SOLUCIÓN DE UN SISTEMA MECÁNICO FERROVIARIO USANDO COMPUTACIÓN PARALELA

Alejandro Bustos Caballero, Higinio Rubio Alonso, Eduardo Corral Abad, Juan Carlos García Prada.

Universidad Carlos III de Madrid. MAQLAB. Dpto. de Ingeniería Mecánica. Avda. de la Universidad, 30 - 28911 Leganés, Madrid. Tfno: 916248402. albustos@ing.uc3m.es

Recibido: 12/Ene/2017 - Revisado: 13/Ene/2017-Aceptado: 28/Mar/2017- DOI: http://dx.doi.org/10.6036/NT8288

TO CITE THIS ARTICLE:
BUSTOS-CABALLERO, Alejandro, RUBIO-ALONSO, Higinio, CORRAL-ABAD, Eduardo et al. PARALLEL COMPUTING USED TO SOLVE A RAILWAY MECHANICAL SYSTEM. *DYNA New Technologies*, Enero-Diciembre 2017, vol. 4, no. 1, p.[18 p.]. DOI: http://dx.doi.org/10.6036/NT8288

RAILWAY MECHANICAL SYSTEM SOLUTION USING PARALLEL COMPUTING

ABSTRACT:
The Graphics Processor Unit (GPU) attends the computers problems related with phenomena presented in Applied Engineering, Scientific Areas and specifically, in the Railway industry.
So that specifically, we present the resolution process of a mechanical system through its implementation taking advantage of the GPU features. A technique that could be applied into railways mechanical system.
Several analytical models are proposed for mechanical devices, each of them with different implementation techniques; but our advantage is to use the GPU multiple processors over two programming environments executed in a traditional form.
The computational times are picked up to determine the strengths and weaknesses using GPU with respect to conventional computers.

Keywords: GPGPU, GPU, CUDA, parallel computing, simulation, railway mechanism

RESUMEN:
La unidad de procesamiento gráfico (GPU) colabora en la resoluciónde fenómenos presentes en la Ingeniería Aplicada, Áreas Científicas, y en la industria ferroviaria.
Específicamente, se presenta el proceso de resolución de un sistema mecánico mediante su implementación aprovechando las características de las GPUs. Una técnica que puede ser aplicada a sistemas mecánicos ferroviarios.
Se plantean varios modelos analíticos para dispositivos mecánicos, cada uno con diferentes técnicas de implementación; aprovechando los múltiples procesadores de la GPU con respecto a dos entornos de programación ejecutados de forma tradicional.
El análisis de los tiempos de computación permite determinar las fortalezas y debilidades de aplicar la nueva tecnología (utilizando GPU) respecto a los computadores tradicionales.

Palabras clave: GPGPU, GPU, CUDA, computación paralela, simulación, mecanismo ferroviario

1.- INTRODUCCIÓN

Desde su aparición, las tarjetas gráficas y sus unidades de procesamiento gráfico (*Graphics Processing Unit, GPU*), se han destinado al procesado de los gráficos en los videojuegos, simuladores [1] y al renderizado de secuencias de animación. Sin embargo, cuentan con una serie de características que las hacen muy atractivas para el cómputo de problemas con un gran número de operaciones y/o iterativos. Al igual que el procesador de un ordenador cualquiera, denominado unidad central de procesamiento (*Central Processing Unit, CPU*), la GPU es un único chip. La diferencia estriba en que, mientras una CPU actual tiene generalmente 4 u 8 núcleos de proceso, una GPU tiene cientos o miles de núcleos de proceso (ver Fig. 1), aunque ejecutan tareas más básicas. Los pocos núcleos de la CPU están optimizados para el procesamiento secuencial mientras que la GPU tiene una arquitectura paralela formada por miles de núcleos pequeños diseñados para manejar múltiples tareas simultáneamente. Las principales ventajas de cada tipo de chip se

Publicaciones DYNA SL -- c) Mazarredo nº69 - 4º -- 48009-BILBAO (SPAIN)
Tel +34 944 237 566 – www.dyna-newtech.com - email: info@dyna-newtech.com

comparan en la Tabla 1 [2]. Se puede observar que las características de la CPU están orientadas a la eficiencia en tareas de sistema operativo, mientas que la GPU se encamina a operaciones en coma flotante.

En el año 2006, la empresa NVIDIA introduce la arquitectura CUDA (*Compute Unified Device Architecture*) en sus GPUs, a la vez que desarrolla las herramientas necesarias para su programación [3]. Este hecho supone un punto de inflexión, pues permite utilizar las capacidades de las GPUs para la ejecución de toda clase de tareas. Nace el GPU de propósito general o GPGPU (*General-Purpose Computing on Graphics Processing Units*). En los años siguientes, tanto investigadores como empresas de software comercial se han lanzado a incorporar esta tecnología en sus desarrollos [2].

En el ámbito de la investigación, se pueden señalar trabajos como la simulación de la dinámica de complejos sistemas mecánicos [4], nuevos algoritmos matemáticos [5], reconstrucción de imágenes médicas [6], simulación de fluidos [7], etc. En cuanto a software comercial, multitud de programas de diversos ámbitos se han adaptado a esta tecnología. Se podrían destacar ANSYS®, Abaqus®, CST Studio Suite® o BRS Labs AISight for SCADA®, en el campo de la ingeniería, pero también se aplica a finanzas, modelado del clima, análisis de datos, defensa e inteligencia, etc.

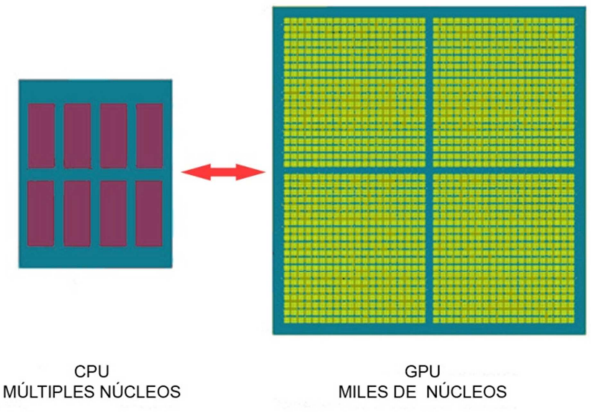


Fig. 1. Comparativa entre CPU y GPU

CPU	GPU
Cachés muy rápidas (ideal para reutilizar datos)	Multitud de unidades matemáticas
Buen acceso a datos	Rápido acceso a su memoria integrada
Multitud de procesos/hilos diferentes	Ejecuta un programa en cada fragmento/vértice
Alto rendimiento en un único hilo de ejecución	Alto rendimiento en tareas paralelas
Ideales para el paralelismo de tareas	Ideales para el paralelismo de datos
Optimizado para alto rendimiento en códigos secuenciales	Optimizado para multitud de operaciones matemáticas de naturaleza paralela (operaciones en coma flotante)

Tabla 1. Comparativa entre CPU y GPU [2]

Desde el punto de vista del usuario final, éste debe considerar que utilizar esta tecnología supone un gran ahorro de tiempo (hasta 10 veces más rápido, dependiendo de la aplicación [8]) y de coste (en la actualidad, las GPU más avanzadas rondan los 4000€, pero una con buenas cualidades se puede encontrar por 500€, y requieren menos consumo energético). Adicionalmente, se debe considerar la relación entre los tiempos de operación del proceso de cálculo y los costes económicos de un producto [9].

En este trabajo se persigue explorar las capacidades de las GPUs mediante su aplicación al cálculo de un mecanismo de utilidad en el sector ferroviario para controlar el movimiento de lazo. Para conseguir este objetivo se proponen varios modelos de análisis del mecanismo que permiten evaluar las capacidades de computación de las GPUs. Adicionalmente, se comparan los resultados obtenidos del cálculo de esos modelos sobre dos lenguajes de programación.

Publicaciones DYNA SL -- c) Mazarredo nº69 - 4º -- 48009-BILBAO (SPAIN)
Tel +34 944 237 566 – www.dyna-newtech.com - email: info@dyna-newtech.com

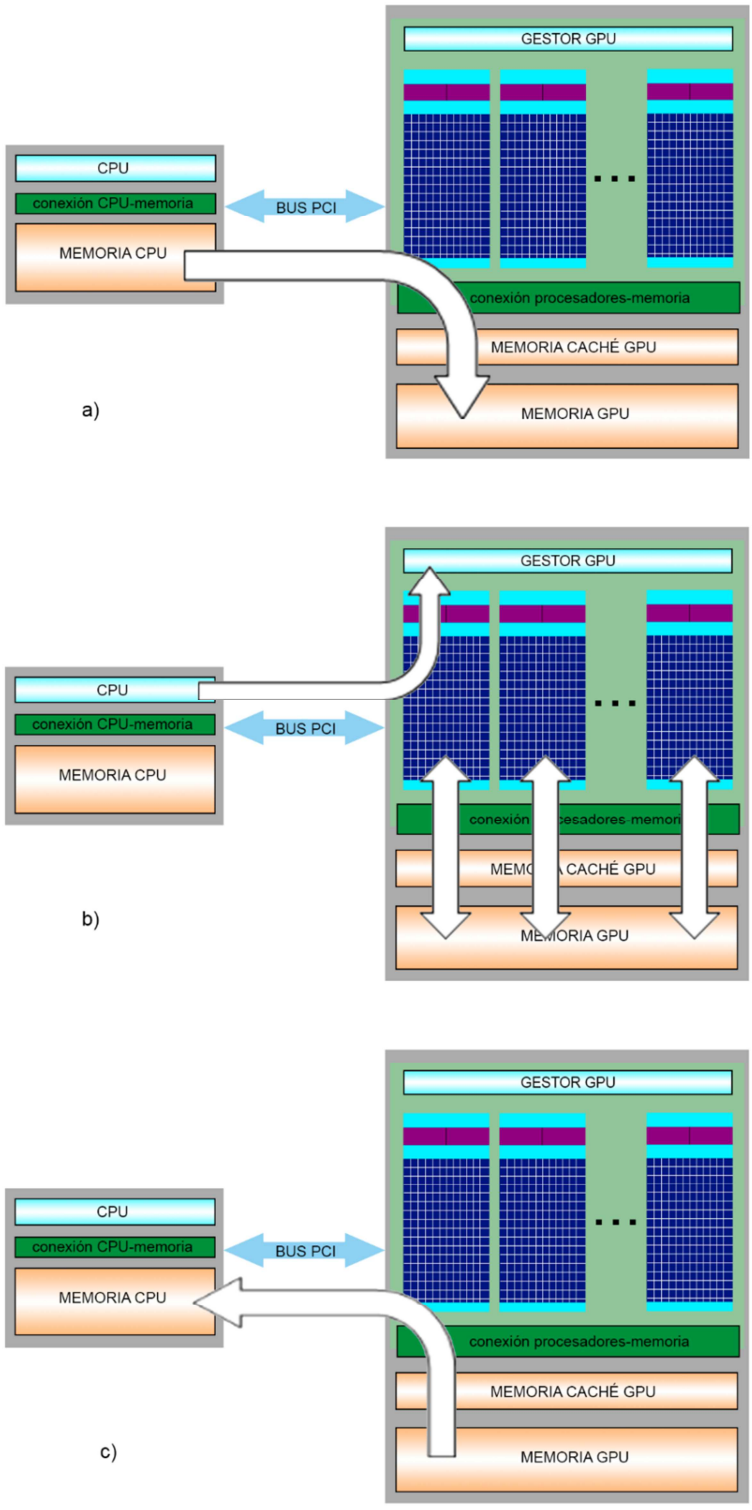


Fig. 2. Proceso de ejecución de un programa en la GPU

Publicaciones DYNA SL -- c) Mazarredo nº69 - 4º -- 48009-BILBAO (SPAIN)
Tel +34 944 237 566 – www.dyna-newtech.com - email: info@dyna-newtech.com

2.- COMPUTACIÓN PARALELA MEDIANTE GPU

A nivel de hardware, en un PC típico, la CPU y la GPU (y sus memorias) son elementos físicamente distintos, conectados entre sí mediante un canal, denominado bus PCI Express, que transmite los datos de un componente a otro. Por tanto, para poder ejecutar rutinas en las tarjetas gráficas y aprovechar sus capacidades de computación, el primer paso es introducir los datos de entrada a través de la CPU y copiarlos a la memoria de la GPU (Fig. 2a). A continuación, se envían las órdenes de la rutina al gestor de la GPU, quien se encarga de ejecutarlas distribuyendo el trabajo entre los múltiples núcleos de cálculo de que consta la tarjeta gráfica (Fig. 2b). Como último paso, se deben copiar los datos de salida de la memoria de la GPU a la memoria de la CPU para poder acceder a ellos (Fig. 2c). Éste el principal inconveniente del GPGPU, pues la velocidad de bus PCI Express es mucho más baja que la de los buses existentes entre la CPU y la GPU con sus respectivas memorias, lo cual crea un inevitable cuello de botella.

El proceso a nivel de software es similar, pues un algoritmo típico de GPGPU ejecuta parte de su código en la CPU y parte en la GPU. En la nomenclatura de CUDA, cada función (código) que se ejecuta en la GPU se denomina kernel. Éste utiliza varios hilos paralelos que realizan la misma instrucción con datos diferentes. Sin embargo, un kernel no llama a los hilos directamente, sino a una malla. Esta malla se compone de varios bloques, que a su vez se componen de múltiples hilos. El número de cada uno de estos elementos depende de la GPU con la que se esté trabajando y se deben tener en cuenta al escribir el código.

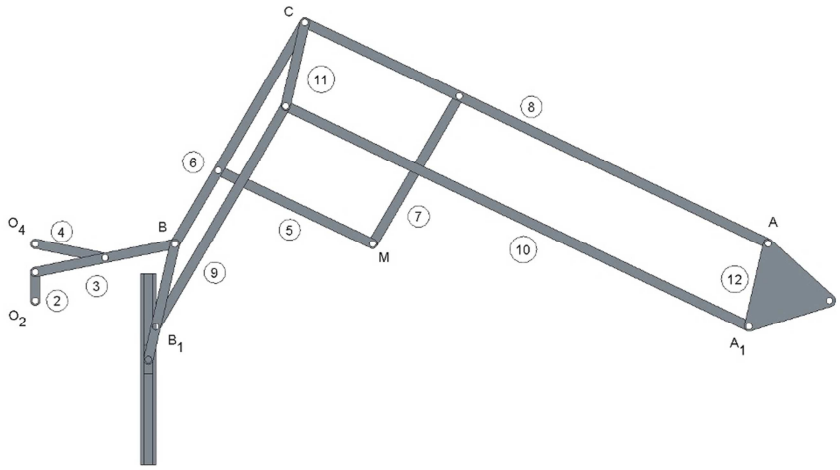
Otro aspecto importante en la programación es la gestión de los datos en la memoria. En las primeras versiones de CUDA se debía reservar espacio en los dos chips y copiar los datos "manualmente". Afortunadamente, las últimas versiones de CUDA han introducido el concepto de *Unified Memory* (UM), o memoria unificada. Aplicando este concepto, basta con definir adecuadamente las variables necesarias una única vez, pues *Unified Memory* se encarga de asignar y transferir datos entre las dos memorias [10].

Todos los cálculos realizados en el presente trabajo se han ejecutado en un PC con un procesador Intel Xeon E5410 a 2.33 GHz con 6 GB de RAM y una GPU NVIDIA GeForce GTX 660 Ti. El sistema operativo es Windows 7. Los programas utilizados han sido MATLAB® R2013b y Visual Studio 2010 Professional con la CUDA Toolkit 6.

3.- MODELOS E IMPLEMENTACIÓN

El control del movimiento de lazo de un vehículo ferroviario es fundamental para garantizar su estabilidad, así como el confort de los pasajeros; para lo cual se disponen complejos sistemas mecánicos en el bogie del vehículo ferroviario. Como se recordará, el bogie es la estructura en la que se alojan dos o más ejes, con sus respectivas ruedas, y sobre la que se apoya el vehículo ferroviario. En este trabajo se presenta el sistema mecánico de la Fig. 3 como un posible dispositivo para el control del movimiento de lazo. Éste se compone de tres mecanismos planos fundamentales: mecanismo de generación (basado en un mecanismo de Tchebyshev), compuesto por los eslabones 1 -soporte-, 2, 3 y 4; un pantógrafo (eslabones 5, 6, 7 y 8); y un mecanismo de refuerzo (resto de eslabones); hábilmente dispuestos para lograr la trayectoria adecuada.

Publicaciones DYNA SL -- c) Mazarredo nº69 - 4º -- 48009-BILBAO (SPAIN)
Tel +34 944 237 566 – www.dyna-newtech.com - email: info@dyna-newtech.com



Eslabón	Longitud (mm)
1	60
2	30
3	150
4	75
5	180
6	270
7	180
8	540
9	270
10	540
11	90
12	90

Fig. 3. Numeración y dimensiones de los eslabones y puntos característicos del mecanismo

Para asegurarse de que el mecanismo propuesto trabaja debidamente, es necesario realizar análisis reiterativos sobre el mismo hasta obtener el funcionamiento deseado. Aprovechando las capacidades de computación de las GPUs, se aplicará esta tecnología al cálculo de la cinemática y de la pseudo-optimización del mecanismo, implementándose los algoritmos de cálculo en C++® y en MATLAB®. El análisis de los tiempos de computación necesarios para realizar los cálculos en ambas plataformas nos permitirá detectar las fortalezas y debilidades de la computación paralela.

El algoritmo completo, que incluye los dos procesos, se muestra en la Fig. 4. Éste comienza con la introducción de los datos necesarios para calcular la cinemática y la pseudo-optimización. A continuación, entra en un bucle en el que realiza los cálculos de los procesos y, por último, almacena los resultados obtenidos. En todos los diagramas de flujo se han sombreado en azul los procesos susceptibles de paralelización. En este trabajo, se considera que los procesos susceptibles de paralelización son aquellas rutinas que ejecutan el mismo código multitud de veces con datos distintos. Como se verá más adelante, el cálculo de la cinemática y de la pseudo-optimización se ajustan a este requisito pues su obtención requiere ejecutar varias veces el mismo proceso en situaciones distintas.

Publicaciones DYNA SL -- c) Mazarredo nº69 - 4º -- 48009-BILBAO (SPAIN)
Tel +34 944 237 566 – www.dyna-newtech.com - email: info@dyna-newtech.com

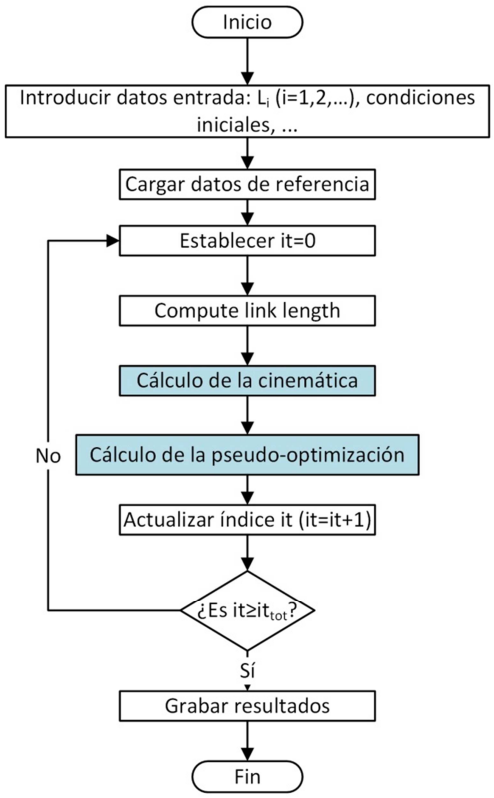


Fig. 4. Diagrama de flujo del algoritmo completo

3.1.- MODELO CINEMÁTICO

Para la obtención de las ecuaciones que definen la cinemática del mecanismo se siguen los ángulos y vectores especificados en la Fig. 5. El primer paso para obtener la cinemática del mecanismo es plantear las ecuaciones de cierre correspondientes a las cadenas cinemáticas. Después de resolver estas ecuaciones, se dispondrá de una serie de ecuaciones explícitas en función del ángulo de entrada (θ_2) [11]. Tomando esto en cuenta, la posición angular de cualquier eslabón se puede escribir:

$$\theta_i = \theta_i(\theta_2), \quad i = 1, 2, \ldots, 1', 2', \ldots \tag{1}$$

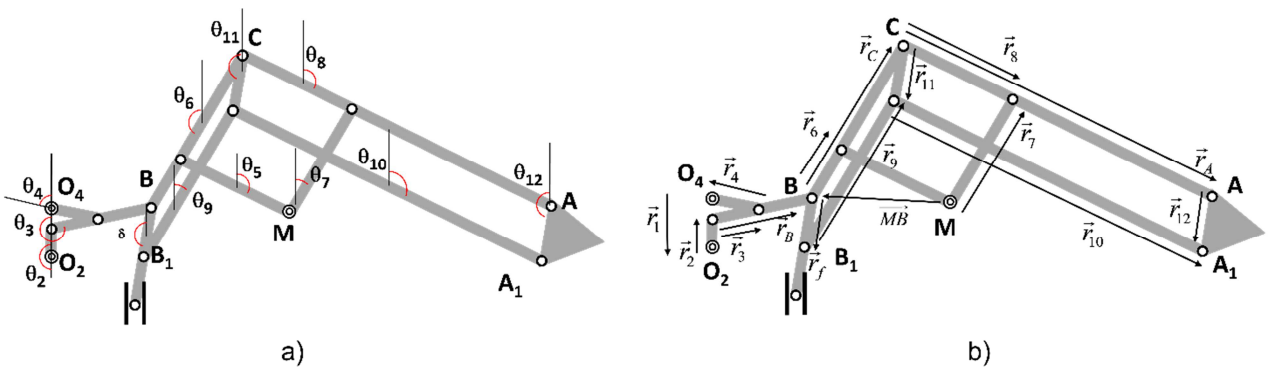


Fig. 5. Ángulos (a) y vectores (b) utilizados para determinar la cinemática del mecanismo

Publicaciones DYNA SL -- c) Mazarredo nº69 - 4º -- 48009-BILBAO (SPAIN)
Tel +34 944 237 566 – www.dyna-newtech.com - email: info@dyna-newtech.com

A partir de aquí, las coordenadas X e Y de los centros de masas (situados en el punto medio de cada barra) se pueden expresar fácilmente con respecto al ángulo de entrada:

$$X_i^{CDM} = X_i^{CDM}(\theta_2);\ Y_i^{CDM} = Y_i^{CDM}(\theta_2); i = 1, 2, \ldots \tag{2}$$

Además, si se conoce la función dependiente del tiempo para el ángulo de entrada, las ecuaciones anteriores se pueden escribir en función del tiempo y obtener las velocidades y aceleraciones mediante la primera y segunda derivadas.

Utilizando el método de Raven, es posible plantear cuatro sistemas de ecuaciones (Ec. 3, 4, 5 y 6) para las cuatro cadenas cinemáticas descritas anteriormente, de forma que se tenga totalmente definido el problema cinemático.

$$r_1 e^{j\theta_1} + r_2 e^{j\theta_2} + r_3 e^{j\theta_3} + r_4 e^{j\theta_4} = 0 \tag{3}$$

$$\overrightarrow{MB} + r_6 e^{j\theta_6} + r_8 e^{j\theta_8} = 0 \tag{4}$$

$$r_C e^{j\theta_7} + r_{11} e^{j\theta_{11}} = r_f e^{j\delta} + r_9 e^{j\theta_9} \tag{5}$$

$$r_A e^{j\theta_8} + r_{12} e^{j\theta_{12}} = r_{11} e^{j\theta_{11}} + r_{10} e^{j\theta_{10}} \tag{6}$$

Donde θ_j es el ángulo y r_j es la longitud de un eslabón j ; MB es la distancia entre los puntos M y B ; y δ es el ángulo definido entre la deslizadera y el eslabón del mecanismo de refuerzo, en sentido antihoriario.

Resolviendo el sistema de ecuaciones correspondiente al mecanismo de Tchebychev, es posible expresar los ángulos θ_4 y θ_3, como funciones de θ_2, siendo a_{aux}, b_{aux} y c_{aux} variables auxiliares dependientes de θ_2 que se han introducido para reducir el tamaño de la ecuación.

$$\theta_4 = \cos^{-1}\left(\frac{2a_{aux}c_{aux} + 2b_{aux}\sqrt{b_{aux}^2 - c_{aux}^2 + a_{aux}^2}}{2\left(a_{aux}^2 + b_{aux}^2\right)}\right) \tag{7}$$

$$\theta_3 = 2\pi - \cos^{-1}\left(\frac{-a_{aux} - r_4\cos\theta_4}{r_3}\right) \tag{8}$$

El mismo proceso aplicado al resto de sistemas de ecuaciones nos permite obtener los ángulos θ_8 , θ_6 , θ_7 (obtenido mediante una sencilla relación geométrica con θ_6) , θ_9 , θ_{11} , θ_{10} y θ_{12} como funciones de θ_2. Como en el caso anterior, se han introducido los parámetros dependientes de θ_2, f_{aux}, a_g, b_g, a_p, b_p, y c_{aux} con el fin de reducir el tamaño de las ecuaciones.

$$\theta_8 = -\cos^{-1}\frac{-f_{aux}x_{MB} - y_{MB}\sqrt{4r_8^2\left(x_{MB}^2 + y_{MB}^2\right) - f_{aux}^2}}{2r_8\left(x_{MB}^2 + y_{MB}^2\right)} \tag{9}$$

$$\theta_6 = \cos^{-1}\left(\frac{x_{MB} + r_8\cos\theta_8}{r_6}\right) \tag{10}$$

Publicaciones DYNA SL -- c) Mazarredo nº69 - 4º -- 48009-BILBAO (SPAIN)
Tel +34 944 237 566 – www.dyna-newtech.com - email: info@dyna-newtech.com

$$\theta_7 = \theta_6 - \pi = \cos^{-1}\left(\frac{x_{MB} + r_8\cos\theta_8}{r_6} - \pi\right) \tag{11}$$

$$\theta_9 = -\cos^{-1}\left(\frac{a_g x_{aux} \pm b_g\sqrt{4r_9^2\left(a_g^2 + b_g^2\right) - c_{aux}^2}}{2r_9\left(a_g^2 + b_g^2\right)}\right) \tag{12}$$

$$\theta_{11} = \cos^{-1}\left(\frac{r_f\cos\delta - r_C\cos\theta_7 + r_9\cos\theta_9}{r_{11}}\right) \tag{13}$$

$$\theta_{10} = -\cos^{-1}\left(\frac{a_p c_{aux} \pm b_p\sqrt{4r_{10}^2\left(a_p^2 + b_p^2\right) - c_{aux}^2}}{2r_{10}\left(a_p^2 + b_p^2\right)}\right) \tag{14}$$

$$\theta_{12} = \cos^{-1}\left(\frac{r_{11}\cos\theta_{11} - r_A\cos\theta_8 + r_{10}\cos\theta_{10}}{r_{12}}\right) \tag{15}$$

El siguiente paso es obtener la posición de los puntos de interés del mecanismo, tarea trivial una vez se conocen las posiciones angulares de los eslabones y sus dimensiones. Por último, tomando la primera derivada de esas ecuaciones se obtendrían las velocidades angulares (y, seguidamente, las velocidades lineales de los puntos); y tomando la segunda derivada, las aceleraciones. Por tanto, la cinemática del mecanismo queda totalmente definida.

3.2. IMPLEMENTACIÓN DEL MODELO CINEMÁTICO

Para realizar el cálculo del modelo cinemático se han desarrollado dos algoritmos, uno secuencial y otro paralelo, que se han implementado sobre C++® y sobre MATLAB®, y cuya complejidad es de orden *O(n)*. En la Fig. 6a se muestra el diagrama de flujo del algoritmo secuencial. La rutina comienza con la introducción de los datos de entrada (dimensiones de los eslabones, condiciones iniciales, condiciones de iteración y tiempo total de la simulación). A continuación, se calcula la cinemática del mecanismo para cada paso de tiempo. En la implementación en C++® esto se hace mediante un bucle. Sin embargo, en MATLAB® se puede prescindir de este bucle si se aprovechan las capacidades de este lenguaje de programación y se utiliza la vectorización. En último lugar, se almacenan los resultados, pudiéndose graficar los mismos para un análisis visual.

Publicaciones DYNA SL -- c) Mazarredo nº69 - 4º -- 48009-BILBAO (SPAIN)
Tel +34 944 237 566 – www.dyna-newtech.com - email: info@dyna-newtech.com

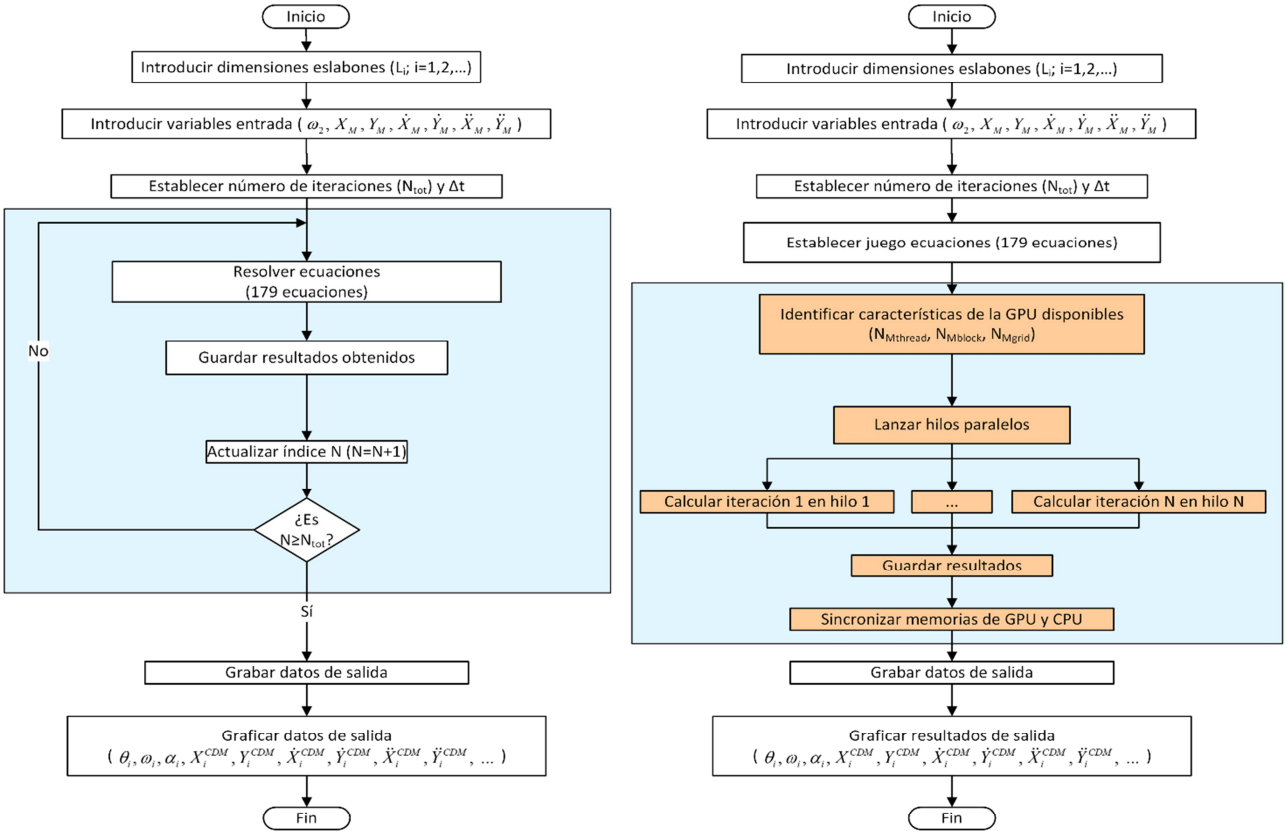


Fig. 6. Diagramas de flujo de los algoritmos secuencial (a) y paralelo (b) para el cálculo de la cinemática del mecanismo propuesto

En lo que respecta al algoritmo paralelo, éste se implementa también sobre MATLAB® y C++® (con la ayuda de la toolkit CUDA). En MATLAB®, los dos algoritmos son bastante similares y únicamente se deben añadir unas pocas líneas de código requeridas para llevar a cabo la computación paralela.

Sin embargo, la implementación del algoritmo en CUDA C++® requiere más operaciones, pero también permite tener control total sobre la gestión de la memoria y de los hilos de ejecución. El bucle del algoritmo secuencial se sustituye por "n" hilos de ejecución, tantos como pasos de tiempo a calcular, que se lanzarán y ejecutarán en paralelo, en la GPU.

El diagrama de flujo del algoritmo paralelo se ilustra en la Fig. 6b. Los primeros pasos son idénticos a los del algoritmo secuencial (Fig. 6a): introducción de datos de entrada, condiciones iniciales, etc. En el siguiente paso comienzan las diferencias. En primer lugar, se identifican las características de la GPU que determinan el número máximo de hilos paralelos que se pueden lanzar y cómo debe hacerse. Concretamente, se identifican el número máximo de hilos por bloque (NMthread), el número máximo de bloques (NMblock) y el tamaño máximo de la malla (NMgrid). Básicamente, estos números determinan cuántos hilos por bloque y cuántos bloques pueden ejecutarse en la GPU. A continuación, se asigna la memoria necesaria para las variables a utilizar y se ejecutan los hilos paralelos para calcular la cinemática. Con el fin de no exceder las capacidades de la GPU, los hilos se reparten en tantos bloques como sean necesarios para no sobrepasar el número máximo de hilos por bloque. Una vez realizados los cálculos, se sincronizan las memorias de la GPU y la CPU, se almacenan y se grafican los resultados.

Publicaciones DYNA SL -- c) Mazarredo nº69 - 4º -- 48009-BILBAO (SPAIN)
Tel +34 944 237 566 – www.dyna-newtech.com - email: info@dyna-newtech.com

3.3.- MODELO DE PSEUDO-OPTIMIZACIÓN

En el modelo de pseudo-optimización propuesto se pretende aproximar la trayectoria del mecanismo a una trayectoria de referencia. Para ello, es necesario definir un parámetro (o juego de parámetros) que evalúe el objetivo que se quiere conseguir. En este trabajo se presenta un índice denominado Amplitud del Factor de Escala (Scale Factor Amplitud, SFA), el cual parte del cálculo de la distancia de Hausdorff [12]. La distancia de Hausdorff mide la separación a la que se encuentran dos subespacios de un espacio métrico y cuyo resultado es la distancia más alejada entre los puntos recíprocos de ambos subespacios o, en otras palabras y aplicado a nuestro caso, el supremo valor de la distancia entre las dos trayectorias.

El procedimiento para calcular el SFA primeramente centra las trayectorias de referencia y calculada en el origen de coordenadas y, a continuación, determina la distancia mínima entre los puntos, utilizando el algoritmo de Hausdorff. Para cada par de puntos más cercanos obtenidos, se calculan sus distancias al origen y la relación entre esas distancias, obteniendo algo similar a un factor de escala para cada par de puntos. Después se obtiene la amplitud de este factor, es decir, la diferencia entre el máximo y el mínimo totales. Si las dos trayectorias son similares, el factor de escala debería ser constante, por lo que la mejor solución es aquella cuya amplitud es mínima.

Este modelo de pseudo-optimización se aplica a los eslabones 3, 6 y 8 (ver Fig. 3) debido a la influencia que éstos tienen en la trayectoria final. Para ello, se modifican sus longitudes de forma individual, dividiendo el proceso en 400 pasos por eslabón. En total se obtienen 1200 nuevos casos de cómputo de la trayectoria.

3.4. IMPLEMENTACIÓN DEL MODELO DE PSEUDO-OPTIMIZADO

El algoritmo que calcula el mejor índice se ha desarrollado en dos versiones, una secuencial (implementada en MATLAB®) y otra paralela (implementada en CUDA C++), ambas con complejidad de orden $O(n^2)$. En la Fig. 7a se muestra el diagrama de flujo del algoritmo secuencial. Los primeros pasos involucran la carga y adecuación de los datos de la trayectoria de referencia. A continuación, se establece el número de iteraciones (igual al número de modificaciones de las longitudes de los eslabones) y se entra en un bucle. Dentro del bucle, se cargan y adecúan los resultados de las simulaciones realizadas anteriormente. Seguidamente, se calculan las distancias entre los puntos de las dos trayectorias (de referencia y calculada), el vector de distancias mínimas, las distancias de los puntos al origen de coordenadas y el índice SFA. Por último, tras salir del bucle, se almacenan y grafican los datos obtenidos.

El algoritmo paralelo (ver Fig. 7b) sigue la misma estructura que el secuencial, pero algunos procesos se han paralelizado. Los primeros pasos son idénticos en ambos, incluido el bucle de iteraciones, la única diferencia hasta este punto es que los datos de entrada deben copiarse a la memoria de la GPU. A continuación, el algoritmo ejecuta, en paralelo en la GPU, la rutina para calcular las distancias entre puntos de las dos trayectorias. Los datos se envían a la CPU, donde se computa el vector de distancias mínimas y se retornan a la GPU, donde se calculan las distancias de cada par de puntos al origen de coordenadas. Seguidamente, los datos pasan a la CPU y se calcula el índice SFA para la iteración actual. Una vez finalizado el bucle, se guardan los resultados obtenidos. Todo el proceso se ilustra en la Fig. 7b.

Publicaciones DYNA SL -- c) Mazarredo nº69 - 4º -- 48009-BILBAO (SPAIN)
Tel +34 944 237 566 – www.dyna-newtech.com - email: info@dyna-newtech.com

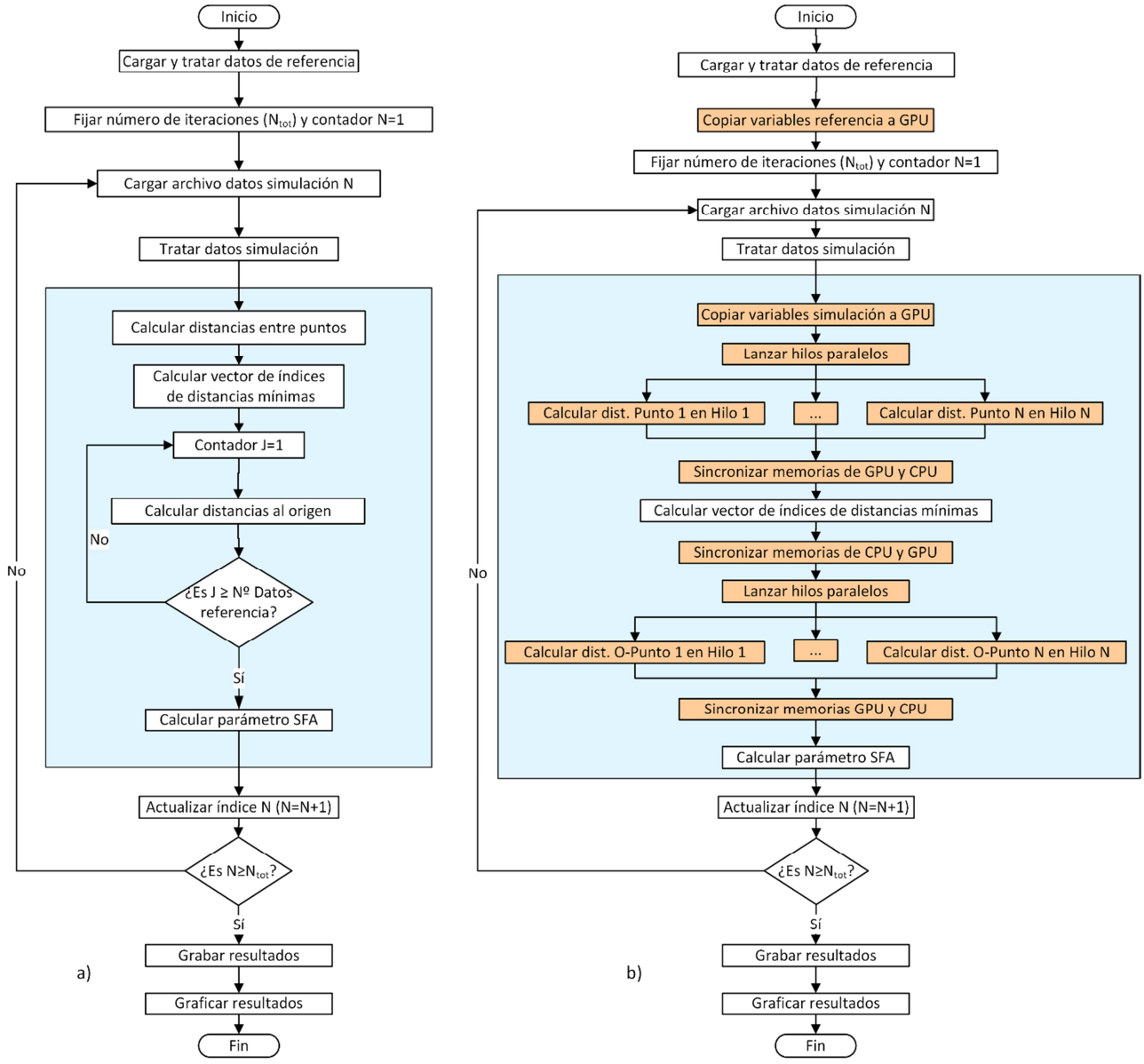


Fig. 7. Diagramas de flujo de los algoritmos secuencial (a) y paralelo (b) para el cálculo de la pseudo-optimización del mecanismo propuesto

4.- RESULTADOS

4.1.- MODELO CINEMÁTICO

Se han realizado experimentos con diez niveles de coste de computación, para cada algoritmo y plataforma. Para cada nivel de coste de computacional se han realizado 20 experimentos, con el fin de tener suficientes datos para el tratamiento estadístico de los datos. Hay que señalar que, debido al elevado tiempo de ejecución de los 3 últimos niveles de cómputo bajo MATLAB® con CUDA, y que la tendencia ya era clara, estos resultados no se han incluido.

Publicaciones DYNA SL -- c) Mazarredo nº69 - 4º -- 48009-BILBAO (SPAIN)
Tel +34 944 237 566 – www.dyna-newtech.com - email: info@dyna-newtech.com

Los resultados obtenidos en todos los experimentos se compendian en la Tabla 2. La primera columna corresponde al nivel de cómputo y muestra el número de pasos de iteración computados en cada simulación. Las siguientes columnas muestran el tiempo medio de ejecución y el factor de aceleración (Factor-X), el cual resulta de dividir el tiempo medio de ejecución de cada nivel de computación y plataforma entre el tiempo medio obtenido en las simulaciones bajo CUDA C++. Los datos mostrados en la Tabla 2 únicamente contabilizan el tiempo empleado en la resolución de las ecuaciones, no tienen en cuenta el tiempo requerido para asignar memoria u otras variables. Además, los datos de la Tabla 2 se representan en gráficas con el objetivo de facilitar su discusión e interpretación.

Número de iteraciones	CUDA C++		C/C++®		MATLAB® CUDA		MATLAB®	
	Tiempo (ms)	Factor-X	Tiempo (ms)	Factor-X	Tiempo (s)	Factor-X	Tiempo (ms)	Factor-X
201	0,58 ± 0,002	1,00	2,25 ± 0,44	3,87	3,52 ± 0,11	6048,81	4,56 ± 1,89	7,84
601	0,92 ± 0,002	1,00	6,85 ± 0,37	7,46	9,87 ± 0,16	10754,34	10,73 ± 0,09	11,69
1101	1,37 ± 0,002	1,00	12,25 ± 0,44	8,91	17,97 ± 0,18	13077,19	18,81 ± 0,14	13,69
2501	1,38 ± 0,02	1,00	22,35 ± 0,49	16,17	43,71 ± 1,61	31621,25	34,44 ± 5,27	24,91
5001	1,39 ± 0,002	1,00	56,00 ± 0,46	40,16	101,80 ± 2,73	73013,90	52,82 ± 4,18	37,88
10001	2,69 ± 0,003	1,00	111,90 ± 0,55	41,55	297,06 ± 14,55	110292,64	86,12 ± 5,11	31,97
20001	4,03 ± 0,006	1,00	232,95 ± 6,39	57,74	764,86 ± 54,91	189591,03	171,45 ± 9,46	42,50
100001	17,50 ± 0,97	1,00	1233,35 ± 5,67	70,49	--	--	908,15 ± 96,65	51,90
500001	81,14 ± 0,23	1,00	7507,65 ± 60,91	92,52	--	--	4974,47 ± 248,12	61,31
1000000	165,67 ± 0,20	1,00	14687,80 ± 210,34	88,66	--	--	9527,56 ± 382,29	57,51

Tabla 2: Resultados para los experimentos del cálculo de la cinemática del mecanismo propuesto

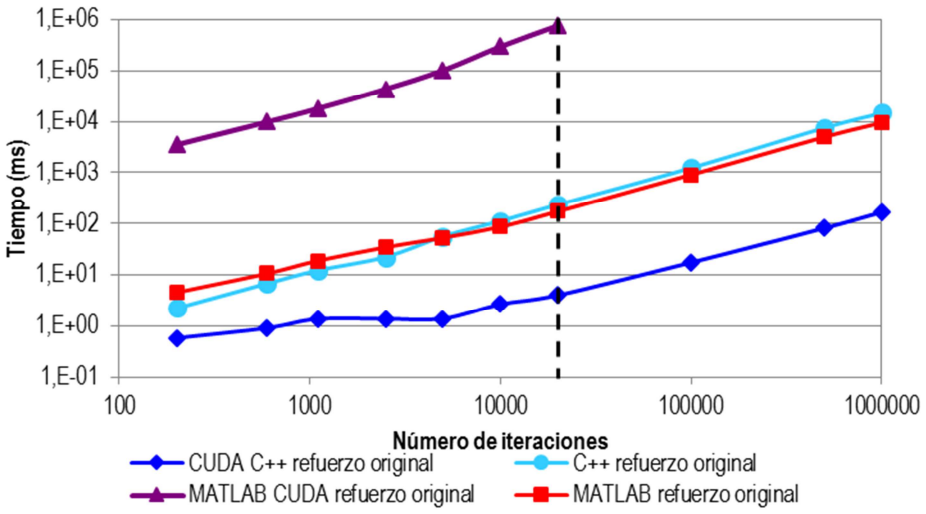


Fig. 8. Evolución de los tiempos de ejecución obtenidos en función del número de iteraciones

En la Fig. 8 se representan los tiempos medios de ejecución, con ambos ejes en escala logarítmica. Se puede ver que el mejor rendimiento se obtiene usando CUDA C++. Los algoritmos sobre C++® y MATLAB® muestran un desempeño similar, aunque ambos son más lentos que el código de CUDA C++. El peor resultado se obtiene combinando MATLAB® y la GPU, denominado MATLAB® CUDA. En teoría, y según se contempla en la documentación de MATLAB® [13], esta combinación debería ser más rápida que MATLAB® pues usa las ventajas del GPGPU para ejecutar las rutinas. Sin embargo, los resultados de los experimentos llevados a cabo en este trabajo muestran un

Publicaciones DYNA SL -- c) Mazarredo nº69 - 4º -- 48009-BILBAO (SPAIN)
Tel +34 944 237 566 – www.dyna-newtech.com - email: info@dyna-newtech.com

comportamiento contrario a esto, a pesar de seguir las recomendaciones del fabricante y aplicar la vectorización en lugar de utilizar bucles. Durante estos experimentos, también se ha comprobado que la precisión doble en coma flotante y, particularmente, la existencia de sentencias de control como "if-end" perjudican la ganancia de tiempo en MATLAB® CUDA.

Así mismo, resulta interesante graficar los factores de aceleración del cómputo utilizando CUDA C++ respecto a C++® y MATLAB®, tal y como se muestra en la Fig. 9. En esencia, se representa cuántas veces es más rápido el cálculo bajo CUDA C++ respecto a C++® (en azul) y respecto a MATLAB® (en rojo). Por cuestiones de claridad en la gráfica, no se han incluido los resultados obtenidos de las simulaciones con MATLAB CUDA (resultados desfavorables, como se comentó anteriormente). Se aprecia que CUDA C++ ejecuta las rutinas hasta 90 veces más rápido que C++® y hasta 60 veces más rápido que MATLAB®. También se observa que, a partir de 5000 iteraciones, MATLAB® es más rápido que C++®.

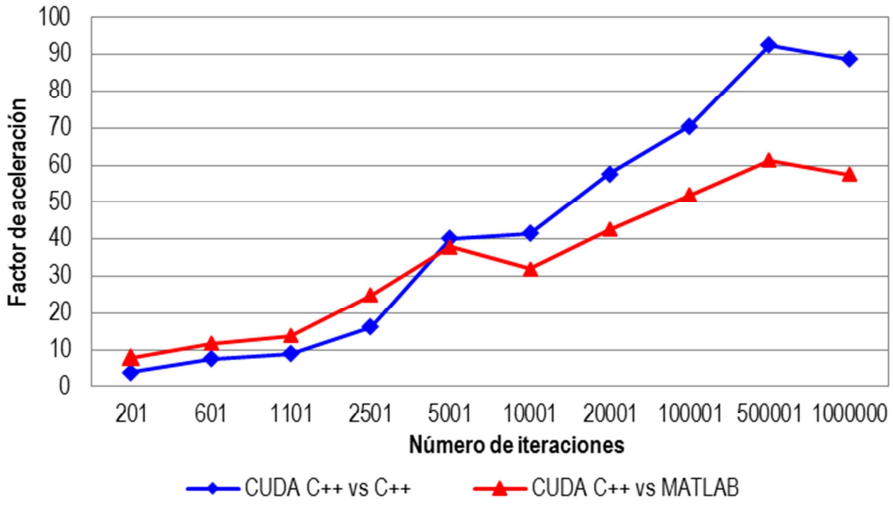


Fig. 9. Evolución de las aceleraciones relativas de CUDA C++ respecto a C++® y respecto a MATLAB®

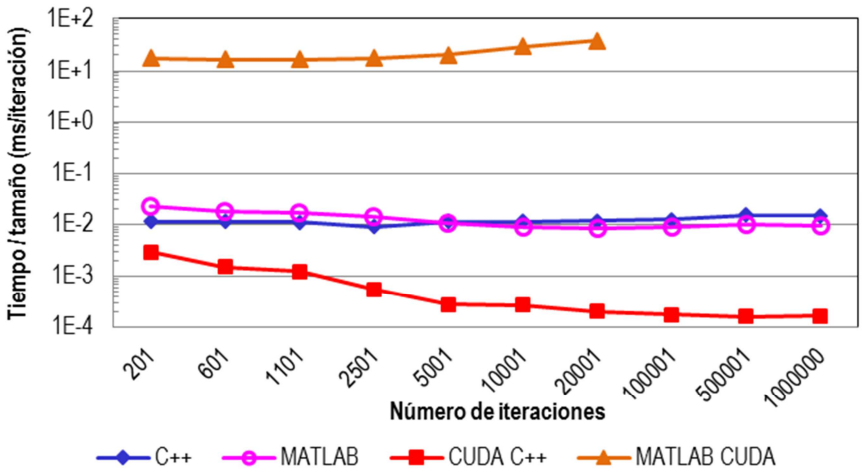


Fig. 10. Evolución de los tiempos de ejecución por iteración

Publicaciones DYNA SL -- c) Mazarredo nº69 - 4º -- 48009-BILBAO (SPAIN)
Tel +34 944 237 566 – www.dyna-newtech.com - email: info@dyna-newtech.com

Además, si se analiza el tiempo empleado en cada iteración (ver Fig. 10), se puede ver que el tiempo por iteración que emplean los algoritmos implementados en CUDA C++ se reduce a medida que aumenta el número de iteraciones, hasta llegar a las 100000. A partir de aquí, se estabiliza en torno a $1{,}6 \cdot 10^{-4}$ ms por iteración, valor resultante de dividir el tiempo total de cómputo entre el número de iteraciones realizadas. El tiempo de cómputo por iteración para la implementación en C++® es prácticamente constante, con un valor de 0,01 ms/iteración. Sin embargo, la versión de MATLAB® de la rutina sigue una ligera pendiente descendente que comienza en 0,02 ms/iteración y acaba en 0,01 ms/iteración. En lo referente al código en MATLAB® CUDA, el tiempo por iteración se reduce al principio y luego aumenta con el número de iteraciones.

4.2.- MODELO PSEUDO-OPTIMIZADO

Los resultados obtenidos del modelo pseudo-optimizado se resumen en la Tabla 3. En ella se pueden ver los valores del parámetro SFA obtenidos para la menor y mayor longitud de los eslabones analizados, así como las longitudes correspondientes al mínimo y mayor valor del SFA, para las tres barras seleccionadas. Las longitudes ideales, según este método, corresponden a los valores mínimos del SFA. El resto de valores de la tabla nos dan una idea de la sensibilidad del factor SFA. En la Fig. 11 se muestra la evolución del parámetro SFA, en función de la longitud del eslabón para cada uno de los tres eslabones analizados. Analizando los datos obtenidos, se observa que las longitudes óptimas obtenidas para los eslabones 3 y 8 son las originales. Para el eslabón 6 se obtiene que la longitud óptima aplicando este método es ligeramente inferior (266,5 mm) a la longitud original (270 mm).

Eslabón	Longtitud mínima		Longitud máxima		SFA mínimo		SFA máximo	
	L (mm)	SFA (L_{min})	L (mm)	SFA (L_{max})	L (mm)	SFA_{min}	L (mm)	SFA_{max}
Eslabón 8	432	0.4685	647.5	0.5695	**540**	**0.3412**	647.5	0.5695
Eslabón 3	120	0.5867	179.8	0.9402	**150**	**0.3412**	179.8	0.9402
Eslabón 6	216	0.4886	288.6	1.905	**266.5**	**0.2988**	288.6	1.905

Tabla 3: Resultados del proceso de pseudo-optimización de los eslabones analizados del mecanismo propuesto

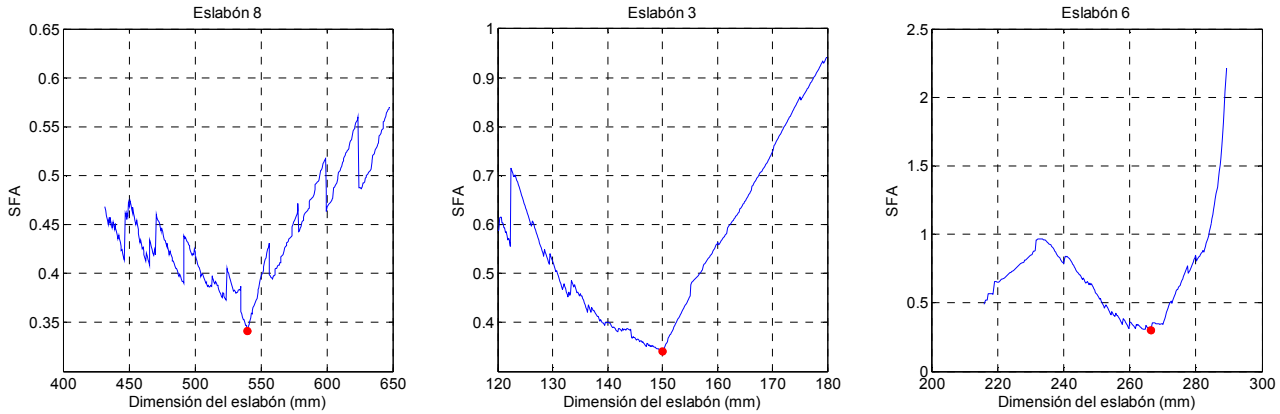


Fig. 11. Valores del parámetro SFA obtenidos, en función de las longitudes de los eslabones

Para analizar los tiempos de cálculo, se ha aplicado el algoritmo a cada eslabón por separado y, posteriormente, de forma secuencial a todos. Los resultados se pueden ver en la Tabla 4 y en la Fig. 12. El análisis de estos tiempos nos muestra que, en esta ocasión, la reducción de tiempo de computación obtenida al utilizar la GPU es insignificante. Esto se debe a que, según el algoritmo, se cargan los resultados calculados con anterioridad a la ejecución de la rutina de pseudo-optimización. De este modo, se realizan constantes transferencias de datos entre la CPU y GPU y las posibilidades de la computación utilizando GPUs se ven seriamente mermadas.

Publicaciones DYNA SL -- c) Mazarredo nº69 - 4º -- 48009-BILBAO (SPAIN)
Tel +34 944 237 566 – www.dyna-newtech.com - email: info@dyna-newtech.com

Eslabón	CUDA C++		MATLAB®	
	Time (s)	Factor-X	Time (s)	Factor-X
Eslabón 8	175,92 ± 1,08	1	183,632304±4,06	1,04
Eslabón 3	176,31 ± 1,98	1	187,063523±5,71	1,06
Eslabón 6	175,36 ± 1,09	1	127,619623±1,98	0,73
Eslabón 8, eslabón 3, eslabón 6	510,2 ± 11,9	1	484,39864±23,14	0,95

Tabla 4: Resultados de los tiempos de ejecución de la pseudo-optimización de los eslabones analizados

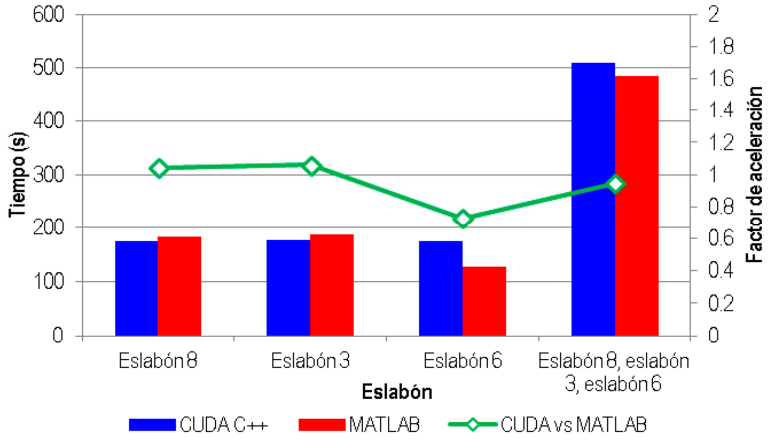


Fig. 12. Evolución de los tiempos de ejecución y factor de aceleración en la pseudo-optimización de los eslabones analizados

4.3.- ALGORITMO COMPLETO

Como última aplicación, se ha realizado una batería de ensayos con el *algoritmo completo* que incluye el cálculo combinado de la cinemática y de la pseudo-optimización. El algoritmo, cuya complejidad es de orden $O(n^2)$, se ha implementado sobre MATLAB® y sobre CUDA C++. En este último caso, en dos versiones: una, cargando los datos y ejecutando todas las rutinas en la GPU; y, otra, en la que se mueven constantemente los datos entre las dos memorias. Los resultados obtenidos se muestran en la Tabla 5 y se representan en la Fig. 13.

El análisis de los datos muestra que la mejor opción consiste en la carga de todos los datos en la memoria de la GPU y ejecutar allí todas las rutinas que sean necesarias. La ganancia de tiempo siguiendo esta estrategia es unas 10 veces superior a ejecutar el mismo algoritmo en MATLAB®. Según los resultados, la peor estrategia de todas consiste en ejecutar tareas en la GPU al tiempo que se transfieren datos entre las memorias de la CPU y la GPU. Haciendo esto se desaprovechan las ventajas del cómputo paralelo al depender totalmente del lento bus PCI Express.

Eslabón	Completo (CUDA)		CompletoMem (CUDA)		Completo (MATLAB®)	
	Tiempo (s)	Factor-X	Tiempo (s)	Factor-X	Tiempo (s)	Factor-X
Eslabón 8	4,9 ± 0,1	1	394,7 ± 57,70	80,54	52,52 ± 0,97	10,72
Eslabón 3	4,89 ± 0,12	1	339,98 ± 49,69	69,54	53,29 ± 2,16	10,9
Eslabón 6	4,9 ± 0,1	1	322,71 ± 30,68	65,8	63,06 ± 1,58	12,86
Eslabón 8, eslabón 3, eslabón 6	12,68 ± 0,12	1	1016,24 ± 71,34	80,17	169,08 ± 2,07	13,34

Tabla 5: Evolución de los tiempos de ejecución del algoritmo completo

Publicaciones DYNA SL -- c) Mazarredo nº69 - 4º -- 48009-BILBAO (SPAIN)
Tel +34 944 237 566 – www.dyna-newtech.com - email: info@dyna-newtech.com

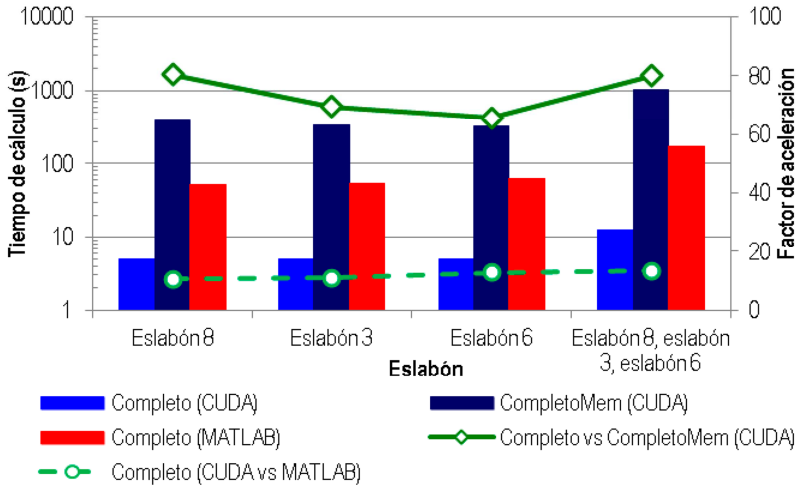


Fig. 13. Evolución de los tiempos de ejecución y factor de aceleración en el algoritmo completo

5.- CONCLUSIONES

A modo de resumen diremos que en este trabajo se ha abordado el estudio del coste computacional de la resolución de varios procesos matemáticos iterativos usados para calcular tres problemas diferentes de un dispositivo ferroviario, utilizando el método de computación clásico (sólo utiliza la CPU) o empleando computación paralela (incorpora GPU). Los tiempos de ejecución necesarios para el cálculo de cada proceso se han analizado sobre cuatro plataformas distintas: C++®, MATLAB®, C++ con CUDA y MATLAB® con CUDA, y con diferentes volúmenes del problema.

Del análisis de los resultados del modelo cinemático de mecanismo ferroviario, con cálculos puramente iterativos, se desprende que la opción que presenta menores tiempos de cálculo es C++® con CUDA. Además, esta opción es más recomendable cuanto mayor es el tamaño del problema de computación a resolver, obteniendo reducciones temporales de hasta 60 veces respecto a MATLAB® y 90 veces respecto a C++®. Sin embargo, la alternativa de combinar MATLAB® con CUDA es la opción más lenta de todas. Estos resultados están en el mismo orden de magnitud que los obtenidos en [14] para simulaciones de dinámica multicuerpo.

En el segundo proceso, el modelo de pseudo-optimización del mecanismo ferroviario, el algoritmo de cálculo se implementó únicamente sobre CUDA C++ y MATLAB®, por ser los dos entornos que arrojaron mejores resultados en el primer modelo. En este algoritmo, además de los cálculos operativos, se precisan numerosas operaciones de carga y descarga de datos. Este proceso repercute en la rapidez de cómputo de la GPU debido a las constantes transferencias de datos que se realizan entre las memorias de CPU y GPU. Como resultado, los tiempos de ejecución que se obtuvieron son similares en MATLAB® y CUDA C++.

Con los resultados del proceso mixto, combinación de los dos anteriores, con tres estrategias distintas de operaciones entre CPU y GPU, se comprobó que el proceso más eficiente se obtiene mediante la transferencia a la GPU de todos los datos de entrada necesarios y la ejecución allí de todas las rutinas programadas y la posterior descarga de datos a la CPU. En este caso, la resolución del algoritmo fue del orden de 10 veces más rápida sobre CUDA C++ que sobre MATLAB®.

Finalmente, en este trabajo se demostró que el uso de la computación paralela puede ser una herramienta de cálculo de interés para reducir los costes de computación en la resolución de problemas de mecánica ferroviaria con alto nivel de repeticiones. Sin embargo, se ha comprobado también que, para problemas donde el intercambio de información entre

Publicaciones DYNA SL -- c) Mazarredo nº69 - 4º -- 48009-BILBAO (SPAIN)
Tel +34 944 237 566 – www.dyna-newtech.com - email: info@dyna-newtech.com

GPU y CPU es frecuente, no es recomendable el uso de GPUs pues los costes de computación son similares con su uso o sin él. En los sistemas mixtos, debido a la importante reducción del coste computacional del primer caso y su repercusión en el sistema mixto, usando computación paralela se consigue una importante reducción de tiempos de cálculo. También, se ha contrastado que hay programas comerciales, muy utilizados por el cálculo en ingeniería, que todavía no han optimizado la computación paralela, al menos en la versión del mismo utilizada en este trabajo.

Los autores consideramos que la metodología propuesta de computación paralela con GPUs, en este caso para el análisis de modelos matemáticos de un sistema mecánico ferroviario, puede ser aplicada a la solución de problemas en otros sectores productivos como el automotriz, el de inyección de plástico, el análisis de imágenes médicas o aéreas, sistemas mecatrónicos, entre otros. Además, el estudio de los costes temporales de ejecución de los diferentes tipos de problemas puede servir de guía para la toma de decisiones del tipo de herramienta computacional a usar.

BIBLIOGRAFÍA

[1] Tovar-Perez C, Cabanellas-Becerra JM, Jimena de Dios GJ. "Una metodología eficiente para la generación de entornos virtuales en simuladores de conducción ferroviaria" *DYNA*. 2013 Vol. 88 p. 433-443 DOI: http://dx.doi.org/10.6036/5524.

[2] Ghorpade J, Parande J, Kulkarni M, et al. "GPGPU processing in CUDA architecture". *Advanced Computing: An International Journal (ACIJ)*, 2012. Vol.3-1, p.105-120. DOI: http://dx.doi.org/10.5121/acij.2012.3109.

[3] Tasora A, Negrut D, Anitescu M. "GPU-based parallel computing for the simulation of complex multibody systems with unilateral and bilateral constraints: an overview" En: Arczewski K. *Multibody Dynamics. Computational Methods and Applications*. Varsovia: Springer, 2011. p.283-307.

[4] Tasora, A. and Anitescu, M. A matrix-free cone complementarity approach for solving large-scale, nonsmooth, rigid body dynamics. Computer Methods in Applied Mechanics and Engineering, 200, pp 439 - 445, 2011.

[5] Mawson MJ, Revell AJ. "Memory transfer optimization for a lattice Boltzmann solver on Kepler architecture NVIDIA GPUs". *Computer Physics Communications*. Octubre 2014. Vol.185-10 p. 2566–2574. DOI: http://dx.doi.org/10.1016/j.cpc.2014.06.003.

[6] Garcia-Blas J, Abella M, Isailaa F, et al. "Surfing the optimization space of a multiple-GPU parallel implementation of a X-ray tomography reconstruction algorithm". *Journal of Systems and Software*. Septiembre 2014. Vol. 95 p. 166–175 DOI: http://dx.doi.org/10.1016/j.jss.2014.03.083.

[7] Sierakowski A. "GPU-centric resolved-particle disperse two-phase flow simulation using the Physalis method". Computer Physics Communications. Octubre 2016. Vol. 207 p. 24-34 DOI: http://dx.doi.org/10.1016/j.cpc.2016.05.006.

[8] Castonguay P." Accelerating CFP simulations with GPUs". En: *Clumeq NVIDIA CUDA/GPU workshop*. Montreal: 2012.

[9] Sandoval-Gutierrez J, Herrera-Lozada JC, Álvarez-Cedillo JA, et al. "Design, Manufacturing and Performance of a Low Cost Delta Robot". *DYNA*, 2016, Vol. 91, no. 3, p.346-352. DOI: http://dx.doi.org/10.6036/7687

[10] Harris M. "Unified Memory in CUDA 6". *NVIDIA Developer Zone*. Noviembre 2013. Disponible en web: <http://devblogs.nvidia.com/parallelforall/unified-memory-in-cuda-6/> [Consulta: Febrero 2015].

[11] Bustos A, Rubio H, García Prada JC, et al. "The Evolution of the Computing Time in the simulation of Mimbot-Biped Robot using Parallel Algorithms". En: *Proceedings of the 2015 IFTOMM World Congress*. Taipei: 2015. ISBN 978-986-04-6098-8.

[12] Guerra C, Pascucci V. "Line-based object recognition using Hausdorff distance: from range images to molecular secondary structures". *Image and Vision Computing.* 2005. Vol. 23-4 p. 405-415. DOI: http://dx.doi.org/10.1016/j.imavis.2004.11.002.

[13] "Measure and Improve GPU Performance". Ayuda de MATLAB® R2013b.

[14] Negrut D, Tasora A, Mazhar H, et al. "Leveraging Parallel Computing in Multibody Dynamics", Multibody System Dynamics, Enero 2012. Vol.27 p. 95-117. DOI: http://dx.doi.org/10.1007/s11044-011-9262-y

AGRADECIMIENTOS

Los autores desean agradecer el apoyo brindado por el Gobierno español para la financiación de este trabajo a través del proyecto MAQ-STATUS DPI2015-69325-C2-1-R.