Instituto de Computación
Facultad de Ingeniería
Universidad de la República
Montevideo, Uruguay

PROYECTO DE GRADO

Computación de alto desempeño para el estudio de medios granulares

Daniel Frascarelli
dsanfra@gmail.com

Febrero de 2018

Tutor: Sergio Nesmachnow

Co-Tutor: Gonzalo Tancredi

VERSIÓN PARA REVISIÓN

Resumen

Este documento presenta el proyecto de grado “Computación de alto desempeño para el estudio de medios granulares”, desarrollado en el Centro de Cálculo del Instituto de Computación, Facultad de Ingeniería de la Universidad de la República.

El proyecto propone una implementación paralela del cálculo del potencial gravitatorio generado por un conglomerado de partículas en una grilla tridimensional. El cálculo del potencial gravitatorio demanda una gran capacidad de cómputo y de tiempo, y en este proyecto se propone una implementación que utiliza técnicas de programación paralela con la finalidad de disminuir el tiempo de ejecución. La evaluación experimental de los algoritmos paralelos desarrollados permite concluir que el speedup obtenido se aproxima a ser lineal para sistemas de 2 x 10^5 partículas.

Distintos artículos han sido publicados en el transcurso del mismo [3], [4]. La implementación realizada se utilizó como base para otros trabajos de maestría [19]. Además, el desarrollo realizado ha sido integrado al paquete de simulación numérica ESyS-Particle y a futuro será posible descargarlo como parte del paquete principal por parte de la comunidad.

Índice general

Capítulo 1

Introducción

Los objetos que orbitan el sol pueden clasificarse en planetas, planetas enanos y en cuerpos menores del sistema solar (en inglés Small Solar System Body o SSSB)[13]. Tanto los planetas como los planetas enanos tienen masa suficiente para su autogravedad superar las fuerzas rígidas del cuerpo, de modo que asumen un equilibrio hidrostático (casi redondeado). De este modo, estos objetos tienen un interior monolítico que sufre deformación plástica durante un largo período de tiempo. Los SSSB, que incluyen cometas y la mayoría de los asteroides y objetos trans-neptunianos, tienen menos de unos pocos cientos de kilómetros de diámetro y tienen varias estructuras que van desde cuerpos monolíticos, objetos fracturados internamente hasta pilas gravitacionales[14].

Los astrónomos han especulado que algunos pequeños asteroides parecen ser grumos de grava unidos por un campo de autogravedad muy débil[15], basado en imágenes recogidas del asteroide 25143 Itokawa por la sonda espacial japonesa Hayabusa en 2005. Sin embargo, la estructura de estos aglomerados de grava y el tamaño de sus componentes internos son desconocidos. La composición química de los SSSB también es diversa: los asteroides están formados por rocas y metales, mientras que los cometas y posiblemente los objetos trans-neptunianos son una mezcla de hielo y material rocoso.

Hay muchas pruebas de que algunos asteroides y cometas son aglomerados de partículas pequeñas o granos, (a) la mayoría de los asteroides presentan periodos de rotación mayores que el límite de inestabilidad rotacional[16], que es consistente con una estructura de pila de escombros; (b) los asteroides y los cometas sufren alteraciones de las mareas cuando entran dentro del límite de Roche de un objeto masivo, lo que indica que tienen muy poca fuerza interna[17] [18]; (c) las cadenas de cráteres, como las que se observan en la luna de Júpiter, se forman tras la colisión de un asteroide o un cometa interrumpido; y (d) muchos asteroides y cometas tienen una densidad baja, que es sólo compatible con un material poroso que tiene grandes vacíos internos.

Además de la autogravedad, los granos o partículas que componen el aglomerado experimentan interacciones mecánicas (elásticas, friccionales y de colisión) con otros granos. Dichos aglomerados también están sujetos a influencias externas, tales como impactos con otros objetos pequeños y fuerzas gravitacionales y de marea ejercidas por objetos más grandes. Las interacciones entre los granos pueden ser simuladas usando métodos de elementos discretos (en ingles Discrete Element Method o DEM). Para reducir el costo computacional, los DEM asumen interacciones mecánicas de corto alcance, considerando que, los granos sólo interactúan con sus vecinos más cercanos. En consecuencia, los investigadores se han limitado hasta ahora a utilizar DEM para simular la física granular en entornos unidireccionales de gravedad, incluyendo recientemente casos de baja gravedad.

Una deficiencia típica de la mayoría de los códigos DEM, es la falta de modelos para simular fuerzas de largo alcance como la gravedad entre las partículas del aglomerado. Este es un problema relevante en el caso de los asteroides y los cometas, que en muchos casos son aglomerados de rocas con una cierta distribución de tamaños. Al tratar con SSSB realistas (aglomerados del orden de millones de partículas) calcular las interacciones entre cada par de partículas tiene un costo computacional muy alto, haciendo difícil estudiar el movimiento del aglomerado por un periodo de tiempo significativo. Se necesita incorpora un nuevo enfoque DEM para calcular la auto-gravedad de sistemas compuestos de millones de partículas que permita estudiar el comportamiento de los SSSBs.

Se propuso la realización de un desarrollo que permite abordar el análisis de aglomerados incluyendo un nuevo enfoque a las técnicas DEM existentes. Además, el método propuesto es adecuado para los cálculos de auto-gravedad en los sistemas de partículas densamente empaquetados, como los SSSBs.

El análisis experimental realizado demuestra que el algoritmo desarrollado escala de forma lineal con el número de partículas del sistema, así como con el numero de recursos de computo. Diferentes estrategias de asignación de trabajo fueron desarrolladas permitiendo reducir el tiempos de computo final y logrando una eficiencia computacional de 0.85 para algomerados de millones de partículas.

La principal contribución de este proyecto es el desarrollo de un algoritmo paralelo para calcular eficientemente la auto-gravedad de un aglomerado de granos. Este desarrollo permite simular y estudiar el comportamiento de aglomerados del orden de millones de partículas en tiempos razonables.

En el transcurso del proyecto se realizaron diversas actividades vinculadas al mismo. En el año 2012 se obtuvo una beca de Iniciación a la Investigación subvencionada por la ANII. En la edición 2013 de Ingeniería Demuestra, el proyecto ganó el primer premio en la categoría *Baja Interpretación Intuitiva*, con un bono premio de USD 300[1]. En agosto de 2014, se participó en el congreso SBD2014 con la presentación *Computing of Self-gravity for Small Solar System Bodies*[2]. En setiembre 2014, se publicó un artículo en la revista IEEE, bajo el titulo *High-Performance Computing of Self-Gravity for Small Solar System Bodies* [3]. En abril 2016, se publicó un articulo como parte de la participación del congreso ISUM 2015, bajo el título *A parallel multithreading algorithm for self-gravity calculation on agglomerates* [4]

La visualización gráfica de simulaciones numéricas permite a los investigadores comprender aspectos claves del comportamiento de aglomerados. Para poder visualizar gráficamente los resultados numéricos obtenidos por el algoritmo desarrollado se integró el código implementado con un paquete de simulación numérica. Como resultado de esta integración los investigadores del área tienen una herramienta que permite no solo realizar cálculos numéricos mediante modelos DEM, sino que pueden visualizar gráficamente el comportamiento obtenido.

Lo que resta de la monografía se organiza de la forma que se describe a continuación. En el Capítulo 2 se describe el problema principal de diseño y desarrollo de un algoritmo para el calculo de la auto-gravedad de un aglomerado de partículas y una reseña de los principales trabajos del estado del arte. En el capítulo 3 se presenta el diseño y arquitectura del algoritmo propuesto. En el capítulo 4 se reporta el análisis experimental realizado para analizar el comportamiento del algoritmo implementado. A continuación, el capítulo 5 presenta las conclusiones y las principales líneas de trabajo futuro.

Capítulo 2

Marco teorico

En este capítulo se presentan conceptos teóricos relacionados que se utilizan a lo largo de los capítulos siguientes. Se realiza una introducción conceptual a los siguientes temas: 1. Método de los elementos discretos, 2. Algoritmo Evolutivo, 3. Auto gravedad, 4. Computación de alto rendimiento, 5. Interpolación y 6. ESyS-Particule.

2.1. Método de los elementos discretos

En esta sección se introduce el concepto del método de los elementos discretos (DEM por su sigla en ingles).

Los medios granulares están formados por un cierto número de objetos macroscópicos (llamados granos) que interaccionan por medio de contactos temporales o permanentes. Todos los materiales que se presentan en forma de granulados (cereales, arena, etc.) o polvos (talco, harina, etc.) son estudiados por la física de medios granulares. Un desafío en las investigaciones actuales es la posibilidad de realizar simulaciones realistas de dichos medios granulares consistentes en miles o millones de partículas y analizar su comportamiento teórico basado en las micro interacciones a nivel de cada partícula.

Las simulaciones mediante DEM han sido empleadas y validadas en diversas áreas, como: geo-mecánica (rocas, tierra), agricultura (transporte, manejo de grano y semillas), farmacia (manejo de polvo y píldoras), química (fluidización, ruptura de partículas), ingeniería de procesos (peening, efecto de virutas y aglomeración), petróleo y gas (bloqueo de conductos por la arena), y medio ambiente y biología (purificación de agua, sangre) entre otras.

El primer hito importante en la historia de DEM se enmarca en 1970 [20], cuando aparecen las primeras simulaciones destinadas a estudiar los sistemas granulares (mecánica de rocas con pocas partículas muy grandes). Las primeras implementaciones fueron mejoradas hacía 1979 [21], para permitir el análisis de partículas más pequeñas, incrementando el rango de sistemas físicos a los cuales el modelos se podían aplicar. Finalmente hacia 1992 [20], se realiza una revisión de los métodos de simulaciones numéricas y se materializan códigos de DEM que permiten: desplazamientos finitos y rotaciones de los elementos discretos, así como el reconocimiento de nuevos contactos a medida que el cálculo avanza. A partir de las siguientes décadas, las publicaciones y referencias al DEM aumenta [22].

Actualmente, el DEM es cada vez más aceptado como un método eficaz para hacer frente a los problemas de ingeniería de materiales granulares o discontinuos[22], especialmente en flujo de granos, mecánica de polvos y mecánica de rocas, dado que se puede adaptar fácilmente al área al cual se está aplicando.

Los procesos que involucran medios granulares se han estudiado experimentalmente, mediante el desarrollo de experiencias de laboratorio que reproducen los fenómenos, y en las últimas décadas en forma numérica. El DEM simula el comportamiento mecánico de un medio formado por un conjunto de partículas las cuales interaccionan entre sí a través de sus puntos de contacto. La disposición de las partículas dentro del medio granular es aleatoria, por lo que se pueden formar medios con diferentes tamaños de partículas distribuidos a lo largo del conjunto.

Principalmente se pueden distinguir las siguientes propiedades básicas que definen el método de análisis numérico: 1. las partículas se consideran como elementos discretos que en su conjunto conforman el sistema complejo de partículas, 2. estas partículas se desplazan independientemente una de otras e interaccionan entre sí en las zonas de contacto y 3. a nivel de cada partícula se hace uso de la mecánica del cuerpo rígido y los elementos discretos se consideran elementos rígidos en sí.

Habitualmente las partículas pueden tener cualquier forma, son elásticas y no sufren deformaciones permanentes.

Una simulación usando DEM comienza por generar un modelo sistema, es decir, posicionar espacialmente todas las partículas asignando a cada partícula sus propiedades físicas (masa, forma, velocidad, densidad, centro de masa, elasticidad, etc.) que la caracterizan en la simulación. Las fuerzas que actúan sobre cada partícula son calculadas usando el modelo inicial, las leyes físicas que actúen sobre ellas y los modelos de contacto entre partículas. Generalmente, el proceso de simulación se entiende como la ejecución reiterada de las siguientes etapas:

- Inicialización del sistema de partículas
- Inicialización de cada partícula y sus propiedades

- Paso de integración
- Cálculo de la nueva posición de cada partícula tomando en cuenta sus propiedades, las fuerzas aplicadas e interacciones con partículas vecinas.
- Post-procesamiento
- Tareas de aceptación del nuevo sistema.

El DEM hace un uso intenso de CPU, lo cual limita la cantidad de tiempo que se puede simular así como también la cantidad máxima de partículas que se pueden tomar en consideración. Algoritmos secuenciales para simulaciones DEM de sistemas de tamaño medio, compuesto de hasta unos pocos miles de partículas, pueden ser ejecutados en computadores personales. La potencia de cálculo requerida para las simulaciones de sistemas más grandes, del orden de un millón de partículas, supera ampliamente las capacidades de un computador personal si se espera obtener resultados útiles y en un tiempo razonable. Para estos casos es recomendable la aplicación de técnicas especiales de procesamiento que permiten, con una infraestructura de hardware adecuada, obtener mejores tiempos de ejecución, así como escalar la cantidad de partículas que forman parte del sistema simulado. De todas formas hay situaciones en las cuales la capacidad real de cómputo que se puede conseguir para hacer frente a un problema, se ve limitada por razones tanto teóricas (complejidad de algoritmos) como prácticas (hardware disponible). En estos casos surge como una alternativa realizar aproximaciones cuando el efecto de considerar cada partícula por separado o una aproximación de ellas, es un error acotado que se puede asumir.

2.2. Algoritmo evolutivo

En esta sección se introduce el concepto de algoritmo evolutivo. Con el advenimiento de la informática surgieron problemas computacionales y el concepto de complejidad computacional comenzó a limitar los problemas que se lograban atacar utilizando herramientas determinísticas de resolución. Se clasificó entonces las clases de complejidad en los cuales un problema podía caer. Definiéndose clase P y clase NP. Un problema pertenece a la clase P si puede ser resuelto en un tiempo polinómico en una computadora determinística. Ejemplos de estos problemas son el ordenamiento, búsqueda binaria, multiplicación matricial, etc. Un problema pertenece a la clase NP si puede ser resuelto en un tiempo polinómico, pero usando una computadora no determinística. Son la clase de problemas para los cuales es difícil encontrar una solución, pero una vez hallada, es fácil verificarla.

Dentro del conjunto de problemas NP se pueden encontrar comúnmente problemas de optimización. Estos problemas resultan fáciles de resolver cuando el espacio de direcciones a tomar en consideración es reducido, pero aumenta su complejidad conforme aumenta el espacio de valores a tomar en cuenta. Un típico ejemplo de estos problemas es el problema de viajero (TSP por sus siglas en inglés, Travelling Salesman Problem), que se define como encontrar una permutación que represente el recorrido de una serie de ciudades de tal forma que todas sean visitadas (sólo una vez) minimizando la distancia total viajada.

En el transcurso del proyecto se utilizaron técnicas de computación evolutiva para optimizar la compactación de aglomerados de partículas.

Dado que la complejidad asociada a los problemas de clase NP crece a medida que se incrementa el tamaño del problema, se hace necesario utilizar algoritmos de resolución más eficientes que la búsqueda exhaustiva. En estos casos es cuando se vuelve útil explorar otras técnicas de resolución, como ser: 1. métodos de aproximación (encuentran una solución buena con un error conocido), 2. métodos aleatorios (con cierta probabilidad, encuentran una solución con un error máximo dado), 3. técnicas heurísticas (técnicas basadas en procesamientos conceptualmente simples encuentran soluciones de buena calidad a problemas difíciles de un modo sencillo y eficiente) y 4. meta-heurísticas (se definen como heurísticas de nivel más alto de abstracción, definen estrategias que guían el proceso de búsqueda, incluyendo en general heurísticas subordinadas).

Dentro del gran conjunto de meta-heurísticas existen las de computación evolutiva, estas técnicas trabajan sobre una población de soluciones que evoluciona mediante mecanismo de selección y construcción de soluciones candidatas por recombinación de características de las soluciones anteriores.

Estas técnicas trabajan en las siguientes etapas: 1. evaluación de la función de fitness (en esta etapa se evalúa la calidad de las soluciones), 2. selección de individuos adecuados (de acuerdo al fitness calculado), 3. aplicación de operadores evolutivos y 4. reemplazo o recambio generacional.

2.3. Auto gravedad

En esta sección se introduce el concepto de auto-gravedad. La gravedad es una de las cuatro interacciones fundamentales observadas en la naturaleza. Origina la aceleración que experimenta un cuerpo físico en las cercanías de un objeto astronómico. También se denomina interacción gravitatoria o gravitación. La auto-gravedad es una fuerza que intenta comprimir un objeto hacia un punto en su centro de gravedad. Este concepto de auto-gravedad puede aplicarse tanto a nivel de partículas como electrones en un nivel cuántico, como a objetos de mayor escala como la tierra o asteroides en un nivel clásico.

Si se considera un aglomerado de partículas y se lo contextualiza en un ambiente de baja gravedad (alejado de cualquier fuente de gravedad), se puede intuir que la auto-gravedad ejercida por las propias partículas que conforman el sistema cobra relevancia en el comportamiento del medio granular. Si además se pretende estudiar el comportamiento de un medio granular por la acción de objetos externos (como por ejemplo choques de proyectiles o influencia de fuerzas a distancia) resulta imprescindible conocer el comportamiento del sistema en su conjunto.

Se sabe que se puede considerar por ejemplo ciertos asteroides como medios granulares, es decir conformados por un conjunto variado de partículas, las cuales se mantienen unidas por efecto de la auto-gravedad entre ellas. Dado que los asteroides se encuentran mayormente alejados de grandes objetos gravitacionales, la auto-gravedad existente cobra relevancia en su comportamiento.

Se plantea a continuación el problema formalmente:
Considérense N esferas de radios diferentes, con una distribución en un cierto rango de tamaños.

Dado este conjunto de N esferas en un volumen cúbico, se solapa el mismo con una grilla definida, y sobre esta grilla se procede al cálculo del potencial gravitatorio. Sobre cada nodo de la grilla se calcula el potencial ejercido por las N partículas, y este proceso se repite por cada nodo de la grilla.

El potencial calculado sobre un nodo de coordenadas (l, m, n) de la grilla tiene la forma:

$$V(l,m,n) = \sum_{n=1} V(i) = \sum_{i} \frac{GM_i}{\|R_0(i) - R(l,m,n)\|}$$

Donde M_i es la masa de la partícula i, G la constante de la Gravitación Universal, $R_0(i)$ es la posición de la partícula i y $R(l,m,n)$ la posición del punto (l,m,n) de la grilla. El denominador es la norma de la resta de los vectores posición $R_0(i)$ y $R(l,m,n)$.

2.4. Computación paralela

La computación paralela es una forma de cómputo en la que muchas instrucciones se ejecutan simultáneamente, operando sobre el principio de que problemas grandes, a menudo se pueden dividir en unos más pequeños, que luego son resueltos simultáneamente (en paralelo). Los programas informáticos paralelos son más difíciles de escribir que los secuenciales, porque la concurrencia introduce nuevos tipos de errores de software, siendo las condiciones de carrera los más comunes. La comunicación y sincronización entre diferentes subtareas son algunos de los mayores obstáculos para obtener un buen rendimiento del programa paralelo.

La computación en paralelo utiliza simultáneamente múltiples elementos de procesamiento para resolver un problema. Esto se logra mediante la división del problema en partes independientes de modo que cada elemento de procesamiento pueda ejecutar su parte del algoritmo de manera simultánea con los otros. Los elementos de procesamiento son diversos e incluyen recursos tales como una computadora con múltiples procesadores, varios ordenadores en red, hardware especializado, o cualquier combinación de los anteriores.

En el contexto de la computación paralela, se denomina proceso a una unidad de actividad que se caracteriza por la ejecución de una secuencia de instrucciones, un estado actual, y un conjunto de recursos del sistema asociados. Cada proceso tiene su contador de programa, registros y variables, aislados de otros procesos, incluso siendo el mismo programa en ejecución 2 veces. Cuando este último caso sucede, el sistema operativo usa la misma región de memoria de código, debido a que dicho código no cambiará, a menos que se ejecute una versión distinta del programa.

Los procesos son gestionados por el sistema operativo y están formados por:

- Las instrucciones de un programa destinadas a ser ejecutadas por el microprocesador.
- Su estado de ejecución en un momento dado, esto es, los valores de los registros de la unidad central de procesamiento para dicho programa.
- Su memoria de trabajo (memoria crítica), es decir, la memoria que ha reservado y sus contenidos.
- Otra información que permite al sistema operativo su planificación.

Esta definición varía ligeramente en el caso de sistemas operativos multihilo, donde un proceso consta de uno o más hilos, la memoria de trabajo (compartida por todos los hilos) y la información de planificación. Cada hilo consta de instrucciones y estado de ejecución.

Formalmente un hilo de ejecución, hebra o subproceso es la unidad de procesamiento más pequeña que puede ser planificada por un sistema operativo. La creación de un nuevo hilo es una característica que permite a una aplicación realizar varias tareas a la vez (concurrentemente). Los distintos hilos de ejecución comparten una serie de recursos tales como el espacio de memoria, los archivos abiertos, situación de autenticación, etc. Esta técnica permite simplificar el diseño de una aplicación que debe llevar a cabo distintas funciones simultáneamente.

Dado un problema complejo a resolver, los modelos de programación paralelo determinan la forma en cómo subdividir el problema inicial en subproblemas de menor complejidad los cuales se puedan resolver de forma independiente. La identificación de la división del problema original en subtareas y la posterior recomposición de la solución a partir de la resolución de los sub-problemas resulta crucial.
Existen dos estrategias para dividir el problema original en subproblemas:

Descomposición por dominio Técnica enfocada en dividir los datos manejados por el programa en (preferentemente) partes disjuntas, a ser procesadas en paralelo por diferentes unidades de procesamiento.

Descomposición funcional Técnica basada en identificar unidades funcionales que realizan diferentes tareas en el programa, esas tareas son entonces ejecutadas en paralelo en diferentes unidades de procesamiento.

Ambas técnicas dependen en comunicación y sincronización a través de un recurso compartido (usualmente una memoria compartida) o pasaje explícito de mensajes, para implementar la cooperación que permite resolver los subproblemas e integrarlos para construir la solución original.

Luego de aplicadas técnicas de HPC en la resolución de un problema, resulta muy importante evaluar la performance obtenida por los cambios introducidos. Generalmente comparando los tiempos de ejecución de nuestra solución paralela con los de una solución secuencial, o con una estimación, podemos saber si nos lleva menos tiempo, pero también es de interés, saber qué tanto mejora nuestra solución al agregar recursos de cómputo. Es decir, se necesita conocer como escala nuestra solución tanto para saber qué tan capaces somos de resolver el problema más rápido, como de resolver problemas más complejos.
A continuación, se explican algunas de las métricas que se utilizan habitualmente para evaluar los resultados:

Speedup Es una medida de la mejora del rendimiento (performance) de un algoritmo paralelo al aumentar la cantidad de procesadores (comparado con el rendimiento al utilizar un solo procesador). Según cómo se interpreten ejecución serial y paralela, se puede arribar a distintas variantes del speedup, en nuestro caso comentaremos sólo dos:

Speedup absoluto Se compara el tiempo de ejecución de un algoritmo paralelo utilizando N procesadores contra el tiempo de ejecución del mejor algoritmo secuencial. El speedup absoluto SABS(N) se define como SABS(N) = T / TN donde T es el tiempo que toma al mejor algoritmo secuencial en resolver el problema, "N" es la cantidad de procesadores utilizados y TN el tiempo que le toma a nuestro algoritmo paralelo en resolver el problema con N procesadores. Esta no es muy utilizada porque normalmente no se conoce el mejor algoritmo secuencial que resuelve el problema.

Speedup relativo o algorítmico Conceptualmente similar al speedup absoluto, pero en vez de comparar con el tiempo que le lleva al mejor algoritmo secuencial, se compara con el tiempo de ejecución en un procesador serial. Se calcula como S(N) = T1 x TN siendo T1 el tiempo de ejecución en un procesador serial. Al igual que para el speedup absoluto no significa nada si no viene acompañado por el número de procesadores para el que fue calculada.

Eficiencia computacional Es una medida relativa que permite comparar el desempeño de un algoritmo paralelo en diferentes entornos, consiste en dividir el speedup por la cantidad de procesadores. La eficiencia computacional E(N) se define como E(N) = T(1) / N x T(N). Corresponde a un valor normalizado del speedup (entre 0 y 1), respecto a la cantidad de procesadores utilizados. Valores cercanos al uno indicarán en general situaciones casi ideales de mejora de performance.

Paralelicibilidad Es una medida para evaluar el desempeño de un algoritmo paralelo, que consiste en comparar el tiempo de ejecución de este en un computador paralelo utilizando N procesadores, con el tiempo de ejecución del mismo algoritmo utilizando el mismo computador, pero solo un procesador. Se calcula como P = TP1 / TPN, donde TP1 el tiempo que toma a un computador paralelo ejecutar un algoritmo paralelo en un único procesador y TPN el tiempo que toma al mismo computador paralelo ejecutar el mismo algoritmo paralelo en N procesadores.
La paralelicibilidad y el speedup si bien similares, miden aspectos diferentes del algoritmo paralelo. El speedup considera el tiempo de un algoritmo secuencial (el mejor existente o conocido) para la comparación, mientras que la paralelicibilidad toma en cuenta el tiempo de ejecución de un algoritmo paralelo en un único procesador. El speedup evalúa la mejora de desempeño al utilizar técnicas de programación paralela. La paralelicibilidad da una medida de cuán paralelizable o escalable resulta el algoritmo paralelo utilizado

2.5. Computación de alto rendimiento

Cada vez se hace más difícil y a la vez más importante, el poder satisfacer los requisitos crecientes de poder de cómputo. Esto se debe a que cada vez más los informáticos están atacando problemas más complejos, aplicando modelos más complejos, analizando volúmenes de datos más grandes o necesitando una capacidad de respuesta en tiempo limitado para nuevos problemas (sistemas de tiempo real y semejantes).

El objetivo de este proyecto es desarrollar una herramienta que permite atacar problemas con volúmenes de datos y complejidad matemática considerable, utilizando técnicas informáticas conocidas como 'Computación de Alto Rendimiento' las cuales consisten en la aplicación de una familia de técnicas, que se utilizan para lograr tareas concurrentes, con el fin de reducir el tiempo de procesamiento. Se debe contar con una arquitectura de hardware paralelo que permita el procesamiento simultáneo de tareas. La cantidad requerida de circuitos para el procesamiento paralelo es mayor que en el procesamiento serial, pero en la actualidad esto no es un problema, ya que los costos de hardware son relativamente bajos.

Este tipo de infraestructura de cómputo compuesta por agregación de recursos de cómputo, potencialmente heterogéneos, conectados a través de una red, son las computadoras más potentes de hoy en da y se llaman clusters ó grids. En general por clúster se denota a recursos de cómputo integrados en un mismo lugar físico, mientras que el término grid en general se refiere a la integración de recursos de más de un dominio de administración (i.e. de más de una institución), potencialmente con distribución geográfica. Existe un ranking de las 500 computadoras más potentes del mundo en cuanto a capacidad de procesamiento (en adelante top500) que anualmente publica dicho ranking en el sitio www.top500.org. El top500 hoy en día se encuentra dominado por los clústeres, tanto homogéneos como heterogéneos, muchos de estos incluyendo procesadores gráficos. También existen otros dos rankings que reordenan las mismas 500 computadoras del top500, pero en vez de hacerlo por capacidad de cómputo lo hacen por consumo de energía (www.green500.org), y capacidad de procesamiento intensivo en datos (www.graph500.org).

Las infraestructuras de cómputo con arquitecturas de múltiples procesadores se dividen en dos grandes grupos: las de múltiples procesadores en memoria compartida; o las de múltiples procesadores en memoria distribuida, o sea, múltiples computadoras con uno o más procesadores en memoria compartida, conectadas a través de una red. Las arquitecturas de múltiples procesadores en memoria compartida, pueden dividirse dependiendo de si estos procesadores están integrados en el mismo chip, en cuyo caso comparten además de la memoria principal de la computadora, memoria más cercana al procesador llamada caché. O si estos se encuentran en chips separados, compartiendo únicamente la memoria principal. También pueden dividirse dependiendo de si todos los procesadores acceden a la memoria principal de manera uniforme, o no, en cuyo caso la memoria principal está dividida en regiones que se corresponden a procesadores, un procesador accede de forma directa a una región, pero para acceder a las demás debe comunicarse con el procesador correspondiente.

En las arquitecturas actuales en general se tiene múltiples chips con múltiples procesadores, y en estos los procesadores en un mismo chip acceden de forma uniforme a memoria, mientras que aquellos en chips diferentes acceden de forma no uniforme. En general en las infraestructuras de cómputo de alto desempeño actuales, como la mayoría de las que figuran en el top500, son sistemas en memoria distribuida, compuestos por nodos de múltiples procesadores en memoria compartida. Para sacar provecho de dichas infraestructuras hacen falta técnicas o paradigmas de computación, que para el caso de múltiples procesadores en memoria compartida se suelen llamar técnicas de programación paralela, y para el caso de memoria distribuida, de programación distribuida.

Un clúster, tal como se mencionó anteriormente, es una supercomputadora construida por agregación de recursos de computo conectados en una red de alta velocidad, dispuestos cercanos entre sí para optimizar las comunicaciones por red. La capacidad de computo de un clúster estará dada entonces tanto por la capacidad de computo de sus procesadores, como por la velocidad de la red de comunicaciones que lo integra.

En este proyecto se utilizo el Cluster FING. El clúster FING es una infraestructura de computo paralelo distribuido de alto desempeño perteneciente a la Facultad de Ingeniería.

2.6. Interpolación

La interpolación es un proceso por el cual se define un valor en un punto cualquiera a partir de los valores conocidos en algunos puntos dados. En general, tenemos un conjunto finito de nodos o puntos fijos con valores nodales asociados a cada uno de ellos: (P_i, V_i), donde $Pi = x_i, y_i, z_i)$ es la posición del i-ésimo punto y Vi el valor conocido en el punto fijo; se pretende encontrar el valor v asociado a un punto de coordenadas x cualquiera. En el contexto del proyecto, resulta necesario utilizar una técnica de interpolación para poder conocer con mejor detalle cómo se distribuye el dominio solución para el cual se aplicó el algoritmo de cálculo de potencial. Resulta más performante aplicar el cálculo de potencial sobre una malla gruesa y luego una técnica de interpolación apropiada sobre una malla una poco más fina, que aplicar el cálculo de potencial sobre una malla fina desde el principio. El proceso de interpolación define el valor asociado al punto variable. Podría, por ejemplo, asignarse el valor del punto más cercano o una combinación de valores cercanos o de todos los conocidos. En una sola dimensión es muy sencillo encontrar el punto más cercano o los dos nodos que "encierran" el punto buscado, pero en más dimensiones resulta más complicado.

La forma estándar, para variables continuas, consiste en hacer una suma de valores ponderados: $V(x) = \sum \alpha_i(x)V_i$. El valor v en el punto variable x se calcula asignando, a cada punto fijo, un peso α_i que depende de la posición del punto variable. Normalmente se pretende que el peso de cada punto esté en relación inversa con la distancia al punto móvil, de ese modo el valor estará más influenciado por los puntos más cercanos. Pero hay muchas formas de definir los pesos. Es justamente, la selección de pesos la que define el método de interpolación.

Una clasificación posible para los métodos de interpolación es en métodos locales y métodos globales: los métodos globales consideran todos los nodos. Para los métodos locales, sólo se consideran los nodos cercanos. En general se utilizan estos últimos métodos pues son algorítmicamente más veloces, sobre todo cuando se dispone de un método rápido para encontrar el conjunto de nodos cercanos.

Debido a que en un principio no se conocía como se iban a comportar las distintas técnicas de interpolación, se estudiaron y luego aplicaron distintas técnicas posibles, entre las cuales las siguientes se destacan:

- Nearest neighbor
- Ponderación lineal
- Kriging

- Inverse distance weighting (IDW)

A continuación, una descripción general de cada técnica y un breve resumen de su aplicación y/o resultados esperados:

Nearest neighbor El algoritmo del vecino más cercano selecciona el valor del punto más cercano y no tiene en cuenta los valores de otros puntos vecinos cercanos, produciendo un interpolante constante por trozos. El algoritmo es muy simple de implementar y se utilizó como referencia base.

Ponderación lineal sobre la distancia La ponderación lineal es un método que permite dado un conjunto de puntos y sus valores asociados, realizar estimaciones respecto a puntos donde se desconoce su valor asociado a través de una combinación lineal de los valores de los puntos conocidos. Además, se caracteriza por obtener el valor buscado a través de una asignación de pesos a los puntos conocidos. Este método toma la forma de:

$$\sum_{i=0}^{N} \frac{w_i u_i}{\sum_{j=0}^{N} u_j(x)}$$

donde $W_i(x)$ representan los valores de los puntos conocidos, y u_i representa la ponderación asignada. Dado que estamos queriendo interpolar una magnitud que varía con la distancia al punto a interpolar, resulta intuitivo que los valores ponderadores se correspondan con la distancia entre el punto a interpolar al punto conocido correspondiente, por lo que entonces u_i representaría la distancia entre el punto w_i conocido al punto a interpolar.

Kriging El método de Kriging es un método geoestadístico de estimación de puntos. Utiliza un modelo de variograma para la obtención de los ponderadores que se darán a cada punto de referencias usados en la estimación. Esta técnica de interpolación se basa en la premisa de que la variación espacial continúa con el mismo patrón. Fue desarrollada inicialmente por Danie G. Krige a partir del análisis de regresión entre muestras y bloques de mena, las cuales fijaron la base de la geoestadística lineal.

La precisión del estimador usado depende de varios factores:

- *el número de muestras tomadas*
- *la calidad de la medición en cada punto*
- *las ubicaciones de las muestras en la zona:* si las muestras son igualmente espaciadas se alcanza una mejor cobertura, dando mayor información acerca de la zona que aquella que se obtendría de muestras muy agrupadas en unos sectores y separadas en otros. Sin embargo, en la práctica, debido a las características de las regiones de estudio, muchas veces es preciso tomar muestras irregularmente espaciadas.

- *las distancias entre las muestras:* para la predicción es más confiable usar muestras vecinas que muestras distantes, esto es, la precisión mejora cuando la cercanía de las muestras aumenta, y se deteriora cuando esta disminuye.
- *la continuidad espacial de la variable o atributo en estudio:* es más fácil estimar el valor de una variable bastante regular en una región que una que presenta grandes fluctuaciones.

Existen distintos tipos de algoritmos categorizados como Kriging. Por un lado tenemos el kriging simple, que asume que las medias locales son relativamente constantes y de valor muy semejante a la media de la población que es conocida. La media de la población es utilizada para cada estimación local, en conjunto con los puntos vecinos establecidos como necesarios para la estimación. Existe además el kriging ordinario, en el que las medias locales no son necesariamente próximas de la media de la población, usándose apenas los puntos vecinos para la estimación. Es el método más ampliamente utilizado en los problemas ambientales.

Ponderación de distancia inversa Ponderación de Distancia Inversa o IDW por su sigla en inglés, es un tipo de método determinista utilizado para interpolaciones multivariables con un conjunto disperso de puntos conocidos. Los valores asignados a los puntos desconocidos son calculando como el promedio ponderado de los puntos conocidos disponibles. El nombre dado a este método es motivado por el promedio ponderado aplicado, puesto que dicha ponderación es calculada usando como base el inverso de la distancia a cada punto conocido. El resultado esperado es una asignación discreta de la función desconocido u en la región D de estudio:

$$u(x) : x \to \mathbb{R}, x \in D \subset \mathbb{R}^n$$

El conjunto de los N puntos conocidos y sus valores asociados se describe como una lista de tuplas:

$$[(X_1, u_1), (x_2, u_2), ..., (X_N, u_N)]$$

La fórmula general para encontrar los valores interpolados u en los puntos desconocidos $x : u_i = u(x_i)$: para $i = 0, 1, ..., N$ usando IDW es una función interpoladora tal como:

$$u(x) = \sum_{i=0}^{N} \frac{w_i(x)u_i}{\sum_{j=0}^{N} w_j(x)} i$$

Donde los pesos w_i se calculan como:

$$w_i(x) = \frac{1}{d(x, x_i)^p}$$

En esta ecuación, d es la distancia entre el punto conocido x_i al punto desconocido x, N es el número total de puntos conocidos y usados en la interpolación y p es un número real positivo, llamado parámetro de potencia.

En este caso, los pesos asignados disminuyen su valor a medida que la distancia a los puntos conocidos aumenta. Además, valores más grandes de p asigna mayor influencia a los puntos que se encuentran más cercanos al punto que se desea interpolar. Para dos dimensiones, un parámetro de potencia $p <= 2$, causa que la interpolación sea dominada por puntos lejanos. En nuestro contexto, utilizaremos diferentes configuraciones de p hasta encontrar aquella interpolación que se ajusta mejor a la magnitud interpolada que nos compete.

2.7. ESyS-Particle

Cuando la cantidad de datos que se obtienen como resultado de un algoritmo numérico es tan grande que analizar los datos de forma individual resulta inmanejable, se necesitan herramientas que permitan visualizar los resultados de forma gráfica. De esta forma una vez que se obtienen los datos crudos, se genera una representación gráfica o simulación mediante la cual resulta más intuitivo realizar conclusiones acerca de la solución alcanzada y potenciales mejoras. Además, cuando se maneja gran cantidad de datos, las herramientas de simulación a utilizar deben aprovechar la capacidad de procesamiento paralelo que los nuevos hardwares ofrecen.

Para cubrir esta falencia, se utiliza la herramienta o paquete de simulación numérica denominado ESyS-Particle. Teniendo presente el análisis realizado en el proyecto de grado “Paralelismo aplicado al estudio de medios granulares”, se asume como propia la conclusión a la que allí se arriba, en la cual se define ésta como la herramienta que mejores prestaciones brinda (considerando el contexto en el que se trabaja).

ESyS-Particle es un paquete de software para modelados numéricos basados en partículas. El software implementa el DEM, una técnica ampliamente utilizada para el modelado de procesos que involucran grandes deformaciones, flujos granulares y /o fragmentación. Las características relevantes de la herramienta son:

- Motor de simulación paralelo basado en MPI
- API en Python para la preparación y ejecución de simulaciones.
- Preparación mediante scripts de modelos geométricos
- Partículas esféricas rotacionales y no rotacionales

- Mallas triangulares para especificación de condiciones de borde y pared.
- Visualización de partículas utilizando POV-Ray y VTK.
- Variedad de interacciones partícula-partícula y partícula-pared:

  Repulsión elástica lineal entre partículas no enlazadas en contacto.

  Repulsión elástica lineal entre pares de partículas enlazadas

  Interacciones friccionales tanto rotacionales como no rotacionales entre partículas no enlazadas.

  Enlaces rotacionales implementando torsión y coeficientes de rigidez.

ESyS-Particle ha sido desarrollado dentro del Earth Systems Science Computational Centre (ESSCC) en la Universidad de Queensland, Brisbane, Australia desde 1994. El software fue denominado en sus comienzos como Lattice Solid Model, y más tarde se llamó LSMEarth antes de que Australian Computational Earth Systems Simulator (ACcESS) comenzara el financiamiento del desarrollo del software en el año 2002. Desde 2002 al 2007 y gracias al financiamiento económico el desarrollo ha avanzado gratuitamente hasta convertirse en un software DEM de uso comercial, distribuido gratuitamente bajo la licencia de software libre versión 3. Al día de hoy el desarrollo de ESyS-Particule continúa siendo financiado por el gobierno australiano, contando con la colaboración de desarrolladores de la universidad de RWTH en Aachen, Alemania, y de varios miembros de la comunidad de usuarios (incluyendo recientes incorporaciones por grupos de la Facultad de Ingeniería – Universidad de la República).

ESyS-Particle ha sido utilizado para simular terremotos, flujo de granos en silos, fragmentación de rocas, apertura de fallas, etc.

ESyS-Particle está diseñado para ejecutar sobre supercomputadoras paralelas, clusters o computadores con varios núcleos corriendo Linux como sistema operativo base. El motor de simulación en C++ implementa una descomposición espacial del dominio mediante el uso de Message Passing Interface (MPI). Una API en Python provee flexibilidad en el diseño de modelos numéricos, en la especificación de parámetros del modelo e interacciones de contacto entre las partículas; así como interfaces con diferentes plataformas de visualización de los resultados de las simulaciones.

ESyS-Particle es una implementación paralela del DEM, que se caracteriza por ser OpenSource, está disponible gratuitamente, y puede ser modificada y extendida libremente. Se basa en una arquitectura de memoria distribuida utilizando MPI. Mientras que el tamaño del problema sea escalado adecuadamente con la cantidad de procesadores, el beneficio de agregar procesadores es casi lineal. Soporta varios millones de partículas en las simulaciones; en pruebas realizadas por los propios desarrolladores de la herramienta, llegaron a utilizar más de 10 millones de partículas.

Este paquete está diseñado para trabajar con otros programas individuales, que se encargan del pre-procesamiento de los datos que son la entrada de la simulación y post-procesamiento de los resultados de la misma. Esto posibilita que sea más fácil el desarrollo y mantenimiento de las distintas herramientas, en lugar de un paquete que incluya todo. Las funcionalidades de ESyS-Particle se manejan mediante scripts en Python y no posee un GUI interactiva. El uso de Python como lenguaje de scripting proporciona una programación orientada a objetos, fácil de utilizar, que posee una extensa librería estándar, y que además ya existe una implementación de la interfaz entre Python y el motor en C++ de ESyS-Particle (mediante la librería Boost-Python).

ESyS-Particle está compilado como una librería compartida y es utilizada como un módulo de Python, los scripts importan las librerías como mecanismo para hacer accesibles las funciones exportadas por ESyS-Particle. Las funciones de los scripts llaman a las funciones de C++ para inicializar el modelo, preparar las interacciones, mover los objetos, salvar datos, etc.

El proceso de simulación por pasos se describe gráficamente en la figura 2.1.

Lo pasos de la simulación se dividen entonces en:

- inicialización de la simulación
- actualización de la tabla de partículas vecinas
- cálculo de fuerzas
- actualización de posición y velocidad de las partículas
- escritura de datos en flujo de salida
- si existen pasos de simulación volver al punto 2, en caso contrario se termina la simulación

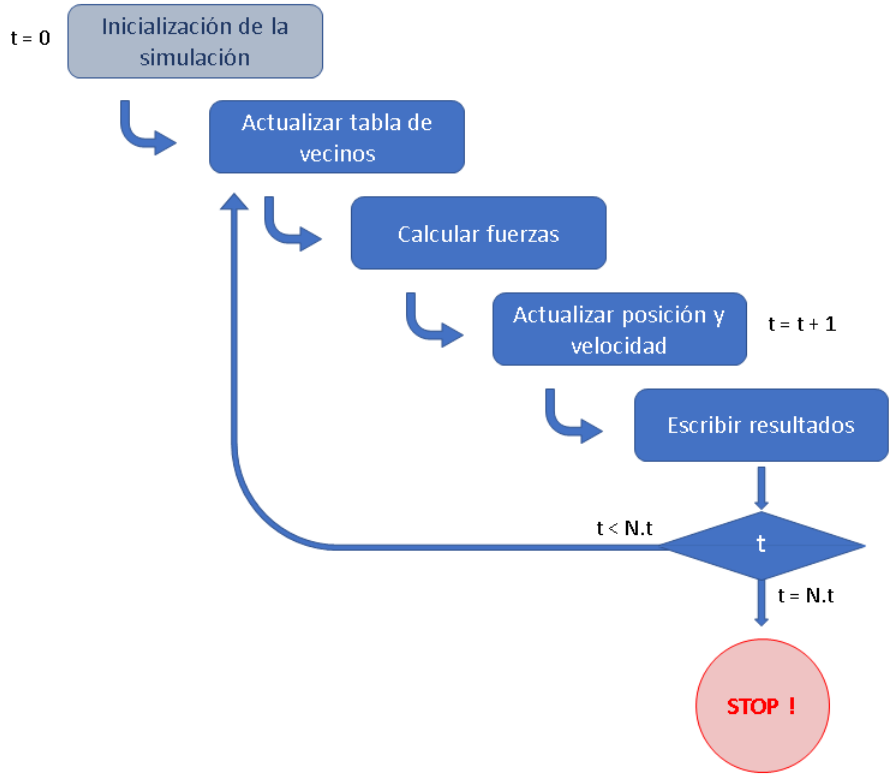


Figura 2.1: Proceso de simulación por pasos del ESyS-Particle.

Una particularidad de los DEM aplicado a la interacción de partículas por contacto, es que las fuerzas son de corto alcance, es decir que una partícula sólo interactúa con las partículas más cercanas. ESyS-Particle toma en cuenta esta peculiaridad logrando reducir los costos computacionales de simulación. En su implementación interna utiliza para cada partícula, una lista de potenciales partículas vecinas a las cuales afecta (ver figura 2.2). Dicha lista no necesita ser re-calculada en cada paso de simulación sino cuando alguna partícula se mueve más que un valor Dv (verlet distance) que dependerá de cada simulación.

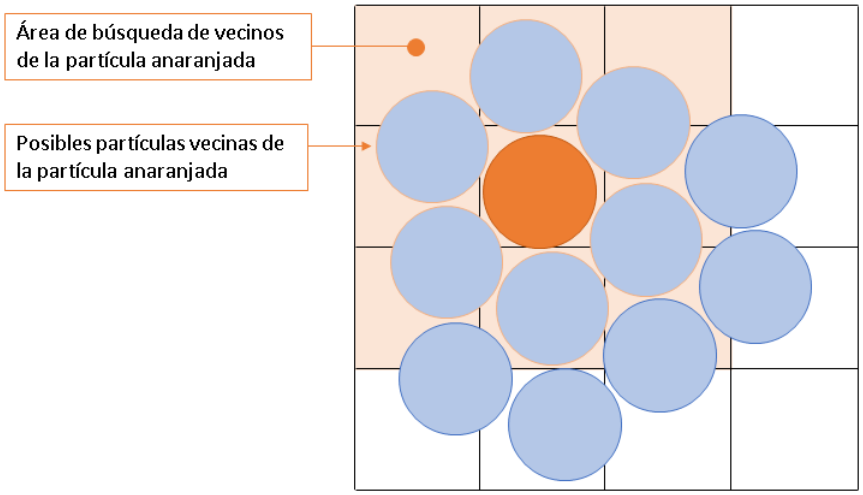


Figura 2.2: Búsqueda de partículas vecinas por ESys-Particle.

ESySParticle puede hacer uso de más de un recurso de cómputo en la computación de la simulación. A continuación se explica el procedimiento de asignación: dado un dominio de simulación con partículas como el de la imagen a) de la figura 2.3, ESySParticle implementa una descomposición de dominio espacial para dividir el área de trabajo entre los recursos de cómputo disponibles. El dominio inicial es subdividido en N_w sub-dominios ordenados en forma de matriz (x, y, z) como se muestra en la imagen b) de la misma figura. Dado que las partículas muy cercanas a los límites tendrán efectos en las partículas del subdominio contiguo, cada recurso de cómputo recibe su subdominio y un buffer alrededor que incluye dichas partículas bordes tal como se muestra en la imagen c) en la misma figura.

De esta forma cada recurso de cómputo actualiza la posición y velocidad de las partículas de su subdominio estricto utilizando las partículas del subdominio con el buffer agregado. Para obtener una escalabilidad decente se requiere que los subdominios estrictos sean grandes en comparación con los subdominios con el buffer . Al final de cada paso de simulación las partículas de los bordes son comunicados de un recurso a otro.

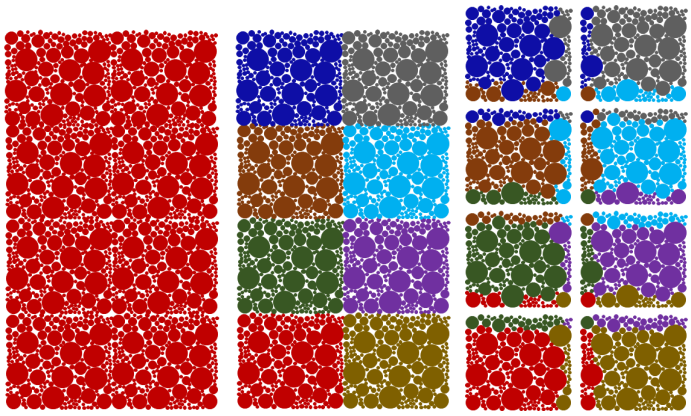


Figura 2.3: Descomposición por dominio de ESyS-Particle.

Capítulo 3

Antecedentes

En este capítulo se presentan los antecedentes del proyecto. El cálculo exacto del potencial gravitatorio generado por n partículas implica $O(n^2)$ operaciones, lo cual indica una complejidad computacional costosa cuando n es del orden de millones de partículas. Diversas estrategias se han desarrollado para reducir el número de operaciones.

The techniques presented are classified as static or dynamic domain decomposition techniques. Static hierarchical domain decomposition Static techniques for particle interaction are based on a domain decomposition that stays invariable during a simulation. According to Hockney and Eastwood (1988), static techniques are divided in three models: Particle-Particle (PP) methods, Particle-Mesh (PM) methods, and Particle-Particle Particle-Mesh (P3M) methods. PP are the simplest methods and consist of computing the forces directly between all the particles in the system. Despite being the most accurate, the PP model lacks the ability to scale in the number of particles due to its O(N2) execution time. PM methods consist of using a mesh over the simulated space and calculating the potential for each particle that belongs to the mesh. After that, the speed for the particles is calculated via an interpolation. According to Hockney and Eastwood (1988), although PM methods are less accurate than PP methods when computing the forces, PM methods are in general significantly faster than PP methods. However, in many practical applications, PM may need a mesh resolution that should result to be slower than a PP method; thus the PM model is not usually used to calculate contact forces. Finally, P3M methods combine the PP and the PM models. The P3M model divides the forces applied to a particle into short range and long range forces. The short range forces are calculated using a PP method and the long range forces are calculated using a PM method. The combination results in a fast and accurate method to simulate particle interaction. Many implementations and variations of the aforementioned three models are present in the literature. Some of the more relevant implementations are reviewed next. Couchman (1991) proposed a variation of the P3M algorithm, that applies a selective refinement of the grid depending on the particle density of a cell. The refinement is performed recursively while a density threshold is exceeded. 8 Given that the refinement is performed recursively, the particle mesh is an spatial division without overlapping zones to be calculated. This way, assigning a similar number of particles to each processing unit, a load balancing scheme is implemented. The results obtained after the refinements are then passed over to the father cell to be integrated via direct summation. There is a correlation between the level of refinement of the grid and the time spent to integrate the results calculated for the individual cells. The refinement level has to be calibrated for this method to be efficient. The method presented by Couchman has the constraint that the domain of the simulation must be cubic and also the space must be divided into an integer number of cells of the same size. Couchman stated that, in this method "The gain is in simplicity"(p. 24). Results presented by Couchman indicate that the algorithm is up to 20 times faster than a P3M algorithm and also requires less memory. Kravtsov et al. (1997) presented an N-body solver called Adaptive Refinement Tree based on the particle-mesh method over a multilevel grid to perform the calculations of the forces on the system. The mesh is created over a cubic space that is divided by regular cells with cubic shape and a predefined size. A multilevel mesh is defined by dividing large cells into eight parts that have the same size depending on the particle density. The multilevel mesh is created at the beginning of the simulation and then it is partially updated when the forces need to be recalculated. The smallest size of an element of the grid is the resolution of the mesh (i.e., a cell). The implementation presented by Kravstov et al. was partially parallel. The section of the application that was parallelized is the one in which the forces are updated for the particles (i.e., the force interpolation). Kravstov et al. stated that the solution is half as fast as the

Una estrategia es sustituir el valor exacto del potencial gravitatorio por una función suave de la posición. Se han utilizado armónicos esféricos y biesféricos como funciones de ajuste [6]. Cada partícula se propaga en este campo de fondo durante un breve periodo de tiempo antes de que se vuelva a calcular el ajuste. Este método requiere operaciones del orden de $O(n.log(n))$, pero su desventaja es la pérdida de precisión. Otro enfoque propone el uso de un método de agrupación jerárquica. Appel[7] utilizó una estructura de árbol para representar los sistemas de n-cuerpo, almacenando las partículas en las hojas y agrupando partículas cercanas como hojas de ramas cercanas. Una implementación eficiente de un enfoque de árbol jerárquico completo fue presentado por Barnes y Hut[8], y posteriormente incorporado en un algoritmo de integración por Richardson[9] para simular la evolución dinámica de planetesimales y la evolución rotacional y colisional de asteroides. Este enfoque también requiere del orden de $O(n.log(n))$ operaciones. Usando estructuras paralelas fueron capaces de simular hasta millones de partículas que interactúan[10]; pero el paso de tiempo utilizado (5s) era demasiado grande para simular adecuadamente las interacciones mecánicas de las partículas blandas (DEM simulaciones por lo general requieren timesteps de 10^{-2}s [11]). Por lo tanto, si el timestep se reduce al valor requerido por las simulaciones DEM, el cálculo de la auto-gravedad sigue siendo costoso.

Sánchez y Scheeres[12] utilizaron un enfoque ligeramente diferente: mientras que en un código de árbol la subdivisión del espacio es dinámica y depende de la distribución espacial de las partículas, ellos utilizaron una cuadrícula regular y estática para dividir el espacio. Este enfoque parece ser más adecuado para los agregados granulares auto-gravitantes en una fase densa como un asteroide de escombros. Sin embargo, el código no fue paralelizado y las simulaciones se limitaron a unos pocos miles de partículas.

Para tratar de manera eficiente los aglomerados auto-gravitantes formados por un número mayor de partículas que las abordadas por los esfuerzos anteriores, nuestro algoritmo de cálculo incorpora elementos de todos ellos, incluida la sustitución de una función de ajuste liso por el valor exacto del potencial gravitatorio, un método de agrupación jerárquica para acelerar el cálculo de autogravedad de los nodos de cuadrícula y una cuadrícula regular y estática para dividir el espacio.

Capítulo 4

Desafio planteado

En este capítulo se presenta el problema a resolver.

4.1. Simulación de medios granulares bajo los efectos de fuerzas gravitatorias

El problema a resolver es el cálculo y simulación del comportamiento de un aglomerado de partículas bajo los efectos de fuerzas gravitatorias en ambientes de baja gravedad.

esto es parte central. necesita mas contenido

Para esto se debe calcular la auto-gravedad en cada partícula del aglomerado a pequeños intervalos de tiempo. Este procedimiento si bien es simple, es computacionalmente demasiado costoso cuando se tienen del orden de un millón de objetos. Debido a esto, se necesitan aplicar técnicas de computación de alto desempeño para disminuir el tiempo de cómputo y así permitir abordar simulaciones en tiempos decentes.

Para conseguir este objetivo se identifican las siguientes tareas a abordar: 1. calculo de auto-gravedad: dado un aglomerado de N partículas, desarrollar un algoritmo que permita calcular al auto-gravedad en cualquier punto del mismo, 2. algoritmo paralelo de auto-gravedad: inclusión de técnicas de computación de alto desempeño al algoritmo de cálculo de auto-gravedad anterior y 3. integración a un paquete de simulación: inclusión del algoritmo paralelo a un paquete de simulación numérica que permita visualizar los resultados.

4.2. Compactación de aglomerados

En el transcurso del proyecto se abordo la compactación de partículas esféricas como problema relacionado. Este problema si bien puede ser relativamente fácil de resolver si se consideran partículas perfectamente esféricas y de igual tamaño, crece exponencialmente su complejidad si se consideran partículas de distintos tamaños. El problema se puede pensar como un problema de optimización, donde se debe minimizar el espacio libre en la caja, es decir el espacio no ocupado por las partículas.

En la simulación numérica del comportamiento de medios granulares resulta de gran importancia la compactación y aleatoriedad que estos medios presentan. Técnicas utilizadas actualmente se basan en generar valores de radios de partículas (con una distribución de probabilidad dada) los cuales se ordenan de forma descendiente. Se realiza luego el llenado aleatorio de la caja comenzando por las partículas más grandes y terminando por las más pequeñas (verificando que ninguna partícula se superponga con las ya colocadas). Utilizando esta técnica se logra una compactación real no mayor a 0.45, la cual es considerada por debajo de lo deseado.

Debido a la importancia y calidad de las simulaciones numéricas que se desea obtener en este proyecto, se decide implementar un algoritmo propio de generación de aglomerados de partículas con una distribución de tamaños dada con alto grado de compactación.

Teniendo en cuenta que se toman en consideración del orden de un millón de partículas sobre un espacio tridimensional (cada partícula se toca con la siguiente en algún punto de su superficie y con los bordes del contenedor) este problema es de complejidad NP.

Capítulo 5

Diseño y arquitectura

En este capítulo se presenta el diseño y arquitectura de la solución.

5.1. Compactación de aglomerados

En esta sección se presentan las características del algoritmo evolutivo desarrollado para resolver el problema de compactación de aglomerados.

5.1.1. Representación

El algoritmo evolutivo diseñado considera a cada individuo como una caja de dimensiones conocidas. Cada individuo contiene un conjunto de partículas. Se generaliza entonces para entender el concepto de población a evolucionar como el conjunto de individuos (es decir de cajas) con cierta compactación definida de partículas.

Cada partícula generada ha de contener la información de posición dentro de la caja y su radio correspondiente. Además, cada individuo debe contener la información de todas las partículas y su fitness asociado.

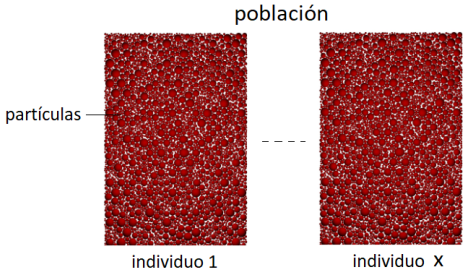


Figura 5.1: Representación de la población.

5.1.2. Población inicial

La población inicial se construye generando valores de radios de partículas (con una distribución de probabilidad dada) los cuales se ordenan de forma descendiente. Se realiza luego el llenado aleatorio de la caja comenzando por las partículas más grandes y terminando por las más pequeñas (verificando que ninguna partícula se superponga con las ya colocadas). Utilizando esta técnica se logra una compactación real no mayor a 0.45 de individuos.

Para ubicar cada partícula en la caja se generan tres números aleatorios (r_x, r_y, r_z) tal que cada valor corresponde a una coordenada de la caja. Los valores se generan cumpliendo una distribución uniforme de parámetros $[0, M]$, donde M representa la mayor dimensión de la caja para la coordenada a generar. Para cada individuo generado, se tiene:

- un conjunto de partículas del individuo
- un conjunto de partículas que no pertenecen al individuo puesto que no se pudieron ubicar en las dimensiones de la caja, pero que pertenecen a la distribución inicial. Este conjunto de partículas se llama stock, y se utiliza en el proceso de mutación.

Una vez generada una población inicial, comienza el proceso iterativo del algoritmo evolutivo.

5.1.3. Operadores

En esta sección se definen los operadores evolutivos involucrados.

Operador de selección Se implementaron dos operadores de selección: selección por torneo y selección por torneo.

Operador de cruzamiento El operador de cruzamiento consiste en dividir la caja horizontalmente e intercambiar las secciones entre diferentes individuos, como se indica en la figura 5.2.

Figura 5.2: Representación del operador de cruzamiento en una sección.

Inicialmente se propuso un operador de cruzamiento que divide a cada individuo en cinco secciones, como se representa en la figura 5.3. Sin embargo realizar tantas divisiones sobre el individuo para cruzar las parejas de padres degrada el fitness, impidiendo al algoritmo evolutivo de convergir. Para evitar ello, se decidió dividir cada individuo en solo una sección horizontal, obteniendo dos partes (superior e inferior).

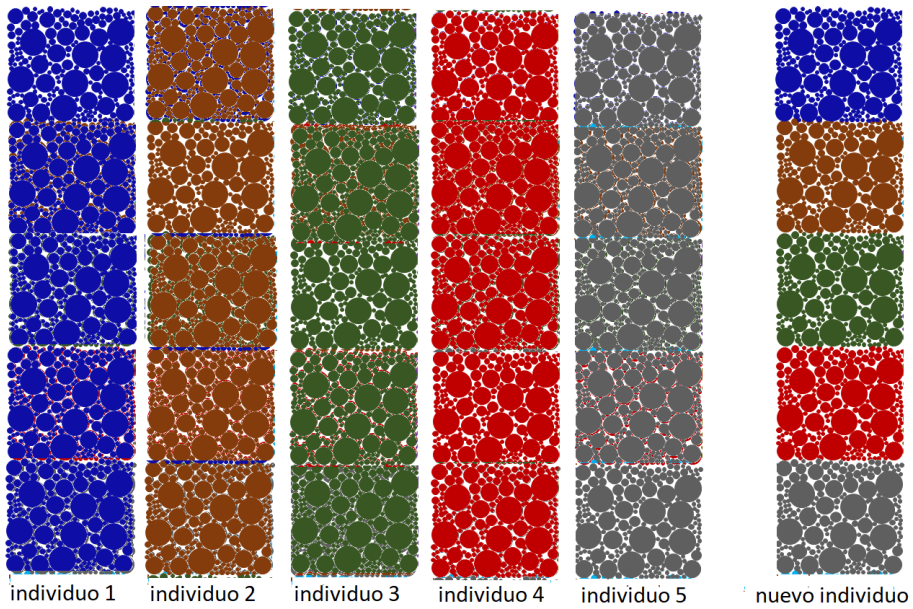


Figura 5.3: Representación del operador de cruzamiento en múltiples secciones.

Operador de mutación El operador de mutación implementado resulta de aplicar las siguientes transformaciones: 1. mover la partícula y 2. intercambiar partículas con el stock de partículas disponibles. El operador de mutación consiste en los siguientes pasos: Para una cantidad de veces i:

1. Elegir una partícula p_i de la caja
2. Identificar si la partícula p_i está contenida en una esfera de radio mayor al radio de la partícula, de forma tal de que si la partícula tuviera dicho radio, no colisionaría con otra partícula de la caja.
3. Si p_i está contenida en un radio mayor

   Mover la partícula p_i de forma de que quede lo más próxima posible a la primera partícula con la que eventualmente colisionaría.

   Al mover la partícula p_i quedó un espacio libre, intentar posicionar una de las partículas que se tienen en el stock de partículas.

4. Si p_i no está contenida en un radio mayor se continua con la siguiente partícula.

La mutación elegida identifica espacios vacios, compacta las partículas hacia esos huecos e inserta partículas nuevas en los espacios libres (respetando la distribución de probabilidad).

Operador de reemplazo de la generación El operador de reemplazo corresponde a un modelo poblacional llamado solapado, debido a que los individuos padres y sus descendientes compiten por la descendencia generada. El operador sigue los siguientes pasos: 1. se ordena toda la población de forma decreciente según su valor de fitness y 2. se quitan los peores n primeros individuos, con n igual a la cantidad de hijos producidos.

Siendo que se descartan tantos individuos como nuevos hijos generados, el tamaño de la población se mantiene constante.

5.2. Simulación de medios granulares bajo los efectos de fuerzas gravitatorias

En esta sección se presentan el algoritmo de simulación de medios granulares.

5.2.1. Metodología

Una vez que se conocen las posiciones de las partículas que componen el medio granular, se genera una grilla que se solapa en el espacio con el medio granular. La grilla divide el medio granular en diferentes celdas o cubos. Considérese una grilla en 2D resultante de dividir el dominio en 3 divisiones en el eje X y en el eje Y, subdividiendo el espacio en 16 celdas tal como se muestra en la figura 5.4. En la figura 5.4 las partículas se representan como puntos azules, el centro espacial O de la celda con coordenadas $(x = 1, y = 2)$ y el centro de masa CM de las partículas p_1 y p_2 contenidas en la celda $(x = 2, y = 3)$.

El algoritmo calcula en cada centro espacial de cada celda la fuerza gravitatoria generada por cada partícula del aglomerado. El cálculo de la fuerza gravitatoria en cada punto O es independiente al del resto de las celdas de la grilla. Cuando la fuerza gravitatoria es calculada para todos los centros espaciales de todas las celdas entonces el algoritmo finaliza.

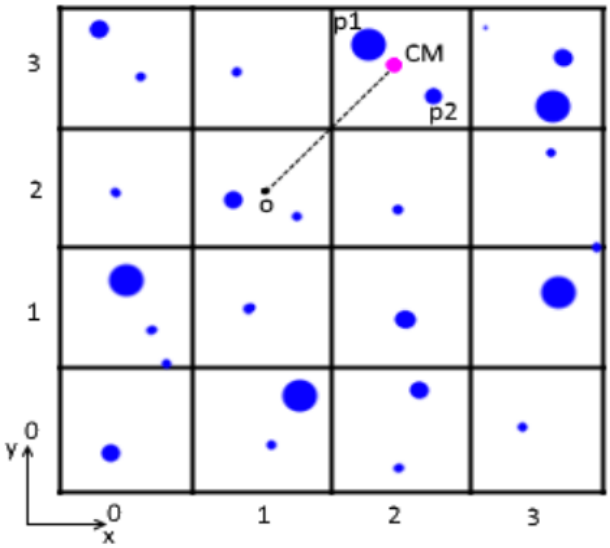


Figura 5.4: Metodología de calculo de autogravedad.

5.2.2. Paralelicidad del algoritmo

En el algoritmo paralelo desarrollado se tienen secciones secuenciales y paralelas. Se considera la paralelización de una sección dependiendo de la necesidad y/o ganancia que se obtendría en la paralelicidad y la complejidad resultante.

Las secciones que requieren mayor poder de cómputo, y por consecuencia donde el algoritmo gasta la mayor parte del tiempo, son las secciones donde la paralelicidad obtiene la mayor ganancia. En el algoritmo paralelo existen dos secciones paralelas, en la primera se calcula la fuerza gravitatoria en cada celda de la grilla y en la segunda se realiza el cálculo de la interpolación. En la figura 5.5 se muestran las distintas secciones:

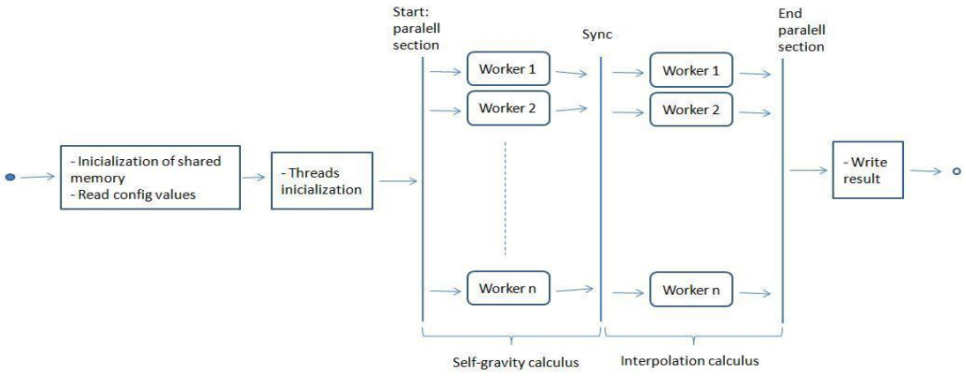


Figura 5.5: Paralelicidad del algoritmo.

En el gráfico anterior se puede reconocer las secciones principales que conforman el algoritmo paralelo:

- Etapa 1. Initialization of shared memory: inicializa estructuras de memoria compartida. Read config values: se leen valores de configuración
- Etapa 2. Threads initialization: se crean las estructuras de hilos y objetos necesarios.
- Etapa 3. Self-gravity calculus: sección paralela donde se procesa el cálculo de la auto-gravedad. Los objetos Worker representados en el diagrama, son los encargados de ejecutar el trabajo pesado de la sección paralela.
- Etapa 4. Interpolation calculus: sección paralela donde se procesa el cálculo de la interpolación. Los objetos Worker representados en el diagrama son los mismo que para la etapa 3.
- Etapa 5. Output result: etapa en la cual ya se ha terminado el procesamiento pesado y se procede a devolver los resultados obtenidos.

5.2.3. Modelo de programación paralela

Uno de los aspectos importantes a la hora de definir un algoritmo paralelo es definir el modelo paralelo que se aplicará y cómo. En esta sección se describe el modelo de programación paralela utilizado y su implementación conceptual.
Tal como se describieron existen dos modelos, uno basado en descomposición de dominio o datos y otro basado en descomposición funcional. Analizando el algoritmo de calculo gravitatorio, se concluye que se tiene fundamentalmente una sola función a computar que se repite en el dominio de estudio. Por esa razon, se optó por implementar un modelo de programación paralela basado en la descomposición de datos, siendo la grilla el dominio a descomponer y las celdas la mínima unidad a procesar.

La forma en como las celdas a procesar se asignan a los recursos de computo disponible impacta en el comportamiento del algoritmo. Se pueden diferenciar dos categorías de estrategias para asignar las celdas, una categoría estática y una categoría dinamica. En la categoría estática se encuentran aquellas estrategias que no toman en cuenta el comportamiento de la ejecución, mientras que en la categoría dinamica se encuentran aquellas estrategias que modifican su comportamiento segun se comporte la ejecución.

En esta sección se analiza el comportamiento de diferentes estrategias de asignación de nodos o celdas de la grilla a los diferentes recursos de cómputo. En el proceso de estudiar el comportamiento del algoritmo se implementaron las siguientes estrategias de recorrida del dominio de datos:

1. Estrategia de interbloqueo lineal con igual comienzo
2. Estrategia circular concéntrica
3. Estrategia aislada lineal básica
4. Estrategia aislada lineal avanzada

Estrategia de interbloqueo lineal con igual comienzo Los hilos son ordenados lexicográficamente desde 1 a n, y todos los hilos empiezan desde el mismo punto inicial. La primera celda libre (celda nro. 1) es asignada al primer hilo, la segunda celda libre (celda nro. 2) es asignada al segundo hilo, y así sucesivamente, hasta que todos los hilos tienen una celda para realizar el procesamiento. Luego, cada vez que un hilo h termina de procesar una celda c y requiere una celda para seguir trabajando, se busca la siguiente celda sin procesar a partir de c y se le asigna al hilo h.

La figura 5.6 muestra la estrategia de interbloqueo lineal, donde cada color es usado para representar las celdas siendo asignadas a diferentes hilos (colores completos representan celdas ya procesadas, colores en degradé representan celdas siendo procesadas y celdas blancas son sin procesar):

Figura 5.6: Estrategia de interbloqueo lineal con igual comienzo.

Algo evidente es la necesidad de sincronizar el acceso a cada celda, para evitar que una misma celda sea procesada repetidas veces por los diferentes hilos. Para eso se agregó un semáforo que permite sincronizar el acceso al área de memoria compartida en la cual se especifica el estado de las celdas (es decir el conjunto de celdas sin procesar, ya procesadas, etc.). Si bien esto garantiza la correctitud en la solución, la sincronización de una cantidad relativamente alta de hilos en un solo punto crea un cuello de botella que enlentece la ganancia agregada con la paralelización. Para solucionar dicho cuello de botella se sustituyó el único semáforo por un conjunto de semáforos, asignando uno a cada celda, de esta forma se tiene una solución más sencilla, mejor control y más fino de qué hilo se está bloqueando.

Una vez ésta estrategia de recorrida lineal con un mismo comienzo estaba funcionando correctamente, se buscó mejorarla teniendo en cuenta los cálculos, el orden y los valores cacheados en el proceso.

Estrategia circular concéntrica En esta estrategia la mitad de los hilos empiezan desde el centro de la grilla, y la otra mitad de los hilos empiezan desde las celdas más alejadas del centro de la grilla. Cuando el algoritmo ejecuta, para cada celda a procesar cada hilo debe calcular el centro de masa de la celda. Este valor solo depende de las particulas contenidas en la celda y no cambia hasta que termina la ejecucion, por lo cual el valor es almacenado en memoria como dato derivado de la celda. Lo que se busca con esta estrategia, es reutilizar dichos valores de centro de masa por los distintos hilos en el transcurso de su procesamiento. Los hilos que ejecutan desde el centro hacia afuera calculan y almacenan ciertos valores que seran reutilizados por los hilos que ejecutan en sentido contrario, y viceversa. Las ideas básicas de esta estrategia se muestran en la figura 5.7.

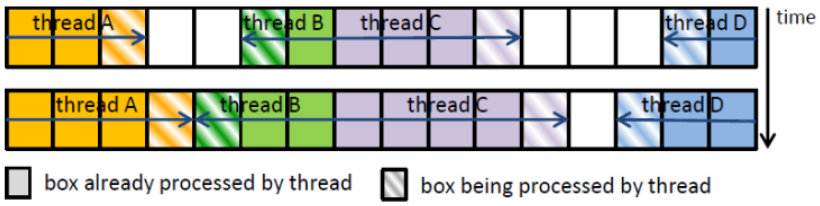


Figura 5.7: Estrategia circular concéntrica.

Dado que la mitad de los hilos arrancan desde la misma celda en el centro de la grilla y la otra mitad de los hilos arranca en una de las celdas más alejadas del centro de la grilla, igualmente necesitamos semáforos para sincronizar la ejecución y asegurar que no se procesa una misma celda múltiples veces. Nótese que la forma en cómo comienzan las dos mitades de los hilos (en el centro de la celda y en la celda más alejada del centro) es similar a la forma en cómo ejecuta la estrategia anterior.

La poca complejidad algorítmica involucrada en el calculo del centro de masa en comparacion con el calculo de autogravedad, hacen que esta estrategia no tenga una ganancia significativa como veremos en los siguientes capitulos.

Estrategia aislada lineal básica Esta estrategia surge al considerar que los hilos no comienzan en la misma celda, sino que distribuidos de forma tal de minimizar las colisiones en el acceso a la memoria compartida. Esta estrategia se caracteriza por realizar una recorrida lineal de las cajas, tal como la primera estrategia descrita, pero asignando a cada hilo un comienzo distinto, no consecutivo y distribuido uniformemente. Es decir, si se considera el conjunto de celdas c_i con i desde 0 a $n-1$ y el conjunto de hilos h_i con i desde 0 a $h-1$, se divide dicho conjunto n de celdas entre el conjunto h de hilos disponibles, de tal forma que para cada hilo h_i, se le asigna como comienzo la celda $h_i * (n/h)$. De esta forma los hilos tienen el mismo porcentaje de dominio libre para procesar antes de toparse con una celda que ya fue calculada por otro hilo. En esta estrategia no se necesita manejar semáforos para cada celda, por lo que podría apreciarse una mejora en la performance. La estrategia se puede visualizar en la figura 5.8.

Figura 5.8: Estrategia aislada lineal básica.

Esta estrategia pertenece a la categoria de estrategias estaticas, es decir los hilos no cambian su comportamiento segun se comporte la ejecución (el resto de los hilos). Para ejemplificar un posible escenario, un hilo que termina su sección asignada estáticamente no continuará ejecutando nuevas celdas en otra sección, sino que finalizará la ejecución. Esto impacta en el tiempo total que le toma al algoritmo.

Notar cómo en esta estrategia se sacrifica el dinamismo en la asignación de celdas en pos de mejorar la performance obtenida.

Estrategia aislada lineal avanzada Esta estrategia incorpora a la estrategia anterior la capacidad de ejecutar nuevas celdas en otras secciones una vez que la sección asignada al inicio se completa.

La idea fundamental de esta estrategia es presentada en la figura 5.9.

La estrategia divide el procesamiento en dos etapas: la primera etapa es idéntica a la estrategia anterior hasta que el espacio de celdas asignado a un hilo se completa, en ese momento comienza la segunda etapa que se comporta como la estrategia nombrada .^Aislada lineal básica". En esta etapa se comienzan a procesar celdas en los espacios de celdas de otros hilos. Esta estrategia combina las mejores características de ambas estrategias.

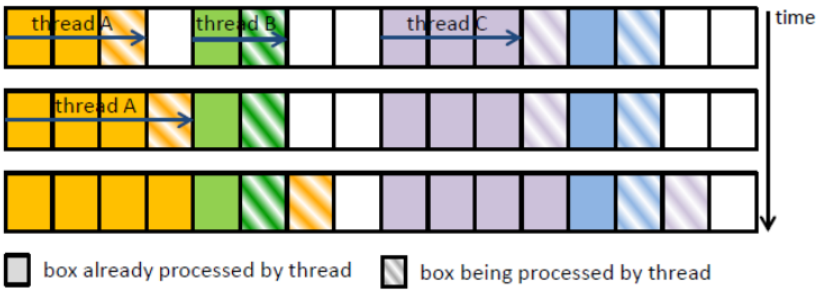


Figura 5.9: Estrategia aislada lineal avanzada.

5.2.4. Aproximación de masas por distancia

Motivación Como ya se mencionó, uno de los aspectos complejos del problema resulta en el alto número de partículas a considerar. Para soportar el cálculo (desde un punto de vista computacional) cuando la cantidad de partículas crece, se necesita reducir el número total de cálculos efectuados. Para eso, hay que recordar que, a una distancia suficientemente grande, desde el punto de vista del efecto gravitatorio, la masa de dos cuerpos puede ser considerada como condensada a su centro de masa (sin generar esto errores significativos). Por tanto, para disminuir el orden de magnitud del cálculo, se puede aplicar un algoritmo con una estructura de datos adecuada que agrupe a las partículas lejanas sustituyendolas por una masa combinada y una distancia media o centro de masa. Este concepto nos permite abordar magnitudes del orden de millones de particulas.

Metodología Considérese un conjunto de partículas con sus coordenadas correspondientes, y sobre cada grupo de partículas una celda perteneciente a la grilla que las contiene, tal como se muestra en la figura 5.9.

Desde el punto origen situado en O, realizar el cálculo de la auto-gravedad utilizando cada partícula contenida en la celda A o utilizando solamente el centro de masa de las partículas de la celda A (punto P_j), se pueden aproximar (cuando la distancia de O a P_j es significativa) sin ser esto una fuente de error considerable.

Siguiendo ese razonamiento para todas las partículas dentro de la grilla, se puede calcular su centro de masa y posición y usar dichos valores para el cálculo del potencial en cualquier punto. En este caso, en vez de iterar sobre todas las partículas para calcular el potencial en un punto O, solo se toman en cuenta los centros de masa de cada celda de la grilla definida. Considerando que enfocamos nuestra problematica en aglomerados densos, se tienen menos centros de masa que particulas, por lo que la complejidad algoritmica disminuye. Se le denomina MADA a dicha estrategia por su sigla en ingles (Mass approximation distance algorithm).

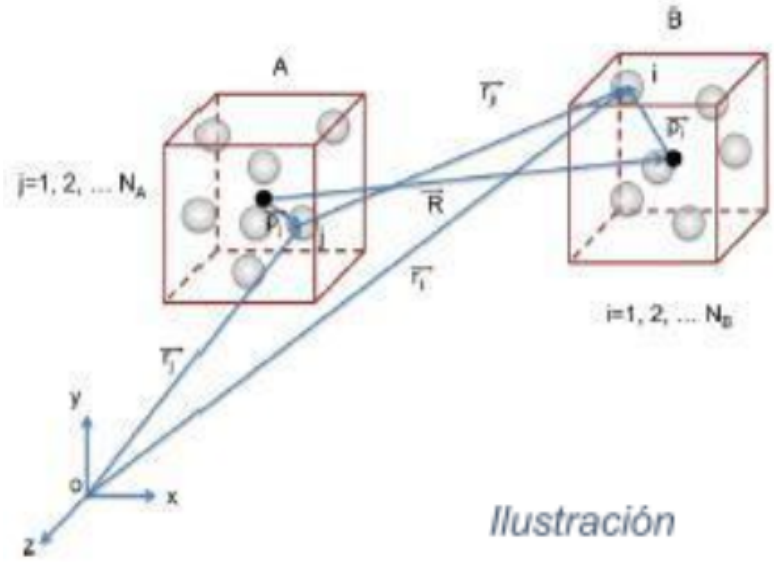


Figura 5.10: Aproximación de masas por distancia.

Celdas virtuales La aplicación de MADA incurre en errores significativos en aquellos escenarios en los que el punto origen O se encuentra próximo al centro de masa. Una mejora de esta estrategia surge al subdividir las celdas de la grilla de forma dinámica dependiendo de la distancia al punto origen O. Para explicar este concepto resulta útil definir el siguiente vocabulario: se denomina celda de primer nivel a las celdas tal como se presentaron hasta ahora (en la figura 5.11 tales celdas estan representadas con borde negro), se denomina celdas de segundo nivel a las celdas con una división interna con lo que definen 4 nuevas sub-celdas (en la figura 5.11 dichas celdas estan representadas con borde azul) finalmente se denomina celdas de tercer nivel a las celdas con dos divisiones con lo que definen 16 nuevas sub-celdas (en la figura 5.11 son las celdas de borde rojo).

Lo que esta mejora propone es dividir las celdas en sub-celdas de diferentes niveles (mayor nivel a medida que se acerca al punto origen O) y utilizar los centros de masa de las partículas contenidas por dentro de estas sub-celdas en vez del centro de masa de la celda de primer nivel. Nótese que la celda que contiene el punto origen O, no utiliza el centro de masa de la celda, sino que calcula directamente utilizando las partículas. Con esta mejora mantenemos la disminución algorítmica lograda con MADA para partículas distantes mientra que se asegura no cometer errores con partículas cercanas.

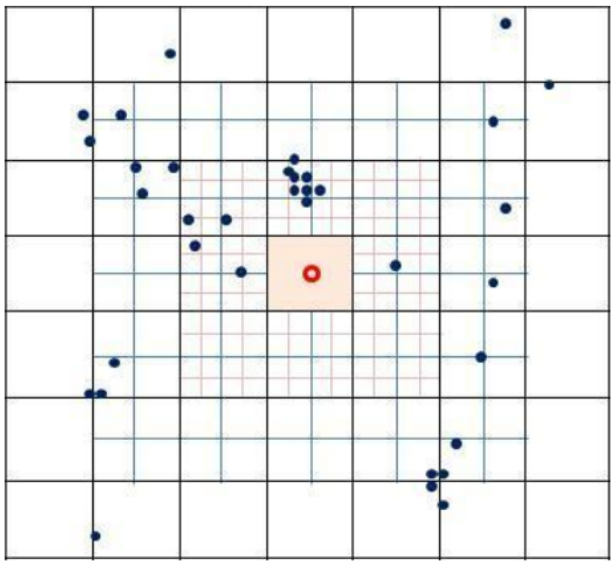


Figura 5.11: Subdivision de celdas de MADA.

Configuración de la malla El MADA es influido por la cantidad de subdivisiones totales que se realizan a las celdas de primer nivel. Por otro lado, según el aglomerado que se está estudiando se preferirá una u otra forma de dividir las celdas. Por las dos razones anteriores, es que se necesita permitir configurar al usuario que ejecuta el algoritmo la forma en como se dividen las celdas de la grilla.

Las implicancias que tiene la elección de los valores configurables son:

- marco de trabajo: en su forma más intuitiva la grilla define el área de interés. En caso de que la resolución de la grilla sea muy gruesa, o no se defina correctamente (corrida respecto a las coordenadas del medio granular) se pueden estar perdiendo valores de interés. En el otro extremo, si se define una grilla muy grande (para asegurar que todo el medio granular es cubierto) puede generar muchas celdas vacías, se estarían procesando celdas sin datos útiles, por lo que no se estaría haciendo un uso útil de los recursos de cómputo disponibles.

- paralelicidad: influye en la paralelicidad debido a que cada celda representa el mínimo elemento que se asigna para procesar. Por lo que el tamaño de la grilla influirá en la resolución de la paralelicidad posible: una grilla fina permitirá tener un control más riguroso en la asignación de celdas, pero si ésta resulta excesivamente fina hará que cada hilo necesite constantemente pedir nuevas celdas que procesar, agregando sobre procesamiento inútil.

- volumen de resultados: debido a que es en el centro de cada celda donde se calcula el valor de auto-gravedad. Por lo que una grilla fina incrementará el poder de cómputo requerido pudiendo suceder que se obtengan más datos de los necesarios, mientras que una grilla gruesa generará pocos resultados, pudiendo no ser realmente útil la ejecución en este último caso.

- una malla gruesa podría no capturar todas las características del problema y podría conducir a resultados no exactos; una rejilla fina proporcionará un control más estricto sobre la asignación de las celdas, pero una muy fina rejilla obligará a cada hilo para aumentar significativamente la comunicación y tiempos de sincronización.

5.2.5. Modelo de memoria compartida

El algorítmo desarrollado se basa en el modelo de memoria compartida, en el cual todos los hilos de ejecución tienen acceso a un único repositorio central de datos.

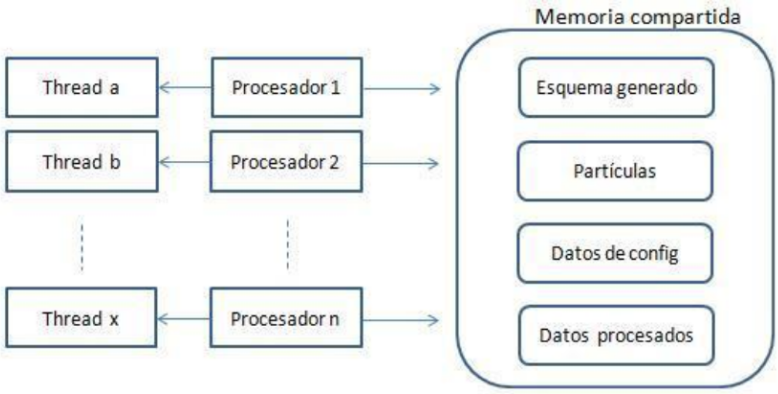


Figura 5.12: Modelo de memoria compartida.

En el diagrama 5.12 se muestran las áreas de memoria que se comparten:

1. Esquema generado: el esquema generado con su conjunto de celdas y estructuras necesarias.

2. Partículas: todas las partículas iniciales una vez se terminan de leer desde archivo.

3. Datos de configuración: el conjunto de valores de configuración utilizados por los diferentes hilos.

4. Datos calculados: todos los datos que se obtienen como resultado de la ejecución del algorítmo.

Los puntos del 1 al 3 son datos que una vez leídos desde archivo se mantienen en memoria y solo son leídos por el algoritmo, no se actualizan durante la ejecución, por lo que no se necesita sincronizar el acceso desde múltiples hilos.

En el punto 4, se mantienen en memoria compartida los datos producto de la ejecución del algoritmo. Para ésta área de memoria compartida, se sincroniza el acceso al espacio de memoria a través de un semáforo dedicado.

5.2.6. Interpolación

La motivación detras de aplicar técnicas de interpolación se basa en disminuir el costo computacional. Aplicar el cálculo del potencial gravitatorio sobre una malla gruesa y luego una técnica de interpolación sobre una malla más fina, es menos costoso computacionalmente que aplicar el cálculo del potencial sobre una malla fina.

A continuación enumeramos las técnicas de interpolación y luego se realiza un análisis teórico de cada una de ellas:

- Nearest neighbor
- Ponderación lineal
- Kriging
- Inverse distance weighting (IDW)

La primer técnica de interpolación simplemente asigna al nuevo valor interpolado el valor más cercano (como distancia euclideana). Está técnica es simple de implementar y brinda una referencia base contra que comparar.

A través del análisis de la ecuación de la segunda técnica de interpolación, se concluye que se está asignado a los puntos mas lejanos al valor origen O a interpolar, mayor importancia en el resultado que los puntos cercanos. Esto se contradice con la magnitud que se está interpolando (potencial gravitatorio), la cual se hace más débil con la distancia. Esta técnica de interpolacion produciría valores incorrectos. Como conclusión de este análisis, concluímos que necesitamos una técnica que proponga lo opuesto, es decir asignar mayor relevancia a puntos cercanos al valor a interpolar, o que permita decidir cuánta relevancia se quiere asignar a los valores conocidos respecto a la distancia al punto origen O a interpolar.

La técnica de Kriging es un algoritmo muy conocido para estimar variables estadísticas. El método puede ser entendido como una predicción lineal o una forma de inferencia bayesiana. Esta técnica parte del principio de que puntos próximos en el espacio tienden a tener valores más parecidos que los puntos más distantes. La técnica de kriging asume que los datos recogidos de una determinada población se encuentran correlacionados en el espacio. A modo de ejemplo, si en un vertedero de residuos tóxicos y peligrosos la concentración de zinc en un punto p es x, será muy probable que se encuentren resultados muy próximos a x cuanto más próximos se esté del punto p (principio de geoestadística). Sin embargo, desde una cierta distancia de p, ciertamente no se encontrarán valores próximos a x porque la correlación espacial puede dejar de existir.

A diferencia del método de Ponderación lineal, la técnica IDW sí se adapta a la magnitud interpolada, puesto que asigna mayor relevancia a puntos cercanos al valor a interpolar. Además, la técnica de Kriging permite indicar cuánta relevancia queremos asignar a los puntos respecto a la distancia al punto a interpolar.

Luego de haber descartado durante esta etapa de análisis teórico la técnica de "Ponderación lineal", se vera en el capitulo 6 el análisis experimental de ambas técnicas de Kriging y IDW.

5.2.7. Paralelicidad en el cálculo de la autogravedad y cálculo de la interpolación

Los calculos de autogravedad e interpolacion son los que demandan mayor recursos de computo. Debido a esta demanda de computo se integraron técnicas de paralelización en dichas etapas.

En vez de crear y destruir el conjunto de hilos a usar en cada sección paralela, se utilizan las mismas instancias de los hilos en todo momento. Esto tiene como ventaja que no se necesita hacer un llamado al sistema operativo para crear y/o destruir hilos, logrando reutilizar recursos costosos como son los hilos. Igualmente, la desventaja que esto acarrea es que para diferentes secciones que presentan diferente complejidad y necesidad de cálculo no se podría dedicar diferente cantidad de hilos. Por ejemplo, en el cálculo de autogravedad se dedica una cantidad de hilos X que se adapta dependiendo de la dimensión del problema planteado. Sin embargo, por lo general el cálculo de la interpolación (que es la sección que ejecutaría a continuación) no demanda tanto poder de cálculo, pero igualmente no se puede evitar destinar la misma cantidad de hilos a su ejecución. Esto puede generar una pérdida de la performance por un cuello de botella en la sincronización entre más hilos de los necesarios. De todas formas, esta rigidez en la asignación de hilos se podría superar fácilmente agregando más hilos a los ya creados si eso fuera necesario, o utilizar un subconjunto de los disponibles manteniendo el resto no utilizado en un estado en espera.

Una vez seleccionado el algoritmo resta decidir cuáles puntos utilizar como entrada en la interpolación. La técnica IDW no nos limita un máximo ni un mínimo, sin embargo puntos muy lejanos tienen una mínima influencia comparado con puntos vecinos. Luego de algunos análisis experimentales se definió que utilizar los ocho puntos más cercanos nos brinda un buen compromiso entre cantidad de datos y calidad del resultado obtenido.

5.2.8. Integración al paquete ESyS-Particle

En esta sección se describen los cambios realizados en el algoritmo para integrarlo con el paquete ESyS-Particle.

Involucrados:En las distintas etapas del proceso de integración con el paquete ESyS-Particle hubieron distintos involucrados, entre los cuales están: Steffen Abe (Investigador Postdoctoral, Instituto de Gestión de Recursos Geotérmicos, Mainz, Alemania), Dion Weatherley (Investigador Postdoctoral en la Universidad de Queensland, Australia), Sergio Nesmachnow (FING - UdelaR), Gonzalo Tancredi (FCIEN - UdelaR), Francisco Lopez (FCIEN - UdelaR), Néstor Rocchetti (FING - UdelaR) y Daniel Frascarelli.

Se realizaron dos tipos de integraciones al paquete ESyS-Particle, una que denominamos integración FewParticles y otra que en contraposición denominamos integración ManyParticles. La integración FewParticles, hace un cálculo exacto sobre todas las partículas del dominio. Para ellos es necesario de disponer de todas las partículas en el recurso de cómputo, por lo que esta forma de integración limita la escalabilidad de la solución final (recuérdese que ESyS-Particle divide partículas entre recursos de cómputo). La segunda forma de integración ManyParticles no hace un cálculo exacto sino que realiza diferentes aproximaciones (como la ya vista MADA) comunicando los valores de autogravedad calculados entre los distintos recurso de cómputo. A continuación se describe en profundidad cada forma de integración.

Integración FewParticles Para poder simular el comportamiento de partículas bajo el efecto de una fuerza de auto-gravedad es necesaria la incorporación de nuevas interacciones al producto ESyS-Particle. ESyS-Particle permite agregar nuevos tipos de fuerzas que aún no estén implementadas por dentro del paquete de forma relativamente sencilla. Sin embargo, dada la arquitectura paralela maestro-esclavo y la interfaz Python, es necesario agregar pequeños trozos de código a lo largo de todo el código fuente.

Existen cuatro tipos de interacciones en ESyS-Particle que difieren en la forma en que son implementados:

- Interacciones no enlazadas entre pares de partículas.
- Interacciones enlazadas entre pares de partículas.
- Interacciones entre partículas y muros o mallas.
- Campos de fuerza tales como la gravedad, que son implementados como interacciones de una sola partícula.

El tipo de interacción que se debe agregar coincide con el cuarto tipo de interacción antes comentado. En el proyecto de grado Paralelismo aplicado al estudio de medios granulares (L. Heredia y P. Richeri) se integra una interacción de tipo Hertzian Visco Elastic Friction, que si bien difiere en algunos aspectos resulta similar el esfuerzo necesario. agregar referencia al proyecto de grado. como se agrega la referencia ? @Nestor tenes algun ejemplo?

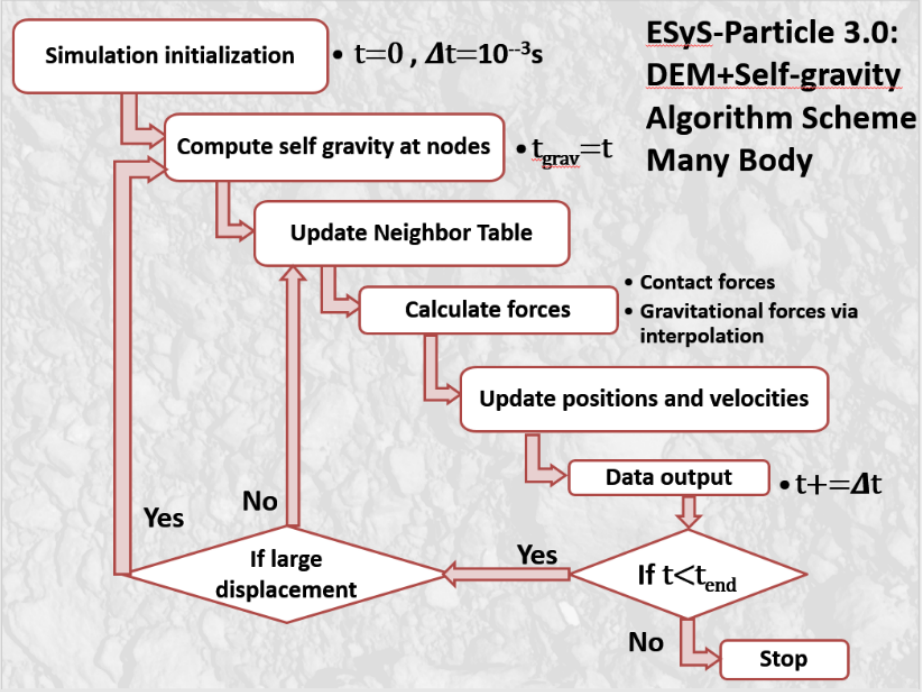
Esta interacción calcula el potencial gravitatorio V_j en cada partícula j de un aglomerado de N partículas de masa M_i y posición r_i mediante la siguiente fórmula:

$$\sum_{i \neq j} \frac{GM_i}{\| \overrightarrow{r_j} - \overrightarrow{r_i} \|}$$

En cada paso de simulación se calcula para cada partícula el potencial gravitatorio generado por el resto de las partículas y se aplica la fuerza resultante. Esta forma de integrar el calculo de auto-gravedad en ESyS-Particle esta bien soportado y resulta simple considerando la arquitectura interna del paquete. Por otro lado, una desventaja clara es la limitación impuesta en la escalabilidad resultante puesto que solo se puede utilizar un recurso de cómputo.

Integración ManyParticles Esta integración con el paquete ESyS-Particle está diseñada para simular del orden de millones de partículas. Esto no es viable con la integración FewParticles, en el cual solo se puede utilizar un recurso de cómputo para procesar todas las partículas. Para esta integracción denominada *ManyParticles* es necesario utilizar más de un recurso de cómputo y habilitar la subdivisión del dominio de trabajo del ESyS-Particle.

Cuando el número de partículas crece la fórmula anterior de orden $O(N^2)$ resulta demasiado costoso computacionalmente. Para ello se diseña una solución que no re-calcula la autogravedad en todos los pasos de simulación, sino que solo recalcula la auto-gravedad cuando ocurren una de dos cosas: o pasaron un número de pasos de simulación dado o las partículas en su promedio se mueven más que un valor D_a. El gráfico a continuación, inspirado en el diseño de ESyS-Particle básico, describe el flujo de cálculo resultante:

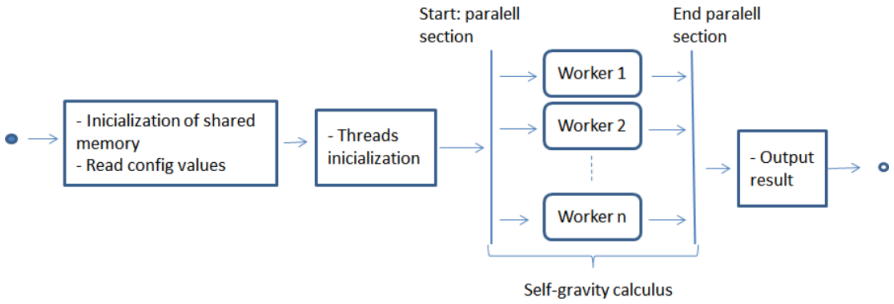


Como se puede ver en la imagen, el paso de calculo de la autogravedad, solo se ejecuta segun cierta condicional. Debido a ello, el cálculo de auto-gravedad obtenido sólo es exacto en el primer paso de simulación, subsecuentes pasos de simulación utilizarán un valor aproximado. Si bien la aproximación es muy buena (puede verse una análisis del error en el Análisis experimental), este valor aproximado genera en simulaciones largas un error que se arrastra y modifica notoriamente el comportamiento del aglomerado. Se diseñaron e implementaron diferentes estrategias para corregir el efecto obtenido, dichas estrategias se describen más adelante en las mejoras posteriores realizadas.

Para atacar del orden de millones de partículas sin que los tiempos de simulación se disparen, se diseñó un procedimiento de cálculo que, como ya se comentó, permite que el cálculo de auto-gravedad sea independiente del número de partículas. Para eso es que se define una malla que se solapa con el medio granular, y en los puntos de la malla es donde se calcula la autogravedad. Dicho cálculo de autogravedad se realiza en un recurso de cómputo principal de entre los que corre el ESyS-Particle, y dichos valores y puntos calculados se comunican a los otros recursos de cómputo (posiblemente distribuidos) que realizarán los cálculos de cada paso de simulación.

Una de las ventajas obvias de esta integración es que permite trabajar con del orden de millones de partículas. Además los distintos parámetros que permiten por ejemplo configurar el uso de recursos usados para calcular la autogravedad, la frecuencia con la que se realiza el cálculo, la resolución de las mallas de cálculo de autogravedad, etc permiten adaptar y hacer un uso adecuado de los recursos de hardware disponibles para cada problema específico. Como desventaja, esta forma de integración no realiza un cálculo exacto de la autogravedad, sino que realiza aproximaciones (necesarias para impedir que el orden de los cálculos sea dependiente del orden de las partículas), las cuales afectan la simulación generada.

Una vez integrado el algoritmo, las secciones anteriormente diseñadas cambian ligeramente:



Estos cambios son debido a que al integrar el cálculo de autogravedad en el paquete ESyS-Particle, el cálculo de autogravedad en los nodos de la maya configurable se realiza en el nodo maestro. Una vez obtenidos dichos valores se envían a los procesos esclavos del ESyS-Particle (posiblemente distribuidos) los cuales utilizando estos valores interpolan el valor de la fuerza de auto-gravedad en las posiciones de las partículas del subdominio que le corresponde. Esta distribución de los cálculos impide tener el algoritmo de interpolación por dentro del código de auto-gravedad, sino que se migró a los procesos esclavos.

Analizando la dinamica de ejecucion del paquete ESySParticle surge la necesidad de modificar cómo se calcula la interpolación, por lo que el código de interpolación ya no forma parte del algoritmo paralelo de autogravedad, sino que se agregó por dentro del código de ESyS-Particle. Si bien esto implica que el desarrollo de la interpolación en paralelo quedará obsoleta, el análisis experimental sobre cuál técnica se aplica mejor en nuestro caso sigue siendo válida.
Estos cambios se ven con mayor profundidad en la siguiente sección.

Capítulo 6

Análisis experimental

6.1. Algoritmo evolutivo

Una vez diseñado e implementado el algoritmo evolutivo, se realizó el análisis experimental para determinar los nivel de compactación.

no tengo resultados útiles del algoritmo evolutivo, que hacer?

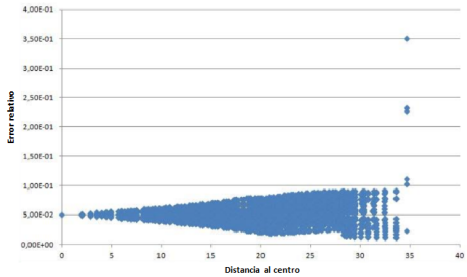
6.2. Interpolación

Se analizó el comportamiento de los métodos de interpolación Kriging e IDW. Los errores relativos para el algoritmo de Kriging e IDW se muestran en la figura 6.1. De la figura 6.1 se desprende que los algoritmos IDW tienen menor error relativo. En la figura 6.2 se muestran los errores relativos mínimo, promedio y máximo para los algoritmos IDW. En las figura 6.2a y 6.2b se observa que los errores relativos promedio y máximo no varía considerablemente para los algoritmos IDW. En la figura 6.2c se observa que el algoritmo IDW con parámetro cuatro presenta menor error relativo mínimo. Por tener menor error relativo mínimo pero similares errores promedio y máximo se seleccionó el algoritmo de IDW con parámetro cuatro.

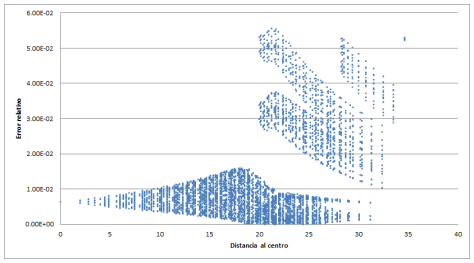
6.3. Cálculo paralelo de simulación de medios granulares

Esta sección presenta el análisis experimental del calculo paralelo de la simulación de medios granulares. El análisis experimental fue realizado en un servidor AMD Opteron 6172 (2.1GHz) 24-Core, 24 GB RAM con CentOS Linux, del Cluster Fing. Se analizó el error relativo del algoritmo implementado, la eficiencia computacional y el speedup.

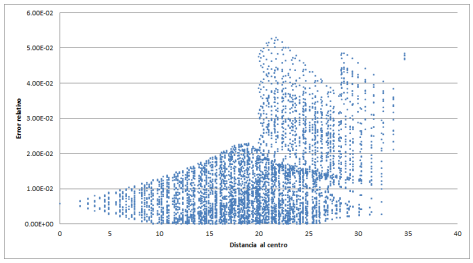
Se definieron seis escenarios para evaluar el algoritmo:



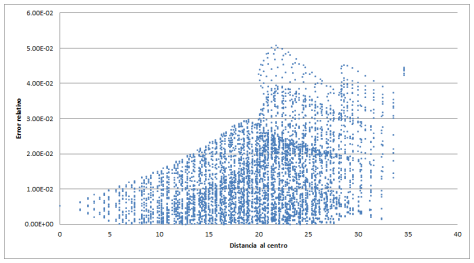
(a) Error relativo del algoritmo Kriging



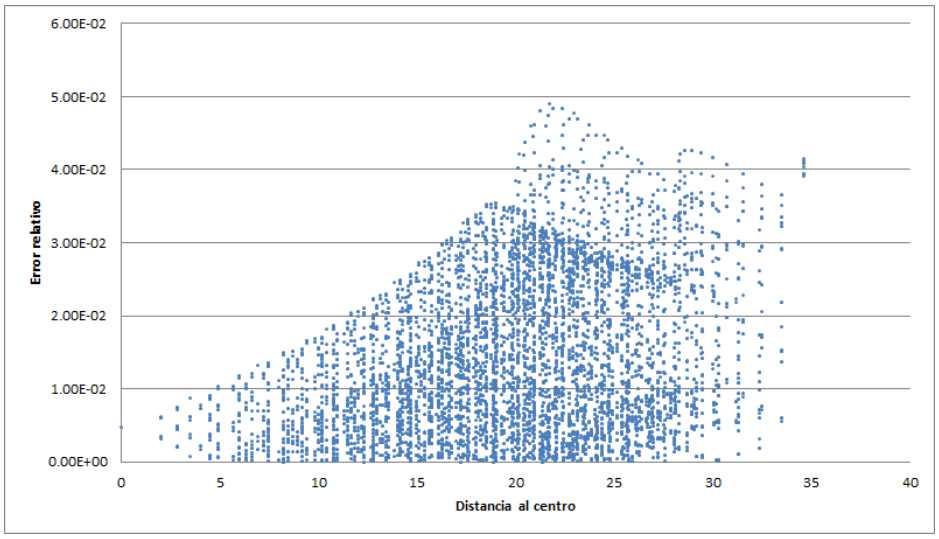
(b) Error relativo del algoritmo IDW con parámetro $p = 1$



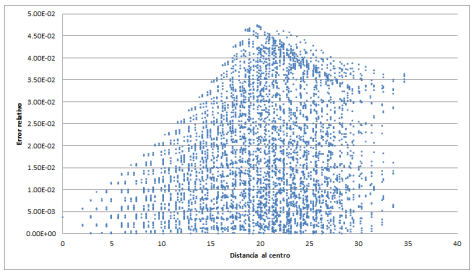
(c) Error relativo del algoritmo IDW con parámetro $p = 2$



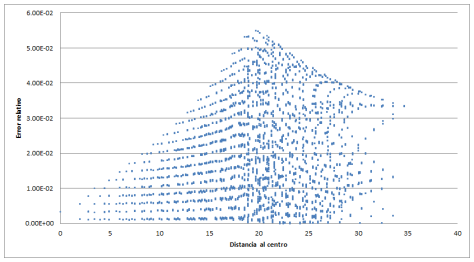
(d) Error relativo del algoritmo IDW con parámetro $p = 3$



(e) Error relativo del algoritmo IDW con parámetro $p = 4$

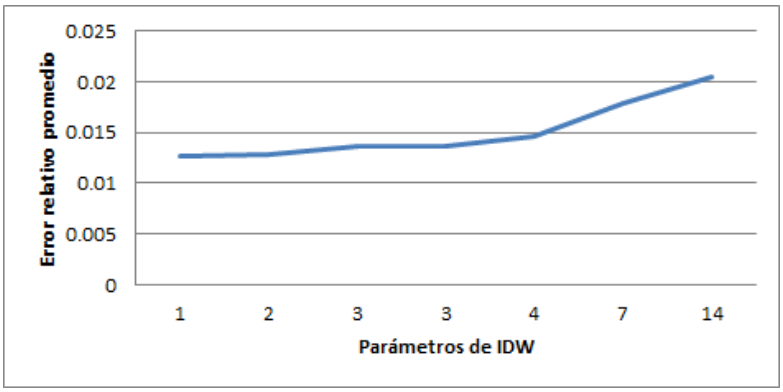


(f) Error relativo del algoritmo IDW con parámetro $p = 7$

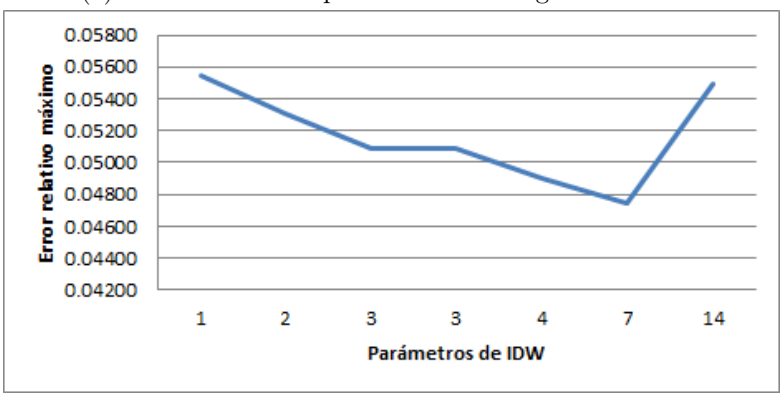


(g) Error relativo del algoritmo IDW con parámetro $p = 14$

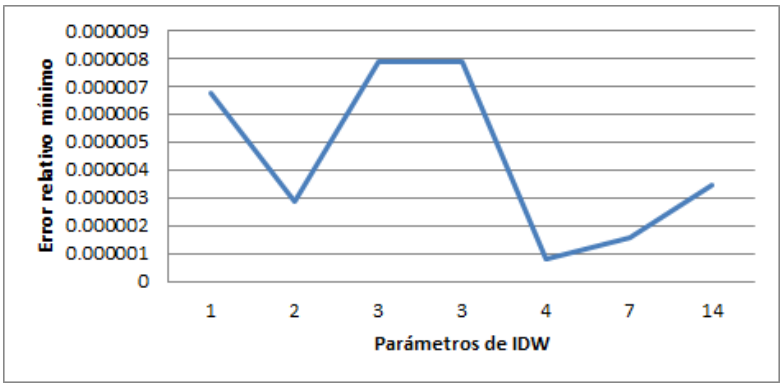
Figura 6.1: Errores relativos de los algoritmos de interpolación



(a) Errores relativos promedio de los algoritmos IDW



(b) Errores relativos máximos de los algoritmos IDW



(c) Errores relativos mínimo de los algoritmos IDW

Figura 6.2: Errores mínimos, promedio y máximo relativos de los algoritmos IDW

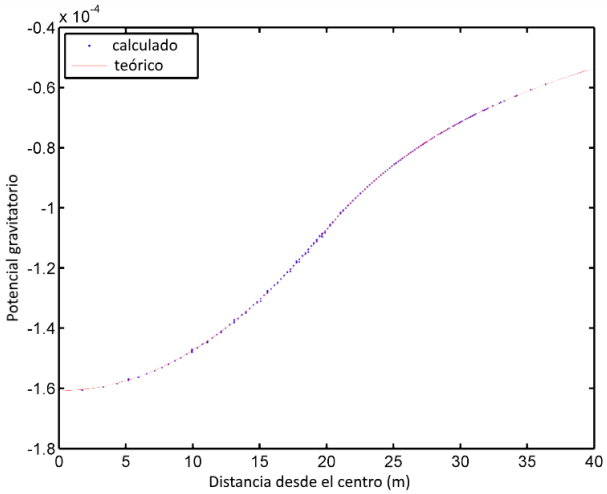


Figura 6.3: Potencial gravitatorio versus la distancia al centro de masa para cerca de un millón de partículas (el valor teórico está sobre dibujado).

- A: 26.344 partículas de radio XXXm contenidas en una esfera de radio YYm
- B: 61.432 partículas de radio XXXm contenidas en una esfera de radio YYm
- C: 101.849 partículas de radio XXXm contenidas en una esfera de radio YYm
- D: 257.776 partículas de radio XXXm contenidas en una esfera de radio YYm
- E: 605.808 partículas de radio XXXm contenidas en una esfera de radio YYm
- F: 1.022.208 partículas de radio XXXm contenidas en una esfera de radio YYm

Los escenarios utilizados tienen simetría esférica, por lo que existe una expresión analítica para calcular el potencial gravitatorio en función de la distancia al centro de masa. La figura 6.3 muestra el potencial gravitatorio calculado a partir de la expresión analítica en función de la distancia al centro. En la figura 6.3 también se muestra superpuesto al valor teórico, el potencial gravitatorio calculado a partir del computo del algoritmo implementado.

La figura 6.4 muestra el error relativo del potencial gravitatorio computado respecto al valor teórico en función de la distancia al centro del aglomerado. El error relativo en el interior de la esfera que contiene las partículas es menor que $8\text{x}10^{-3}$ y en el exterior de la esfera es menor que $2\text{x}10^{-3}$.

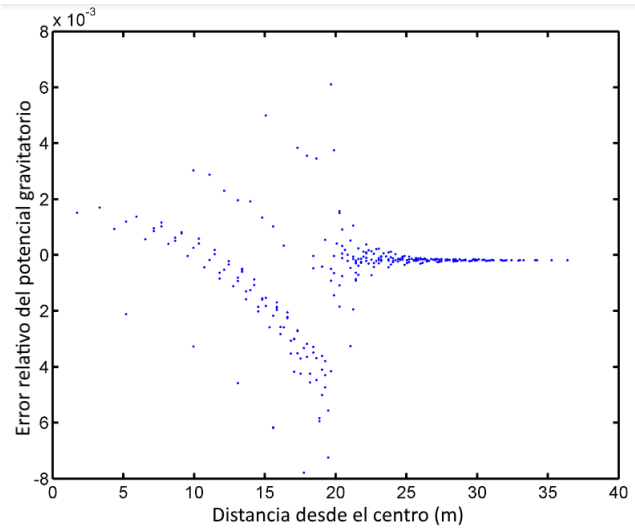


Figura 6.4: Error relativo del potencial gravitatorio computado respecto al valor teórico en función de la distancia al centro del aglomerado.

Para estudiar la eficiencia computacional del algoritmo, se analizó el tiempo de ejecución de los distintos escenarios al utilizar 1, 3, 6, 9, 12, 15, 18, 21 y 24 hilos de ejecución. La figura 6.5 muestra el tiempo de ejecución en función del número de hilos. Una función potencial inversa se puede ajustar a cada curva.

El speed-up y los valores de eficiencia son reportados en las siguientes figuras respectivamente. Para el sistema de más de un millón de partículas y usando hasta 24 hilos, el cálculo tomó cerca de 100s mientras que el cálculo secuencial demanda 1100s. Usando 20 hilos obtuvimos un speedup de factor 10 y una eficiencia en el rango de 0.4-0.8, dependiendo del número de partículas usadas. Para dicho conjunto de más de un millón de partículas, obtuvimos un speedup cerca del lineal, pero la eficiencia se redujo para largos conjuntos.

Además también analizamos la dependencia de los tiempos de ejecución con el número de partículas cuando se varían la cantidad de hilos utilizados en el procesamiento. Los resultados se muestran en la siguiente figura.

Las curvas se pueden ajustar a una función polinomial, que corresponde a un valor casi constante del tiempo de ejecución normalizado con respecto al número de partículas, como se observa en la figura siguiente para un número dado de hilos y más de 210^5 partículas. Teniendo en cuenta que la dependencia con el número de hilos (n) se puede ajustar a una función de ley potencial inversa y el número de partículas (p) con una funciona polinomial lineal, se puede ajustar el tiempo de ejecución con $t = a + b.p.n^{-}c$ usado coeficientes $a = 13{,}65, b = 1{,}049x10^{-}3yc = 0{,}8682$.

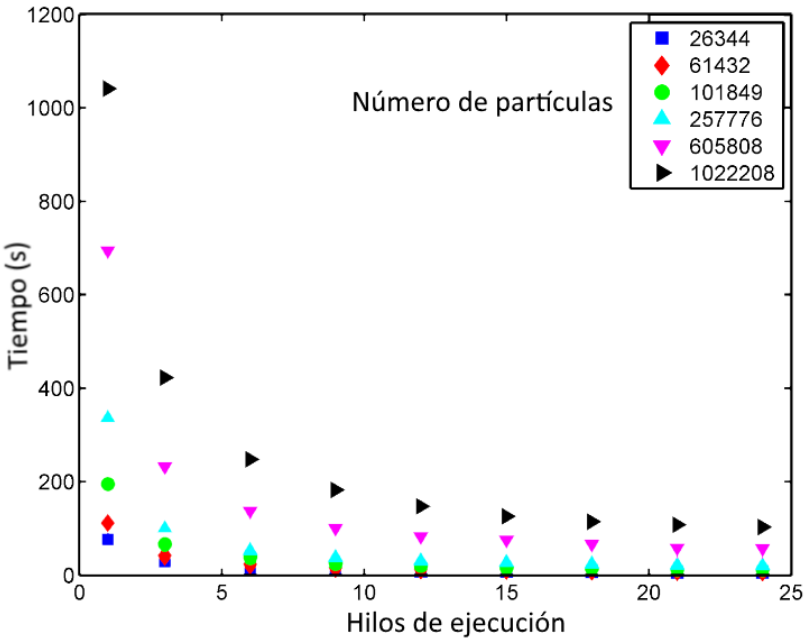
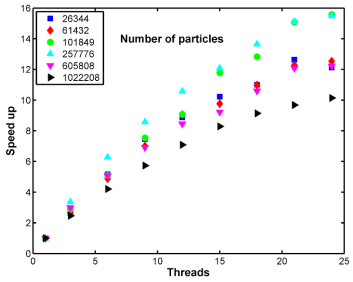
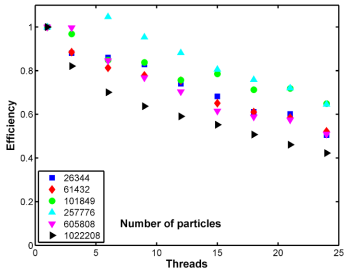
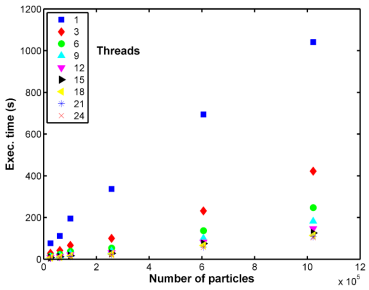
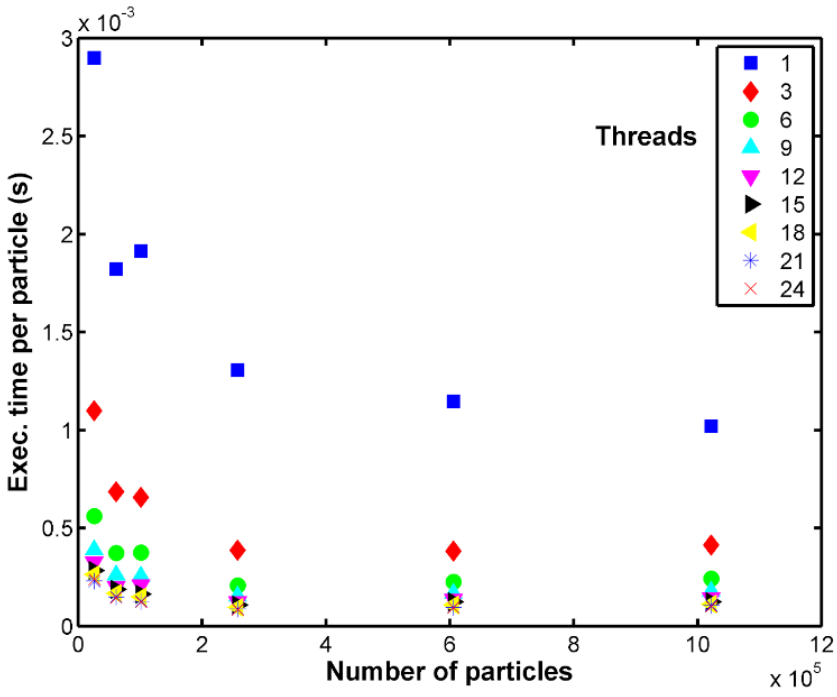


Figura 6.5: Tiempo de ejecución en función de la cantidad de hilos para los distintos escenarios.






Threads
1
3
6
9
12
15
18
21
24
Exec. time (s)
Number of particles
x 10^5


x 10^-3
Threads
1
3
6
9
12
15
18
21
24
Exec. time per particle (s)
Number of particles
x 10^5

strategy	*execution time (s)*	
	average	σ
Interlocking linear	186.2	4.6
Circular concentric	198.3	10.6
Basic isolated linear	254.0	19.2
Advanced isolated linear	180.1	2.7

Table 1. Execution time analysis of the proposed strategies for workload distribution in the parallel algorithm.

6.3.1. Evaluación de las estrategias de asignación de trabajo

Los distintos algoritmos de asignación de nodos a recursos de cómputo fueron evaluados sobre distintos modelos de aglomerados realistas. Se trabajo con partículas de distintos tamaños encerradas en una esfera mayor. Se definieron tres escenarios para evaluar las estrategias de paralelización implementadas:

- escenario A: contiene 17621 partículas en una esfera de radio 10 m.
- escenario B: contiene 148435 partículas en una esfera de radio 20m.
- escenario C: contiene 1218024 partículas en una esfera de radio 40m.

Para evaluar la eficiencia computacional de las diferentes estrategias de distribución de trabajo, realizamos pruebas utilizando el escenario C y 12 hilos de ejecución en paralelo. La tabla siguiente muestra los promedios y la desviación estándar, para cinco ejecuciones independientes por cada estrategia.

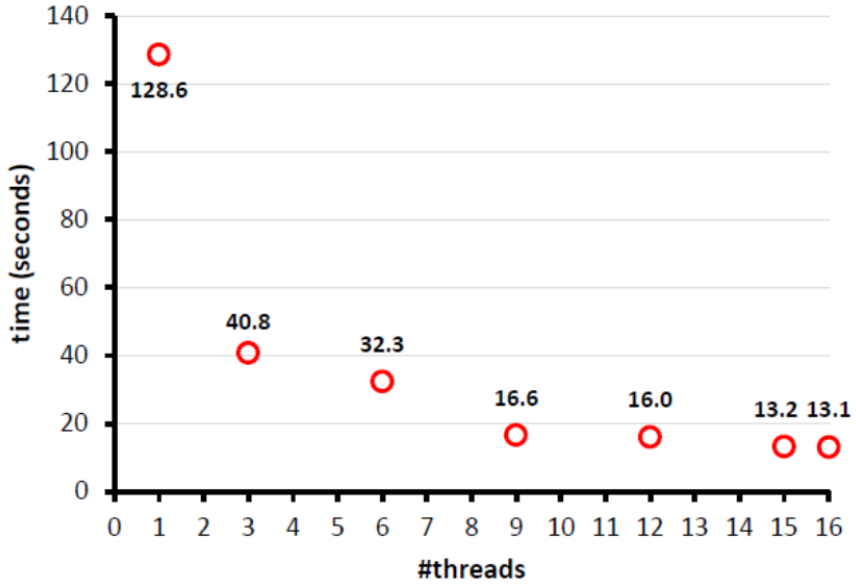
Para los resultados mostrados en la tabla anterior, concluimos que la estrategia “Advanced isolated linear” logra obtener la mejor performance para el escenario trabajado. Es además la más robusta, considerando que es la que arroja menor desviación estándar. La estrategia “Interlocking linear” es la segunda mejor opción. La estrategia circular concéntrica no toma ventaja de los valores pre calculados, y por el contrario la compleja lógica incluida para manejar la sincronización de hilos y la distribución circular hace que la estrategia sea menos eficiente que la estrategia de “Interlocking linear”. La estrategia “Basic isolated linear” es la que peor se comporta, fundamentalmente porque la carga de trabajo no está balanceado entre los distintos hilos. En esta estrategia, la performance del algoritmo en su totalidad está limitada por la performance obtenido por el hilo que más tiempo demore. Al incluir la metodología de dos etapas, con la estrategia “Advanced isolated linear” se logra reducir el tiempo considerablemente.

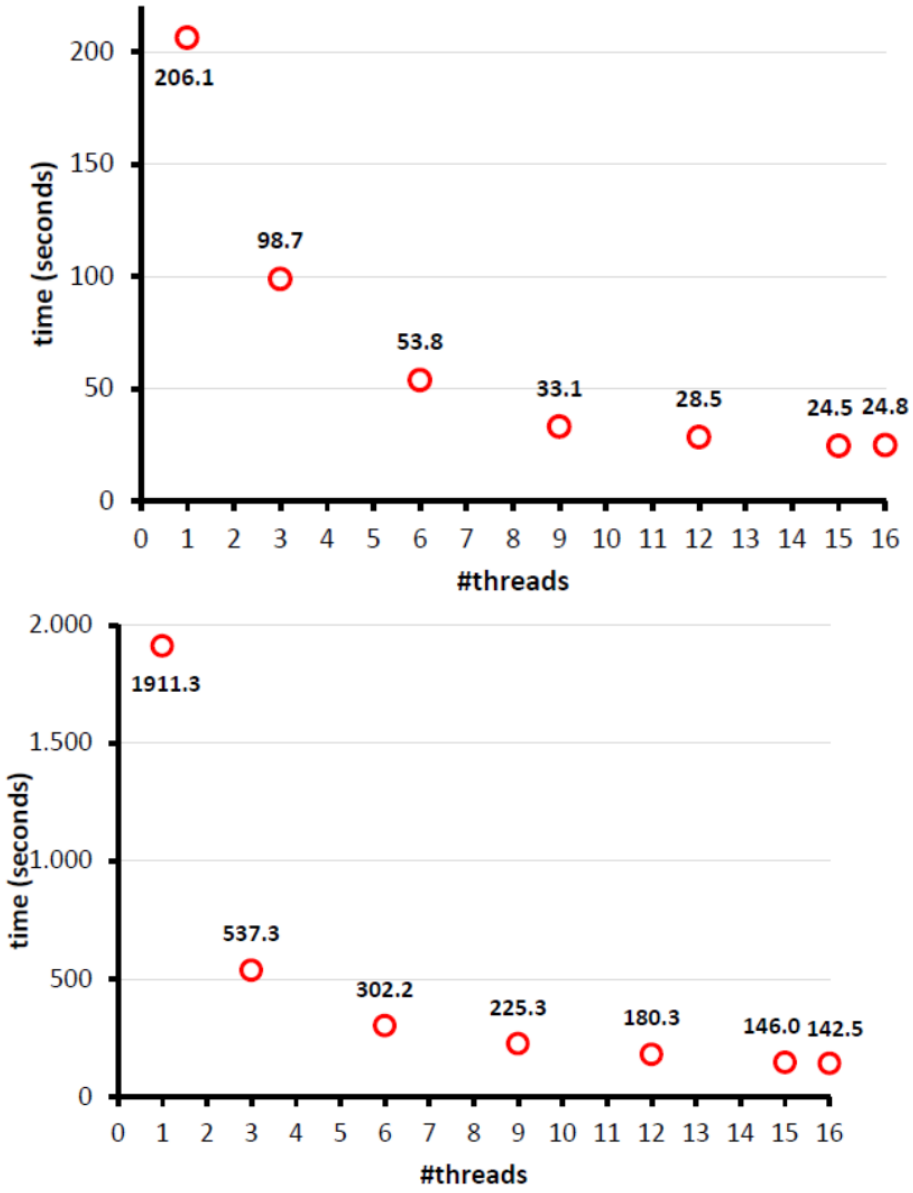
La tabla siguiente muestra la eficiencia computacional (tiempo de ejecución y speedup) para la estrategia "Advanced isolated linear" utilizando diferente cantidad de hilos y para los tres escenarios estudiados. El promedio y la desviación estándar resultante son calculados a partir de seis ejecuciones independientes sobre cada escenario para cada hilo.

scenario	*radius (m)*	*#particles*	*#threads*	*time (s)*	*speedup*
A – small	10	17621	1	128.6±2.8	1.0
			3	40.8±6.0	3.2
			6	32.3±1.2	4.0
			9	16.6±1.0	7.7
			12	16.0±0.4	8.0
			15	13.2±1.0	9.7
			16	13.1±1.6	**9.8**
B – medium	20	148435	1	206.1±1.3	1.0
			3	98.7±1.7	2.1
			6	53.8±0.7	3.8
			9	33.1±0.9	6.2
			12	28.5±0.7	7.2
			15	24.5±0.9	**8.4**
			16	24.8±1.4	8.3
C – large	40	1218024	1	1911.3±9	1.0
			3	537.3±0.8	3.6
			6	302.2±3.9	6.3
			9	225.3±3.4	8.5
			12	180.3±3.1	10.6
			15	146±1.6	13.1
			16	142.5±3.8	**13.4**

Table 2. Experimental results: execution time and speedup of the multithreading algorithm using different number of threads, for the three studied problem scenarios.

Las siguientes figuras muestran la ejecución para cada escenario cuando se varía el número de hilo.





Al utilizar 16 hilos se logra para el escenario más grande un speedup de factor 13.4, lo cual se corresponde con una eficiencia computacional de 0.85. El comportamiento del speedup para el algoritmo paralelo propuesto es casi linear. Además, el mejor speedup/eficiencia se obtuvo para la simulación que manipula la mayor cantidad de partículas, demostrando un buen comportamiento de escalabilidad. Simulaciones de más de un millón de partículas, que demanda 1920s (más de media hora) pueden realizarse en dos minutos y medio utilizando 16 hilos.

Capítulo 7

Conclusiones, mejoras y trabajos a futuro

7.1. Conclusiones

En el contexto de este proyecto se diseñó e implementó un algoritmo de memoria compartida totalmente paralelo para calcular de forma eficiente el potencial gravitacional de un aglomerado de granos. El método propuesto es adecuado para los cálculos de autogravedad en los sistemas de partículas densamente empaquetados. El algoritmo escala de forma lineal con el número de partículas del sistema, y escala con una ley de potencia inversa de exponente 0.87 con el número de hilos utilizados en el cálculo. El comportamiento del speedup está cerca de ser lineal para sistemas de 210^5 partículas.

Se analizaron diferentes estrategias de paralelización para calcular de forma eficiente y escalable, escenarios de simulación realistas en tiempos de ejecución reducidos. Se realizaron evaluaciones experimentales de las diferentes estrategias, surgiendo que la estrategia denominada "Advanced isolated lineal" utilizando diferentes comienzos fue la más eficiente de las estudiadas. Esta técnica aplica un procesamiento en dos etapas, reduciendo la complejidad en la sincronización entre hilos mientras que se reducen demoras debido al desbalance. El código implementado se integró satisfactoriamente en el paquete de simulación ESyS-Particle en una fructífera colaboración con los desarrolladores del mismo Steffen Abe y Dion Whesley. Actualmente el código integrado se encuentra en etapa de testing en un repositorio privado, por lo aún no se encuentra a disposición para ser descargado por la comunidad. El objetivo es poder integrar el código en la rama principal del paquete ESyS-Particle en un futuro cercano.

7.2. Mejoras realizadas

Con la visita de Dion Weatherley, se cursó una asignatura dictada por él, en la cual se realizaron mejoras al código anterior desarrollado. Una mejora propuesta fue no realizar el cálculo de auto-gravedad en aquellas celdas en las cuales no se tenían partículas. Esto implico integrar contra el núcleo de ESyS-Particle para obtener aquellas celdas ocupadas. Este cambio permitió acelerar el calculo de auto-gravedad en hasta 50 veces.

Por dentro del trabajo de maestría de Nestor Rochetti se tomó el código desarrollado como base y se propusieron diversas mejoras. Una de las mejoras implementadas consistió en mejorar el proceso de interpolación agregando un componente predictivo tomando en cuenta la aceleración calculada en la etapa de autog-gravedad. Esto permitió corregir errores intrínsecos en el propio diseño de calculo de auto-gravedad solo cada ciertos pasos de simulación. Otra mejora propuesta fue la implementación del algoritmo de Barnes-Hut para determinar y mejorar la elección de cuales partículas se toman en cuenta al calcular la auto-gravedad [5]

7.3. Trabajos a futuro

Como trabajo a futuro se propone mejorar el algoritmo evolutivo para así alcanzar los umbrales perseguidos inicialmente. Esto implica invertir tiempo en estudiar mejor el estado del arte de la temática y volcar la experiencia acumulada en una re-estructuración del código desarrollado.

Respecto al método de interpolación utilizado consideramos que los pendientes visualizados fueron atacados en el proyecto de maestría de Nestor R.

Para integrar efectivamente el código desarrollado en la rama principal del paquete ESyS-Particle, es necesaria una revisión del código desarrollado con la finalidad de cumplir con ciertos estándares de código.

Bibliografía

[1] *Proyectos premiados 2013*, https://www.fing.edu.uy/node/9577 [citado Enero 2018]

[2] *Computing of Self-gravity for Small Solar System Bodies*, http://sbd14.sciencesconf.org, SBD2014, Small Bodies Dynamics 2014 [citado Enero 2018]

[3] D. Frascarelli, S. Nesmachnow and G. Tancredi, *High-Performance Computing of Self-Gravity for Small Solar System Bodies*, http://ieeexplore.ieee.org/xpl/articleDetails.jsp?arnumber=6898782

[4] S. Nesmachnow, D. Frascarelli and G. Tancredi, *A parallel multithreading algorithm for self-gravity calculation on agglomerates*, https://link.springer.com/chapter/10.1007/978-3-319-32243-8_22

[5] N. Rocchetti, D. Frascarelli, S. Nesmachnow and Gonzalo Tancredi, *Performance improvements of a parallel multithreading self-gravity algorithm*, https://link.springer.com/chapter/10.1007/978-3-319-73353-1_21

[6] R. Hockney and J. Eastwood, *Computer Simulation Using Particles* New York: McGraw-Hill, 1981.

[7] A. W. Appel, *An efficient program for many-body simulation* SIAM J. Sci. Stat. Comp., vol. 6, no. 1, pp. 85–103, 1985.

[8] J. Barnes and P. Hut, *A hierarchical O(N log N) force-calculation algorithm* Nature, vol. 324, no. 6096, pp. 446–449, 1986.

[9] D. Richardson, *Tree Code Simulations of Planetary Rings*, Mon. Not. Roy. Astron. Soc., vol. 269, p. 493, 1994.

[10] D. Richardson, P. Michel, K. Walsh, and K. Flynn, *Numerical simula¬tions of asteroids modelled as gravitational aggregates with cohesion*, Planet. & Space Science, vol. 57, pp. 183–192, 2009.

[11] G. Tancredi, A. Maciel, L. Heredia, P. Richeri, and S. Nesmachnow, *Granular physics in low-gravity environments using discrete element method*, Mon. Not. Roy. Astron. Soc., vol. 420, pp. 3368–3380, 2012.

[12] D. Sánchez and D. Scheeres, *DEM simulation of rotation-induced reshaping and disruption of rubble-pile asteroids*, Icarus, vol. 218, pp. 876–894, 2012.

[13] IntÍ Astronomical Union, *Definition of a Planet in the Solar System*, Resolution B5 of the XXVIth IAU General Assembly, 2006; `http://www.iau.org/static/resolutions/Resolution_GA26-5-6.pdf`.

[14] *Planets, Dwarf Planets and Small Solar System Bodies*, `https://www.iau.org/public/themes/pluto`

[15] E. Asphaug, *The Shifting Sands of Asteroids*, *Science*, vol. 316, pp. 993–994, 2007.

[16] A.W. Harris, *The Rotation Rates of Very Small Asteroids: Evidence for 'Rubble Pile' Structure*, abstract no. 1247, Abstracts of Papers Submitted to the 27th Lunar and Planetary Science Conf., 1996, pp. 493–494.

[17] M. Luciuk, *Roche Limit: Why Do Comets Break Up?*, `http://www.asterism.org/tutorials/tut25-1.htm`.

[18] B. Meinke, *When Comets Crack*, `http://hubblesite.org/hubble_discoveries/comet_ison/blogs/when-comets-crack`.

[19] N. Roccheti, *Parallel multithreading algorithms for self-gravity computation in ESyS-Particle*, `https://www.colibri.udelar.edu.uy/jspui/bitstream/20.500.12008/27171/1/ROC20.pdf`.

[20] A A Mahmood, M Elektorowicz, *A Review of Discrete Element Method Research on Particulate Systems*, `https://iopscience.iop.org/article/10.1088/1757-899X/136/1/012034/pdf`.

[21] Cundall, P. and Strack, O, *A discrete numerical model for granular assemblies*, `https://www.icevirtuallibrary.com/doi/10.1680/geot.1979.29.1.47`.

[22] Ricardo Dobry, Tang-Tat Ng *Discrete modeling of stress-strain behavior of granular media at small and large strains*, `https://www.researchgate.net/publication/235300289_Discrete_modeling_of_stress-strain_behavior_of_granular_media_at_small_and_large_strains/link/55e7215708ae3e1218420404/download`.