Proyecto de Computación Paralela y Distribuida

Facundo García Cárdenas, Fernando García y Alessio Ghio
Departamento de Ciencia de la Computación, Universidad de Ingenieria y Tecnologia - UTEC, Lima, Peru

***Resumen*—El algoritmo de N-cuerpos es un método ampliamente difundido, en la comunidad científica, para simular la interacción de un conjunto de partículas en base a unos parámetros que orientan su comportamiento general. Mediante esta herramienta, es posible analizar escenarios complejos y extremos como la colisión de cuerpos celestiales, proximidades de un agujero negro, etc. En este trabajo, se presenta el análisis, implementación y optimización de un código paralelo híbrido, utilizando MPI y OMP, para la simulación de la colisión de dos galaxias. Con el fin de obtener resultados más significativos, se realizaron pruebas en el clúster Khipu de UTEC, en donde se tomaron medidas para hallar el tiempo de ejecución promedio, velocidad y eficiencia del código presentado.**

***Index Terms*—N-cuerpos, MPI, OMP, híbrido, eficiencia.**

I. Estudio del Algoritmo

El proceso de simulación de la colisión de galaxias se realiza comúnmente por medio del algoritmo de N-cuerpos, el cual consta de calcular la interacción entre cada par de partículas en un conjunto que simula cada cuerpo espacial de una galaxia. En su implementación paralela distribuida [1], este algoritmo comienza alocando un espacio de memoria correspondiente al total de partículas en cada proceso MPI, lo cual permite predecir el efecto del conjunto de partículas asignadas a cada proceso sobre el conjunto global. Seguidamente, se cargan los parámetros de simulación como cantidad de cuerpos, tiempo de inicialización, estado inicial de las partículas (masa, posición, velocidad y tiempo), y el orden deseado para las ecuaciones de Taylor empleadas en el proceso de predicción de estado de cada partícula. Debido a que se trabaja con el paradigma de memoria distribuida, estos parámetros son enviados a cada proceso MPI por medio de broadcasts.

Una vez establecida la configuración de la simulación, se inicializa el estado de las particulas en cada proceso MPI. Para ello, se instancia un vaticinador, denominado *Predictor*, el cual estima la posición y velocidad de cada partícula en la siguiente iteración de la simulación. Luego, se calculan las fuerzas ejercidas por cada cuerpo del conjunto local sobre el conjunto global de partículas. Durante este proceso, se calculan las diferencias entre las posiciones y velocidades de cada partícula global con cada partícula local, cuyos valores se utilizan para computar sus correspondientes nuevas aceleraciones y tirones (*jerks*). De igual manera, se computa la energía potencial por kilogramo utilizando la distancia entre cada par de partículas por medio de la siguiente ecuación:

$$E_p = \sqrt{\frac{m}{\epsilon^2 + \Delta x^2 + \Delta y^2 + \Delta z^2}}. \tag{1}$$

En otras palabras, se calcula el efecto de las distancias, aceleraciones y tirones de todo el conjunto de partículas locales sobre cada partícula global. Seguidamente, estos valores son utilizados para estimar las fuerzas ejercidas f_i entre cada par de partículas. Una vez calculadas las fuerzas locales, se realiza un proceso de reducción MPI para obtener las fuerzas resultantes. Esto es posible dado que la fuerza resultante f_r es igual a la sumatoria de todas las fuerzas, expresado como $f_r = \sum_{i=0}^{N} f_i$, donde N es el número total de cuerpos. En base a esta fuerza, se actualiza el estado de cada particula.

Tras actualizar el estado de las partículas, se multiplican las masas por la energía potencial por kilogramo y por la velocidad de cada cuerpo, para obtener las energías potencial y cinética, respectivamente. Con estos valores, se obtiene la energía total del sistema y se verifica que no se haya creado o destruido energía dentro del sistema, como una medida de verificación del comportamiento simulado.

Luego de haber inicializado correctamente el estado de todas las partículas, se procede a simular el comportamiento de estas en el intervalo de tiempo asignado. Para ello, se genera una lista que almacena todas las partículas activas; es decir, aquellas que ejerceran fuerzas o serán afectadas en el intervalo de tiempo analizado. Seguidamente, tal como se realizó durante el proceso de inicialización, se calculan las fuerzas ejercidas sobre cada par de partículas y se verifica la energía total del sistema. Finalmente, se genera un resumen con los tiempos de computo y comunicación para cada sección del código, siendo los tiempos más relevantes:

- Scan: Hallar todas las partículas activas.
- Pred: Predecir el estado futuro de cada partícula.
- Force: Computar las fuerzas resultantes.
- Comm: Comunicación entre procesos MPI (All-reduce).
- Corr: Actualización y correción de estados.
- Tot: Tiempo de ejecución total del algoritmo.

En este trabajo, el análisis del comportamiento del algoritmo se centrará en examinar los efectos generados por la variación en la cantidad de procesadores empleados, la precisión seleccionada y la cantidad de cuerpos simulados. Adicionalmente, para mejorar la eficiencia del algoritmo, se paralelizarán los bucles internos del código por medio de OMP, empleando el paradigma de memoria compartida. A continuación, se detallará cuáles son las condiciones iniciales escogidas para este análisis, las cuales serán empleadas para todas las simulaciones presentadas en esta informe.

II. Modificación de Condiciones Iniciales

El algoritmo de N-cuerpos se centrá en simular la interacción entre un conjunto de N particulas, donde la precisión de su comportamiento está determinada por el orden Y de la serie de Taylor, la cual es empleada para la estimación del estado

de cada cuerpo. Sumado a esto, el análisis preliminar de este algoritmo se centra en su implementación paralela distribuida, en donde se emplean p procesos MPI. Para determinar el efecto de cada variable en el tiempo de ejecución y la eficiencia del algoritmo, se seleccionaron los siguientes valores:

- $Y : 4, 6, 8$
- $N : 1{,}000, 6{,}000, 10{,}000$
- $p : 1, 2, 4, 8$

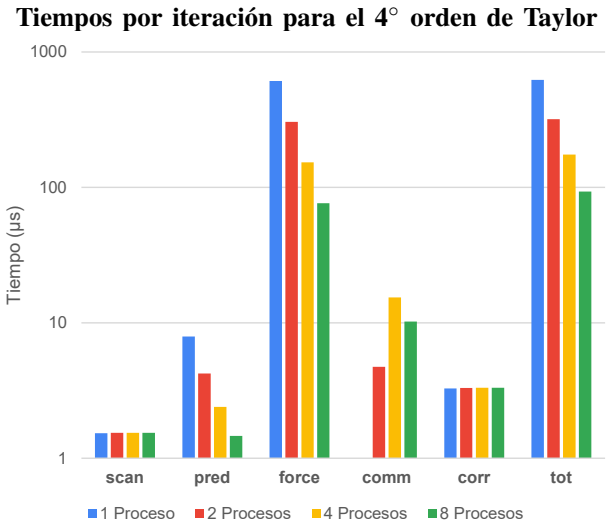
Para escoger el estado inicial de las particulas y simular la colisión de 2 galaxias, es necesario generar un archivo de inicialización que contenga la cantidad de cuerpos a simular, el tiempo de inicialización, y el estado inicial de cada partícula, lo cual incluye su identificador único, masa, posición y velocidad. Para ello, se empleó un archivo que, a partir de constantes físicas y parámetros de simulación deseados, genera un archivo de texto con los datos necesarios para la simulación. En base a esto, se inicializa la simulación de una colisión empleando 1.000, 6.000 y 10.000 particulas. A continuación, se detalla el desarrollo de las simulaciones, en donde se explica el proceso para obtener los resultados de la simulación en el clúster y el análisis realizado.

III. Desarrollo de Simulaciones

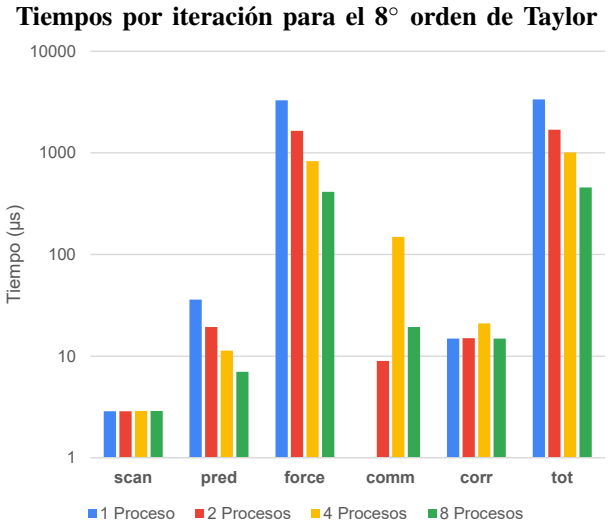
El algoritmo de N-cuerpos contiene una gran variedad de parámetros que afectan su desempeño. Para determinar la influencia de cada variable en un entorno controlado, se empleó el clúster Khipu de UTEC. En este, se evaluó el comportamiento del código para cada combinación de las variables descritas en la sección anterior. Para evitar que la inicialización de las partículas afecte el desempeño del algoritmo y genere acumulaciones de error durante las pruebas, se procuró que su estado inicial y los parámetros empleados sean equivalentes a lo largo de todas las pruebas. A continuación, se presentan los tiempos de ejecución promedio obtenidos.

III-A. Simulación con 1.000 cuerpos

La primera prueba realizada en el clúster Khipu consta de calcular los tiempos de ejecución de cada sección del código para una simulación de 1.000 cuerpos. Como se observa en la Fig. 1, el tiempo de ejecución total del algoritmo disminuye drásticamente en su implementación paralela. Sumado a esto, se observa que el cálculo de las fuerzas resultantes de cada partícula corresponde al mayor tiempo empleado en el código, siendo el menor de estos el computado por 8 procesos, por lo que esta corresponde a una sección crítica a optimizar. Por otro lado, se observa que el tiempo de comunicación no representa una parte significativa del código, lo cual nos indica que este algoritmo podría beneficiarse de utilizar más núcleos y particionar el espacio en mayor medida, generando un tipo de compensación entre la partición del espacio y el tiempo de comunicación entre procesos. Al comparar estos tiempos con los ordenes mayores de la serie de Taylor, se observa que solo aumenta significativamente el tiempo empleado en el cálculo de las fuerzas, por lo que el análisis previo se mantiene sin importar el orden empleado.



(a)

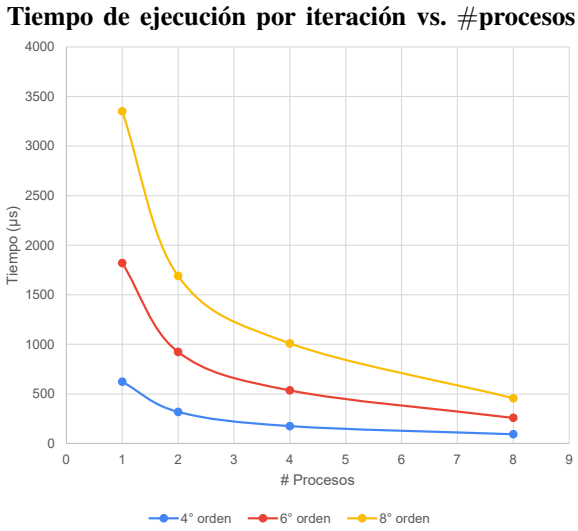


(b)

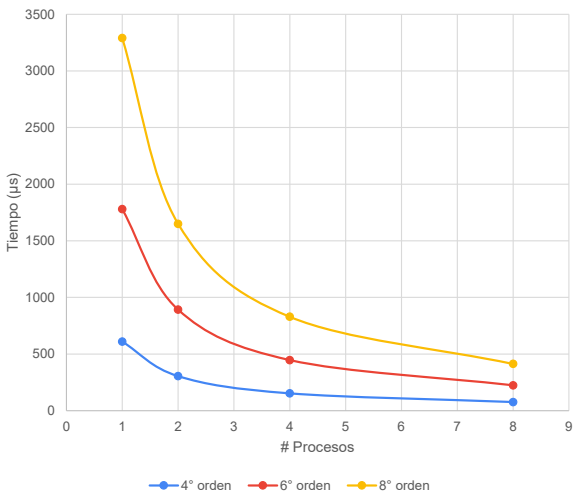
Figura 1: Proporciones de los tiempos de ejecución por iteración, para la simulación de 1.000 cuerpos con MPI.

Para realizar una comparación más profunda con respecto a la proporción del tiempo empleado en el cálculo de las fuerzas y el tiempo de ejecución total del algoritmo, se presenta una gráfica de comparación de tiempos vs. número de procesos empleados, donde cada curva representa el comportamiento para cada orden análizado de la serie de Taylor. Como se observa en la Fig. 2a, el 8° orden de la serie de Taylor presenta el mayor tiempo de ejecución para todos los procesos empleados. Esto corresponde a nuestra apreciación inicial, en donde se estima que el tiempo de cálculo de las fuerzas domina la ejecución, tal como se demuestra en la Fig. 2b. Debido a que el orden de la serie de Taylor dicta cuántas operaciones se realizan en el cálculo de las fuerzas, siendo esta la operación más costosa, este afecta directamente al tiempo de ejecucion de esta sección. Sumado a esto, podemos observar

una disminución exponencial en el tiempo de ejecución total del algoritmo en relación a la cantidad de procesos empleados. Este comportamiento indica que el código analizado es escalable en relación a la cantidad de nodos utilizados, hipótesis que será corroborada en las siguientes secciones.



(a) Tiempos de ejecución totales



(b) Tiempos de cálculo de fuerza

Figura 2: Comparación de tiempo de ejecución total y cálculo de fuerzas para la simulación de una galaxia con 1.000 cuerpos.

Como se puede determinar tras el análisis de estas gráficas, el orden de la serie de Taylor solo genera un aumento en el tiempo de cálculo de las fuerzas. Por lo tanto, solo nos centraremos en el análisis del algoritmo cuando $Y = 4$, para poder determinar los factores que afectan en mayor medida a la eficiencia del algoritmo y su escalabilidad.

III-B. Simulación con 6.000 cuerpos

En el caso de la segunda prueba, donde se realizó la simulación de una galaxia con 6.000 cuerpos, los resultados se asemejan en gran medida a la primera prueba. Esto se puede observar en la Fig. 3, la cual muestra que el tiempo de ejecución del cálculo de fuerzas predomina sobre las demás secciones. Sin embargo, se observa que todos los tiempos de ejecución por iteración incrementaron ligeramente, con excepción del cálculo de fuerzas. Esto se debe a que la cantidad de operaciones en la inicialización, comunicación y corrección incrementa linealmente, mientras que en la predicción y actualización de estado crece exponencialmente.

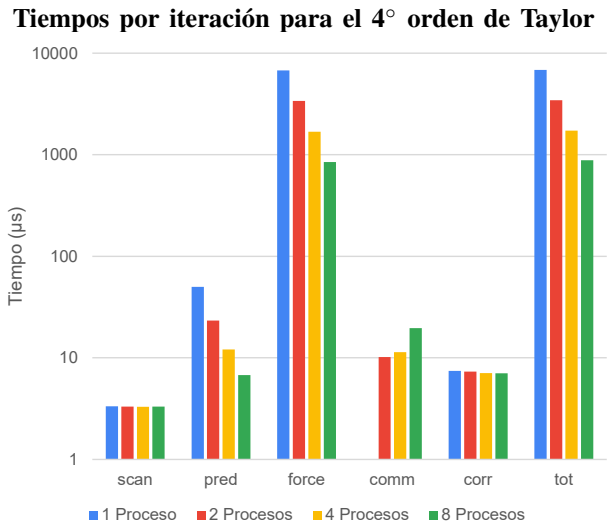


Figura 3: Proporciones de los tiempos de ejecución total, para la simulación de 6.000 cuerpos con MPI.

Por otro lado, la Fig. 4 muestra los tiempos de ejecución totales, para cada orden de la serie de Taylor, en la simulación de 6.000 cuerpos. Al igual que en la Fig. 2a, se observa que el tiempo de ejecución por iteración disminuye exponencialmente a medida que incrementa la cantidad de procesos empleados. No obstante, los valores obtenidos son aproximadamente 11 veces mayores, lo cual se debe a que se simuló una mayor cantidad de interacciones entre partículas.

III-C. Simulación con 10.000 cuerpos

Al incrementar aún más la cantidad de cuerpos a simular, los efectos del análisis anterior se repiten en mayor medida. En otras palabras, como se muestra en la Fig. 5, el incremento del tiempo de ejecución, en el cálculo de fuerzas, se vuelve más considerable; mientras que los demás tiempos aumentan levemente con respecto al tiempo de ejecución total, pero significativamente en comparación con la misma sección para una cantidad de cuerpos menor.

De la misma manera, la Fig. 6 muestra el mismo comportamiento de las pruebas anteriores, donde el tiempo de ejecución por iteración disminuye de manera exponencial a medida que aumenta la cantidad de procesos. Sin embargo, el incremento total de tiempo con respecto a la prueba anterior se duplica, aproximadamente, mientras que en la prueba anterior incrementó casi 11 veces más con respecto al primer resultado.

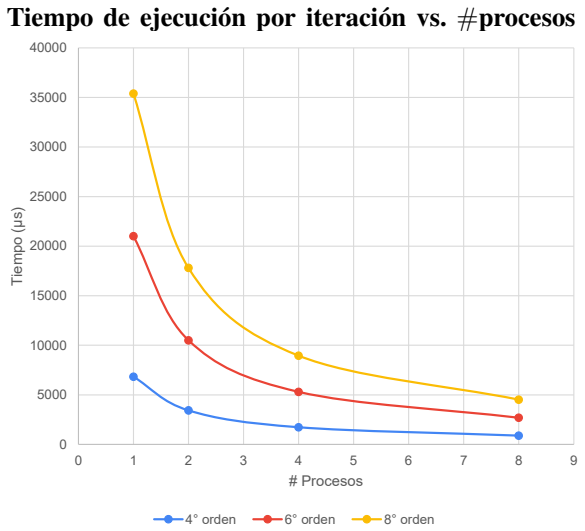


Figura 4: Tiempos de ejecución totales, para la simulación de 6.000 cuerpos con MPI.

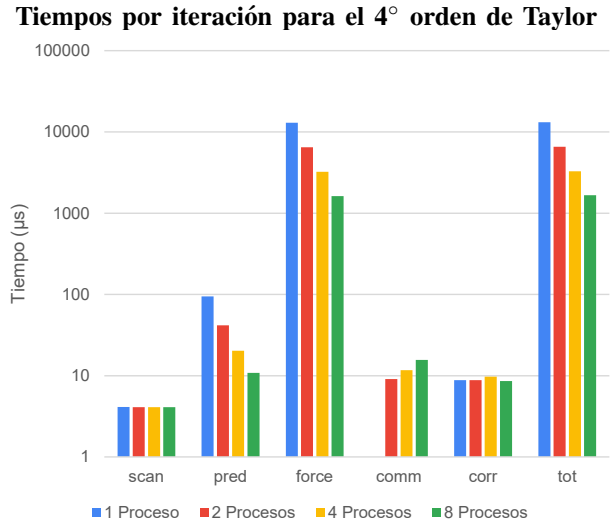


Figura 5: Proporciones de los tiempos de ejecución total, para la simulación de 10.000 cuerpos con MPI.

IV. Análisis de Complejidad

Para calcular la complejidad total del algoritmo, es necesario definir individualmente el costo de cálculo y el costo de comunicación. Para ello, se observa que el código presenta dos partes principales, las cuales se denominan bucle de inicialización de estados y bucle temporal. El primer bucle se encarga de predecir el comportamiento del conjunto local de partículas, con respecto al conjunto global, y estimar su futura posición y velocidad. Por otro lado, el bucle temporal simula el comportamiento de todas las partículas en el intervalo de tiempo asignado, en donde se estima el estado actual y futuro de cada partícula, la interacción entre los cuerpos, y se verifica que la energía total del sistema se mantenga constante. En base a esto, el costo de computar el algoritmo está dado por

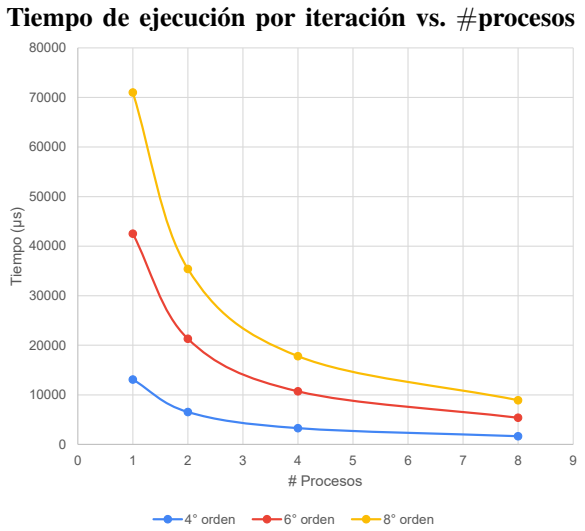


Figura 6: Tiempos de ejecución totales, para la simulación de 10.000 cuerpos con MPI.

la siguiente ecuación:

$$T_{comp} = [i \cdot (\frac{n^2}{p} + n(1 + \frac{1}{p}) + p)] + [w \cdot (\frac{n^2}{p} + n(1 + \frac{1}{p}) + p)] \quad (2)$$

donde p es la cantidad de procesos empleados, n es la cantidad de partículas simuladas, i es la cantidad de iteraciones del bucle de inicialización, y w es la cantidad de iteraciones del bucle temporal. Es importante resaltar que i suele ser un valor entero positivo cercano a 1, por lo que su efecto no es significativo. De igual manera, p debe ser siempre un valor entero positivo mayor o igual a 1, por lo que el término $\frac{1}{p}$ se puede eliminar de la ecuación. Por lo tanto, la complejidad de computo se puede reducir a la siguiente expresión:

$$T_{comp} = (1 + w) \cdot (\frac{n^2}{p} + n + p) \quad (3)$$

En caso se desee estimar la complejidad total del algoritmo secuencial, solo se debe reemplazar la Eq. 3 con el valor $p = 1$, dado que solo se emplea un proceso y no se emplean recursos en la comunicación. Caso contrario, al trabajar con un sistema de memoria distribuida, es necesario calcular el tiempo empleado para la comunicación entre nodos. Con esto en mente, se observa que el código emplea un total de 9 operaciones broadcast, 2 all-reduce y un reduce, las cuales se encuentran distribuidas de la siguiente manera:

- 1 broadcast (*short*) de 1 entero.
- 7 broadcasts (*short*) de 1 doble.
- 1 broadcast (*short*) de 128 caracteres.
- 2 all-reduce (*long*) de $14n$ dobles.
- 1 reduce (*short*) de 1 doble.

No obstante, para hallar el costo total de comunicación, es necesario definir la complejidad individual de cada operación.

A continuación, se presenta un pequeño resumen de la complejidad de las operaciones más frecuentes en base a la longitud del mensaje enviado [2].

- Broadcast (*short*): Se utiliza el método de *Binomial tree* con una complejidad $(\log p)(\alpha + \beta(n \cdot T_{byte}))$.
- Broadcast (*long*): Se emplea un Scatter seguido de un All-gather, con una complejidad de $\alpha(\log p + p - 1) + 2\beta(p-1)(n \cdot T_{byte})/p$.
- Reduce (*short*): Se utiliza el método de *Binomial tree* con una complejidad $(\log p)(\alpha + \beta(n \cdot T_{byte}))$.
- Reduce (*long*): Se emplea un Reduce-Scatter seguido de un Gather, con una complejidad de $2\alpha \log p + 2\beta(p-1)(n \cdot T_{byte})/p$.
- All-reduce (*short*): Se utiliza el método de *Recursive doubling* con una complejidad $(\log p)(\alpha + (n \cdot T_{byte})\beta)$.
- All-reduce (*long*): Se emplea un Reduce-Scatter seguido de un All-gather, con una complejidad de $2\alpha \log p + 2\beta(p-1)(n \cdot T_{byte})/p$.

En base a estos valores, se reemplaza el valor de T_{byte} en cada expresión, el cual corresponde al tiempo empleado para enviar la cantidad de bytes contenidos en un mensaje, y se obtiene la siguiente ecuación para la complejidad de la comunicación en el algoritmo analizado:

$$\begin{aligned} T_{comm} = &[7\log(p)(\alpha + 8\beta)] + [\log(p)(\alpha + 4\beta)] \\ &+ [\log(p)(\alpha + 512\beta)] + [\log(p)(\alpha + 8\beta)] \\ &+ [4\alpha\log(p) + 4\beta(p-1)(112n)/p] \end{aligned} \tag{4}$$

donde α representa el retraso de comunicación y β corresponde al ancho de banda de la arquitectura empleada. Una vez hallados estos valores, la complejidad teórica total del código analizado está dada por:

$$T_{tot} = T_{comp} + T_{comm} \tag{5}$$

Para verificar la precisión de la expresión hallada, se debe reemplazar los valores de las variables previamente mencionadas y compararla con el tiempo de ejecución observado en la sección anterior. La Fig. 7 presenta dicha comparación, donde se reemplazaron los siguientes valores:

- α: 1
- β: 0.8
- n: 1,000
- w: 100

donde α y β fueron estimados en base a un ajuste con la curva real. En otras palabras, se puede decir que dichos parámetros son representativos para la arquitectura del clúster Khipu. Finalmente, para obtener la precisión del modelo, se calculó el valor RMSE (Root-Mean-Square Error), el cual es de 0.996 %. Esto nos permite concluír que el modelo generado predice eficientemente el comportamiento del algoritmo.

V. Análisis de Velocidad y Escalabilidad

El algoritmo de N-cuerpos se basa en calcular las interacciones entre cada par de partículas, por lo que se espera ver una dependencia cuadrática en la cantidad de FLOPs procesados

Tiempo de ejecución total teórico vs. real para el 4° orden de la serie de Taylor

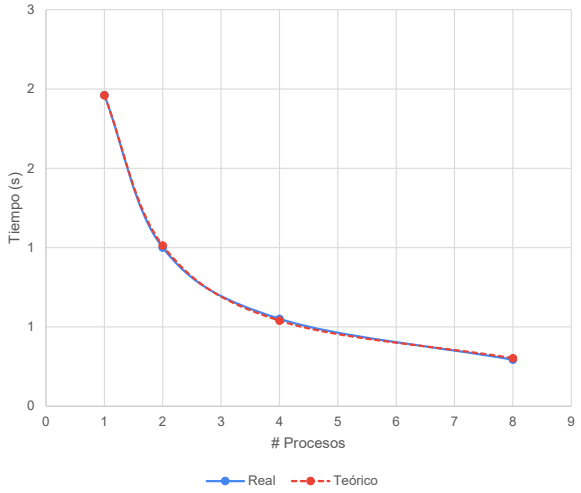


Figura 7: Tiempos de ejecución totales teóricos y reales, para la simulación de 1.000 cuerpos con MPI.

con respecto a la cantidad de partículas simuladas. Por lo tanto, para poder corroborar esta hipótesis, se requiere analizar las operaciones realizadas dentro del bucle de inicialización y el bucle principal (denominado bucle temporal). Para simplificar este análisis, nos centraremos en el caso $Y = 4$, dado que el efecto del orden de la serie de Taylor es pequeño en comparación a la dependencia cuadrática sobre la cantidad de cuerpos evaluados. A continuación, se presenta la ecuación que modela la cantidad de operaciones de punto flotante realizadas por el proceso de simulación de la colisión de 2 galaxias.

$$\begin{aligned} FLOPs = p \cdot [(&\frac{43n^2}{p} + \frac{37n}{p} + 57n + p) \cdot (i + w) \\ &+ w \cdot (129n + 52) + 29n + 47 + p] + 83 \end{aligned} \tag{6}$$

donde p es la cantidad de procesadores empleados, n es la cantidad de partículas simuladas, i es la cantidad de iteraciones del bucle de inicialización, y w es la cantidad de iteraciones del bucle temporal. Como se puede observar, la cantidad de FLOPs procesados por el algoritmo presenta una dependencia cuadrática con respecto a n y lineal con respecto a p. Por lo tanto, estas son las variales que afectan en mayor medida a la complejidad del algoritmo. Adicionalmente, es importante resaltar que i suele ser un valor entero positivo cercano a 1, por lo que su efecto no es significativo.

Por otro lado, se calculó el speedup de los tiempos de ejecución por iteración, utilizando el 4° orden de la serie de Taylor, para las simulaciones de 1.000, 6.000 y 10.000 cuerpos. El resultado se muestra en la Fig. 8, donde se puede observar que en todas las pruebas, el speedup aumenta de manera lineal. Sin embargo, en el caso de 1.000 cuerpos, la pendiente no es tan pronunciada como en las otras simulaciones. Esto indica que si se aumenta la cantidad de procesos en la primera prueba, la reducción de tiempo de ejecución no sería tan sigificativa

en comparación con las otras pruebas. No obstante, se puede notar que el speedup para la simulación de 6.000 cuerpos empieza a decaer, lo cual quiere decir que, con más procesos, la diferencia de pendientes entre esta simulación y la de 10.000 cuerpos se volvería más notoria.

Figura 8: Speedup para la simulación de 1.000, 6.000 y 10.000 cuerpos con MPI.

Estas observaciones se refuerzan en la Fig. 9, la cual muestra los resultados del cálculo de eficiencia. En la primera simulación, la eficiencia disminuye en mayor medida que las siguientes dos simulaciones, lo cual es de esperarse debido a que la pendiente del speedup no es muy pronunciada. Asimismo, se puede observar que la eficiencia con 8 procesos no llega a disminuir en gran medida para la simulación de 6.000 y 10.000 cuerpos, pero se puede estimar que, a medida que se aumenten los procesos, la diferencia entre estos se volvería significativa. Esto último se debe al declive de la pendiente del speedup de la simulación de 6.000 cuerpos, mencionada anteriormente.

En base a estos valores, se puede observar que la escalabilidad del algoritmo sigue la ley de Gustafson, dado que el speedup presenta un comportamiento lineal al aumentar la cantidad de nodos, en el rango de n empleado para la simulación. Sumado a esto, se puede concluir que este algoritmo, en su implementación distribuida, es escalable pero no en gran medida. Esto se debe a que el tiempo empleado en el cálculo de fuerzas aumenta exponencialmente y no se encuentra paralelizado. Por lo tanto, para un valor de n mucho mayor, la eficiencia del algoritmo disminuiría drásticamente. De esta manera, se determina que es necesario paralelizar el bucle de estimación de fuerzas para optimizar este algoritmo, lo cual se discutirá en la siguiente sección.

Finalmente, para determinar si la eficiencia se mantiene constante durante las pruebas realizadas, se calculó el error estadístico $\sigma_x = \frac{\sigma}{\sqrt{n}}$ para un conjunto de 100 muestras, donde σ es la desviación estándar y n la cantidad de pruebas realizadas. En la Tabla I, se presentan los resultados para

Figura 9: Eficiencia para la simulación de 1.000, 6.000 y 10.000 cuerpos con MPI.

$p = [1, 2, 4, 8]$ y $n = [1{,}000, 6{,}000, 10{,}000]$. Como se puede observar, todos los errores calculados son menores al 1 %. Por lo tanto, se concluye que la eficiencia hallada para la implementación en memoria distribuida de este código se mantiene constante a lo largo de todas las pruebas realizadas.

Cuadro I: Error estadístico de la eficiencia.

	1 proceso	2 procesos	4 procesos	8 procesos
1.000 cuerpos	0.0106 %	0.0148 %	0.5366 %	0.4398 %
6.000 cuerpos	0.0055 %	0.1521 %	0.0096 %	0.3186 %
10.000 cuerpos	0.0034 %	0.0187 %	0.0590 %	0.0206 %

VI. Optimización

En base al análisis realizado sobre la implementación paralela distribuida del algoritmo de N-cuerpos, se pudo determinar que este método presenta un bajo costo de comunicación y que la mayor parte de su procesamiento se halla en el cálculo de las fuerzas resultantes de la interacción entre partículas. Debido a esto, una forma intuitiva de optimizar el código se centra en paralelizar el bucle de fuerzas por medio de OMP, el cual se basa en el paradigma de memoria compartida. Este enfoque consiste en distribuir el conjunto de partículas correspondiente a cada nodo MPI entre h hilos OMP. Este método presenta la ventaja de evitar aumentar el tiempo de comunicación del algoritmo y, debido a que el efecto de las partículas es independiente de los resultados previos del bucle de fuerzas, no es necesario lidiar con comunicaciones entre fronteras, en el dominio evaluado. En otras palabras, el conjunto asignado a cada hilo OMP es independiente del trabajo realizado por el resto. A continuación, se presenta una gráfica comparativa de los tiempos de ejecución resultantes del algoritmo de N-cuerpos paralelizado con MPI y su implementación híbrida, para un total de 1.000 cuerpos.

Como se observa en la Fig. 10, la paralelización del bucle de fuerzas, mediante OMP, resulta en promedio 1.9 veces más

Tiempo de ejecución en memoria distribuída vs. híbrida [$n = 1{,}000$]

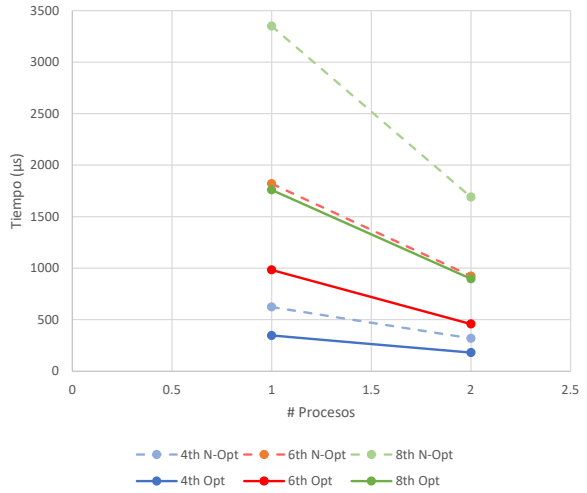


Figura 10: Comparación de tiempos de ejecución entre la implementación distribuída e híbrida [$n = 1{,}000$].

rápido que su contraparte secuencial, para todos los casos analizados. Con el fin de evaluar el efecto de incrementar n, se presenta la misma gráfica comparativa cuando $n = 6{,}000$.

Tiempo de ejecución en memoria distribuída vs. híbrida [$n = 6{,}000$]

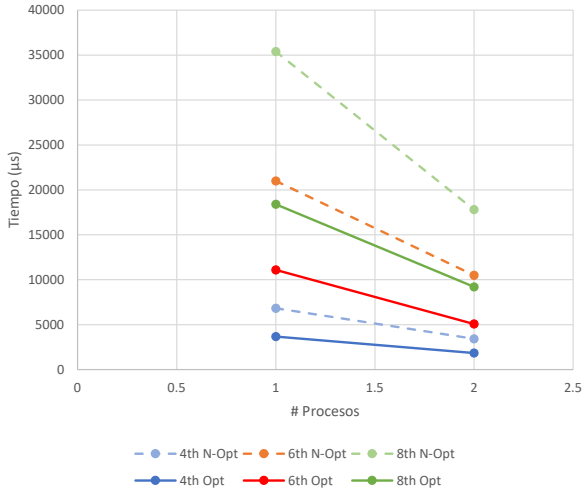


Figura 11: Comparación de tiempos de ejecución entre la implementación distribuída e híbrida [$n = 6{,}000$].

Al igual que en el caso anterior, se observa que la paralelización del bucle de fuerzas resulta en promedio 1.95 veces más rapido. Finalmente, en la Fig. 12 se presenta la misma gráfica comparativa para $n = 10{,}000$.

De igual manera, la paralelización resulta en promedio 1.9 veces más rápido. Por lo tanto, esto nos permite concluir que la aceleración del algoritmo, generada por la paralelización en memoria compartida del bucle de fuerzas, es independiente del tamaño de n. Así mismo, se observa que la variación en

Tiempo de ejecución en memoria distribuída vs. híbrida [$n = 10{,}000$]

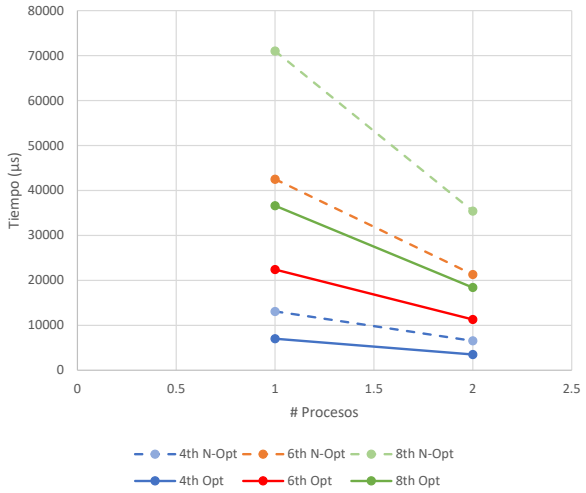


Figura 12: Comparación de tiempos de ejecución entre la implementación distribuída e híbrida [$n = 10{,}000$].

la aceleración del algoritmo resulta mínima entre los ordenes empleados para la serie de Taylor, por lo cual es también independiente de estos. En base a esto, con el fin de analizar la eficiencia de la optimización planteada, se presenta un gráfico del speedup para todos los n evaluados, cuando $Y = 4$.

Speedup de la implementación híbrida

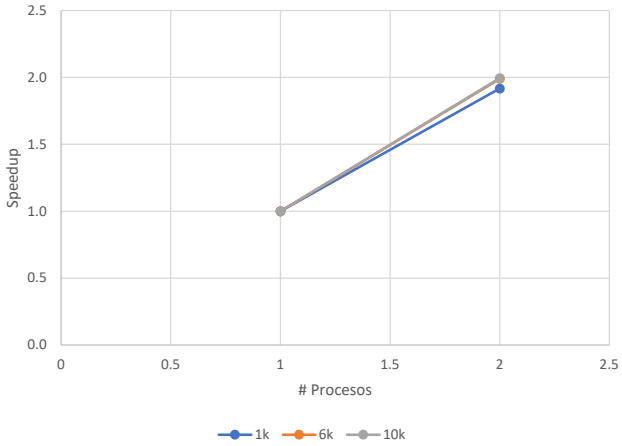


Figura 13: Comparación del speedup de la implementación híbrida para $n = 1{,}000, 6{,}000, 10{,}000$.

Como se observa en la Fig. 13, el speedup del código optimizado presenta un comportamiento lineal, el cual se encuentra en acorde con lo predicho por la ley de Gustafson. Finalmente, la Fig. 14 presenta una gráfica de eficiencia para el algoritmo con paralelismo híbrido. En esta, se observa que la eficiencia se mantiene constante y cercana a 1. A su vez, a medida que se incrementa la cantidad de partículas empleadas, mayor es la eficiencia resultante del algoritmo, la cual presenta

un comportamiento positivo creciente. Entonces, se concluye que el algoritmo propuesto es estable y altamente escalable para el dominio y rango analizado. No obstante, debido a límitaciones en los recursos disponibles en el clúster Khipu, no se pueden realizar pruebas con una mayor cantidad de nodos e hilos. Sin embargo, si se extrapola el comportamiento observado en los casos analizados, se puede estimar con una cierta precisión que el comportamiento del speedup del algoritmo seguirá mostrando un comportamiento líneal con pendiente positiva y, a su vez, la eficiencia seguirá siendo cercana a 1. Esto se debe principalmente a que se está acelerando la sección de código que presenta la mayor latencia del algoritmo, la cual corresponde al bucle de fuerzas.

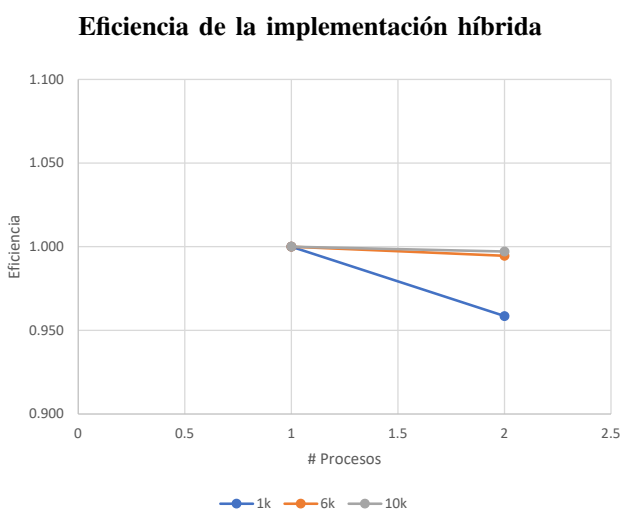


Figura 14: Comparación de la eficiencia de la implementación híbrida para $n = 1{,}000, 6{,}000, 10{,}000$ e $Y = 4$.

VII. Conclusiones

La implementación paralela híbrida del algoritmo de N-cuerpos presenta una ventaja significativa frente a su contraparte secuencial y distruibuida. Esto se debe a que el código es capaz de distribuir su dominio entre n procesos, los cuales a su vez lo subdividen en h hilos para acelerar aún mas la simulación. Debido a que los tiempos de comunicación en este algoritmo son bajos, y la mayor parte del procesamiento se centra el cálculo de las fuerzas ejercidas, entre cada par de partículas, este algoritmo es altamente paralelizable y escalable. Tal como se demostró a lo largo de este trabajo, este algoritmo es capaz de determinar la precisión deseada en base al orden de la serie de Taylor empleado, el cual incrementa directamente el tiempo de ejecución del bucle de fuerzas y se ve altamente beneficiado por su paralelización en memoria compartida. En base a esto, se determina que la implementación propuesta en este trabajo es altamente eficiente y escalable. Para un futuro trabajo, se recomienda mejorar el proceso de distribución del dominio, a fin de emplear más procesadores para el cálculo de interacciones. Finalmente, se recomienda evaluar este algoritmo en un sistema con mayor disponibilidad de recursos para estudiar el comportamiento de la eficiencia de este método con una cantidad mayor de procesos e hilos.

Referencias

[1] R. Spurzem, P. Berczik, I. Berentzen, K. Nitadori, T. Hamada, G. Marcus, A. Kugel, R. Männer, J. Fiestas, R. Banerjee *et al.*, "Astrophysical particle simulations with large custom gpu clusters on three continents," *Computer Science-Research and Development*, vol. 26, no. 3-4, pp. 145–151, 2011.

[2] The Open MPI Development Team, Jun 2020. [Online]. Available: https://www.open-mpi.org/doc/