación de mejor alineamiento entre el evento i de T_1 y el evento j de T_2. $F(i,j)$ es construida recursivamente, inicializando los valores $F(0,0)$, $F(0,1)$,y $F(1,0)$ en 0. Los valores de la matriz F se calculan desde la izquierda superior hasta la derecha inferior. Para obtener el valor $F(i,j)$ se tiene en cuenta si el evento i de T_1 y el evento j de T_2 son iguales, para lo cual se realizan operaciones de edición descritas previamente. El valor de $F(i,j)$ se calcula como se muestra en la ecuación 2:

$$F(i,j) = máx \begin{cases} F(i-1,j-1) + S(T_1(i), T_2(j)), \\ F(i-1,j) + I(T_1(i-1), T_1(i)), \\ F(i,j-1) + I(T_2(j-1), T_2(j)). \end{cases} \quad (2)$$

Luego de calcular la matriz, para encontrar el alineamiento, es necesario buscar el camino que mejor puntuación tenga. Para lo cual de la posición (i,j) se camina hacia la posición $(i-1,j-1)$,$(i-1,j)$ o $(i,j-1)$ siempre buscando el valor mayor que contenga cada celda. El movimiento hacia la posición $(i-1,j-1)$ significa $T_1(i)$ y $T_2(j)$, hacia la posición $(i-1,j)$ significa $T_1(i)$ y el símbolo de hueco $(-)$ y hacia la posición $(i-1,j)$ símbolo de hueco $(-)$ y $T_2(j)$. El recorrido termina cuando se llega a la posición $(0,0)$. El procedimiento descrito se llama *traceback* [3]. La figura 1 muestra el procedimiento descrito.

Una vez alineados los pares es necesario alinear los conjuntos de pares más cercanos entre sí. La alineación de múltiples trazas sigue el principio descrito, pero tiene en cuenta la frecuencia de repetición de los eventos. La ecuación 3 muestra dicho procedimiento.

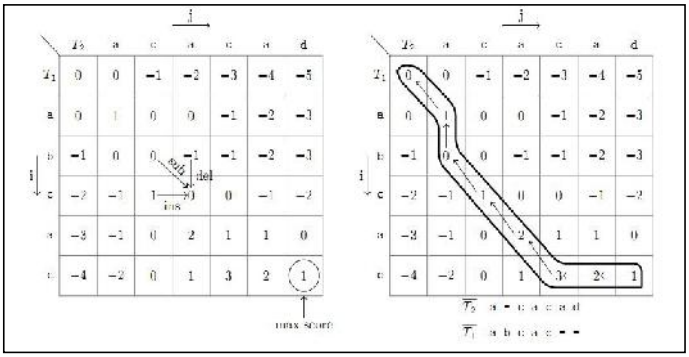

Fig. 1. Matriz F para calcular el alineamiento [3].

$$F(i,j) = máx \begin{cases} F(i-1,j-1) + \bar{S}(C_A^i, C_B^j), \\ F(i-1,j) + \bar{I}(C_A^{i-1}, C_A^i), \\ F(i,j-1) + \bar{I}(C_B^{j-1}, C_B^j). \end{cases} \quad (3)$$

Siendo $\bar{S}(C_A^i, C_B^j)$ el valor de sustituir la columna i del alineamiento A con la columna j del alineamiento B, donde $n_A^i(a)$ es la frecuencia de repetición del evento a en la columna i del alineamiento X y $\bar{I}(C_A^{i-1}, C_A^i)$ la puntuación de insertar la columna i en el alineamiento A dado su columna izquierda $i-1$ donde $f_A^i(a,b)$ es la frecuencia de la actividad b en la columna i del alineamiento A dado su evento vecino a en la columna $i-1$.

Antecedentes del algoritmo "Alineamiento de Trazas"

El algoritmo para alinear trazas tiene como antecedentes la alineación de secuencias biológicas [3]. En el siguiente acápite se realiza un estudio del estado del arte de los algoritmos de este tipo, que emplean técnicas de programación paralela. Los algoritmos para alinear secuencias biológicas se pueden clasificar atendiendo a la cantidad de secuencias que alinean. La figura 2 muestra dicha clasificación.

Las alternativas básicas para realizar el alineamiento de un par de secuencias son: el alineamiento local (Smith-Waterman) [4, 6, 8, 10, 11] y el alineamiento global (Needleman-Wunsch) [5]. Los dos algoritmos se basan en la programación dinámica [12] para encontrar la alineación óptima de dos secuencias. Ambos se apoyan en la construcción de una matriz F (matriz de alineamiento) para calcular el alineamiento.

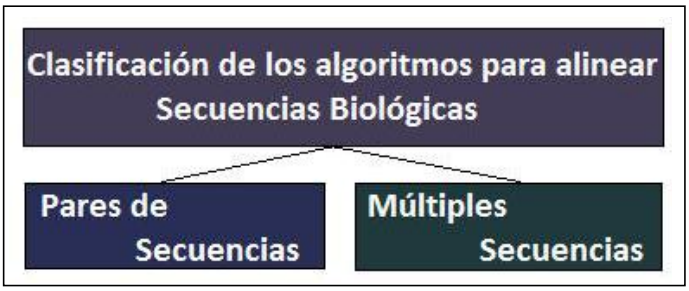

Fig. 2. Clasificación de los algoritmos para alinear secuencias biológicas teniendo en cuenta la cantidad de secuencias a alinear.

Los trabajos [4-8] hacen reseña a la alineación de dos secuencias. Las pruebas realizadas en los artículos indican que a medida que aumenta el tamaño de la secuencia alinear el tiempo de ejecución se eleva. Por ende, se enfatiza en la paralelización para disminuir el tiempo. El largo de las secuencias evaluadas oscila entre los 100 a 1000 caracteres, en algunos de los trabajos, el largo de las secuencias evaluadas supera en miles a estos números.

La figura 3 muestra el comportamiento seguido en la mayoría de diseños paralelos propuestos en [4-8]. La técnica frente de onda (*wave front*) (figura 4) es ampliamente empleada en varios documentos como son [4-6] y básicamente lo que plantea es calcular los elementos que se encuentren en la misma diagonal a la vez, puesto que ya debieron investigarse los elementos necesarios para el nuevo cálculo.

Fig. 3. Técnicas y modelos de programación paralela aplicados a la alineación de pares de secuencias.

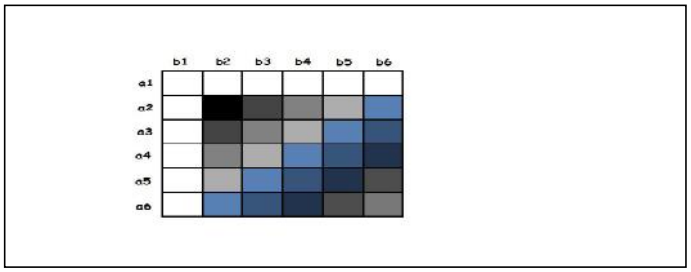

Fig. 4. Técnica frente de onda.

La figura 4 muestra dos secuencias S_1 y S_2. En una primera instancia deben calcularse los elementos de la columna y la fila de color blanco. Luego pueden calcularse en cada iteración los elementos de una diagonal paralelamente. La figura 4 muestra con los mismos colores los elementos que pertencen a cada diagonal y que pueden ser calculados paralelamente.

Además de los algoritmos para alinear pares de secuencias, también existen algoritmos para alinear múltiples secuencias [13, 14]. La estrategia empleada [15-18] es alinear progresivamente las secuencias, donde son alineadas una por una teniendo en cuenta el orden seguido en un árbol guía previamente calculado. La primera etapa del algoritmo es calcular la distancia entre todos los pares de secuencias. El resultado de este, es usado como distancia para construir el árbol guía. La última etapa es donde se realiza el alineamiento progresivo de las secuencias. La figura 5 muestra las variantes de paralelización seguidas por algunos autores en [15-17, 19, 20] .

Los algoritmos paralelos implementados generalmente emplean un clúster para paralelizar los algoritmos de este tipo. El paradigma que emplean [15, 16, 19] es MPI*, aunque existen reportes que emplean CUDA [17] para alinear múltiples secuencias.

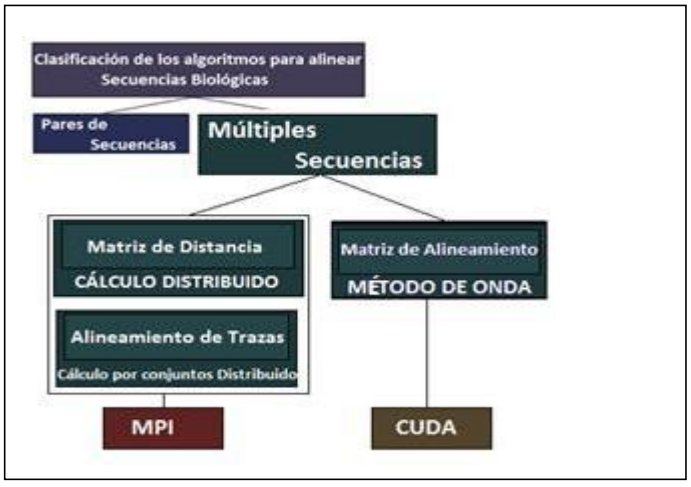

Fig. 5. Técnicas y modelos de programación paralela aplicados a la alineación de múltiples secuencias.

*MPI del inglés: Message Passing Interface (Interfaz de Paso de Mensajes).
**CUDA del inglés Unified Device Architecture (Arquitectura Unificada de Dispositivos de Cómputo).

Un diseño paralelo se detalla en [19]. Las secciones que paralelizan son el cálculo de la matriz de distancia que se divide en dependencia de la cantidad de nodos del clúster y la alineación de varios conjuntos de secuencias paralelamente. La aceleración obtenida con el algoritmo paralelo es de 5,5 veces.

Otro diseño se propone en [17] donde se enfocan solo en la alineación de las secuencias. El algoritmo implementado recibe un árbol guía como entrada, el cual obtiene del sitio oficial de Clustal W del EMBL-EBI. Emplean el método frente de onda (figura 4) para alinear los conjuntos de secuencias. El modelo de programación que emplean es CUDA. La aceleración obtenida por el algoritmo paralelo oscila entre los valores 6 y 15 veces.Teniendo en cuenta el estudio realizado se plantea lo siguiente:

- Los algoritmos para alinear pares de secuencias generalmente emplean el método frente de onda.
- Los trabajos donde se reflejan algoritmos paralelos para alinear múltiples secuencias se enfocan en dos secciones: la creación de la matriz de distancia y la alineación de secuencias.
- Los tiempos logrados empleando el método frente de onda [17] logran disminuir el tiempo de ejecución 6 veces más que los reportados con la alineación de varios conjuntos de trazas (MPI) paralelamente [19].

Parallel TA, un diseño e implementación paralela

El caso de estudio en el cual se enmarca dicha investigación radica en las trazas generadas por un proceso registro de personal. Las trazas se guardan en bases de datos, de las cuales solo se alinean las trazas que sean diferentes. Las trazas más largas contienen solo 49 caracteres y las más cortas tienen un solo carácter. A pesar de que las bases de datos contienen miles de trazas solo diferentes tienen 4 350.

Con el *objetivo de identificar* la región que más se demora en el algoritmo, se decide emplear la mayor cantidad de datos, 4 350 trazas. De las secciones descritas del algoritmo, calcular la matriz de distancia para construir el árbol guía y alinear las trazas son las únicas que permiten introducir técnicas de programación paralela. De estas la sección que más demora el tiempo de ejecución es alinear las trazas la cual representa un 68 % del tiempo total del algoritmo, el porcentaje restante equivale a las secciones faltantes.

Por lo tanto, se propone en el presente artículo paralelizar la matriz de distancia y emplear el método frente de onda para alinear las trazas. Al nuevo algoritmo paralelo que se propone, se le llamará Parallel TA, cuyo diseño es el siguiente:

Sección 1: Paralelización de la matriz de distancia

La matriz S o matriz de distancia es una matriz de orden $n \times n$, donde n es el total de trazas a alinear. La figura 6 muestra los pasos seguidos. Dicha matriz es simétrica por ende se propone solo calcular los elementos que se encuentren por debajo de la diagonal principal (diagonal de color negro). Cada uno de los elementos de la matriz puede ser calculado independientemente. Se pretende calcular la distancia de Levenshtein por cada elemento $S(i,j)$ (la distancia entre T_1 y T_2 es la misma que entre T_2 y T_1) y escribir el valor calculado en la posición $S(j,i)$.

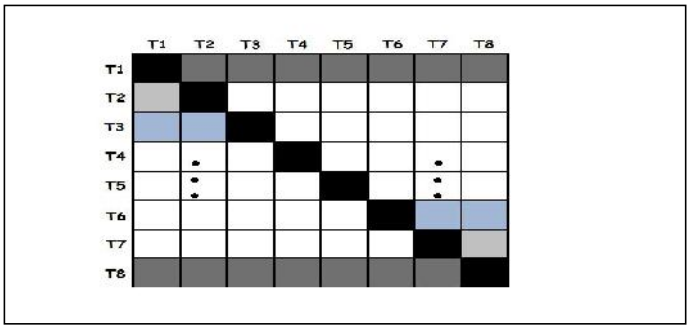

Fig. 6. Paralelización de la matriz S.

Los elementos de la matriz de distancia se dividen por filas. Los elementos de la fila x, representados con el color azul, se dividen en grupos en dependencia de la cantidad de hilos de ejecución disponibles en la arquitectura de hardware donde se ejecute el algoritmo. El objetivo de ello es que los hilos no hagan lecturas dispersas en la memoria, sino que trabajen con los elementos que le siguen.

Una vez calculada la matriz de distancia se necesita buscar la traza con menor valor. Se propone buscar el menor valor por cada fila de matriz y luego comparar dichos valores para encontrar el menor. La figura 7 muestra el procesamiento seguido para encontrar el menor en cada fila (paso 1) y la figura 8 muestra el comportamiento empleado para encontrar la traza con menor distancia entre todas (paso 2).

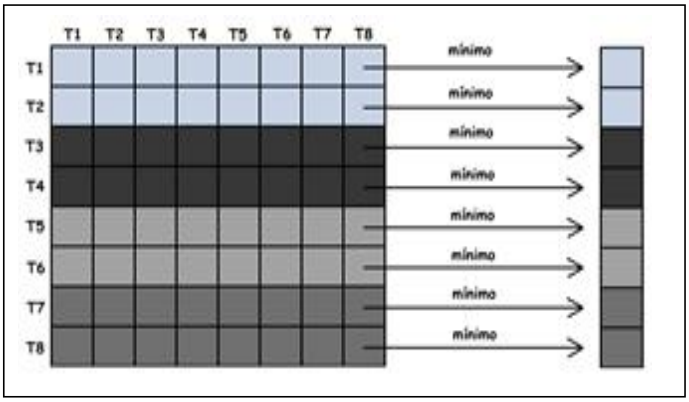

Fig. 7. Cálculo del mínimo asociado a cada hilo.

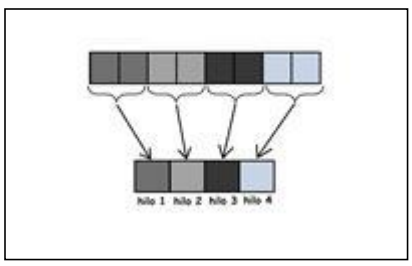

Fig. 8. Cálculo del mínimo por hilo de procesamiento.

La búsqueda del menor valor de distancia requiere la misma filosofía explicada con anterioridad. A cada hilo se le asignó un conjunto de filas para buscar. Las filas del mismo color representadas en la figura, son calculadas por un hilo. Los menores de cada fila se guardan en un arreglo de tamaño N, donde N es la cantidad de trazas a alinear. Para buscar el menor de todos, el arreglo también es divido en dependencia de la cantidad de hilos, cada uno de ellos busca su menor guardando su resultado en un arreglo de tamaño *M*, donde *M* es la cantidad de hilos disponibles. El arreglo de tamaño *M* es representado en la figura, donde los elementos del mismo color son los calculados por cada uno de los hilos. Luego un solo hilo es el encargado de comparar los resultados en el arreglo de tamaño *M*.

Sección 2: Paralelización de la matriz F

A pesar de que cada par de trazas o conjunto de trazas a alinear en el nivel *x* de árbol es independiente, se propone realizar un solo alineamiento a la vez y calcular empleando el método de onda la matriz F. Esta matriz tiene la propiedad de que se calcula en una primera instancia la primera fila y la primera columna, luego se calcula el resto de la matriz. Por lo tanto se propone también calcular la primera fila y la primera columna a la misma vez. La figura 9 muestra el procedimiento descrito.

El problema en este caso es que sin importar cuántos hilos de ejecución provea la arquitectura de hardware, solo en esta sección se empleará dos, un primer hilo para calcular la columna refleja con color azul y un segundo hilo para calcular la columna reflejada con color gris. El cálculo de los elementos restantes de la matriz se evidencia en la figura 10.

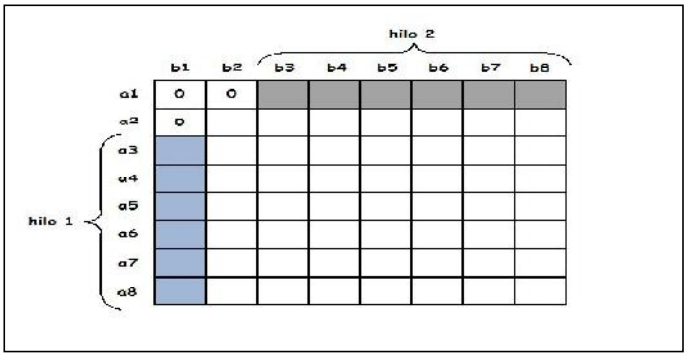

Fig. 9. Cálculo de la primera fila y columna a la vez.

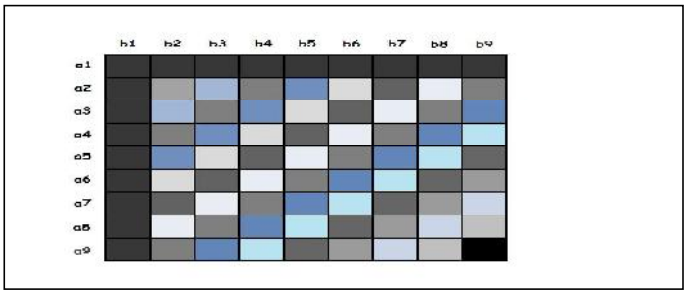

Fig. 10. Cálculo de los elementos que se encuentren en la misma diagonal a la vez.

El problema en este caso es que sin importar cuantos hilos de ejecución provea la arquitectura de hardware, solo en esta sección se empleará dos, un primer hilo para calcular la columna refleja con color azul y un segundo hilo para calcular la columna reflejada con color gris. El cálculo de los elementos restantes de la matriz se evidencia en la figura 10.

La figura 11 muestra los elementos, que una diagonal, pueden ser calculados a la vez. Por ejemplo, primeramente se calcularían los elementos representados por la diagonal de color azul claro, luego los elementos de la diagonal gris y así sucesivamente. Solo en las iteraciones cuya cantidad de elementos sea igual o mayor que la cantidad de hilos de ejecución disponibles en la arquitectura podrían aprovechar todos los hilos (suponiendo que se ejecuta el algoritmo en una arquitectura con cuatro hilos de ejecución de procesamientos disponibles). Los elementos de un mismo color son calculados por un mismo hilo.

Fig. 11. Cálculo de los elementos de una diagonal ejecutados por hilos de ejecución diferentes.

Cada conjunto de elementos a calcular, en una iteración, se divide también en dependencia de la cantidad de hilos de ejecución disponibles.

Las consideraciones realizadas respecto al diseño e implementación son aplicadas al paradigma de memoria compartida CUDA [21-23].

RESULTADOS

Las pruebas realizadas pretenden evaluar el algoritmo paralelo implementado. Para ello se emplea la métrica aceleración [24-26], comúnmente empleada para evaluar algoritmos paralelos. Además, se emplean varios escenarios de prueba los cuales se describen a continuación:

- Arquitectura 1: Intel Core 2 Duo E7300 a 2.6 GHz con 2 GB de memoria RAM.
- Arquitectura 2: Intel Core 2 Quad Q9300 a 2.5 Ghz con 4 GB de memoria RAM.
- Arquitectura 3: Intel Core i7 920 a 2.67 GHz con 6 GB de memoria RAM.

Para cada arquitectura descrita se realizaron pruebas con tres juegos de datos distintos, los cuales se detallan a continuación:

- Juego de datos 1: cantidad de trazas 350, tamaño mínimo de traza 1, tamaño máximo 10, promedio 5,5.
- Juego de datos 2: cantidad de trazas 2 350, tamaño mínimo de traza 1, tamaño máximo 26, promedio 13,5.
- Juego de datos 3: cantidad de trazas 4 350, tamaño mínimo de traza 1, tamaño máximo 49, promedio 25.

Los tiempos de ejecución que se exponen es el promedio de 10 ejecuciones del algoritmo para cada arquitectura descrita. Las pruebas se ejecutaron sobre el sistema operativo Windows 7 de 64 bits.

Experimento 1

El experimento se realiza para arquitectura descrita para el juego de datos 1 (350 trazas). Los tiempos que se exponen en la tabla pertenecen a las dos secciones paralelizadas propuestas. Ver figura 12 y tabla 1.

Experimento 2

El experimento se realiza para arquitectura descrita para el juego de datos 2 (2 350 trazas). Los tiempos que se exponen en la tabla pertenecen a las dos secciones paralelizadas propuestas. Ver figura 13 y tabla 2.

Experimento 3

El experimento se realiza para arquitectura descrita para el juego de datos 3 (4 350 trazas). Los tiempos que se exponen en la tabla 3 pertenecen a las dos secciones paralelizadas propuestas. Ver figura 14.

Fig. 12. Aceleración obtenida empleando juego de datos 1.

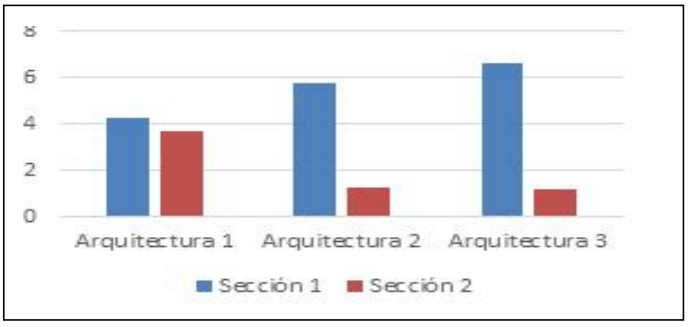

Fig. 13. Aceleración obtenida empleando juego de datos 2.

Tabla 1
Promedio de tiempos de ejecución (segundos) y desviación estándar obtenidos para juego de datos 1

	Tiempo de ejecución(segundos)					
	Arquitectura 1		Arquitectura 2		Arquitectura 3	
	Sección 1 Prom./Desv.	Sección 2 Prom./Desv.	Sección 1 Prom./Desv.	Sección 2 Prom./Desv.	Sección 1 Prom./Desv.	Sección 2 Prom./Desv.
Secuencial	0,080 0/0,010	0,211 3/0,013 1	0,088 4/0,238	0,285 6/0,001	0,051 5/0,139	0,141 0/0,037
Paralelo	0,046 0/0,004 4	0,663 1/0,029 6	0,030 3/0,005 5	2,541 8/0,065	0,015 0/0,056	3,097 4/0,001

Tabla 2
Promedio de tiempos de ejecución (segundos) y desviación estándar obtenidos para juego de datos 2

	Tiempo de ejecución(segundos)					
	Arquitectura 1		Arquitectura 2		Arquitectura 3	
	Sección 1 Prom./Desv.	Sección 2 Prom./Desv.	Sección 1 Prom./Desv.	Sección 2 Prom./Desv.	Sección 1 Prom./Desv.	Sección 2 Prom./Desv.
Secuencial	5,466/0,002 8	157,61/40,234	4,632/0,900 3	203,79/40,234	3,987 1/0,005 1	199,381/1,161
Paralelo	1,293/0,064	43,077/0,272 6	0,805/0,158 5	166,92/0,023 1	0,604 6/0,158 5	166,784/0,171

DISCUSIÓN

La desviación estándar mostrada en la tabla 1, para los tiempos promedio de ejecución, demuestra que los tiempos obtenidos, para todas las secciones, son similares. Se observa además, que para todas las arquitecturas de prueba empleada, el tiempo de ejecución de la sección 1 es inferior a su tiempo secuencial. La sección 2 no disminuye en ninguna de las arquitecturas de prueba. Se concluye que para esta cantidad de trazas la paralelización de la sección 2 no logra disminuir el tiempo de ejecución del algoritmo. Los tiempos logrados con GPU superan el tiempo secuencial. El tiempo de ejecución secuencial es inferior al tiempo de ejecución paralelo. CUDA al emplear las tarjetas de video (GPU) necesita transferir los datos desde la memoria principal hacia la GPU y viceversa. En esta ocasión las matrices de alineación son pequeñas para 350 trazas, por lo tanto, la cantidad de hilos de ejecución que pueden emplearse son pocos. Además, el tiempo adicional que introduce CUDA para el envío y recibo de datos ralentiza el tiempo de ejecución en comparación con el secuencial.

Revista Cubana de Ingeniería. Vol. VII, No. 1, enero - abril, 2016, pp. 27 - 35, ISSN 2223 -1781

Tabla 3
Promedio de tiempos de ejecución (segundos) y desviación estándar obtenidos para juego de datos 3

	Tiempo de ejecución(segundos)					
	Arquitectura 1		Arquitectura 2		Arquitectura 3	
	Sección 1 Prom./Desv.	Sección 2 Prom./Desv.	Sección 1 Prom./Desv.	Sección 2 Prom./Desv.	Sección 1 Prom./Desv.	Sección 2 Prom./Desv.
Secuencial	8,503/1,308 9	2 220,56/46,21	9,069/0,258	2 484,03/0,182	6,859/0,508	3 365,48/1,161
Paralelo	4,844/0,078	207,76/0,639	2,634/0,124	688,64/95,192	2,145/0,064	688,89/0,204

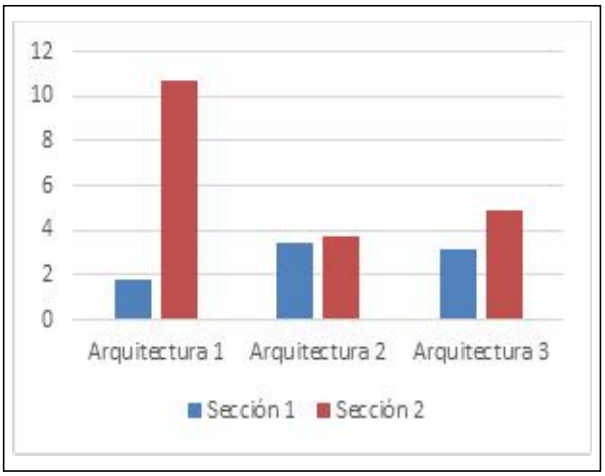

Fig. 14. Aceleración obtenida empleando juego de datos 3.

La figura 13 muestra la aceleración obtenida en cada arquitectura de prueba. La barra azul representa la aceleración alcanzada con la paralelización la sección 1, y las restantes barras rojas son la aceleración obtenida con la implementación de la sección 2. La arquitectura que mejor aceleración logra disminuir el tiempo con la sección 1 es la 3. Las implementaciones realizadas a la sección 2, no logran disminuir el tiempo de ejecución secuencial, por lo tanto el resultado de la fórmula de aceleración obtenida para todo los casos es menor que 1. Lo que indica que el tiempo de ejecución paralelo es peor que el tiempo de ejecución secuencial.

Los resultados expuestos en la tabla 2, respecto a la sección 1 tienen un comportamiento similar respecto a la tabla 1, en cambio para la sección 2, para esta cantidad de trazas, el algoritmo paralelo logra disminuir el tiempo de ejecución en todas las arquitecturas de prueba. La figura 14 muestra la aceleración obtenida para ambas secciones en cada arquitectura de prueba. Se evidencia que la arquitectura donde se logra mejor aceleración para la sección 1 es la arquitectura 3, mientras que para la sección 2, es la arquitectura 1. La sección 1 al encontrarse paralelizada con OpenMP, logra mejores tiempos en la arquitectura 3, pues comparando los CPU de prueba, es el que provee mejores prestaciones; lo mismo ocurre para la sección 2, que la mejor tarjeta de video se encuentra en la arquitectura 1.

La tabla 3 muestra el promedio del tiempo de ejecución obtenido por ambas secciones para el juego de datos 3. Se evidencia que todos los tiempos paralelos obtenidos son inferiores a los tiempos secuenciales. La desviación estándar es mayor para las arquitectura 1 y 2 en la sección 2, lo que demuestra que los tiempos obtenidos, para estos casos, son un poco divergentes. Como se presenta en la figura 15 la aceleración obtenida para la sección 2 es aproximadamente 11 veces mejor que el tiempo secuencial. Respecto a la sección 1, la mejor aceleración se logra con la arquitectura 2, aunque el tiempo reflejado en la tabla 3 indica que el mejor tiempo de logra con la arquitectura 3.

Los experimentos demuestran que cuando la cantidad de trazas es mayor, dígase experimentos 2 y 3, a pesar que siguen existiendo las trasferencias, la cantidad de hilos de procesamiento que pueden ser empleados paralelamente es mayor, pues las matrices de alineación también son más grandes. Por lo tanto, se concluye que, a mayor cantidad de trazas a alinear, la implementación paralela logra disminuir, para todas las arquitecturas de prueba, el tiempo de ejecución del algoritmo secuencial.

CONCLUSIONES

En la presente investigación se aborda el estudio de métodos que permiten reducir el tiempo de ejecución de la implementación del algoritmo “Alineamiento de Trazas”. Los resultados indican que:

- Es posible paralelizar la región *Crear Matriz de Distancia y Alinear Trazas*.
- A pesar que la matriz de distancia es una región cuyo tiempo no es elevado, se propone un diseño que logra disminuir para todos los juegos de datos y arquitecturas de pruebas el tiempo de ejecución.
- La región *Alinear Trazas* es paralelizada, siguiendo los métodos empleados para la alineación de secuencias biológicas.
- La región paralelizada *Alinear Trazas* disminuye el tiempo de ejecución del algoritmo secuencial para una cantidad de trazas superiores a 350.
- Los mejores tiempos paralelos en las pruebas realizadas se logran con la tarjeta de video NVidia GTX 550.
- Para un total de 4 350 trazas, el tiempo de ejecución secuencial de 37 min. (4 350 trazas en un Core 2 Duo), disminuyó a 4 min.
- El algoritmo paralelo es aproximadamente 11 veces más rápido que el algoritmo secuencial.

Por todo lo planteado se concluye que el objetivo de investigación ha sido cumplido satisfactoriamente. La investigación realizada permite obtener un algoritmo que puede ser extrapolado a cualquier escenario que necesite alinear un conjunto de trazas.

Se recomienda implementar el algoritmo siguiendo el diseño propuesto con otros modelos de programación, específicamente OpenCL, y comparar los resultados obtenidos con los expuestos para CUDA, en el trabajo. Además de optimizar la solución propuesta haciendo uso de las diferentes memorias que poseen las tarjetas de video.

REFERENCIAS

1. **VAN DER AALST, W.** *Process mining: discovery, conformance and enhancement of business processes*. Springer Science & Business Media. 2011, ISBN:978-3-642-19344-6.
2. **WEIJTERS, A.; VAN DER AALST, W. M.; DE MEDEIROS, A. A.** "Process mining with the heuristics miner-algorithm". *Technische universiteit eindhoven, tech. Rep. Wp,* 2006. vol. 166**,** pp.1-34. DOI 10.1.1.118.8288.
3. **BOSE, R. J. C.; VAN DER AALST, W. M.** "Process diagnostics using trace alignment: opportunities, issues, and challenges". *Information systems,* 2011, vol. 37**,** pp.117-141. ISSN:0306-4379.
4. **DU, Z.; YIN, Z.; BADER, D.** 2010. "A tile-based parallel viterbi algorithm for biological sequence alignment on gpu with cuda". *Proc. of the ieee international symposium on parallel and distributed processing, workshops and phd forum (IPDPSW).* Atlanta, GA: IEEE. DOI 10.1109/IPDPSW.2010.5470903.
5. **SIRIWARDENA, T.; RANASINGHE, D.** *Accelerating global sequence alignment using cuda compatible multi-core gpu*. Information and automation for sustainability (ICIAFS), 5th International Conference on, 2010 colombo. IEEE, pp.201-206. ISBN:978-1-4244-8549-9.
6. **HASAN, L.; KENTIE, M.; AL-ARS, Z.** *Gpu-accelerated protein sequence alignment*. Engineering in medicine and biology society, embc, 2011 annual international conference of the ieee, 2011. IEEE, pp. 2442-2446. ISBN:978-1-4244-4121-1.
7. **PANDEY, J.; KHARE, N.; PANDEY, R.** "A survey of parallel models for sequence alignment using Smith Waterman algorithm *IOSR". Journal of computer engineering*. 2015, vol. 17, pp. 48-52. ISSN:2278-0661.
8. **DESHMUKH, K. B.; KHARAT, M. U.** "Accelerating smith-waterman alignment based on gpu". *International journal of advanced research in computer science and software engineering.* 2015, vol.5**,** pp.1162-1168. ISSN:2277 128X.
9. **GONZÁLEZ, A. E. C.** "La métrica de levenshtein". *Revista de ciencias básicas ujat* [online]*,* vol.7**,** pp. 35-43, 2008 [cit 2015-11-15]. Available at: <http://www.publicaciones.ujat.mx/publicaciones/revista_dacb/Acervo/v7n2OL/v7n2.pdf#page=37>.
10. **KHALED, H.; EL GOHARY, R.; BADR, N.; FAHEEM, H.** "Accelerating pairwise dna sequence alignment using the cuda compatible gpu". *International journal of computer applications,* 2013 [online]*,* vol.84**,** pp.25-31 [cit 2015-10-20]. Available at: < http://research.ijcaonline.org/volume84/number1/pxc3892619.pdf >.
11. **HALIM, A. K.; HARUN, M. H.; MOHAMED, S.; MOHAMED, S.** Low power study on trace back and reconstruction modules for dna sequences alignment accelerator. Cambridge computer modelling and simulation (UKSIM). 2012, UKSIM 14th international conference on, pp.117-125. ISBN: 978-1-4673-1366-7.
12. **SMITH, T. F.; WATERMAN, M. S.** "Identification of common molecular subsequences". *Journal of molecular biology,* 1981, vol.147**,** pp. 195-197.ISSN: 0022-2836.
13. **BABU, M. R.** "Parallelized hierarchical expected matching probability for multiple sequence alignment". *Journal of theoretical & applied information technology,* 2014, vol.64**,** pp.378-385. ISSN:1992-8645.
14. **DO, C. B.; KATOH, K.** *Protein multiple sequence alignment*. *Functional proteomics.* Springer. 2008. ISBN: 1588299716.
15. **NOTREDAME, C.** "Recent progress in multiple sequence alignment: a survey". *Pharmacogenomics.* 2002, vol.3**,** pp.131-144. ISSN: 1462-2416.
16. **CHENNA, R.; SUGAWARA, H.** *et. al.* **"**Multiple sequence alignment with the clustal series of programs". *Nucleic acids research.* 2003, vol.31**,** pp.3497-3500. ISSN:1992-8645.
17. **HANIF, M. K.; ZIMMERMANN, K. H. "**Graphics card processing: acceleration of multiple sequence alignment". *Austin j comput biol bioinform* [online]. 2014, vol.1**,** 6 [cit 2015-10-05]. Available at: < http://austinpublishinggroup.com/computational-biology-bioinformatics/fulltext/ajcbb-v1-id1008.php >.
18. **ZAFALON, G. F.; MARUCCI, E. A.** *et. al*. Improvements in the score matrix calculation method using parallel score estimating algorithm. *Journal of biophysical chemistry.* 2013, vol. 4. DOI 10.4236/jbpc.2013.42006.
19. **ZOMAYA, A. Y.** *Parallel computing for bioinformatics and computational biology: models, enabling technologies, and case studies*. John Wiley & Sons. 2006. ISBN:978-0-471-71848-2.
20. **SHARMA, C.** Parallel approaches in multiple sequence alignments. *International journal of advanced research in computer science and software engineering* [online]. 2014, vol. 4**,** pp. 130-137 [cit 2015-10-05]. Available at: < www.ijarcsse.com >.
21. **WILT, N.** *Cuda handbook: a comprehensive guide to gpu programming*. Addison-Wesley. 2013. ISBN:978-0-321-80946-9.

22.COOK, S. *Cuda programming: a developer's guide to parallel computing with gpus*. Newnes, 2012. ISBN: 978-0124159334.

23.SANDERS, J.; KANDROT, E. *Cuda by example: an introduction to general-purpose gpu programming*. Addison-wesley, 2010. ISBN:978-0131387683.

24.STALLINGS, W. *Computer organization and architecture: designing for performance*. Prentice Hall,2010. ISBN:978-0-13-607373-4.

25. GEBALI, F. *Algorithms and parallel computing*. Wiley,2011. ISBN:978-0-470-90210-3.

26. PACHECO, P. *An introduction to parallel programming*. Elsevier, 2011. ISBN:0080921442.

AUTORES

Marlis Fulgueira Camilo
Ingeniera Informática, Máster en Informática, Complejo de Investigaciones Tecnológicas Integradas (CITI), La Habana, Cuba

Ernesto Insua Suárez
Ingeniero Informático, Complejo de Investigaciones Tecnológicas Integradas (CITI), La Habana, Cuba

Humberto Díaz Pando
Ingeniero Informático, Doctor en Ciencias Técnicas, Profesor Auxiliar, Facultad de Ingeniería Informática, Instituto Superior Politécnico José Antonio Echeverría, Cujae, La Habana, Cuba

Accelerating Trace Alignment Algorithm Applying Cuda

Abstract
Currently, the business processes that run businesses generate large volumes of traces. These traces are stored in event logs for further analysis. The use of tools to extract useful knowledge from information recorded possible to know exactly happens in a company and the existence of anomalies executed process. The algorithm *Trace Alignment* allows to identify the most likely common behavior or executed process, the occurrence of deviations and the contexts in which one or more activities are carried out. Experiments show that the runtime depends on the number of traces desired alignment. The present article introduces techniques of parallel programming, CUDA, with the aim of reducing the execution time of the algorithm. The main features of the algorithm, as well as other parallel implementations are analyzed in order to unify the techniques that can achieve the best time. Parallel TA proposed algorithm decreases about 11 times, compared to the sequential implementation.

Key words: CUDA, trace alignment, business process